Universidad Nacional Autónoma de México

Posgrado en Ciencia e Ingeniería de la Computación

Sistema de visión para el reconocimiento y seguimiento de objetos basado en la extracción de características naturales

T E S I S

que para obtener el grado de:

Maestro en Ingeniería
(Computación)

P R E S E N T A

Javier Jiménez Juárez

Director: Dr. Jesús Savage Carmona

Ciudad Universitaria, México D.F., 2007

Quienquiera que seas, me temo que caminas por los caminos de los sueños,
Me temo que estas supuestas realidades están destinadas a desvanecerse bajo tus pies y tus manos,
Incluso ahora tus rasgos, alegrías, idiomas, casa, negocio, problemas, locuras, disfraz, delitos, se disipan de ti,
Tu cuerpo y alma verdaderos aparecen ante mí,
Abandonan los asuntos, el comercio, las tiendas, el trabajo, las granjas, la ropa, la casa, la compra, la venta, la comida, la bebida, el sufrimiento, la muerte.

Quienquiera que seas, ahora pongo mi mano sobre ti, para que seas mi poema,
Te susurro con mis labios al oído,
Que he amado a muchas mujeres y hombres, pero a nadie tanto como a ti.

Oh, he tardado mucho tiempo y estuve mudo,
Tendría que haber ido directamente a ti desde el principio,
Tendría que haber hablado sólo de ti, tendría que haberte cantado sólo a ti.

Dejaré todo y vendré a componer tus himnos,
Nadie te ha entendido, pero yo te entiendo,
Nadie te ha hecho justicia, no te has hecho justicia a ti mismo,
Nadie ha dejado de verte imperfecto, yo soy el único que no te ve imperfecciones,
Todos han querido subordinarte, yo soy el único que nunca aceptará subordinarte,
Yo soy el único que no pone un amo sobre ti, un dueño, un superior, Dios, más allá de lo que espera intrínsecamente en ti.

Los pintores han pintado sus nutridos grupos y la figura central de todo,
De la cabeza de la figura central se desprende una aureola de luz dorada,
Pero yo pinto miríadas de cabeza, y no hay ninguna sin su aureola de luz dorada,
Mi mano la hace emanar de la cabeza de todos los hombres y las mujeres, resplandeciente por siempre jamás.

¡Oh, podría cantar tantas grandezas y glorias sobre ti!
No supiste lo que eras, has estado durmiendo profundamente toda tu vida,
Tus párpados han estado cerrados la mayor parte del tiempo,
Tus acciones regresan ahora con burlas,
(Tu frugalidad, tu conocimiento, tus oraciones, ¿para qué vuelven sino para burlarse de ti?)

Tú no eres las burlas,
Dentro y debajo de ellas te veo agazapado,
Te persigo hasta donde nadie te ha perseguido,
El silencio, el escritorio, la expresión petulante, la noche, la acostumbrada rutina, aunque te

oculten de los demás o de ti mismo no te ocultan de mí,
El rostro afeitado, la mirada insegura, la tez impura, aunque engañen a los demás, no me engañan a mí,
La ropa atrevida, la actitud deforme, la embriaguez, la codicia, la muerte prematura, las hago a un lado.

No hay cualidad en el hombre o la mujer que tú no poseas,
No hay virtud ni belleza en el hombre o la mujer, que no sea tan buena en ti,
Ni valor ni resistencia en los demás, que no sea tan buena en ti,
Ni placer que espere a los demás y no te espere a ti igualmente.

En cuanto a mí, no doy nada a nadie si no te doy a ti el mismo cariño.

No entono los cantos de la gloria de nadie, ni de Dios, sin antes entonar los cantos de tu gloria.

¡Quienquiera que seas! ¡reclama lo tuyo como sea!
Estos paisajes del Este y del Oeste son sosos comparados contigo,
Estas inmensas praderas, estos ríos interminables, tú eres tan inmenso e interminable como ellos,
Estas furias, elementos, tormentas, movimientos de la Naturaleza, agonía de aparente disolución, tú eres, hombre o mujer, el señor o la señora por encima de ellos,
Señor o señora en tu propio derecho sobre la Naturaleza, los elementos, el dolor, la pasión, la disolución.

Las ligaduras caen de tus tobillos, y descubres una suficiencia inagotable,
Viejo o joven, varón o hembra, tosco, humilde, rechazado por los demás, lo que tú eres se promulga a sí mismo,
A través del nacimiento, la vida, la muerte, el entierro, los medios te son proporcionados, nada es escaso,
A través de las iras, las pérdidas, la ambición, la ignorancia, el aburrimiento, lo que tú eres se abre paso.

A ti, Walt Whitman

Dedicatoria

A la U.N.A.M., mi alma mater y mi casa.
A mis padres, Rodolfo Jiménez y Dominga Juárez.
A Rodolfo, Rocío y mis sobrinos Rodrigo, Emilio y Diego.
Y muy especialmente a ti.

Agradecimientos

Una persona nunca es suficiente para elaborar un trabajo de esta magnitud. Mucha gente es requerida para salir avante en cada una de las metas y obstaculos encontrados. Por ello, quiero agradecer a todos aquellos que me ayudaron, pero en particular. . .

A la Universidad Nacional Autónoma de México, por el privilegio de ser parte de ella.

A CONACYT por el apoyo económico recibido durante mis estudios.

Al personal del posgrado en computación por ser más que eso, por ser amigos. Gracias Amalia, Lulú y Diana.

A Gerardo y Adalberto, compañeros en el laboratorio de Bio-Robótica que me ayudaban a comprender y atacar los problemas mediante su experiencia, conocimientos y diferentes puntos de vista.

A Emmanuel Hernández quien siempre ha demostrado ser un buen amigo, compañero y profesionista. Gracias a sus valiosas aportaciones en el tiempo que tengo de conocerle, he podido refinar y aprender diversas técnicas.

A Uriel Nava, amigo y compañero desde la licenciatura. Gracias por la compañía, consejos y esa sincera amistad.

Al Dr. Jesús Savage Carmona, mi director de tesis y tutor desde la licenciatura. Agradezco la confianza que ha depositado en mi y la estima en que me tiene.

Finalmente, agradezco a todos mis amigos y familiares que siempre se han preocupado e interesado por mis avances como profesionista y ser humano: Helia, Ishtar, Dalia, Iván, Uriel, Juan Monter, Juan Eduardo, Miguel Angel, Alejandro, Gabriel y Carlos Soster. Con su constante apoyo e interés fue más fácil completar esta tarea.

Índice general

Índice de figuras

Índice de cuadros

Resumen

En este trabajo se presenta el desarrollo e implementación de un sistema de visión por computadora, para la detección y rastreo de objetos basándose en técnicas de extracción de características naturales. Consta de una revisión del estado del arte de las técnicas empleadas para realizar tales fines, de una discusión acerca de los algoritmos y el algoritmo que se utilizó, así como una completa explicación de su funcionamiento y sus diferentes implementaciones programadas.

El objetivo principal fue desarrollar un conjunto de bibliotecas de programación *propias* que permitieran realizar operaciones de reconocimiento de patrones y el seguimiento de los mismos, de forma que fueran herramientas útiles para las personas que desarrollan aplicaciones y hacen investigación en el área de la visión por computadora.

Se refieren como propias pues no se hace uso de código ajeno al que se creó para esta tesis. Es así que la integración con sistemas y aplicaciones es más simple, pues no hay que buscar dependencias ni estar lidiando con distribuidores o licencias. Es así que buena parte de las herramientas está dedicada a la implementacíon de tipos de datos básicos que se relacionan con el soporte de carga de imágenes y las operaciones que puedan realizarse con ellas, sólo por mencionar un ejemplo.

Por otro lado, es importante denotar la valía del algoritmo utilizado para la extracción de características naturales: SIFT. Se trata de una de las transformaciones más probadas y utilizadas para detectar puntos de interés y utilizarlos en tareas de reconocimiento y rastreo. Fue desarrollada por David Lowe en la Universidad de British Columbia y bautizada de tal modo por su nombre completo: *Scale Invariant Feature Transformation*.

Dentro de la evaluación del estado del arte, realizada, no se encontraron muchas implementaciones del algoritmo, o al menos que fueran libres o utilizables. Por lo tanto, la implementación aquí presentada es importante, pues le da valor extra al sistema desarrollado. Además, el código será liberado para que la gente aporte en ideas, mejoras y optimizaciones al mismo.

Las bibliotecas fueron desarrolladas en lenguaje C, C++ y C#, dándole variedad a la implementación y abriéndola a más usuarios. Puede ser ejecutada desde modo de línea de comandos y procesar con imágenes previamente adquiridas, o bien, adquirirlas en tiempo real mediante un dispositivo de video y procesarlas de modo inmediato. Asimismo cuenta con una variante multiprocesos (*pipeline*) que utiliza al máximo las capacidades del equipo de cómputo y ofrece una mejora en el rendimiento y velocidad de aproximadamente 30 %.

Los resultados mostraron ser buenos en términos generales. Se puede cargar fácilmente una imagen, obtener su transformada SIFT, hacer reconocimiento contra otras imágenes y encontrar una transformación geométrica entre ellas. Para las pruebas se utilizó un conjunto reducido de objetos, aproximadamente 6 y con ellos hubo variaciones en la calidad de los resultados. Para objetos de manufactura industrial, como cajas, el algoritmo resultó ser muy bueno bajo rotaciones y traslaciones en su plano, y con errores cuando las transformaciones fueron en profundidad, es decir, produjeron un cambio en la perspectiva.

Como trabajo futuro queda realizar diversas métricas para evaluar con mayor precisión el rendimiento de las bibliotecas y compararla contra otras versiones del algoritmo SIFT.

En conclusión, el objetivo planteado al inicio fue cumplido y los resultados han probado el correcto funcionamiento del desarrollo realizado.

Introducción

No se puede concebir al mundo actual sin las tareas desempeñadas por los sistemas computarizados: cajeros electrónicos para la banca, máquinas registradoras y sistemas de control de inventario para empresas de diversas magnitudes, comunicación vía Internet y correo electrónico, sistemas de navegación, supervisión y asistencia computarizada son algunos de los problemas resueltos por las computadoras.

Sin embargo, siguen existiendo tareas dependientes completamente del hombre, muchas de ellas que son resueltas con base en el sentido más utilizado por el ser humano: la vista. En el afán de continuar automatizando procesos, hacerlos más eficientes y relegar tareas más *importantes* al hombre, los científicos han intentado dotar a las máquinas de visión. La tarea no ha resultado ser nada fácil ni simple y gracias a ello se ha desarrollado un área que continúa en expansión: la de la visión por computadora.

La meta de la visión por computadora es proveer a las computadoras con capacidades de percepción *humanas*, de forma que puedan sensar el ambiente, tomar decisiones apropiadas y aprender de experiencias para mejorar el rendimiento futuro.

En los últimos años ha habido una demanda significativa de sistemas de visión por computadora para resolver problemas del mundo real. Sin embargo, la mayoría de los modelos y metodologías no van más allá de resolver dominios muy simples, *de juguete* por catalogarlos de un modo. Por lo tanto, el estado del arte actual en

visión, necesita de avances significativos para tratar aplicaciones del mundo real como navegación autónoma, reconocimiento de objetos, manufactura, interpretaciones de fotografías, sensado remoto, etc. Es bien sabido que este tipo de sistemas requieren de algoritmos robustos que funcionen bajo oclusiones parciales, recorte en la escena vista, bajo contraste y condiciones ambientales no controladas —como la iluminación u otro tipo de factores que interfieran con el sensado de la imagen—.

Entre los campos de la visión por computadora, una de las principales tareas consiste en reconocer objetos, como ya se mencionó. En ocasiones es necesario determinar en que lugar está el objeto buscado y rastrearlo. ¿Cómo se puede resolver esto? Para los humanos es relativamente fácil e incluso trivial esta tarea. Existe un conocimiento previo de geometría y descripción de forma, pero para un sistema de cómputo no resulta serlo.

Una imagen posee muchas características de donde extraer información. La primera de ellas ya se comentó y es la forma de los objetos que contiene. El color y la textura son también un par de características importantes, pero no bastan para clasificar objetos. Los algoritmos de segmentación son ampliamente utilizados y han sido una buena aproximación para determinar características o clasificar por medio de color/textura. Sin embargo, para la forma de los objetos, los estudios son limitados.

Como características de forma, pueden considerarse puntos, líneas (rectas o contornos), o geometrías compuestas por conjuntos de las anteriores. Es así que la herramienta fundamental para el análisis de sistemas de visión por computadora resulta ser el procesamiento digital de imágenes. Se trata de un área científicamente formal y algorítmica que provee de las herramientas necesarias para intentar obtener una semántica de la imagen. Los algoritmos descritos por los investigadores de procesamiento digital de imágenes (*PDI*) conforman la base de cualquier otro algoritmo o técnica empleada para fines de un nivel mayor.

Entre las características que se pueden buscar en una imagen para definir objetos y rastrearlos están aquellas que son naturales o artificiales. Las segundas tienen que ver con *marcas* que son superpuestas en los objetos reales y que ayudan a la identificación de ellos mediante la extracción de ciertos atributos simples como pueden ser líneas o contornos cerrados de figuras geométricas.

Cuando se utiliza el término *característica natural*, se hace referencia a aquellos elementos que componen la imagen del objeto en cuestión, que le pertenecen y siempre aparecerán en él.

Las técnicas basadas en estos atributos también toman como base al *PDI* pero tienen su propio campo de estudio, lo que las convierte en elementos de continua expansión, con nuevas vertientes día a día. Lo importante de estos análisis está en la capacidad de extraer suficiente información para describir una imagen o para reconocer objetos sin incorporar información extra con el motivo de facilitar la detección.

Entre algunas de las aplicaciones que más hacen uso de tareas de reconocimiento y rastreo están las de la manufactura/automatización, robótica y realidad aumentada, por mencionar algunas.

En el laboratorio de Bio-Robótica de la Facultad de Ingeniería, se trabaja en las dos últimas áreas descritas. Una de las necesidades con que se cuenta actualmente es la de reconocer objetos y rastrearlos en ambientes reales, donde puede existir recorte del objeto en la imagen, o estar ocluido. Por ello, la necesidad de trabajar con algoritmos que sean robustos a este par de situaciones y que no necesiten incorporar objetos extra en la escena.

Objetivos

La presente tesis pretende ser la base de futuros desarrollos en el área de la visión por computadora u otra área de aplicación. Como ya se mencionó, actualmente se requieren de sistemas basados en visión en el laboratorio de Bio-Robótica para las aplicaciones ahí desarrolladas.

La base de funcionamiento para los dos campos mencionados (robótica y realidad aumentada) necesitan obtener información a partir de una imagen de entrada que sense el entorno real. Principalmente, lo que se necesita es reconocer objetos, y rastrearlos haciendo su respectivo registro geométrico con la cámara que les observa. El proceso después de este paso inicial, depende de la aplicación. En el caso de robótica puede tratarse de planeación de rutas o toma de decisiones, mientras que para realidad aumentada alinear modelos virtuales que provean de información adicional a la escena/objeto vistos.

Por ello, para esta tesis, el objetivo que se tiene es **desarrollar un conjunto de bibliotecas de programación propias, que permitan realizar operaciones de reconocimiento de patrones y seguimiento de los mismos** para su posterior uso en las aplicaciones ya comentadas.

Al decir *biblioteca de programación propia*, se pretende referir al no uso de código ajeno, para una integración completamente independiente de la aplicación y de los sistemas donde se requiera. Por ello se han creado las rutinas y tipos de datos necesarios para darle soporte a la carga de imágenes y poder operar con ellas dentro del espectro del problema a resolver.

Por otro lado, también es importante denotar la valía del algoritmo utilizado para la extracción de características naturales: SIFT. Se trata de una de las transformaciones más probadas y utilizadas para detectar puntos de interés y utilizarlos en tareas de reconocimiento y rastreo. Fue desarrollada por David Lowe en la Universidad de British Columbia y bautizada de tal modo por su nombre completo: *Scale Invariant Feature Transformation*. Dentro de la evaluación del estado del arte, realizada, no se encontraron muchas implementaciones del algoritmo, o al menos que fueran libres o utilizables. Por lo tanto, la implementación aquí presentada es importante, pues le da valor extra al sistema desarrollado. Además, el código será liberado para que la gente aporte en ideas, mejoras y optimizaciones al mismo.

Organización de la tesis

La tesis está dividida en cinco capítulos:

- El **capítulo 1** ofrece un panorama del área de la visión por computadora, sus aplicaciones y los temas tratados a lo largo de este trabajo. También se comienza el tratamiento sobre características naturales.
- En el **capítulo 2** se hace una revisión del estado del arte de las técnicas usadas en sistemas de visión para el reconocimiento y rastreo de objetos. Luego se revisa uno que quizás sea el más empleado en aplicaciones de realidad aumentada, el ARToolKit. Finaliza con las técnicas basadas en características naturales y la vertiente utilizada en el trabajo.
- El **capítulo 3** explica en detalle el funcionamiento del algoritmo SIFT, puesto que es necesario comprender su funcionamiento para una posterior implementación.
- El contenido del **capítulo 4** describe la elaboración de la biblioteca de programación de inicio a fin, los tipos de datos creados y utilizados, la implementación de cada fase del algoritmo, su encapsulamiento para un modelo orientado a

objetos y ejecución *multihilos*, y como fue llevado a la plataforma .NET para su integración con C#.

- Finalmente, en el **capítulo 5** se discuten las pruebas realizadas al sistema, sus aciertos, errores, así como la formulación de hipótesis de las fallas.

Al final se incluyen las conclusiones y trabajo futuro con respecto del trabajo, así como cuatro apéndices. Primero (apéndice A) se muestra un pequeño resumen acerca de como obtener las derivadas de una imagen. El apéndice B ilustra con un ejemplo, una posible forma de codificar con el *software* elaborado. El tercero, apéndice C, trata con las formalidades de los espacios y transformaciones afines, mientras que el último muestra las imágenes producidas por las pruebas (apéndice D).

Capítulo 1

Visión por computadora y sus aplicaciones

Como resultado de los recientes avances tecnológicos en materia de cómputo, hoy en día se tienen sistemas poderosos formados por *CPUs* con gran densidad de componentes. El equipo de super cómputo de hace unos años (memoria RAM del orden de *gigabytes* y discos duros de gran capacidad) es ahora el *hardware* común de las computadores personales.

Tales avances han producido un vertiginoso desarrollo de *software*, antes difícil de implementar debido a las limitaciones técnicas. Entre las aplicaciones que demandan mayor consumo de recursos, se encuentran aquellas referentes al entretenimiento, el cómputo científico y procesos de automatización. La industria del entretenimiento es de las que más ha propiciado el desarrollo tecnológico. Tan sólo basta con mirar hacia la creciente y ya enorme industria de las gráficas por computadora. En ella pueden considerarse a productos como juegos de video y utilerías de postproducción utilizadas en televisión y cine: *game engines*, recorridos virtuales, diseño asistido

por computadora, aplicaciones de procesamiento de imágenes y video (en etapas *online* y *offline*), entre otras. En la misma industria, en el área de la ciencia e investigación, destacan las aplicaciones empleadas para hacer procesamiento de señales, cálculo numérico, simulaciones, aplicaciones médicas, de ingeniería y militares, sólo por nombrar algunas.

Para el caso particular del estudio aquí presentado, el ámbito que más importa es el de la visión por computadora con un enfoque claro a la detección de características naturales (presentadas en el capítulo 2) y del desarrollo de una biblioteca de programación que permita extraer puntos de interés, para su posterior uso en tareas de reconocimiento y rastreo de objetos.

1.1. Visión por computadora

Como tal, encontrar una definición completa y concisa de lo que es la visión por computadora (*CV* de sus siglas en inglés *Computer Vision*) es muy complicado. De manera general, se puede considerar a la *CV* como *la ciencia y tecnología de las máquinas que ven* [Wik07] y cuya meta es la toma de decisiones a partir de los objetos que componen una escena real tomando como base la descripción de la misma [SS00]. Una buena definición es la que se puede encontrar en [TV98]:

> **Definición 1** *La tarea principal de la* CV *consiste en obtener las propiedades del entorno real a partir de un conjunto de imágenes digitales. Tales características pueden ser geométricas (forma y posición en objetos sólidos) y/o dinámicas (velocidades de los objetos). La mayoría de las soluciones asumen que se cuenta con los parámetros adecuados (obtenidos mediante algoritmos de procesamiento de imágenes) para efectuar los cálculos.*

El intento por hacer máquinas que vean proviene de la década de 1960, cuando los expertos en inteligencia artificial lo consideraban una tarea realizable por estudiantes como parte de su proyecto de verano. Cuarenta años después, la tarea aún no es resuelta, y ha demostrado ser un tema formidable y lleno de retos. Una de las principales causas de la no obtención de resultados claros por los pioneros, consistió en el afán de considerar un vínculo directo entre los algoritmos de procesamiento y del sistema a desarrollar, con la forma de operación del sistema biológico humano, llegando incluso a tratar de imitarlo [ZH03].

A pesar de ello, la investigación en *CV* ha producido sorprendentes avances. Del lado de la práctica está, por ejemplo, la posibilidad de guiar remotamente a vehículos no tripulados. Esto requiere de técnicas capaces de analizar escenas tridimensionales en tiempo real, que son elaboradas y costosas en lo que a capacidad de cómputo se refiere. En téminos de teoría, nuevas ramas han aparecido como respuesta al deseo de elaborar descriptores de escenas. Un campo en particular es el de la *geometría para* CV, que se encarga de describir los cambios que sufre la imagen de los objetos de una escena cuando se les observa desde diferentes posiciones, como función de los mismos y de los parámetros de la cámara utilizada [ZH03].

1.1.1. Elementos de un sistema de *CV*

El término visión nos remite al trabajo con imágenes. Por tratarse de computadoras, la digitalización de las mismas es primordial. Por ello, una de las herramientas fundamentales tiene que ver con el método de adquisición y almacenamiento de las mismas, es decir, el *hardware* de cáptura.

Este puede ser de propósito específico o general dependiendo de la aplicación, sin embargo, el más socorrido es el general, ya que tanto algoritmos como *software* bien diseñados funcionan correctamente en conjunto con el *hardware*. Como ejemplo de sistemas a la medida, considérese aquel que depende de un control autónomo de iluminación combinado con cámaras de alta resolución; técnicas simples de *CV* podrán emplearse para resolver las tareas requeridas, sin necesidad de usar algoritmos sofisticados de calibración de cámaras ni de estandarización de patrones de iluminación [TV98].

Si bien el *hardware* de captura de video desempeña un papel fundamental, también lo hace su liga con el mundo algorítmico: la interfaz de adquisición, una pieza de *software* que obtiene imágenes digitales provenientes de un flujo de video continuo, en un esquema de petición en demanda. Sus tareas consisten en identificar cuando se puede adquirir una nueva imagen, acondicionarla y muestrearla para generar la imagen digital y dejarla a disposición de la aplicación que la pida o procese [NA02].

El siguiente elemento a considerar es el *software* encargado de procesar la imagen. Sus tareas son muy diversas y abarcan desde el almacenamiento de la imagen digital en determinado formato, extracción de características de los elementos compositivos, toma de decisiones, etc. Además, se encuentra directamente relacionado

con el objeto de estudio y toma como base una variedad de áreas de cómputo.

El campo de resolución de problemas que pueden cubrir los sistemas de visión es amplio, multidisciplinario y en continua expansión. Casi puede aplicarse a cualquier tarea que involucre la identificación, discriminación y clasificación de objetos, acciones, eventos, etc. —antes hecha por humanos— y que sea capaz de automatizarse.

En las siguientes secciones se presenta un breve conjunto de disciplinas íntimamente relacionadas con la *CV*, así como algunas áreas específicas en donde es ampliamente utilizada. Además, se define a cada una de ellas de modo que pueda comprenderse su importancia y el modo en el que ayudan a la resolución de problemas, o al constante progreso del campo.

1.1.2. Disciplinas relacionadas

La naturaleza compleja de la visión, ha provocado que ella evolucione como un sujeto multidisciplinario. Sus vertientes confluyen en campos como el de la inteligencia artificial, robótica, procesamiento de señales, reconocimiento de patrones, teoría de control, psicología, neurociencia, entre otros. Durante el breve periodo de evolución —pues se ha visto reflejado un rápido crecimiento— de la disciplina se identifican claramente dos consecuencias:

- La conjunción de objetivos, herramientas y gente de otras disciplinas.
- La definición y los alcances de la *CV* deben considerarse como *granos de sal en un universo aun por explorar*.

Haciendo una breve revisión de las áreas más cercanas e involucradas con la visión por computadora, se encuentran:

Procesamiento de imágenes. Se trata de un área vasta, que (para los propósitos de la *CV*) trata con las propiedades de las imágenes y las transformaciones que puedan existir entre ellas. Como ejemplos de las técnicas más utilizadas están el realce de características, compresión de la información contenida, restauracón de la imagen y, sobretodo, la extracción de características intrínsecas o *naturales* (como puede ser contornos, o texturas, entre otros).

Reconocimiento de patrones. Se han desarrollado diferentes técnicas para reconocer y clasificar objetos. Muchos métodos han probado su eficacia para 2D y 3D bajo ciertas condiciones, pero no son adecuadas en un marco general. Los estudios más recientes abordan esta problemática y son el pináculo de la investigación en *CV*, un ejemplo de ellos es el libro de Zisserman [ZH03], donde se exponen métodos geométricos basados en múltiples vistas.

Fotogrametría. Intenta obtener medidas confiables y precisas a partir de imágenes que no se encuentran en el mismo plano. Aunque es utilizable en *CV*, no está muy relacionada con el área, pues es una disciplina que busca una precisión mayor y que se enfoca por completo la obtención de métricas en imágenes.

Psicología. Aunque los métodos actuales divergen de la concepción inicial de considerar a la visión por computadora como un sistema emulador del biológico, es un área importante. La razón principal consiste en la ayuda que representa —para el investigador en *CV*— conocer el funcionamiento de la percepción humana. Así, mediante un pensamiento más completo, podrá realizar un mejor análisis y desarrollar algoritmos adecuados a los problemas que se presenten.

Neurociencia. Al igual que la psicología, es importante por el estudio que hace sobre el sistema biológico. Estudios extensivos en la última década cubren investigaciones acerca del ojo, de las neuronas, y de las estructuras cerebrales dedicadas al procesamiento de los estímulos visuales. Los resultados obtenidos, han creado un subcampo de la *CV* especializado en el diseño de sistemas artificiales que imiten el procesamiento y comportamiento de los biológicos, en diversos niveles de complejidad.

1.1.3. áreas de aplicación e investigación de la *CV*

La lista aquí mostrada, sólo enumera algunos tópicos del vasto universo de problemas a resolver mediante sistemas de *CV*. Como *áreas de investigación* se han considerado aquellos temas que son referenciados por un número significativo de publicaciones, mientras que las *áreas de aplicación* a los dominios donde se utilizan métodos de visión, posiblemente en combinación con otras tecnologías, para la resolución de problemas.

INVESTIGACIÓN

Detección de características. Un área por demás importante, tal vez la fundamental y en la que se basan los algoritmos de visión. Se trata de extraer propiedades globales —el nivel promedio de gris en una imagen— o locales —círculos, líneas, etc.— en una imagen. En este trabajo se consideran como características a aquellas propiedades locales, significativas y detectables que aparezcan en una imagen. *Significativo* refiere a la asociación de las características extraídas con elementos de interés en la escena (contornos, nivel de gris o estabilidad). Por *detectables* entiéndase que debe de existir algún algoritmo que pueda extraer a la propiedad, de otro modo, por útil que sea al ojo humano, para los fines de la *CV* es inútil. En el capítulo 2 se abordará este tema con mayor profundidad.

Análisis de imágenes de rango. Este tipo de imágenes codifican la forma y distancia de los objetos, mediante el uso de sensores especiales como sonares o rayo laser. Por lo tanto, reproducen la estructura 3D de una escena.

Reconstruccón. Se trata de una de las funciones más simples y básicas en los humanos: distinguir objetos, incluso aquellos que no existen (ilusiones ópticas). Todo esto depende de ćomo es percibido el ambiente por el sentido de la vista. La tarea puede tratarse de identificar forma (análisis 2D) o extraer información 3D, pero siempre a partir de imágenes tomadas desde diferentes ángulos.

Visión estéreo. Un campo importante y ampliamente usado en robótica móvil para detectar obstáculos. Aunque, pricipalmente, se trata de emular al sistema humano (usando dos cámaras), existen técnicas para procesar imágenes en estéreo con una sola cámara. A esto se le conoce por *estructura de movimiento* o *structure from motion* y pretende construir representaciones 3D a partir del video de un cuerpo rígido en movimiento. Es así que este campo se une con el anterior y que trata asuntos de reconstrucción.

Análisis de movimiento. Intenta resolver varios problemas, entre ellos el reconocimiento de actividades humanas y rastreo de movimiento para vigilancia, movimiento del cuerpo o de partes del cuerpo en aplicaciones médicas; utlizado para la corrección de póstura e identificación de posibles lesiones. En el campo de la manufactura es usado para medir el impacto de proyectiles a alta velocidad, por ejemplo en bates de béisbol, raquetas de tenis y bastones de hockey, entre otros.

Aplicación

Industria. Se trata del área que va de la mano con los avances tecnológicos. Algunas tareas incluyen la inspección de procesos, el control de los mismos mediante algún robot o el control de calidad de la producción.

Reconocimiento de características humanas. Aquellas tareas de reconocimiento de rostro, gesticulación, iris, huellas, etc.

Robótica. La intención inicial de proveer a las máquinas con el sentido de la vista, no podría entenderse sin los robots. Pareciera que la ciencia ficción de antaño alcanza la realidad poco a poco. Los robots ahora son capaces de resolver sus tareas, de una manera más aproximada a como lo hacen los humanos. Son capaces de navegar en terrenos hostiles e incluso de forma autónoma, decidiendo a donde ir y la trayectoria que deben tomar. Implícitamente resuelven aspectos complicados como reconocimiento y evasión de obstáculos desconocidos, así como de planeación de rutas.

Milicia. Algunos ejemplos de desarrollo militar son la detección automática de tropas enemigas o la guía remota en vehículos o misiles. También hay una cantidad de simuladores de entrenamiento para soldados, que no pueden ser concebidas sin un sistema de visión.

Medicina. Las aplicaciones de esta área se caracterizan por la extracción de información a partir de imágenes médicas (ultrasonido y *MRI* por mencionar algunas) con el propósito de elaborar diagnósticos para los pacientes. Mediante esta área, cada vez se identifican con mayor precisión diferentes enfermedades como artereosclerosis, tumores, y también se hacen mediciones del tamaño de los órganos, flujo de sangre, etc.

Búsqueda de imágenes. Corresponde a la búsqueda y obtención de imágenes en grandes bases de datos, aprovechando el ancho de banda y la capacidad de las computadores de la actualidad. A pesar de que las técnicas estándar de petición en bases de datos funcionan bien cuando las llaves contienen una descripción de texto, el objetivo del área es hacer análisis basado en contenido, es decir, buscar *dentro* de la imagen. Para ejemplificar, supóngase que un arquitecto o historiador de arte desea buscar imágenes de edificios con un estilo particular de puerta. Sería

de mucha utilidad si, a partir de una imagen que el provea, pudiera pedir al sistema aquellas imágenes parecidas que existan en la base de datos. Características como textura, color y forma pueden emplearse para alcanzar el fin mencionado.

Mediciones meteorológicas. La mayoría de la superficie de la Tierra es regularmente rastreada por satélites artificiales que transmiten imágenes en formato digital. De ellas puede extraerse diversa información, útil para los humanos. Por ejemplo, determinar la cantidad de hielo/nieve en un río permite regular la presa que lo controla, y así, evitar posibles inundaciones.

Realidad virtual. La realidad virtual comprende su propio espectro de selección de problemas e incluso la generación de su entorno, es decir, la imagen sintética. Sin embargo, existen diversos sistemas de visión aplicables a la exploración de mundos virtuales. Como ejemplo considerese un ambiente de simulación para robots móviles. Se pretende implementar algoritmos de manera que, habiéndolos probado, puedan llevarse a la práctica en un robot real. Por ello, si el robot tiene cámaras, entonces el que se simula debe tener algún medio de adquisición de imágenes del ambiente virtual y de procesarlas como si fuera el real.

1.2. Realidad Aumentada

Los objetivos que persiguen las bibliotecas a desarrollar son el reconocimiento y rastreo de objetos en ambientes de realidad aumentada, pero, ¿cómo se define o cuales son las características que la definen?

> **Definición 2** *La* AR *(por sus siglas en ingés* Augmented Reality*) es una variante de la realidad virtual. Por su parte, las tecnologías de realidad virtual, sumergen al usuario en un ambiente sintético donde, mientras permanezca en él, no será capaz de percibir el entorno real. En contraste, los sistema de* AR *permiten que el usuario explore el mundo real, acompañado por objetos virtuales superpuestos. Por lo tanto, es correcto decir que el objetivo principal de los sistemas de* AR*, consiste en mejorar/aumentar la percepción e interacción del usuario en el entorno real; esto por medio del contenido complementario de los objetos virtuales 3D que coexisten en el mismo espacio real.* [MK94, Azu97, AF01]

Es claro que la *AR* complementa a la realidad antes que sustituirla. Idealmente, el usuario percibirá que los objetos reales y virtuales coexisten en el mismo espacio, efecto similar a los logrados en las películas *¿Quién engañó a Roger Rabbit?* y *Parque Jurásico*, por mencionar un par de ejemplos. A pesar de que se cuenta con una definición completa de *AR*, el hecho de mencionar que es una variante de realidad virtual, la sitúa en un punto intermedio entre la llana realidad y lo sintético de lo virtual. [MK94] define a un continuo *Realidad–Virtualidad* donde existen a la par la unión de elementos virtuales y reales. A este *intervalo* se le conoce por *realidad mezclada* y es ilustrado por la figura 1.1.

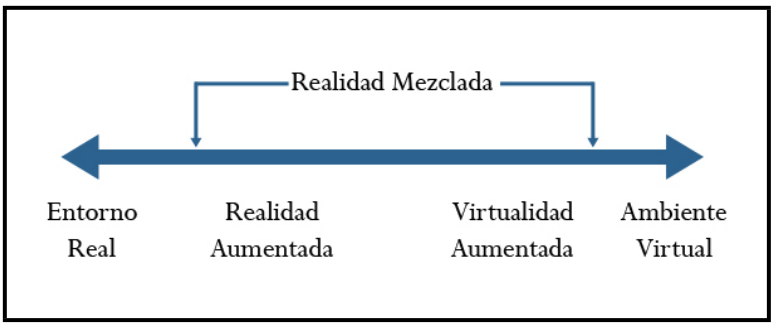


Figura 1.1: Realidad–Virtualidad

1.2.1. Realidad Mezclada

El concepto del continuo se relaciona con la mezcla de las clases de objetos que se presentan en determinada situación, como en la figura 1.1, en donde la *realidad* se encuentra en un extremo y lo totalmente sintético en el opuesto. El primer caso define ambientes que sólo contienen objetos reales e incluyen al video *crudo*, sin haber pasado por una etapa de post-producción. El caso de lo *sintético* define ambientes completamente virtuales. La manera más simple de entender los ambientes de realidad mezclada, es representada por aquel entorno donde conviven objetos de ambos mundos en una misma pantalla.

La existencia de tal continuo, permite que haya diversos sistemas de *realidad mezclada*. El cuadro 1.1 muestra la clasificación que proponen [MK94] y [Azu97], tomando en consideración las tecnologías e interfaces que pueden emplearse.

Clasificación de sistemas de *realidad mezclada*:

1. Proyección. Utilizan monitores convencionales y se trata de interfaces no inmersivas. En ocasiones también son llamadas *ventana del mundo* por la característica de proyectar escenas 3D en tecnología de 2D [FM93].

2. Proyección usando *HMDs*. Es la misma tecnología que la clase 1, pero con la diferencia que la proyección se efectúa en el casco para realidad virtual, logrando una sensación de inmersión.

3. *HMDs* con tecnología *see-through*. Este tipo de *visores*, permiten que el usuario vea realmente a través de ellos, mientras que las gráficas sintéticas son ópticamente superpuestas, generalmente mediante espejos.

4. Proyección usando *HMDs* con correspondencia ortoscópica. El dispositivo de adquisición de video, está adjunto al casco y alineado con respecto a los ojos del usuario.

5. Ambientes gráficos en su totalidad (inmersivos o no) a los que es añadida la percepción del mundo real, mediante video.

6. Ambientes gráficos en su totalidad pero parcialmente inmersivos, ya que los objetos reales interfieren con la escena sintética.

Cuadro 1.1: Taxonomía de [MK94]

En las figuras 1.2 y 1.3 se ilustran a los tipos 1 y 4 de la clasificación descrita. En ellas pueden observarse las etapas involucradas en la generación de la imagen combinada y los dispositivos empleados. Resulta interesante que ambas tecnologías hagan uso de etapas similares, siendo la diferencia principal, el *hardware* en donde se despliega la imagen resultante.

A pesar de que el término *realidad mezclada* es correcto y cubre aspectos generales, en la literatura es más común encontrar *realidad aumentada*, por ello, se abordará de aquí en adelante al tema como *AR* sin olvidar que la generalización corresponde a los ambientes de *realidad mezclada*.

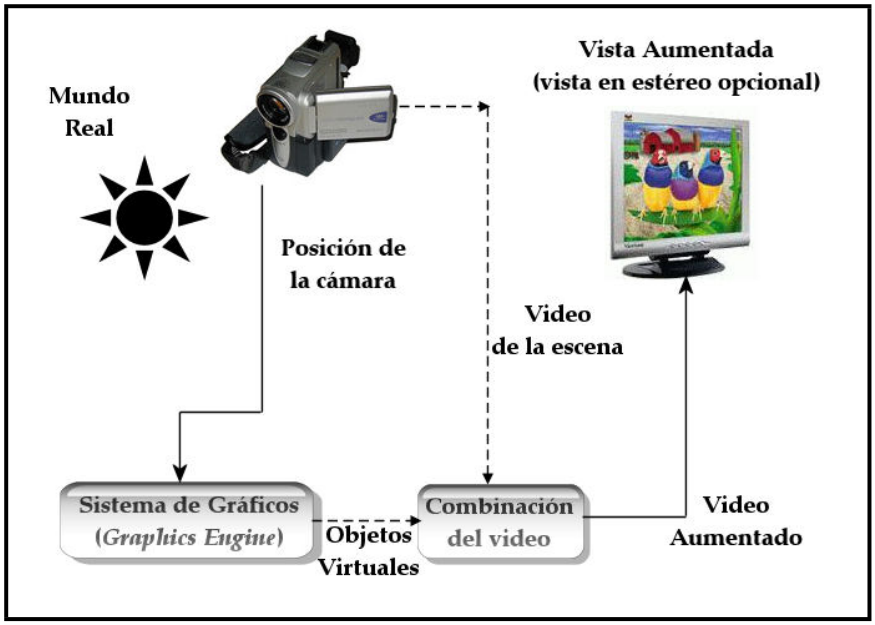


Figura 1.2: Sistema de realidad mezclada *Window of the World*, [AB94].

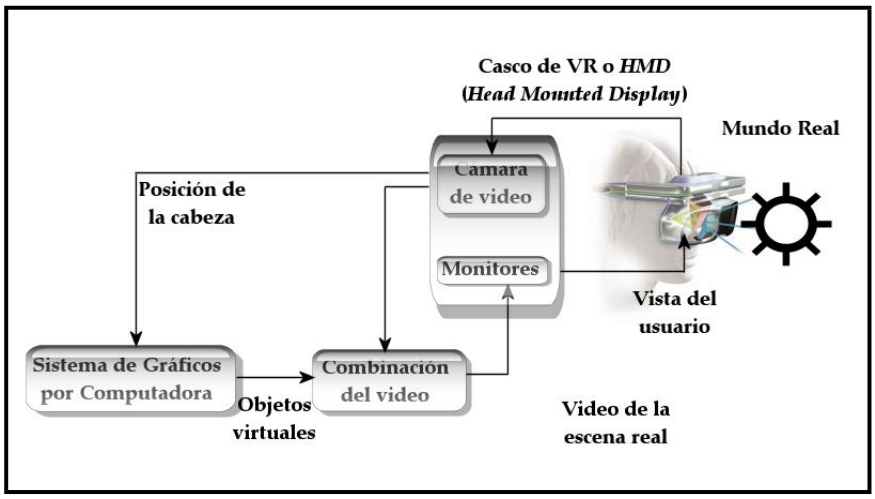


Figura 1.3: Sistema de realidad mezclada con dispositivos *see-through*, [AB94].

1.2.2. Sistemas de *AR*

En el área de los ambientes virtuales, existen diversas tecnologías tanto de procesamiento como de despliegue. Para evitar limitar el estudio de la *AR* a tecnologías específicas —por ejemplo, usando *HMDs*—, [Azu97, AF01] define las siguientes características para clasificar a un sistema/tecnología de *AR*:

1. Combinación de objetos sintéticos con los que existen en la realidad.
2. Interactivos en tiempo real.
3. Estén *registrados en 3D*.

Quizás, uno de los aspectos fundamentales consiste en el registro, que refiere a la correcta alineación entre los objetos reales y los virtuales. Sin ella, la *ilusión* de que los objetos virtuales *existen* en el ambiente real no se obtiene, provocando una mala aceptación por parte del usuario.

En [MK94] se clasifican tecnologías de *realidad mezclada* y en [Azu97, AF01] se hace una amplia revisión de tipos de sistemas para ser utilizados en aplicaciones de *AR*. Una de las decisiones principales, al momento de diseño, consiste en elegir el tipo de despliegue que la aplicación utilizará. Ya se vió que puede tratarse de dispositivos de proyección u ópticos. Por conveniencia, se utilizan los primeros, ya que el *hardware* es menos costoso. Puede utilizarse el monitor de cualquier computadora o unos lentes de precio accesible.

Otro punto medular es el concerniente al método y *hardware* de rastreo y registro. Por un lado están los métodos basados en *trackers* magnéticos y *GPS*, y en el otro los relacionados con visión. Los primeros muestran ser muy estables y precisos, pues operan con base en un sistema de referencia que permanece en la misma posición siempre, haciendo que el rastreo sea constante. En cambio, el que se fundamenta en técnicas de visión por computadora, resulta ser menos exacto y estable, sin embargo, la ventaja que presenta sobre los otros métodos radica nuevamente en el tipo de *hardware* necesario. Al tratarse de cámaras como medio de entrada, el panorama es amplio: *webcams*, cámaras caseras, profesionales, etc., son algunos ejemplos. En relación íntima con ellas, se encuentra el tipo y complejidad de los algoritmos de visión. Entre menor sea la calidad de la cámara, los algoritmos tendrán que ser más efectivos y robustos y viceversa.

Por lo anteriormente expuesto, el sistema de visión que se propone y más comúnmente usado es el que se muestra en la figura 1.4. La escena es vista por un dispositivo de adquisición de imágenes, en este caso se trata de una cámara de video que produce una proyección en perspectiva del mundo 3D como imagen 2D. Los parámetros intrínsecos (longitud focal y distorsión del lente) y extrínsecos (posición y orientación) del dispositivo determinan el contenido *real* del plano de proyección. La figura 1.4 ilustra lo aquí mencionado.

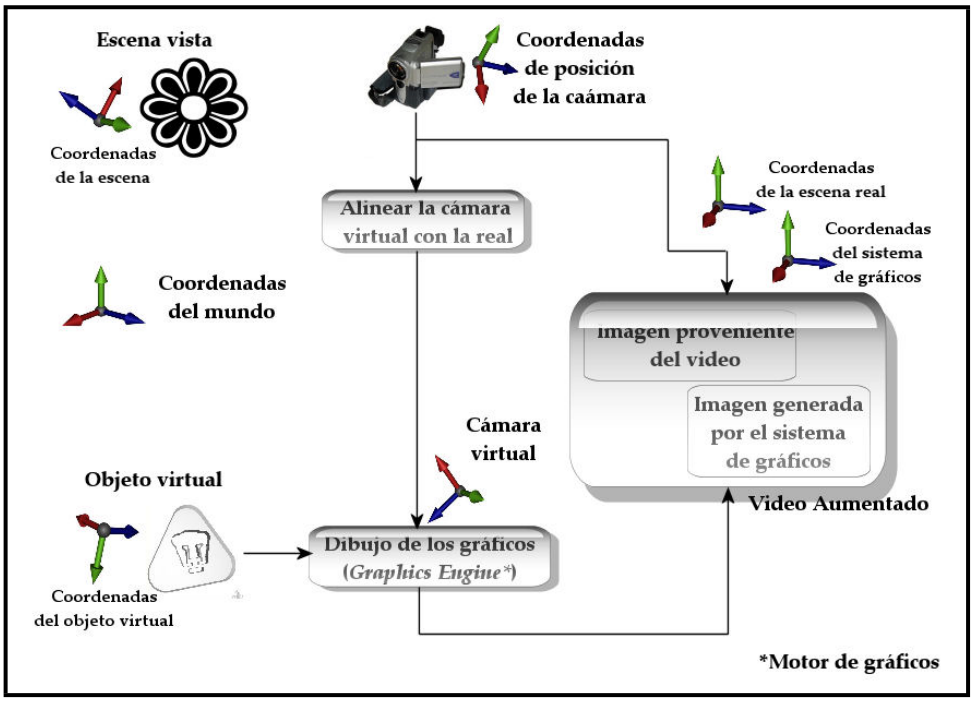


Figura 1.4: Sistema típico de *AR*

La creación de la imagen virtual es realizada con un sistema estándar de gráficos por computadora. Los objetos son modelados con base en su propio marco de referencia. Este sistema requiere de información concerniente a la escena real para que pueda dibujar correctamente los modelos. Dicha información controlará a la cámara virtual que es usada para obtener las proyecciones *empatadas* de los objetos. Finalmente, la imagen generada es entonces combinada con la imagen de la escena real para crear la imagen en *AR*.

1.2.3. áreas de aplicación

Una variedad de temas pueden hacer uso de estos sistemas. En [Azu97, AF01] se encuentra una revisión completa de ellas y aquí solamente se presentan las que se consideran particularmente importantes e interesantes.

ALGUNOS TÓPICOS QUE HACEN USO DE LA *AR*

Medicina. Ya que la tecnología es un elemento muy persuasivo en el campo médico, no es sorprendente que éste sea visto como uno de los más importantes para los sistemas de *AR*. En su mayoría, las intervenciones médicas tienen que ver con cirugía asistida con imágenes, refiriéndose a todos los estudios efectuados al paciente previamente (como ultrasonidos, tomografías, radiografías, etc.) y con el fin de planear la intervención. Entonces, la *AR* puede emplearse como ayuda al equipo de cirugía, de manera que cuente con información precisa acerca de los estudios del paciente al instante de la operación. En ultrasonido para embarazo, por ejemplo, el técnico médico podría tener una representación 3D del feto, empatado y registrado correctamente, dentro del abdomen de la paciente. Véase la figura 1.5(a).

En el campo de la investigación, en Munich, el grupo [NK06] trabaja desde el año 2003 en simuladores médicos utilizando *AR*. Entre sus principales líneas de investigación se tiene el desarrollo de interfaces 3D, cuyo objetivo es la correcta representación, prequirúrgica, postquirúrgica y en cirugía de datos reales de pacientes de ortopedia. Otros proyectos tratan aspectos de cirugía laparoscópica, y un sistema de navegación *in situ* para cirugía asistida.

Entretenimiento. Un esquema simple de *AR* ha sido usado en los negocios del entretenimiento y noticias por ya algún tiempo. Cuando se observa el reporte meteorológico, el reportero es mostrado enfrente de mapas regionales con determinada animación. La situación real en el estudio, es que el reportero se encuentra parado frente a una pantalla de color verde o azul. Esta imagen real es entonces *aumentada* mediante mapas generados por computadora empleando una técnica llamada *chroma–keying*. Otro caso es el de la publicidad virtual que se coloca en ciertos eventos deportivos: juegos de béisbol, de basquetbol, fútbol americano y soccer, entre otros. Por ejemplo, mientras que un juego de béisbol se transmite, se puede agregar la información relevante a la velocidad

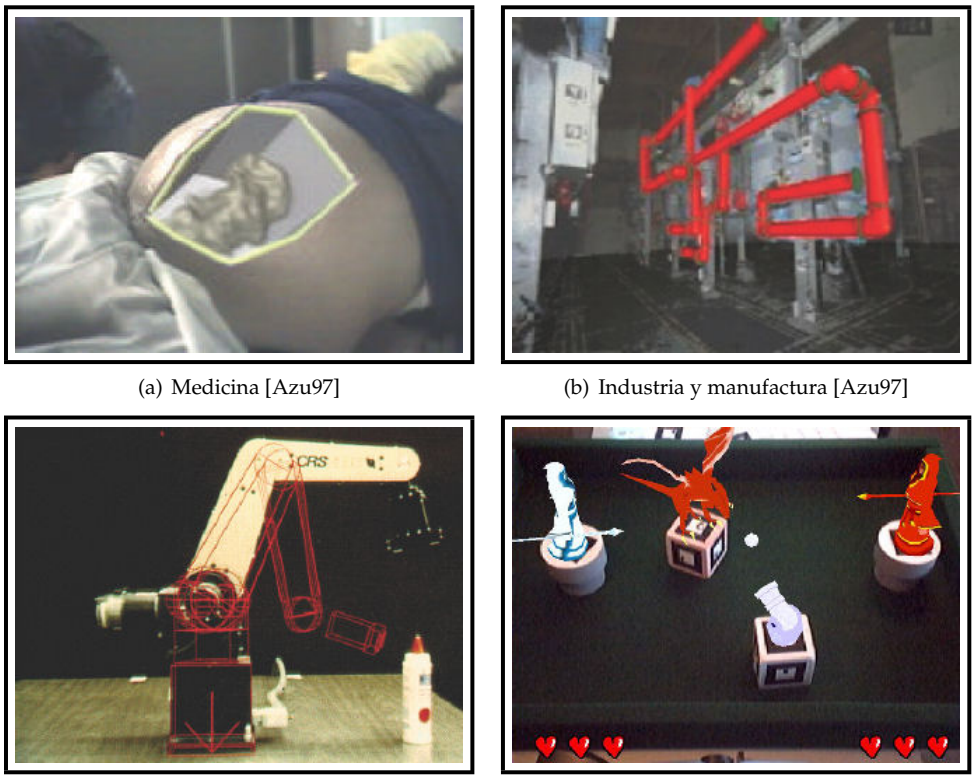
del lanzamiento, o incrustar publicidad que no está presente en el estadio.
En videojuegos, además de haberse desarrollado aplicaciones comerciales como el *eye toy* [Son05], algunos esfuerzos estudiantiles merecen ser nombrados. El principal es el grupo [Jim07]. Entre sus creaciones destacan los juegos de tablero basados en los populares juegos de cartas al estilo *Magic: The Gathering* o *Yugi–Oh!*, véase la figura 1.5(d). Otro juego *retro* es la versión *AR* de *Pong*.

Manufactura y reparación. El proyecto [MTD07] es un buen ejemplo de cómo es empleada la *AR* en esta área. Además de proveer un entorno de trabajo propio para el desarrollo de aplicaciones de *realidad mezclada*, uno de sus prototipos tiene como objetivo entrenar a los técnicos que trabajan en una refinería de petróleo. Véase la figura 1.5(b)

Visualización. La visualización de datos comprende una extensa gama de áreas. Los ejemplos antes y posteriormente enumerados son en sí ejemplos de visualización de datos médicos, industriales, lógicos y educativos. Por ello, la visualización debe entenderse de manera particular y general. En términos de generalidad, puede ejemplificarse con nuevos métodos de interacción en telepresencia, bajo ambientes de colaboración, tal y como se plantea en [Fuh98].

Robótica y planeación de rutas. En su trabajo presentado en noviembre de 2005, [Sti05], propone un esquema novedoso de robótica autónoma. Ellos hacen uso de las más recientes técnicas de visión por computadora para determinar características importantes como posición y reconocimiento de objetos en 3D, además de representaciones sintéticas del entorno y de *AR* para utilizar a los robots en tareas de planeación de rutas. Véase la figura 1.5(c)

Educación. En [Bil02] se menciona que, dada la característica de *crear* nuevos contextos, la *AR* debe ser explorada por investigadores y educadores en conjunto, y así determinar cuales son las características más importantes y cómo aplicarlas en un ambiente escolar. Por su parte, [Ada04] menciona la necesidad de una nueva didáctica más dinámica y que explote las posibilidades tecnológicas de la actualidad.



(a) Medicina [Azu97]

(b) Industria y manufactura [Azu97]

(c) Robótica [Azu97]

(d) Entretenimiento [Jim07]

Figura 1.5: Aplicaciones de la *AR*

1.3. Bibliotecas de programación

Las bibliotecas de programación son una herramienta muy útil para toda la gente encargada de desarrollar *software*. Permiten la reutilización de código que resuelve necesidades específicas, y que son recurrentes en los problemas de cómputo. La reutilización *ad-hoc* ha sido muy socorrida desde los primeros días en que comenzó la programación.

En realidad, se trata de colecciones de sub-programas que ayudan al programador y proveen de servicios a programas independientes con los que se conectan. De este modo, el código puede ser compartido y modificado modularmente. Mediante un proceso de *ligado* o *vinculado*, los programas pueden comunicarse y hacer uso del código externo.

Históricamente, las bibliotecas solamente podían ser estáticas y consistían de un conjunto de rutinas que eran copiadas dentro de la aplicación principal por el compilador o el ligador y de tal modo se producían ejecutables *standalone*. Posteriormente se creo el ligado dinámico, que ya no incluye el código de la biblioteca dentro de la aplicación que hace uso de ella, sino que, en tiempo de ejecución, víncula la aplicación con la utiliería externa.

Para la presente tesis, el *software* desarrollado pretende ser una biblioteca de programación a la que se recurra cuando se necesite hacer detección de objetos y rastreo de los mismos. El enfoque que se ha considerado, es de ser una biblioteca que pueda usarse en ambientes *offline* y *online*, así como en tareas de robótica o más complejas, sirviendo de base para desarrollar futuras aplicaciones de realidad aumentada.

Capítulo 2

Estado del arte

El capítulo 1 ofrece un panorama y define al amplio campo de la visión por computadora. Además se han tratado aspectos relacionados con el objetivo de este trabajo y de las diferentes aplicaciones en las que pueden emplearse sistemas de *CV*. Cómo ya se mencionó, el objetivo es crear una biblioteca de programación que sea usada en problemas de reconocimiento y rastreo de objetos. En específico se busca que sea útil para futuras aplicaciones de robótica y, principalmente, de *AR*.

Existen varias formas de hacer reconocimiento de objetos mediante sistemas de *CV*, dos de ellas son la utilización de marcas artificiales que poseen ciertos atributos que facilitan su detección, reconocimiento y rastreo. El otro enfoque es usar características *naturales* que son inherentes a la imagen del objeto que se analiza.

Por lo tanto, debido a las futuras aplicaciones que pretende servir el *software* en desarrollo, primero se presenta una revisión de las técnicas empleadas por los sistemas de *AR*. Posteriormente se discute el estado del arte respecto de los métodos usados para la extracción de características locales para el reconocimiento y rastreo de las mismas. Finalmente, se concluye con la vertiente elegida para el desarrollo de esta tesis.

2.1. Bosquejo histórico

Los primeros esfuerzos por hacer mundos de *AR* utilizaban medios magnéticos para obtener la posición 3D de los objetos virtuales con respecto del usuario. En una revisión de 1996, [SH96] propone un sistema combinado *magnético–visión* para rastreo y expone algunos problemas relacionados con cada uno de ellos. Los dispositivos magnéticos de la época generaban errores grandes en la ubicación, llegando incluso a ser de 10cm; con una calibración *meticulosa* se reducía a 2cm, error todavía considerable.

Entre los sistemas ópticos que menciona, destacan los usados por [WA92, AB94] ya que mostraron mayor precisión que los magnéticos. Sin embargo, por las limitaciones tecnológicas, el sistema de [WA92] requería de cuatro cámaras para hacer la estimación de pose. En resumen, la recuperación de información 3D a partir de imágenes no resultaba una tarea sencilla y menos aun para las limitantes de aquel entonces.

Un año después, en [KK97] se proponen nuevas técnicas y presentan algunos avances. Entre ellos está la capacidad de rastrear y registrar objetos en tiempo real por los sistemas de [UK95], [BW96] y [GLP94]. Sin embargo, el primero de ellos no es efectivo para *AR* interactiva, puesto que no resuelven el problema de registro en 3D por completo. El segundo lo hace, pero las estimaciones de cámara y objeto tienen que ser descompuestas en dos partes. Por otro lado, el último lo resuelve satisfactoriamente haciendo uso de *marcas* artificiales, perfectamente registradas y registrables en un mundo 3D. A su vez, [KK97] propone una solución propia, estimando la ubicación del observador mediante una sola cámara, algunas *marcas* artificiales y filtrado de Kalman para estimar la posición de la cámara y rastrear las *marcas*.

Para 1998 los métodos basados en *marcas* artificiales se hicieron más famosos, un ejemplo es [Par97]. Además, comenzaron a utilizarse los métodos basados en características de la imagen. Algunos tenían que ser calibrados manualmente mientras que otros encontraban correspondencias automáticamente a partir de imágenes previamente calibradas. A pesar de ello, utilizar estos sistemas en tiempo real era complicado por el tipo de procesamiento que debía ser hecho.

En el 2000, la geometría proyectiva o *geometría para CV*, comenzó a ser utilizada con mayor frecuencia en aplicaciones de *AR*. Por ejemplo, en [SZ00] se presenta un sistema de que rastrea planos en ambientes reales y no conocidos para después *aumentar la escena* con respecto de ellos. El requerimiento fundamental es que

rastree con precisión, sea sustentable, y opere en tiempo real. Por lo tanto, la tecnología que se levanta como triunfadora es la óptica: ofrece la precisión requerida, y los ambientes no conocidos no permiten el uso de *marcas* ni de una construcción de algún modelo 3D *a-priori*. En síntesis, lo que se propone es simplificar el rastreo mediante el reconocimiento de superficies planas en la imagen. Se trata de una solución ingeniosa, ya que, en general, la mayoría de los objetos en el mundo real tienen bordes y muchos de ellos conforman planos.

En 2001, [AF01] hizo nuevamente una recopilación sobre las técnicas y sistemas utilizados en *AR*. Comenta que se han desarrollado sistemas de rastreo precisos usando técnicas híbridas para así explotar las fortalezas y compensar las debilidades de las individuales. Algunos utilizan acelerómetros y *CV* mientras que otros sistemas hacen uso de brújulas/giroscopios e incluso GPS. Por lo general, este tipo de *hardware* es utilizado para aplicaciones al aire libre.

A pesar de estas buenas ideas, para ambientes naturales no hay como basarse en el rastreo de características inherentes al medio, ya que proveen de la información necesaria para identificar objetos o regiones en la imagen. Para la época del estudio, aún era complicado llevar algoritmos robustos a una ejecución en tiempo real.

En artículos más recientes, como [US05], el problema de utilizar algoritmos basados en características naturales ha sido resuelto. Entre las características utilizables destacan: *puntos de interés* como en [CC02], bordes o curvas. Algunas técnicas adicionales que pueden servir operando sobre los descriptores obtenidos son flujo óptico [NY99] para estimar el movimiento de la cámara, registro en tiempo real en aplicaciones de *AR* mediante geometría epipolar [CC02], registro mediante planos ([SZ00] y otros trabajos suyos), entre otras.

El cuadro 2.1 presenta un resumen de los tipos de sistemas de rastreo que se han utilizado en los últimos diez años. Cabe mencionar que, dada la cantidad de algoritmos desarrollados y de la capacidad de cómputo actual, es virtualmente posible implementar cualquier combinación entre los mencionados.

En las siguientes dos secciones se abordará, primero, a quien quizás sea el sistema-biblioteca de programación más famoso para el desarrollo de aplicaciones de *AR*: *ARToolkit*. Posteriormente se especificará un conjunto de técnicas empleadas para la extracción de características naturales, se mencionará cual es la vertiente elegida para este trabajo, así como su relevancia en el campo de la *CV* y no sólo para la *AR*.

Técnicas empleadas para rastreo en aplicaciones de *AR*...

- Basados en algún tipo de *hardware*:
  - Sensores magnéticos.
  - Brújulas/Giroscopios.
  - GPS.
- Sistemas de visión.
  - Por el número de dispositivos de captura:
    - Multicámara. *Visión estéreo, tres cámaras, cuatro cámaras, etc.*
    - Monocámara.
  - Por el procesamiento sobre la imagen adquirida:
    - Basados en *marcas* artificiales o modelos conocidos *a priori*.
    - Basados en características inherentes a la imagen.
- Sistemas híbridos:
  - Visión—Magnéticos
  - Visión—Brújulas/Giroscopios
  - Visión—GPS

Cuadro 2.1: Rastreo en *AR*

2.2. *ARToolkit*

Se trata de una biblioteca de programación que permite que los programadores desarrollen, de una manera fácil, aplicaciones de *AR*. Como ya se mencionó, una de las tareas más complicadas es la del registro. Para resolver esto, el sistema de visión de *ARToolkit* se basa en una secuencia de pasos ilustrados en la figura 2.1.

En síntesis, el funcionamiento es así: la imagen adquirida del video es convertida a una binaria en blanco y negro mediante la aplicación de un umbral. Enseguida se identifican todas las regiones cuadradas que aparezcan en la imagen. Para cada una de ellas, se determina el patrón interior y se empata contra un conjunto previamente entrenado de *marcas*. Cuando existe una correspondencia entre el pa-

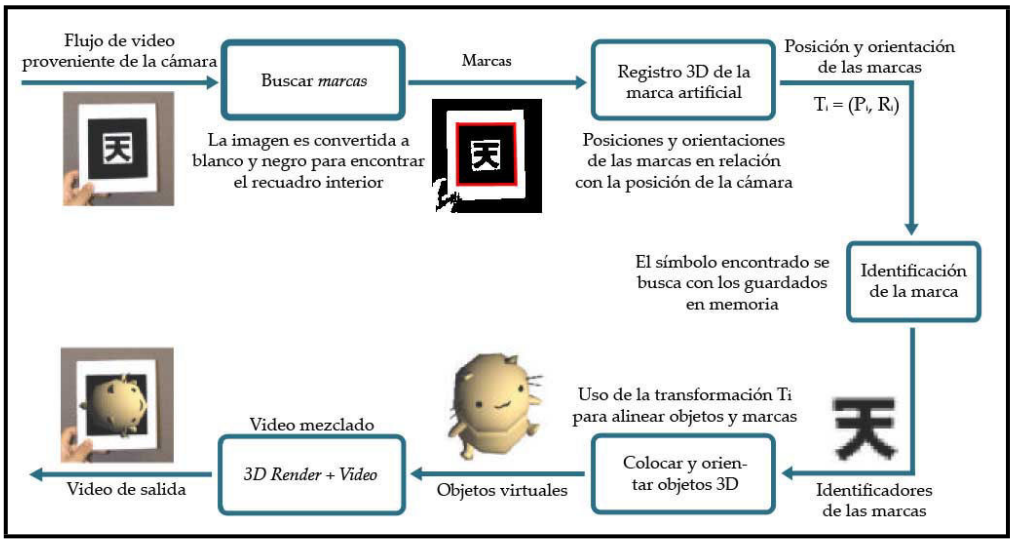


Figura 2.1: Etapas de procesamiento de *ARToolkit*

trón detectado y alguno de los entrenados, *ARToolkit* usa el tamaño —ya conocido— del cuadrado y la orientación del patrón para efectuar el registro del mismo. La matriz calculada es utilizada para mezclar el objeto virtual 3D con el flujo de video y así producir el efecto deseado [KB00].

Como se menciona en [HIT00b], el sistema de visión se basa en la detección de esquinas y de bordes, en conjunto con un algoritmo rápido de para la estimación de pose. Se esquematiza en el cuadro 2.2.

Un poco más de detalle puede encontrarse en [KB99]. Se establece que las marcas, de tamaño previamente conocido, son tomadas como base del marco de referencia en el que el dispositivo de despliegue es representado; véase la figura 2.2. La ecuación 2.1 muestra la matriz de transformación ($\mathbf{T_{cm}}$) utilizada para llevar este sistema de coordenadas al utilizado por la cámara; ha sido estimada mediante análisis de la imagen.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} V_{11} & V_{12} & V_{13} & W_x \\ V_{21} & V_{22} & V_{23} & W_y \\ V_{21} & V_{22} & V_{23} & W_y \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix} = \begin{bmatrix} & \mathbf{V_{3x3}} & & W_{3x1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix} \tag{2.1}$$
$$= \mathbf{T_{cm}} \begin{bmatrix} X_m & Y_m & Z_m & 1 \end{bmatrix}^T$$

```
subroutine DetectMarkers(i, camIntParam)
/// Conversion de la imagen a blanco y negro
i' = binarize(i, thresVal);     /// i': Im. binaria; i: Im. adquirida; thresval: Umbral

/// Etiquetado de la imagen binaria
comList = label(i');            /// comList: Lista de componentes (recuadros)

/// Obtencion de los contornos
cont = detectContours(comList); /// cont: Lista que contiene los contornos detectados

/// Estimacion de lineas en los contornos
lines = detectLines(cont);      /// lines: Lista que contiene las lineas estimadas

/// Deteccion de esquinas
corners = getCorners(lines);    /// corners: Lista de esquinas de los recuadros

/// Normalizacion de los patrones segun los parametros intrinsecos de la camara
markers = normalizePatterns(corners, camIntParam);

/// Reconocimiento y empate de los patrones encontrados con los entrenados
ids = match(markers, knownPatterns); /// ids: Ids de los patrones que hicieron match

/// Calculo de la homografia, se utilizan las lineas encontradas y los identificadores
/// de las marcas que hicieron correspondencia para extraer las dimensiones reales y
/// pose inicial de la marca rastreada
H = homography(lines, ids, markers);

/// Calculo de la transformacion para la camara a partir de la homgrafia
M = findCameraTrans(homography, markers);

/// Optimizacion de las transformaciones
optimize(M);

/// Regresa la matriz encontrada
return M;
```

Cuadro 2.2: Algoritmo de *CV* usado por *ARToolkit*

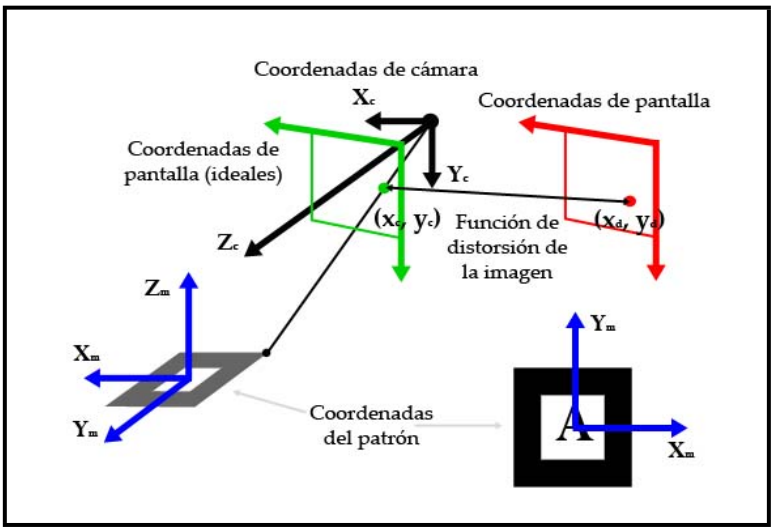


Figura 2.2: Marco de referencia usado por *ARToolkit* [HIT00a]

Como se puede ver en el algoritmo del cuadro 2.2, posterior a la extracción de características (líneas y contornos), se efectúa una normalización y empate del patrón detectado contra los que se tienen entrenados. Para este proceso, la ecuación 2.2 —que representa una transformación de perspectiva— es usada. Las variables son determinadas sustituyendo las coordenadas de pantalla (x_c, y_c) y de *marca* del patrón encontrado (X_m, Y_m), para cada uno de los vértices del recuadro.

$$\begin{bmatrix} hx_c \\ hy_c \\ h \end{bmatrix} = \begin{bmatrix} N_{11} & N_{12} & N_{13} \\ N_{21} & N_{22} & N_{23} \\ N_{31} & N_{32} & N_{33} \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ 1 \end{bmatrix} \tag{2.2}$$

Las ecuaciones que representan a dos segmentos de línea paralelas —del cuadrado— en coordenadas de cámara son:

$$a_1 x + b_1 y + c_1 = 0, \quad a_2 x + b_2 y + c_2 = 0 \tag{2.3}$$

Dada una matriz de proyección **P**, obtenida al calibrar la cámara y mediante la ecuación 2.4, las ecuaciones de los planos que incluyen ambas líneas, respectiva-

mente, se representan por la ecuación 2.5 en espacio de cámara al sustituir x_c y y_c en 2.4 por x y y en 2.3.

$$\mathbf{P} = \begin{bmatrix} P_{11} & P_{12} & P_{13} & 0 \\ 0 & P_{22} & P_{23} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} hx_c \\ hy_c \\ h \\ 1 \end{bmatrix} = \mathbf{P} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{2.4}$$

$$\begin{aligned} a_1 P_{11} Xc + (a_1 P_{12} + b_1 P_{22}) Y_c + (a_1 P_{13} + b_1 P_{23} + c_1) Z_c = 0 \\ a_2 P_{11} Xc + (a_2 P_{12} + b_2 P_{22}) Y_c + (a_2 P_{13} + b_2 P_{23} + c_2) Z_c = 0 \end{aligned} \tag{2.5}$$

Sean $\vec{n_1}$ y $\vec{n_2}$ los vectores normales a estos planos, respectivamente, el producto vectorial $\vec{n_1} \times \vec{n_2}$ es un vector de dirección paralelo a dos lados del recuadro. Sean $\vec{u_1}$ y $\vec{u_2}$ dos vectores unitarios obtenidos a partir de dos conjuntos de lados paralelos de un cuadrado, por lo tanto, son también perpendiculares, sin embargo, en la práctica, diversos errores, algunos ocasionados por los algoritmos de procesamiento de imágenes, provocan que no lo sean. Para compensarlo, desígnense dos vectores perpendiculares entre ellos, $\vec{v_1}$ y $\vec{v_2}$, en el plano que incluye a $\vec{u_1}$ y $\vec{u_2}$, ver figura 2.3.

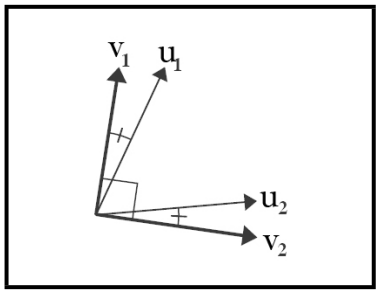


Figura 2.3: Compensación a la normalización

Sea $\vec{v_3}$ un vector perpendicular a $\vec{v_1}$ y $\vec{v_2}$, entonces, la componente de rotación $\mathbf{V_{3x3}}$ en la ecuación 2.1 está dada por $\begin{bmatrix} \vec{v_1}^T & \vec{v_2}^T & \vec{v_3}^T \end{bmatrix}$. Recapitulando, $\mathbf{V_{3x3}}$ fue encontrado mediante las ecuaciones 2.1, 2.4 y los cuatro vértices del patrón (en espacio de *marca* y de cámara). Así, se generan ocho ecuaciones —incluyendo a los componentes de traslación $W_x \quad W_y \quad W_z$ —, pudiéndose ahora obtener tales componentes.

La matriz de transformación calculada contiene error. A pesar de ello, tal puede ser reducido de la siguiente manera. Las coordenadas de los vértices en espacio de *marca* puede transformarse en coordenadas de cámara usando la matriz obtenida.

Enseguida, la matriz de transformación se optimiza, minimizando así las diferencias entre estas coordenadas transformadas y las coordenadas medidas en la imagen. Aunque existen seis variables independientes en la matriz de transformación, solamente las componentes de rotación son optimizadas, mientras que las de traslación son recalculadas utilizando el método descrito. Iterando este proceso varias veces, la transformación resulta más precisa. La razón de dejar fuera de la optimización a las seis variables, es por cuestiones de costo en materia de cómputo.

2.3. Métodos basados en características naturales

Con la explicación arriba expuesta —acerca del funcionamiento interno de la biblioteca *ARToolkit*— es entendible que existan errores durante el registro. Además, la correcta interpretación de los datos es dependiente de que se detecten en la imagen los cuatro vértices que componen al recuadro exterior de la marca. Las oclusiones, recorte en la escena vista y cambios abruptos en el contenido de la misma, provocan variaciones y errores que pueden llegar a ser considerables.

Técnicas más robustas para efectuar este tipo de operaciones son las basadas en la detección de características naturales e inherentes a la imagen. Conceptualmente, el problema de trabajar con imágenes en sistemas de cómputo, radica en el contenido semántico que contienen. En otras palabras, para las computadoras, una imagen no es más que una matriz llena de datos sin significado.

Entre los intentos por obtener o interpretar imágenes digitales se tienen a los basados en extracción de caraterísticas globales y locales. Los primeros toman en cuenta la distribución de color o textura, analizando la imagen como un todo. Desafortunadamente, al igual que *ARToolKit*, no son métodos robustos bajo las condiciones de oclusión, recorte y otros cambios. La segunda vertiente, características locales, resultan una mejor opción, pues definen estructuras locales formadas por pixeles con gran variación en intensidad. En general, a estos puntos se les denomina *puntos de interés* y proporcionan mayor información de la imagen.

En general, en todos los métodos de esta aproximación, primero se detecta un conjunto de puntos de interés. Enseguida se calcula un *descriptor* para cada punto detectado; ellos son usados para encontrar estructuras locales similares. Cuando se trata de hacer reconocimiento en imágenes, los descriptores son usados para determinar correspondencias uno a uno, las cuales permitirán recobrar la *geometría* existente

entre pares de imágenes. Entre las aplicaciones de *empate* entre descriptores están la reconstrucción 3D de la escena vista, la elaboración de un *panorama* a partir de una secuencia de imágenes, y localizar la posición de la cámara, y por ende del observador, entre otras.

La calidad, robustez e invariancia del reconocimiento depende directamente del método de detección. En [Mik02], se presenta un estudio a profundidad acerca de la teoría y métodos para la detección de puntos de interés y cálculo de descriptores locales. Para el desarrollo de las múltiples aproximaciones en materia de elaboración de descriptores, diversos problemas deben ser resueltos. El cuadro 2.3 los esboza y pretende proporcionar cierto entendimiento de ellos.

A continuación se muestra una breve revisión de las técnicas empleadas para la resolución de los problemas descritos en el cuadro 2.3.

2.3.1. Detección de puntos de interés

En la literatura existen numerosas técnicas para detectar puntos de interés, debido a ello, los puntos extraídos difieren en su posición, escala y estructura. La información promedio que proporcionan los puntos, es diferente entre cada método, haciendo que los cálculos sean más, o menos distinguibles. En general, el objetivo es desarrollar un detector que sea invariante a las transformaciones geométricas y de fotogrametría más comunes e introducidas bajo diferentes condiciones de vista. Las escenas reales usualmente contienen superficies *suaves*, que pueden ser aproximadas mediante secciones planas. Con las últimas, es posible determinar la transformación en perspectiva a partir de diferentes puntos de observación. Por lo tanto, se considera un buen detector a aquel que presenta invariancia a transformaciones afines.

A través del tiempo se han propuesto y diseñado diferentes tipos de detectores. Quizá, el más famoso y punto de partida para muchos otros sea el detector de *esquinas* de Harris [HS88]. Este detector se basa en los valores característicos de la matriz de segundos momentos de la imagen (covarianza). Sin embargo, los puntos encontrados no son invariantes a escala. Los puntos encontrados corresponden a valores *grandes* en los valores característicos de la matriz mencionada. Este detector ha probado ser el más robusto en contra de ruido y cambios de iluminación en la escena.

En 1995, Zhang [ZF95] mostró que era posible *empatar* puntos de interés, encontrados mediante el detector de Harris, en una imagen grande, usando una ventana

Punto de interés. Representación de una vecindad local definida por sus coordenadas de imagen, tamaño y forma de la estructura. Las transformaciones de rotación, escala y en perspectiva, así como cambios en la intensidad de los pixeles son los cambios que se presentan con mayor frecuencia en la imagen. La posición del punto de interés es el factor más importante en la detección, ya que los errores en ella causarán fallas en el *matching*. La escala del mismo es el segundo parámetro en orden de importancia. Finalmente, la forma de la estructura local debe ser identificada invariantemente de las transformaciones que pueda haber sufrido la imagen.

Invariancia a escala. La escala de una estructura local está relacionada con la resolución en la que se encuentra representada en la imagen. Para una estructura, existe una resolución mínima —bajo de la cual la estructura carece de significado— y una máxima —que depende principalmente de las restricciones que definen el carácter local de la misma—. El problema consiste en elegir la escala apropiada en donde la estructura es más representativa. Entonces, el escalamiento puede utilizarse para modelar el efecto de cambiar la posición de la cámara a lo largo del eje focal o bien, de los parámetros de longitud focal del lente. Es por ello que este punto resulta crucial para toda vertiente basada en características locales. Sin esta propiedad, las tareas de detección y reconocimiento serían aun más complejas y debería usarse mayor cantidad de información en la imagen, llevando a un incremento en la complejidad de los algoritmos y tiempo de cómputo.

Invariancia a transformaciones afines. Para poder determinar los parámetros ya mencionados es materia primordial determinar las transformaciones afines que puedan presentarse. La traslación no es tan importante, pues el análisis local la remueve de la descripción efectuada. El algoritmo de detección debe ser robusto ante rotaciones debido a que la cámara puede variar su posición en cualquier momento. Los algoritmos pueden resolver estos problemas conjunta o individualmente, pero deben ser considerados.

Descripción local. Es la *semántica* del punto de interés. Los descriptores son necesarios para comparar y encontrar estructuras similares. El problema consiste en definir una descripción simple y compacta que cubra los aspectos mencionados en los anteriores puntos.

Reconocimiento y *matching*. El cómputo de los descriptores puede contener errores y, debido a que se necesita de una métrica confiable para reconocer y *empatar* puntos, algoritmos robustos deben utilizarse. Uno de ellos es RANSAC, que estima la transformación entre puntos candidatos y de referencia, eliminando los erróneos.

Cuadro 2.3: Consideraciones importantes para la elaboración de descriptores locales

de correlación alrededor de cada punto y así elegir puntos parecidos. Posteriormente, los puntos erróneos son eliminados por medio de la obtención de la matriz fundamental para el par de imágenes.

Como se mencionó al inicio de la sección, una forma de conseguir invariancia geométrica es asumir que las regiones a *empatar* son localmente planas y describirlas de forma que sean invariantes al calcular la homografía. Una homografía es una mapeo invertible de puntos y líneas sobre un plano de proyección $\mathbb{P}^2$, [ZH03]. En su trabajo de 1997, Cordelia Schmid y Roger Mohr, [SM97] utilizan derivadas de gaussianas que son rotacionalmente simétricas. El detector de Harris sigue siendo la base para la extracción de los puntos de interés, pero en lugar de utilizar una ventana de covarianza, definen un descriptor invariante a la rotación. Además, demostraron que las múltiples características extraídas permiten generalizar el reconocimiento de objetos y ser utilizadas bajo condiciones adversas como oclusión y recorte de objetos y/o escenas.

Ya que se pretende representar a los puntos bajo diferentes escalas, la imagen debía ser expresada en términos útiles y propios de un análisis multiescala. La representación conocida como *espacio de la escala* (*scale-space*) es vital para ello. En inicios de la década de los 80, Witkin [Wit83] propusó considerar la escala como un parámetro continuo y formuló las reglas principales de la teoría moderna de este tipo de análisis. Desde entonces, las propiedades de la figura *espacio de la escala*, han sido estudiadas, contribuyendo con importancia Koenderink [Koe84], Lindeberg [Lin94] y Florack [Flo93]. En el siguiente capítulo se abordará con mayor detalle lo relevante a este campo de estudio.

Los métodos existentes buscan *extremos* (máximos y/o mínimos) en la representación 3D de una imagen, *espacio de la escala*. Un punto característico es detectado si un *extremo* es localizado alrededor de un cubo 3D y su valor absoluto es mayor a cierto umbral; veánse la figura 2.4 y la ecuación 2.6. Las diferentes técnicas varían en la expresión empleada para construir la representación *espacio de la escala*.

$$\begin{gathered} C_p = F(m, \sigma_n) > F(\bullet, \sigma_l) \\ \text{con } l = n-1, n, n+1 \text{ , } \bullet \text{ son los 26 vecinos de } m \text{ y } |F(m, \sigma_n)| > umbral \end{gathered} \tag{2.6}$$

En la ecuación anterior, C_p es candidato a punto de interés, m es la localidad o pixel analizado y $F(m, \sigma_n)$ el valor de la función en la localidad (nivel de gris en la imagen, por ejemplo). Por otro lado, σ_n indica el valor de la desviación estándar

utilizada en la obtención de la representación *espacio de la escala* y de las diferencias de gaussianas; en el lado derecho de la ecuación, σ_l refiere a los niveles empleados en los *planos* anterior y siguiente de diferencias de gaussianas. Asimismo, el símbolo • se utiliza para hacer mención a los 26 vecinos de la localidad m, para que así $F(\bullet, \sigma_l)$ sea el valor de la función evaluada en cada vecino de m. En otros términos, la ecuación 2.6 busca y cataloga aquellas localidades donde el valor de la función sea superior a sus 26 vecinos.

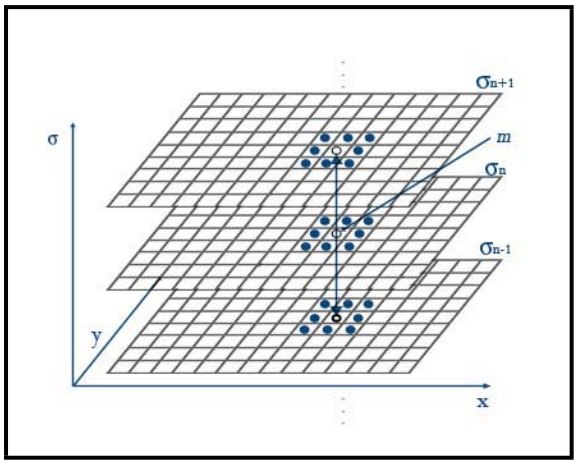


Figura 2.4: Detección de puntos de interés en la representación *espacio de la escala*

Se han desarrollado diferentes aproximaciones que identifican puntos de interés mediante la ecuación 2.6 ilustrada en la figura 2.4. Uno de los pioneros es el desarrollado por Crowley, [CP84], en donde se detectaban *picos* en *espacio de la escala* que posteriormente se colocaban en un árbol. Entonces, por medio de la estructura generada, se efectuaba el reconocimiento y registro entre un par de imágenes, inclusive conteniendo cambios de escala arbitrarios.

Lindeberg en su trabajo [Lin98] hace la búsqueda de *extremos* en la función normalizada de *Laplaciano de Gaussianas* o *LoG*. La representación *espacio de la escala* es construida por suavizado sucesivo sobre la imagen de alta resolución, usando funciones gaussianas con núcleos de diferente tamaño. Esta operación es circularmente simétrica y, por lo tanto, invariante a rotaciones.

Para 1999, [Low99] propusó un algoritmo eficiente para el reconocimiento de objetos basado en la extracción de características locales. Tales características consisten en puntos de interés obtenidos mediante la búsqueda 3D de *extremos* en la pirámide

de *escala -espacio*, construida ahora usando filtros de diferencias de Gaussianas, *DoG*. Se trata de una aproximación más eficaz, ya que la función *DoG* es una aproximación de la *LoG*. Además, su implementación produce una aceleración del proceso y una reducción en el tiempo de cómputo. Es así que se presenta como una herramienta ideal para la detección de características locales en tiempo real.

Con el devenir del tiempo y la tecnología, nuevas vertientes han sido desarrolladas y el campo de estudio en detección de puntos de interés ha sido extendido: Baumberg y Van Gool en 2000 con [Bau00] y [TV00], respectivamente, Mikolajczyk y Schmid en 2002 con [Mik02], Zisserman con [SZ02] y Lowe con [Low01] son sólo algunas de las investigaciones hechas.

En estos estudios se pretende resolver, además del problema de la escala, el de las transformaciones afines. Con ello, se puede efectuar un *empate* invariante sobre superficies planas bajo cambios en proyecciones ortográficas 3D. Sin embargo, ninguna propuesta es totalmente invariante, pues todas comienzan el proceso de identificación con variaciones afines en las imágenes, pues el proceso de explorar el espacio afín de cada una es costoso. Además, los *cuadros* afines son más sensibles al ruido que incluso la escala, por lo que las características puramente afines tienen menos repetibilidad que las de escala, sobretodo cuando la inclinación de las *superficies planas* va más allá de 40 grados [Low04]. A pesar de que presenta un panorama muy adverso esta condición, en práctica no lo es, ya que el conjunto de imágenes de entrenamiento debe tomarse al menos cada 30 grados, en cuanto a la rotación del punto de vista, para resolver cambios no planos y oclusiones que puedan presentarse.

Aun cuando los métodos que sustituyen el *LoG* por *DoG* son más rápidos, y en general ofrecen buenos resultados, pueden fallar cuando el máximo local está contenido en un borde o arista (el cambio en la señal se presenta en una sola dirección). Estos puntos son menos estables pues su ubicación es más sensible al ruido y a pequeños cambios en la textura de la localidad. Algunos métodos más sofisticados eligen la escala donde la traza y el determinante de la matriz hessiana, simultáneamente, asumen un extremo.

Además de los detectores basados en la representación *espacio de la escala*, existen otros como el de Kadir y Brady, [KB01], que usa maximización de la entropía de determinada región, el de Jurie, [JS04], cimentado en la extracción de bordes y el trabajo de Vincent Lepetit, [LF04a, LF04b, LF06]. Los dos primeros son un tanto más lentos en ejecución en contraparte de los últimos, que de hecho son utilizables

en tiempo real y en aplicaciones de *AR*. A diferencia de los métodos descritos con anterioridad, Lepetit no fundamenta su método en el análisis multiescala, sino que estudia el reconocimiento y rastreo como un problema de clasificación.

2.3.2. Descriptores locales

El siguiente paso importante en el proceso de reconocimiento de imágenes es definir *patrones* locales asociados a un punto detectado. El objetivo es obtener una descripción completa y compacta que permita emplear una medida de similitud entre dos imágenes. Los descriptores deben contener la información de la forma y textura de la estructura local. Dos de las propiedades más importantes que deben considerarse son el contenido de información y la invariancia, ya que determinan el nivel de distinción y robustez de los descriptores, además de ser recíprocamente dependientes: a mayor invariancia, menor información.

En materia de descripción, la más simple de ella la tiene la imagen en sí, ya que una medida de correlación cruzada puede utilizarse para determinar la similitud entre regiones o imágenes. Sin embargo, no han probado ser una buena opción, ya que dependen del tamaño de la región elegida. Entre más pequeña sea, se evalua mejor a la vecindad de una localidad, sin embargo, si la vecindad cambia demasiado entre el par o secuencia de imágenes, entonces la correlación resultará afectada y la medida no será adecuada o registrada como efectiva para la localidad y su vecindad. A medida que crece la región, el descriptor pierde la característica de ser local, y su calidad conlleva un detrimento.

Para Johnson y Hebert, [JH97, JH99], la representación es mejor descrita usando imágenes descriptivas asociadas a puntos en la superficie del objeto. Las imágenes —*spin-image*— son generadas usando la posición de los pixeles relativos al punto de interés. Los pixeles son elegidos con base en dos parámetros, produciéndose una imagen por punto de interés. Esta técnica proporciona una descripción de la forma de un objeto, ya que define la proyección 2D de posiciones relativas 3D. La descripción por *spin-images* es invariante a transformaciones rígidas y también robusta frente a oclusiones parciales, debido a la distribución acumulada de puntos. Una medida de correlación cruzada puede usarse para calcular la similitud entre dos *spin-images*.

Los descriptores basados en la previa extracción de contornos también son importantes. Entre ellos están el de [MC02], el de [Mik02] y [NA98]. El trabajo de

Mikolajczyk propone un descriptor que utiliza bordes locales ignorando a los cercanos, con lo que se tiene la posibilidad de encontrar características estables aun estando cerca de fronteras angostas o con forma de pico.

En 1999, Lowe presentó su trabajo [Low99], donde propone un descriptor basado en la respuesta de las neuronas en la corteza visual. Para lograr robustez ante distorsiones geométricas, la vecindad de un punto es representada mediante múltiples *imágenes*, consiguiendo así múltiples planos con diferentes orientaciones. Posteriormente, en [Low04] define un descriptor aun más robusto y que no es más que un histograma de los gradientes locales alrededor del punto de interés. Las posiciones de los histogramas son colocadas en un vector de 128 dimensiones: 8 de orientación, y un sub-esquema de 4x4 para la posición. Esta vertiente será explicada con mayor detalle en el siguiente capítulo. A causa de la calidad de tal descriptor, se han propuesto diversas modificaciones al mismo, entre ellas: *PCA-SIFT* [KS05], *GLOH* [MS05], *Fast Aproximated SIFT* [GB06], entre otras.

Una técnica con base en *momentos de color* es la descrita por Mindru en [MV99] y describe la naturaleza espectral de los datos, véase ecuación 2.7.

$$M_{pq}^{abc} = \int_{\Omega}\int x^p y^q \left[R\left(x,y\right)\right]^a \left[G\left(x,y\right)\right]^b \left[B\left(x,y\right)\right]^c dxdy \qquad \text{con orden } p+q \text{ y grado } a+b+c \tag{2.7}$$

R, G, B indican el nivel de intensidad del pixel (x, y) en cada plano de color: rojo, verde y azul, respectivente. Estos *momentos* son independientes y pueden ser calculados fácilmente para cualquier orden y grado. Además, caracterizan la forma, intensidad y la distribución de color sobre una región delimitada Ω. Por lo regular, se utilizan invariantes de segundo orden y primer grado, lo que provee de 18 componentes en el descriptor. Una nueva versión de este descriptor apareció en [MV02], que además es invariante a cambios afines en la iluminación. En algunos trabajos, Tuytelaars y Van Gool, utilizan la técnica descrita para representar regiones invariantes.

Tuytelaars y Van Gool en [TV06] proponen una vertiente basada en la matriz hessiana usando una aproximación muy básica. Se fundamenta en el cálculo de imágenes integrales para reducir el tiempo de cómputo, siendo así el detector *rápido hessiano*. El descriptor es una distribución de respuestas de *wavelets de Haar* operadas en la vecindad del punto de interés. El resultado final es un vector de 64 dimensiones, reduciendo así el tiempo de cómputo para tareas de reconocimiento.

2.4. Técnica elegida para el presente trabajo

Para la realización de este trabajo, se decidió implementar la técnica de Lowe [Low04]. Según los resultados expuestos en sus trabajos, es una operación robusta para fines de detección y rastreo. Además, ha probado ser ampliamente utilizada en ambientes *offline* y *online*, con lo que no se restringe su uso a aplicaciones de *AR*, sino que puede ser utilizado por sistemas de *image retrieval* —donde se solicitan imágenes que cumplan con ciertas características o bien aquellas que contengan a un objeto en particular—, o bien en aplicaciones de robótica, por mencionar un par.

Además, es conveniente contar con una biblioteca propia que proporcione el cálculo de la transformada SIFT. Durante el tiempo en que se investigó, sólo se encontraron dos implementaciones abiertas del algoritmo: la provista por *libsift* [Now04] y la de *sift*++ [Ved06]. Con el presente desarrollo, se tendrá una implementación más del algoritmo y esta vez hecha en casa. Con trabajo futuro sobre de ella, le dará un valor adicional, al poder ser utilizada por miembros de la comunidad estudiantil de la Universidad.

Retornando a lo concerniente con el algoritmo, también se eligió para probar entre dos vertientes de detectores y descriptores basados en características locales. En el laboratorio de Bio-Robótica de la Facultad de Ingeniería se desarrollaron a la par las dos vertientes. Una es la aquí comentada mientras que la otra es la desarrollada por otro estudiante del posgrado y que utilizar el detector de Harris en combinación con el descriptor de SURF (*Speeded Up Robust Features*) ayudado por un filtrado de Kalman, [Car07].

Otro punto importante es el enfoque que quiere que tenga a futuro esta biblioteca: ser base del proceso de visión utilizado por aplicaciones de *AR*. Por lo tanto, hay diferentes módulos que deben ser construídos para obtener el resultado. Haciendo una enumeración, pueden considerarse los siguientes:

1. Detección de elementos distintivos en la imagen.
2. Codificación de los elementos detectados.
3. *Matching* de las características extraídas contra una base de datos de *patrones*.
4. Cálculo de la transformación geométrica 3D con respecto de la cámara.

El algoritmo de Lowe, provee de todas las etapas comentadas, cubriendo así los requerimientos de los sistemas de *CV* empleados en aplicaciones de *AR*. Las primeras dos las abarca con la detección de puntos de interés, mientras que la segunda con el cálculo del descriptor para cada punto de interés. Los útimas dos módulos pueden cubrirse mediante diferentes técnicas a partir de los descriptores encontrados.

Capítulo 3

Teoría de características locales y SIFT

Una vez descrito el estado del arte de las técnicas para la extracción de características naturales y de la elaboración de su descripción en el capítulo anterior, en el presente se ofrece un breve estudio de la teoría del análisis multiescala en el que se basan y el funcionamiento del algoritmo elegido: SIFT, desarrollado por David G. Lowe.

3.1. Análisis Multiescala

La representación *espacio de la escala* se inició en la década de los 80. Entonces se consideraba a la escala como un parámetro continuo y se formularon la principales reglas de la teoría. Desde entonces, la representación y propiedades de esta teoría han sido estudiadas extensivamente y se han hecho importantes contribuciones. La representación multiescala es crucial para la extracción de características naturales, pues solamente existen en un rango limitado de escalas. Para poder extraer un gran número

de características, se explora a la representación de la imagen en un amplio intervalo de escalas. Por tal motivo, numerosos estudios han sido realizados y mostrado que el núcleo gaussiano es el óptimo para obtener la representación *espacio de la escala* de una imagen.

Las características locales están definidas por su ubicación, escala y forma, que son modificadas por transformaciones afines cuando se observa a la imagen desde diferentes ángulos. Para estimar la deformación afín de un punto de interés, se exploran las propiedades de la matriz de segundos momentos o hessiana.

3.1.1. El núcleo gaussiano y la representación *espacio de la escala*

En el dominio de las imágenes digitales, el parámetro de la escala es discreto, por lo tanto, la representación *espacio de la escala* es un conjunto de imágenes representadas en diferentes niveles discretos de resolución. Una de las propiedades importantes de esta *función* es la de cumplir con la ecuación de difusión, [Mik02], donde la solución es una convolución con un núcleo gaussiano. En dos dimensiones, este núcleo está definido por la ecuación 3.1.

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.1}$$

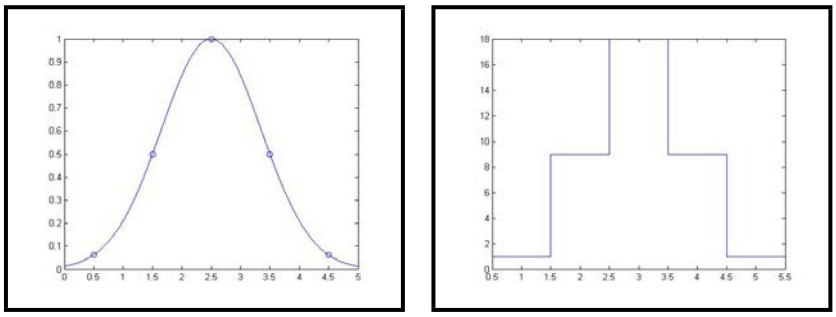
Entre las propiedades de las funciones gaussianas destacan la linealidad, separabilidad, y causalidad, entre otras. La separabilidad es por demás importante. En el caso específico del núcleo gaussiano, el interés radica en la capacidad de obtener núcleos de dimensiones superiores por medio del producto de núcleos unidimensionales, ecuación 3.2. Para fines de implementación, el *suavizado* de una imagen puede descomponerse en dos, uno por cada dimensión (renglones y columnas).

$$G(x, y, \sigma) = g(x, \sigma)\, g(y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2}{2\sigma^2}} + \frac{1}{2\pi\sigma^2} e^{-\frac{y^2}{2\sigma^2}} = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.2}$$

El filtro gaussiano utilizado por las herramientas construidas en este trabajo utiliza la especificación de Trucco en [TV98]. Para construir máscaras discretas de 1D, se debe de muestrear una gaussiana continua, comenzando por determinar el ancho del espacio discreto a utilizar.

El ancho w —típicamente un número impar— está relacionado con el valor de la desviación estándar σ y puede determinarse estableciendo que el rango denotado por w debe cubrir la mayoría del área bajo la curva. En [TV98] se indica que una elección adecuada es de $w = 5\sigma$, que abarca el 98.76 % del área. Ajustando tal porción de la gaussiana entre los extremos del intervalo, se encuentra que, por ejemplo, para un filtro de ancho 3 pixeles, se tiene una σ_3 de 0.6 pixeles y en general que $\sigma_w = w/5$.

Entonces, el muestreo de una gaussiana continua produce entradas discretas de un *núcleo* que, para fines de tiempo de filtrado, deben ser *aproximadas a enteros*. Así, cuando la representación de la imagen está en tipos de datos enteros, las operaciones son optimizadas y se elimina la necesidad de operar con punto flotante. Para ello, se normaliza el núcleo con respecto del dato más pequeño, se redondea el resultado, y se divide cada elemento entre la suma de los datos. La figura 3.1 muestra la gráfica de una gaussiana en 1D y su núcleo entero de tamaño 5: [1, 9, 18, 9, 1].



(a) Gaussiana continua y muestras reales (b) Núcleo gaussiano entero

Figura 3.1: Función gaussiana en 1D

El núcleo gaussiano permite crear espacios uniformes y afines, en el caso de SIFT, que es el algoritmo de interés, se utiliza un espacio uniforme.

Representación uniforme *espacio de la escala*

En este caso, los niveles utilizados son creados mediante la convolución de un núcleo gaussiano y de la imagen.

$$L(\mathbf{x}, \sigma) = G(\mathbf{x}, \sigma) * I(\mathbf{x}) \tag{3.3}$$

donde I es la imagen y $\mathbf{x} = (x, y)$ la ubicación del punto. El núcleo es circularmente simétrico y está parametrizado por un factor de escala σ. Una *escala* es obtenida por medio de la convolución de la imagen y el núcleo, y la sucesión de estas operaciones en los niveles calculados generan la representación multiescala; de forma que se acelere el proceso. Es importante elegir adecuadamente el factor de escala, pues puede conducir a problemas de *aliasing*.

Representación afín *espacio de la escala*

Se trata de una representación más general y sobretodo útil cuando la imagen contiene transformaciones afines. La representación es hecha por la convolución de la imagen con un núcleo gaussiano afín, como el de la ecuación 3.4.

$$G(\Sigma) = \frac{1}{2\pi\sqrt{\det\Sigma}} e^{-\frac{\mathbf{x}^T\Sigma^{-1}\mathbf{x}}{2}} \tag{3.4}$$

La matriz Σ es una matriz identidad multiplicada por dos escalares —σ_x, σ_y en la ecuación 3.5— y corresponde al núcleo gaussiano utilizado. Nótese que, si los escalares son iguales, entonces se tiene la representación uniforme del núcleo gaussiano. La forma de calcular la convolución con este tipo de filtros es utilizar una transformada de Fourier y multiplicar en frecuencia aunque también se han propuesto otros métodos recursivos. En [Mik02] se indica que el costo, en materia de cómputo, de calcular la respuesta del filtro en un punto de la imagen, es idéntico a la convolución sobre una imagen. El método que proponen consiste en descomponer el núcleo en un producto de matrices: rotación y escalamiento.

$$\Sigma = \mathbf{R}^{\mathbf{T}} \cdot \mathbf{D} \cdot \mathbf{R} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix} \begin{bmatrix} \cos(\phi) & \sin(\phi) \\ -\sin(\phi) & \cos(\phi) \end{bmatrix} \tag{3.5}$$

La ecuación 3.5 indica que el suavizado de una imagen usando un núcleo afín, corresponde a la convolución con un núcleo gaussiano elíptico y rotado.

3.1.2. Matriz de segundos momentos

La matriz de segundos momentos ha sido utilizada para la detección de características o la descripción de estructuras locales en una imagen. También es conocida como

matriz de autocorrelación y su uso comenzó a popularizarse cuando Harris, [HS88], creo su detector de *esquinas*. La matriz está definida como:

$$\mathbf{M}(\mathbf{x}) = \begin{bmatrix} \sum_W I_{xx}(\mathbf{x}) & \sum_W I_{xy}(\mathbf{x}) \\ \sum_W I_{xy}(\mathbf{x}) & \sum_W I_{yy}(\mathbf{x}) \end{bmatrix} \tag{3.6}$$

Describe la distribución del gradiente en una vecindad local W alrededor del punto $\mathbf{x}$ (representado por las segundas derivadas con respecto de x, y y la cruzada xy, I_{xx}, I_{yy} e I_{xy}, respectivamente). Los valores característicos de la matriz representan las curvaturas principales del punto; es decir, el cambio que sufre la señal en las direcciones ortogonales. Cuando los puntos cumplen con la característica de presentar cambios *significativos*, son más estables bajo condiciones arbitrarias de iluminación y, por lo tanto, son más representativos que otros.

Cuando se emplea una aproximación de la representación *espacio de la escala* mediante su expansión en serie de Taylor (ver ecuación 3.11), al segundo termino (el cuadrático) se le llama matriz hessiana. Es una matriz útil para describir las propiedades de estructuras locales en la imagen y está construida por las segundas derivadas parciales.

$$\mathbf{H} = \begin{bmatrix} L_{xx} & L_{xy} \\ L_{xy} & L_{yy} \end{bmatrix} \tag{3.7}$$

Las derivadas codifican la información referente a la forma, proveyendo de una descripción de como cambia la normal con respecto de una superficie. Son de interés particular los filtros que se basan en el determinante y la traza de esta matriz, por ejemplo, algunas de los usos especificados en [Mik02] son el uso del determinante para calcular el producto de las curvaturas principales de la superficie de intensidad en la imagen; la traza denota al filtro Laplaciano, utilizado a menudo en detección de bordes isotrópicos.

El Laplaciano es importante cuando se trata de gaussianas ya que es un modo de bio–percepción y puede ser aproximado por diferencias de gaussianas. Esto se verá más adelante, cuando se desglose el funcionamiento del algoritmo SIFT.

3.2. Algoritmo SIFT

En esta sección se explicará el funcionamiento del algoritmo empleado para la detección y descripción de características locales. Se trata del desarrollado por David Lowe [Low04] por las razones expuestas en el final del capítulo 2. El nombre de *SIFT* proviene de sus siglas en inglés: *Scale Invariant Feature Transformation*. Es una transformación sobre una imagen que extrae características locales que son invariantes a cambios en escala, rotación y parcialmente a cambios de iluminación y cambios 3D en la posición de la cámara. En concreto, transforma los datos de la imagen en coordenadas invariantes y relativas a características locales de la misma. Se divide en las siguientes fases:

1. Detección de *extremos*.
2. Designación de puntos de interés.
3. Asignación de orientación.
4. Descripción del punto de interés.

Uno de los aspectos importantes de este algoritmo es que genera un número importante de características que cubren dénsamente la imagen en todo el rango de escalas y localidades. Por ejemplo, una imagen típica de 500x500 pixeles, creará cerca de 2000 características estables (dependiendo del contenido de la imagen y la elección de varios parámetros). La cantidad de puntos detectados es de particular importancia en el reconocimiento de objetos, ya que la capacidad para encontrar objetos pequeños en escenas ocluidas o cortadas requiere que al menos 3 puntos sean *empatados* correctamente.

Los descriptores calculados son altamente distinguibles, lo que permite que un sólo punto de una imagen, encuentre a su correspondiente en otra escena con buena probabilidad en una base de datos con muchas *características locales*. Enseguida se presenta cada una de las etapas del algoritmo.

3.2.1. Detección de *extremos* locales

La vertiente propuesta se basa en un filtrado en cascada que utiliza algoritmos eficientes para identificar ubicaciones *candidatas* que, después, serán analizadas con

mayor detalle. La primera etapa es identificar localidades y escalas que puedan ser asignadas, repetidamente, en diferentes vistas del objeto/escena. Para lograrlo, se buscan características estables en todas las posibles escalas usando una función de escala continua: la representación *espacio de la escala*.

En las publicaciones de [Koe84, Lin94], se ha demostrado que el único núcleo válido para la construcción de la representación *espacio de la escala* es el gaussiano. Por lo tanto, la función que hace el mapeo sobre una imagen se define como $L(x, y, \sigma)$ y que se produce mediante la convolución de una gaussiana $G(x, y, \sigma)$ con la imagen $I(x, y)$. Esto está representado por la ecuación 3.8:

$$\begin{aligned} L(x, y\sigma) &= G(x, y, \sigma) * I(x, y) \\ \text{y } G(x, y, \sigma) &= \frac{1}{2\pi\sigma^2} e^{-\left(x^2+y^2\right)/2\sigma^2} \end{aligned} \tag{3.8}$$

En [Low99], el autor del algoritmo propone utilizar una función de *diferencia de gaussianas* (*DoG*) en convolución con la imagen, para buscar los *extremos* locales en la función *espacio de la escala*. Esto se representa por la ecuación 3.9.

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma) \end{aligned} \tag{3.9}$$

El término k que aparece en la ecuación 3.9 es un factor de separación entre dos escalas. La elección de una función *DoG* se debe, primero, a su facilidad y eficiencia en la implementación. La función L debe ser calculada para la descripción *espacio de la escala* y D mediante restas. Además, la función *DoG* ofrece una aproximación al *Laplaciano de gaussianas*, $\sigma^2\nabla^2 G$, tal y como se establece en [Lin94]. En tal estudio se muestra que el factor σ^2 es requerido para una verdadera invariancia a escala y en el desarrollado por [Mik02] que los máximos y mínimos del laplaciano, producen los puntos más estables en comparación con otras vertientes: gradiente de la imagen, *esquinas* de Harris, o la matriz hessiana, por mencionar algunas.

$$\begin{aligned} \frac{\partial G}{\partial \sigma} &= \sigma\nabla^2 G \\ \sigma\nabla^2 G = \frac{\partial G}{\partial \sigma} &\approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma} \\ (k-1)\,\sigma^2\nabla^2 G &\approx G(x, y, k\sigma) - G(x, y, \sigma) \end{aligned} \tag{3.10}$$

Entonces, la ecuación 3.10 muestra que cuando la *DoG* se crea mediante un factor de separación constante, k, el término necesario σ^2 es automáticamente incorporado, obteniendo así la invariancia a escala establecida en [Lin94]. El factor $k-1$ no influye en la detección y, claramente, el error en la aproximación tiende a cero cuando k tiende a 1, aunque en la práctica, se ha encontrado que casi no hay impacto en la estabilidad de los puntos encontrados.

Para fines de construcción de este módulo, las *DoGs* son obtenidas por el proceso iterativo mostrado en la figura 3.2. La imagen inicial es incrementálmente convolucionada con funciones gaussianas, variando el núcleo según k. Se emplean octavas y escalas, las primeras corresponden a submuestreos de la imagen en factores de 2, a excepción de la primera, donde la imagen inicial es duplicada en tamaño para así detectar más candidatos que contribuyen a la estabilidad, [Low04]. Las escalas corresponden a las imágenes producidas por la convolución de la imagen inicial de la octava y de las gaussianas.

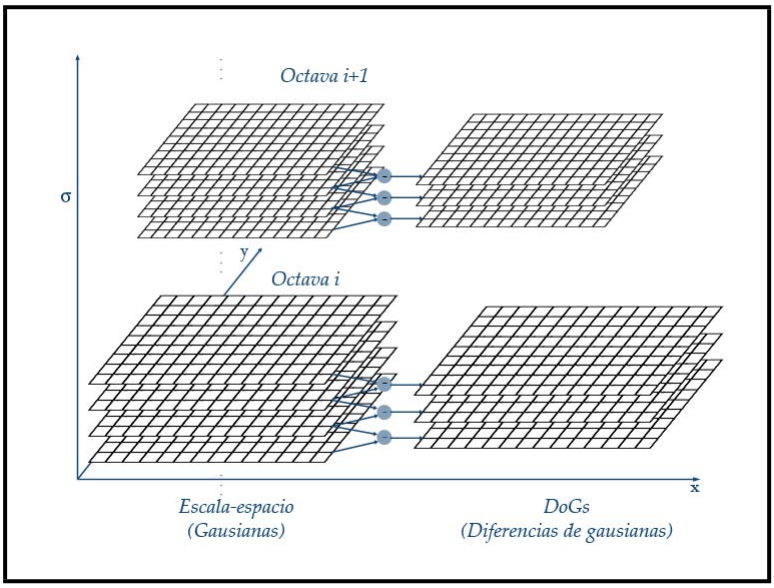


Figura 3.2: Construcción de la función de *espacio de la escala*.

Para tener una buena detección de características (lo que se verá más adelante), se deben producir al menos 4 imágenes por octava, en la representación *espacio de la escala*. Luego, la resta entre ellas define a la función *DoG* –véase figura 3.2– de la octava, y de estas se deben producir al menos 3. La razón es que la búsqueda de

extremos en la imagen, se efectúa escala por escala, tomando cada pixel como pivote de un *cubo* (escala presente, anterior y siguiente) y analizándolo contra los pixeles contenidos en él. Una vez que se calculó una octava completa, se submuestrea la última imagen y se procesa la siguiente octava.

Una vez concluida la construcción de la representación *espacio de la escala*, la función *DoG*, $D(x, y, \sigma)$, es utilizada para detectar puntos candidatos a características invariantes. Como ya se mencionó, cada pixel en $D(x, y, \sigma)$ es comparado contra sus ocho vecinos en la escala de análisis, y dieciocho vecinos en las escalas anterior y posterior. En resumen, el pixel analizado es candidato si cumple con ser mayor a cierto umbral, y mayor o menor a sus 26 vecinos. Esto puede verse en la figura 2.4. El costo, en materia de cómputo, es relativamente bajo, ya que la mayoría de los puntos serán descartados después de unas pocas comparaciones o, incluso, desde el inicio con el umbral.

Al final de este proceso, se tendrá un gran grupo de características identificadas. Algunas de ellas son muy *débiles* y por ello no cumplen con la repetibilidad deseada. A continuación se presenta el esquema propuesto para elegir los puntos más estables, obteniendo así algo más cercano al grado de repetibilidad buscado.

3.2.2. Designación de puntos de interés

Una vez detectado un candidato a punto de interés, le sigue determinar si será lo suficientemente estable haciendo un ajuste detallado de su localización espacial, escala y proporción de sus curvaturas principales (matriz hessiana). Así, se consigue que aquellos puntos con bajo contraste, sensibles a ruido, o pobremente localizados sobre un borde, sean descartados. Este trabajo esta dividido en dos fases, la primera de *interpolación* y la segunda, de eliminación de respuesta en los bordes mediante la matriz hessiana.

Interpolación y ajuste del punto candidato

En su trabajo de 2002, [Low01], Lowe y Brown desarrollaron un método para ajustar una función cuadrática en 3D, a la muestra local de puntos detectados y, así determinar la ubicación interpolada del extremo. Sus experimentos han demostrado que este ajuste produce una mejora sustancial en estabilidad y reconocimiento. En concreto, la

función es aproximada por la expansión en serie de Taylor de la expresión de *espacio de la escala*, hasta los términos cuadráticos. La ecuación 3.11 refiere esto.

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T\frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x} \tag{3.11}$$

donde D y sus derivadas son evaluadas en el punto candidato y $\mathbf{x} = (x, y, \sigma)^T$ es el *offset* de tal punto. Entonces, la ubicación del extremo $\hat{\mathbf{x}}$ es obtenida tomando la derivada de la ecuación 3.11 y haciéndola cero (ecuación 3.12).

$$\hat{\mathbf{x}} = -\frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1}\frac{\partial D}{\partial \mathbf{x}} \tag{3.12}$$

Para tener un cómputo eficiente, las derivadas de D son evaluadas mediante restas entre vecinos. El resultado es un sistema de ecuaciones de 3x3 que es resuelto con costo mínimo. Para entenderlo, desarróllese el contenido de la ecuación 3.11. Se sabe que $D(\mathbf{x})$ es una función escalar de variable vectorial, entonces...

$$D(\mathbf{x}) = D + \begin{bmatrix} \frac{\partial D}{\partial x} & \frac{\partial D}{\partial y} & \frac{\partial D}{\partial \sigma} \end{bmatrix}\begin{bmatrix} x \\ y \\ \sigma \end{bmatrix} + \frac{1}{2}\begin{bmatrix} x & y & \sigma \end{bmatrix}\begin{bmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial xy} & \frac{\partial^2 D}{\partial x\sigma} \\ \frac{\partial^2 D}{\partial xy} & \frac{\partial^2 D}{\partial y^2} & \frac{\partial^2 D}{\partial y\sigma} \\ \frac{\partial^2 D}{\partial x\sigma} & \frac{\partial^2 D}{\partial y\sigma} & \frac{\partial^2 D}{\partial \sigma^2} \end{bmatrix}\begin{bmatrix} x \\ y \\ \sigma \end{bmatrix} \tag{3.13}$$

En 3.13 se encuentra que $\partial^2 D/\partial \mathbf{x}^2$ es una matriz simétrica y para evaluar su inversa, puede utilizarse un método simple como el de la adjunta y el determinante. Entonces, primero se verifica que el determinante sea diferente de cero y, en caso afirmativo, se cálcula la matriz adjunta, 3.14.

$$\begin{aligned}
\det\left(\frac{\partial^2 D}{\partial \mathbf{x}^2}\right) &= \left|D^2_{\mathbf{x}}\right| \\
\left|D^2_{\mathbf{x}}\right| &= D_{xx}\left(D_{yy}D_{\sigma\sigma} - \left(D_{y\sigma}\right)^2\right) - D_{xy}\left(D_{xy}D_{\sigma\sigma} - D_{y\sigma}D_{x\sigma}\right) + D_{x\sigma}\left(D_{xy}D_{y\sigma} - D_{yy}D_{x\sigma}\right) \\
\frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1} &= \frac{\mathrm{adj}\left(\frac{\partial^2 D}{\partial \mathbf{x}^2}\right)}{\left|D^2_{\mathbf{x}}\right|} \\
\frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1} &= \frac{1}{\left|D^2_{\mathbf{x}}\right|}\begin{bmatrix} D_{yy}D_{\sigma\sigma} - \left(D_{y\sigma}\right)^2 & D_{x\sigma}D_{y\sigma} - D_{xy}D_{\sigma\sigma} & D_{xy}D_{y\sigma} - D_{x\sigma}D_{yy} \\ D_{y\sigma}D_{x\sigma} - D_{xy}D_{\sigma\sigma} & D_{xx}D_{\sigma\sigma} - (D_{x\sigma})^2 & D_{x\sigma}D_{xy} - D_{xx}D_{y\sigma} \\ D_{xy}D_{y\sigma} - D_{yy}D_{x\sigma} & D_{xy}D_{x\sigma} - D_{xx}D_{y\sigma} & D_{xx}D_{yy} - \left(D_{xy}\right)^2 \end{bmatrix}
\end{aligned} \tag{3.14}$$

Entonces, a partir de lo anterior, el sistema de la ecuación 3.12 es resuelto con costo mínimo, pues está plenamente definida la solución a la matriz inversa, y la multiplicación por el *vector* $D_{\mathbf{x}}$ es directa. Cuando el *offset* $\hat{\mathbf{x}}$ es mayor al escalar 0.5 en cualquiera de sus dimensiones, el *extremo* está más cercano a otro punto. Es así que el punto candidato cambia, y se efectúa una nueva interpolación alrededor de tal localidad. Finalmente, el *offset* es sumado a la posición original.

La función $D(\hat{\mathbf{x}})$, valuada en el extremo encontrado, es utilizada para rechazar extremos con bajo contraste. Esto puede obtenerse sustituyendo la ecuación 3.12 en 3.11. De ello resulta 3.15:

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2}\frac{\partial D}{\partial \mathbf{x}}^{T}\hat{\mathbf{x}} \tag{3.15}$$

Eliminación de respuesta a bordes

Para ofrecer mayor estabilidad, no es suficiente con rechazar a los candidatos que tengan bajo contraste. La función *DoG* tiene una fuerte respuesta en los bordes en una imagen. Un *pico* vagamente definido en la *DoG* tendrá un valor *grande* de curvatura principal a lo largo del borde, pero uno pequeño en la dirección perpendicular. Las curvaturas principales pueden determinarse a partir de una matriz hessiana de 2x2, **H**, calculada en la ubicación y escala del punto de interés, véase ecuación 3.16.

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{3.16}$$

Las derivadas son estimadas mediante diferencias entre vecinos y los valores característicos de la matriz H son proporcionales a las curvaturas principales de D. Considerando la propuesta de [HS88], no interesa obtener los valores característicos, sino únicamente su proporción. Entonces, sea α el valor característico de mayor magnitud y β el de menor. La traza de **H** representa la suma de ellos y el determinante su producto, véase la ecuación 3.17.

$$\begin{aligned} \mathrm{Tr}(\mathbf{H}) &= D_{xx} + D_{yy} = \alpha + \beta \\ \mathrm{Det}(\mathbf{H}) &= D_{xx}D_{yy} - \left(D_{xy}\right)^2 = \alpha\beta \end{aligned} \tag{3.17}$$

En el caso de que el determinante sea negativo, las curvaturas toman diferentes signos y el punto debe ser descartado. Sea r la proporción existente entre las magnitudes de los valores característicos, dígase $\alpha = r\beta$. Entonces. . .

$$\frac{\mathrm{Tr}\,(\mathbf{H})^2}{\mathrm{Det}\,(\mathbf{H})} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r} \tag{3.18}$$

En 3.18 se nota que el elemento importante es la proporción entre las magnitudes, no de los valores en sí. Cuando los valores caracterśticos son iguales, la cantidad $(r+1)^2/r$ es mínima. Por lo tanto, la verificación se basa en la proporción entre las curvarturas, siendo ésta. . .

$$\frac{\mathrm{Tr}\,(\mathbf{H})^2}{\mathrm{Det}\,(\mathbf{H})} < \frac{(r+1)^2}{r} \tag{3.19}$$

La ecuación 3.19 se puede implementar eficientemente, con menos de 20 operaciones flotantes para probar cada candidato. Con este paso se finalizan las pruebas sobre puntos candidatos y se cuenta con un conjunto estable de características naturales en la imagen.

3.2.3. Asignación de orientación

En la dos secciones anteriores, se ha explicado como detectar candidatos a puntos de interés, y como elegir a los mejores entre ellos. El siguiente paso es sentar las bases para la creación de una descripción de la característica local. Lowe propone hacer una representación relativa a la orientación del punto y así obtener invariancia a rotación. Esta vertiente contrasta con la de Schmid y Mohr, donde cada propiedad tiene una medida rotacionalmente invariante. La desventaja de tal método radica en que limita al descriptor y descarta información al no requerir que todas las medidas se basen en una rotación consistente.

Experimentalmente, en [Low04], se encontró que el siguiente método ofrecía los mejores y más estables resultados. La escala del punto de interés es usada para seleccionar la imagen L más cercana; con esto se consigue la invariancia a escala. Enseguida, para cada muestra $L(x,y)$ en la escala σ del punto de interés se evalúa el gradiente: $\mathrm{m}(x,y)$ y $\theta(x,y)$, ecuación 3.20.

$$\begin{aligned} \mathrm{m}(x,y) &= \sqrt{(L(x+1,y)-L(x-1,y))^2+(L(x,y+1)-L(x,y-1))^2} \\ \theta(x,y) &= \arctan\left(\frac{(L(x,y+1)-L(x,y-1))^2}{(L(x+1,y)-L(x-1,y))^2}\right) \end{aligned} \tag{3.20}$$

Posteriormente, alrededor del punto de interés se determina una vecindad que lo contenga y de la que sea su centro. Luego, para cada punto en la vecindad, se calcula su gradiente y se crea un histograma de orientaciones de 36 posiciones; una posición por cada 10 grados y así cubrir el intervalo de los 360 grados. La magnitud de cada muestra que se agrega al histograma, es ponderada por una ventana circular gaussiana con $\sigma_w = 1.5\sigma_k$. La ventana define a la vecindad y, tal como se especifica en [TV98], el diámetro de ella es $5\sigma_w$.

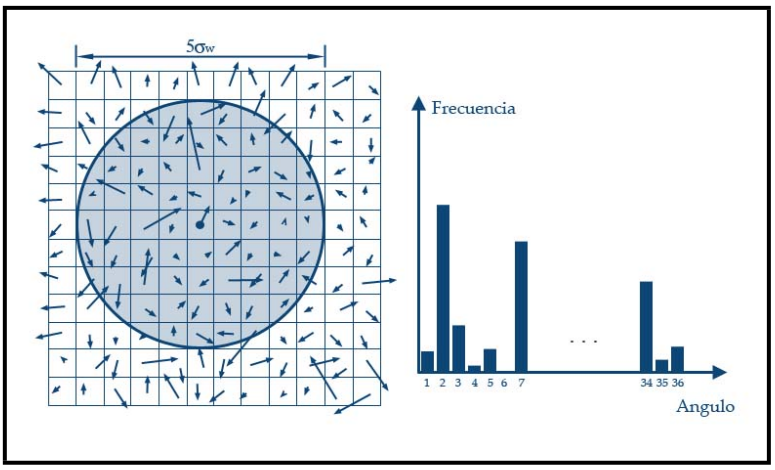


Figura 3.3: Ventana circular e histograma por punto de interés.

Los picos del histograma corresponden a las orientaciones dominantes del gradiente local. El valor del pico más grande es la orientación principal y aquellos mayores al 80 % del principal son usados para crear más descripciones del punto de interés. Por lo tanto, para ubicaciones con múltiples picos, habrá múltiples descripciones con la misma localidad y escala pero diferente orientación. Tener más de una orientación contribuye a la estabilidad del punto de interés. Finalmente, se interpola el valor real del pico ajustando a una parábola con los datos adyacentes en el histograma.

3.2.4. Descripción del punto de interés

En la etapa anterior, a cada punto de interés se le asignó al menos una orientación. En conjunto con sus otros parámetros —ubicación y escala—, se define un sistema de coordenadas 2D que describe la región y provee de invariancia a tales atributos. En la etapa actual, se calcula un descriptor para la región *altamente distinguible* e invariante a otras posibles variaciones.

Una forma obvia consistiría en muestrear las intensidades locales alrededor del punto de interés en la escala apropiada, y utilizar una medida de correlación para determinar las posibles correspondencias. Sin embargo, las medidas de correlación han resultado ser muy sensibles a cambios afines y otros tipos de deformación en proyecciones 2D de mundos 3D.

En un trabajo de Edelman, Intrator y Poggio, [EI97], que Lowe refiere, se propone una representación basada en un modelo biológico, en específico el de la corteza visual primaria. Las neuronas de esa zona responden a un gradiente con una orientación y frecuencia en particular, pero la ubicación de él sobre la retina puede estar *defasado* alrededor de un pequeño campo de receptores y no con una localidad precisa. La hipótesis de aquel entonces consistió en establecer que la función de esas neuronas era *crear vínculos o correspondencias* de objetos 3D entre imágenes a partir de un intervalo de puntos de observación. Los reportes de sus experimentos mostraron excelentes resultados bajo rotaciones, incrementando la taza de reconocimiento y empate de 35 % (usando medidas de correlación) al 94 %.

La descripción de Lowe toma como idea principal la recién explicada y se muestra en la figura 3.4. Primero, se calculan los gradientes de una vecindad alrededor del punto de interés; es importante notar que el vecindario no del mismo tamaño que en la fase anterior y enseguida se explicará su obtención. La región es tomada de la imagen con escala σ, igual a la del punto de interés. Para poder obtener invariancia a rotación, las orientaciones de los gradientes son rotadas con respecto de la orientación asignada en la etapa anterior.

Luego, una función gaussiana con $\sigma = \frac{1}{2}W_{win}$ (donde W_{win} es el ancho de la ventana del descriptor) es utilizada para ponderar la magnitud de cada gradiente muestreado. El propósito de la ventana es darle menor importancia a las muestras de la frontera, pues son las más afectadas por ruido y otros errores. Además, así tienden a suprimirse errores causados por pequeñas fallas en la detección del punto de interés, ya que la información no proviene sólo de él, sino de una región alrededor del mismo.

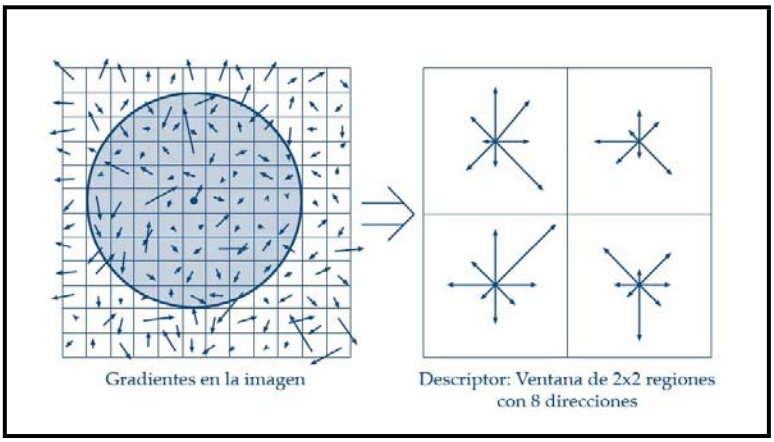


Figura 3.4: Construcción del descriptor.

El lado derecho de la figura 3.4 muestra al descriptor. En realidad, es un arreglo 3D de datos con dimensiones: $W_{win}, H_{win}, N_\theta$ o ancho, alto y *profundidad*, representada como un histograma más de N_θ posiciones. En el caso de la figura 3.4, el descriptor (mostrado en a la derecha) consiste de regiones muestrales de 4x4 pixeles con histogramas de 8 direcciones para producir el descriptor de tamaño 2x2x8. La longitud de las flechas en el descriptor, representa la magnitud de esa posición del histograma. Una muestra del gradiente puede estar defasada hasta cuatro posiciones y todavía contribuir al mismo histograma; así se logra una mejora en estabilidad cuando no hay una buena detección del punto de interés y además está *movido* unos cuantos pixeles respecto de su posición verdadera.

Como parte del proceso, es importante evitar los *efectos de frontera* que puedan presentarse y en donde el descriptor cambia abruptamente por muestras que cambian de un histograma a otro o de una orientación a otra. Por ello, se utiliza una interpolación trilineal para distribuir el valor de cada muestra en adyacencias del histograma. Expresado en otros términos, cada valor del histograma es ponderado por $1 - d$ en cada dimensión, siendo d la distancia del centro de la *dimensión* a la muestra.

El descriptor usado por Lowe se muestra en la figura 3.5. La estructura del descriptor está formada por un vector que contiene los valores de todos los histogramas que corresponden a las flechas de la figura 3.5. El ancho y alto del descriptor es de 4x4 y los histogramas empleados de 8 espacios. El vector resultante es de 128 elementos: $\begin{bmatrix} W_{win} & H_{win} & N_\theta \end{bmatrix} = \begin{bmatrix} 4 & 4 & 8 \end{bmatrix}$.

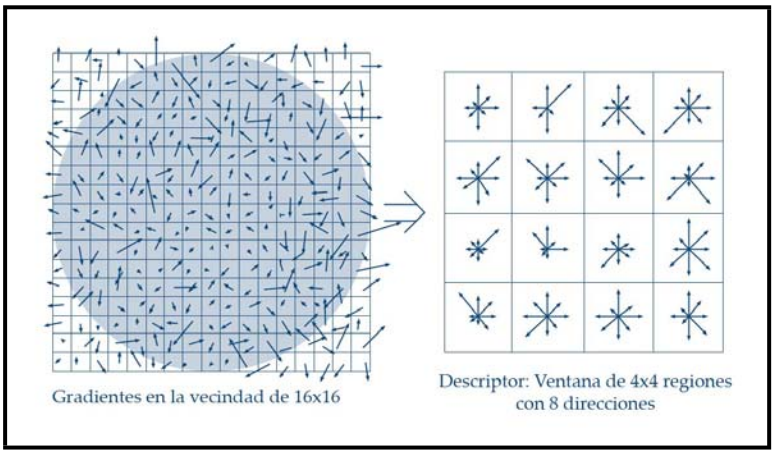


Figura 3.5: Descriptor propuesto por Lowe en [Low04].

Finalmente, el vector es modificado para reducir los efectos producidos por cambios en la iluminación. Primero se normaliza al vector, de forma que se elimine al escalar causante de cambios en contraste de los pixeles y las magnitudes de los gradientes en los histogramas. Luego, aquellas componentes del vector que sean mayores a 0.2 son modificadas, asignándoles el máximo —*0.2*— y el vector es normalizado nuevamente. En la siguiente sección se explica el porque de esto.

3.2.5. Consideraciones experimentales

En el artículo de Lowe, [Low04], se han definido experimentalmente los parámetros de ejecución del algoritmo. Para una correcta implementación, deben conocerse y entenderse, por ello aquí se muestran tal como recomienda el autor que deben de ser utilizados. Los siguientes son los referentes a la detección de puntos de interés y su refinamiento.

- **Número de octavas:** Los experimentos mostraron que para **3** octavas se presenta la máxima repetibilidad y, cuando se utilizan más octavas, la repetibilidad no aumenta. La explicación radica en la fase de detección de candidatos. En efecto, aumentar el número de octavas hace que se detecten más candidatos, pero con mayor sensibilidad al ruido y con bajo contraste. En la siguiente etapa, la de *refinamiento*, aquellos puntos con las características recién enunciadas son eliminados. Es así que no es conveniente utilizar un mayor número de octavas.

- **Número de escalas:** El número de escalas debe ser superior a **4**, pues la resta entre ellas producirán las diferencias de gaussianas y se necesitan al menos 3 para poder detectar candidatos a puntos de interés. Sin embargo, los experimentos hechos por Lowe muestran que cuando se hacen 6 escalas se obtiene la máxima repetibilidad.

- **Valor inicial de σ:** Para comenzar con la construcción de la representación *espacio de la escala*, el valor de la desviación estándar utilizado es de **1.6**. En los resultados expuestos por Lowe, [Low04], este valor permite tener máxima repetibilidad.

- **Duplicado del tamaño de la imagen original:** La imagen con la que se inicia el algoritmo debe ser duplicada en tamaño. La operación es efectuada mediante interpolación lineal y es equivalente a un *suavizado* en la imagen, descartando así las altas frecuencias. Tal filtrado permite detectar más candidatos a puntos de interés.

- **Determinación de bajo contraste:** Los valores del extremo interpolado (ver sección 3.2.2) que cumplan con $|D(\hat{\mathbf{x}})| < 0.3$ serán descartados para proveer mejor estabilidad.

- **Prueba de curvatura:** Los experimentos hechos en el artículo utilizan un valor de $r = 10$, que elimina aquellos puntos con curvaturas mayores a 10.

Para fines de estabilidad del descriptor, el vector resultante debe ser normalizado para reducir/eliminar los efectos que cambios en la iluminación pueden producir. Cambios en el contraste de la imagen en donde el valor de cada pixel es multiplicado por una constante, también escalará las magnitudes de los gradientes. Entonces, la normalización inicial del vector, tiene como finalidad eliminar tal constante de escalamiento. Los cambios de brillantez no afectan a los gradientes, pues son evaluados mediante diferencias entre vecinos.

Con este paso, el descriptor es invariante a cambios afines de iluminación. Sin embargo, otro tipo de efectos pueden producirse por saturación de la cámara, o la iluminación sobre superficices 3D. Los cambios que producen se reflejan en cambios repentinos de las magnitudes de algunos gradientes, pero no tienden a afectar a la orientación. Por ello, para reducir tales cambios, se pueden utilizar un umbral para limitar las magnitudes —experimentalmente, se ha determinado que sea de 0.2— y nuevamente normalizar al vector.

A su vez, el descriptor mostró ser óptimo para 8 direcciones en el histograma y un arreglo de 4x4 histogramas. Mientras que para un sólo arreglo de histogramas el reconocimiento es muy pobre (menor al 5 %), con sólo llevarlo a 9 (3x3) regiones aumentó al 37 % aproximadamente. Los mejores resultados se presentan cuando se tiene un arreglo de 4x4 y 8 direcciones, alcanzando el 50 % en la búsqueda del descriptor más cercano en la base de datos. Es importante mencionar que estos resultados son válidos para una superficie plana que ha sido girada hasta 50 grados en profundidad (una cifra alta para fines de detección) y con adición de 4 % de ruido.

Continuando con más experimentos, se encontró que la repetibilidad del descriptor está entre el 40 % y el 80 % cuando se aplican distorsiones afines en el punto de observación (de 50 a 0 grados, respectivamente).

Capítulo 4

Descripción del sistema desarrollado

Conociendo las técnicas útiles y necesarias para extraer características locales en una imagen, en este capítulo se abordará el asunto específico de su implementación en computadora, los módulos desarrollados y las técnicas auxiliares empleadas para alcanzar el objetivo.

En cuanto al trabajo realizado, se trata de un conjunto de bibliotecas de programación que permiten entrenar, reconocer y rastrear *marcas naturales* para su uso en tareas de visión por computadora. En específico, se han diseñado para emplearse en futuras aplicaciones de *AR*. Debido a la pretensión de utilidad en otras aplicaciones de *CV*, la construcción ha sido realizada desde las *primitivas* de representación, tratando de hacer el menor uso posible de código ajeno para proveer una fácil y amplia integración con otros sistemas en desarrollo.

A su vez, como aportación a las implementaciones disponibles del algoritmo SIFT de David Lowe, éste ha sido llevado a múltiples procesos para una mejora

en el rendimiento y velocidad de ejecución. Ha sido desglosado en 3 etapas que corresponden a:

1. **Detección de puntos candidatos.** Un proceso se especializa en la obtención de la representación *espacio de la escala*, generación de las *DoG* y detección de localidades candidatas a puntos de interés.

2. **Asignación de puntos de interés.** Los puntos encontrados en la etapa anterior son evaluados por este proceso, especializado en su refinamiento por medio de la interpolación y eliminación de respuesta a bordes, métodos descritos en el capítulo anterior.

3. **Cálculo del descriptor del punto de interés.** Simplemente se toman a los puntos de interés *sobrevivientes* y su descriptor es calculado.

Posterior a esto, se definen dos procesos más, específicos para las tareas de *empate* (*matching*) de puntos de interés, respecto de un conjunto de puntos previamente suministrado, y del cálculo de la transformación (*homografía*) que mapea los puntos recien calculados a los suministrados.

En un diagrama de bloques, el sistema *final* operaría como se muestra en la figura 4.1. Como se puede observar, ambos módulos —análisis de video en tiempo real (bloque *Video* y de imágenes (*Imagen referencia*)— utilizan al bloque SIFT. Se trata del *núcleo* del software construido y sus etapas son desglosadas en la parte inferior de la figura. Más adelante, en la sección 4.4 se mostrará a detalle cada uno de los módulos y en las subsecuentes secciones se hace un análisis de las implementaciones orientadas a objetos, multihilos y para .NET.

El esquema multiproceso adoptado es de *pipelining*. Haciendo una analogía con los microprocesadores RISC, en donde toda instrucción toma el mismo número de pulsos de reloj en ser ejecutada, aquí se ha considerado que toda etapa tomará igual o menor tiempo al de la más tardada.

A continuación se presenta el trabajo desarrollado comenzando por las *primitivas* o elementos base de la implementación.

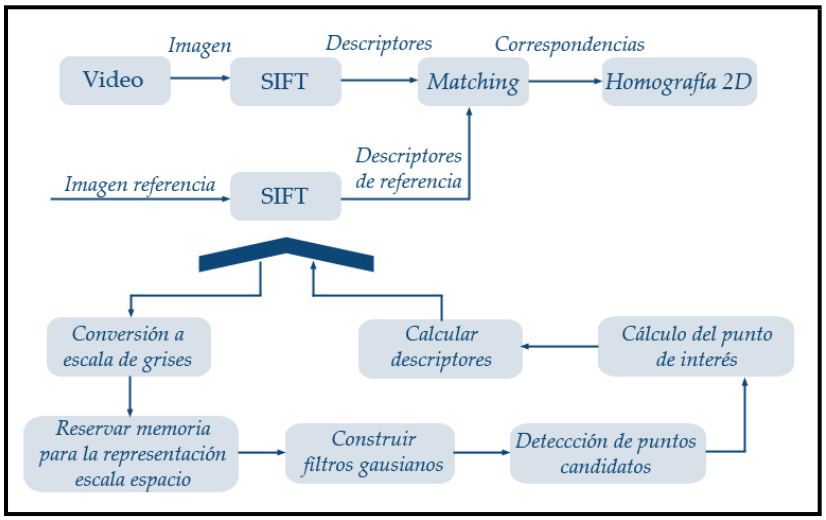


Figura 4.1: Diagrama de bloques del sistema

4.1. *Software* construido

El *sotware* está programado en C/C++ y las aplicaciones con GUI en C#. Consta de varios módulos conectables entre sí y que resuelven tareas específicas. Los datos primigenios —imágenes y filtros, por mencionar un par— son representados mediante clases, aunque ciertas operaciones son definidas en un paradigma estructurado más que orientado a objetos. Lo anterior para promover una sencilla integración en aplicaciones con uno u otro paradigma.

Para la captura y despliegue de video se utilizaron las APIs de Microsoft, *DirectShow* en conjunto con *Direct3D* y como lenguaje a C++. El módulo de video está bien separado del resto del código de procesamiento, de modo que sean piezas conectables, pero no dependientes una de otra. Dado que las GUIs son quienes más uso hacen de utilerías para adquisición de video, el módulo destinado a ello pudo haberse implementado directamente en C#, salvo que no hay forma de vincularlo directamente con las APIs. Por ello se construyó primero la versión de C++ y después, mediante C++/CLI (código administrado para .NET) se elaboró el *wrapper* para C#.

La codificación del algoritmo SIFT está escrita en C y C++ y también cuenta con un *wrapper* para C#. La versión de C++ está construida bajo el paradigma orientado a objetos y cuenta con dos implementaciones, una que se ejecuta en un sólo hilo y otra que es multihilos para realizar las funciones anteriormente comentadas.

4.2. Adquisición de video

El esquema utilizado de adquisición de video es similar al desarrollado en [JJ04]. La captura se hace por medios de *DirectShow*. La filosofía o esquema de programación del API consiste en definir bloques con conexiones y unirlos. A cada uno de ellos se les denomina *filtros*. Un esquema de los pasos involucrados en la inicialización del video se muestra en la figura 4.2.

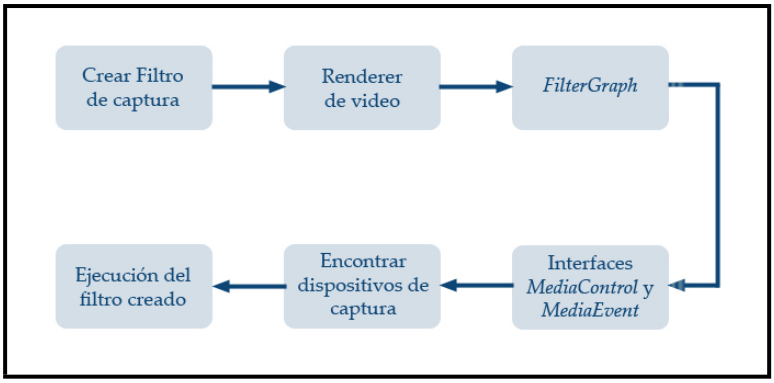


Figura 4.2: Proceso para inicial *DirectShow*

Para comenzar, es necesario definir los filtros y construir su *gráfica* de control: *FilterGraph*. Este elemento es el componente central de toda aplicación que requiera vinculación con *DirectShow*, pues define el comportamiento y tareas del *filtro*; como ejemplo de tareas están la notificación y sincronía con eventos por mencionar un par. Después de la construcción del *renderer* —*filtro* encargado de transformar el video adquirido en una imagen lista para ser dibujada— es importante crear las instancias de las interfaces *MediaControl* y *MediaEvent*. La primera provee de métodos para el control de la ejecución en la adquisición y la segunda para atender notificaciones de tareas asociadas al proceso de captura.

La gráfica de filtros recién comentada es la encargada de la adquisición de video. Para crear un vínculo con *Direct3D* se han diseñado dos clases: *CTextureRenderer* y *CCustomPresentation*; el código base pertenece al SDK de *DirectX*.El primer objeto implementa las tareas relacionadas con la captura y procesamiento de video, mientras que el otro sólo se comunica con el primero para adquirir un *buffer* que contiene a la imagen. Para agilizar el proceso, la imagen se mapea como textura, dejando la tarea de despliegue a *Direct3D*, que lo ejecutará con mejor eficiencia que si se tuviera que

dibujar pixel a pixel en cada cuadro procesado por medios propios; figura 4.3.

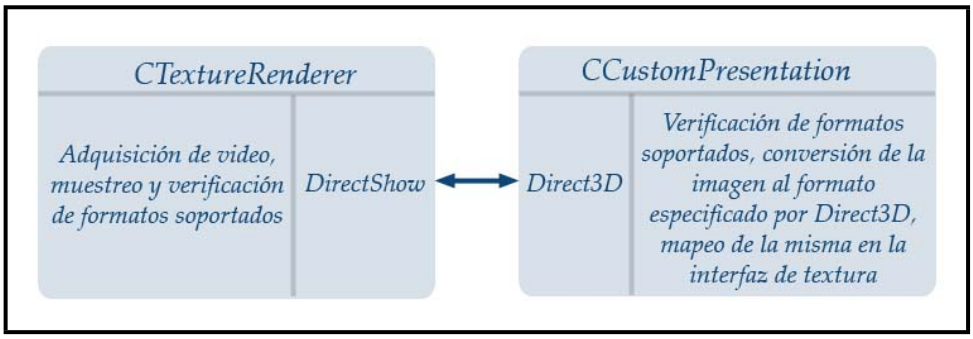


Figura 4.3: Vínculo entre *DirectShow* y *Direct3D*

4.3. Representación de datos

El video y las imágenes digitales adquiridas, corresponden al tipo de información con el que se debe operar. En el caso de video digital, puede ser visto como un flujo de imágenes adquiridas a una frecuencia alta, considérese mayor a 24 imágenes por segundo; las imágenes adquiridas con tazas superiores a la cifra anterior, producen en el ojo humano un efecto de continuidad.

Las imágenes componen a la primitiva esencial, pues son la información con que operar. A su vez, es necesario definir un conjunto de operaciones sobre ellas: a nivel de imagen y de modificadores *externos*. Para el caso de la implementación requerida, se definieron dos clases: imagen y filtro. A continuación se describen cada una de ellas.

4.3.1. Clase *Image*

Es la clase principal de la que depende el funcionamiento interno de métodos, procedimientos y programas de mayor nivel. Provee de los métodos necesarios para cargar imágenes en diferentes formatos, acceder a sus propiedades y guardarlas. Por otro lado, la interacción con otros objetos se efectúa mediante un esquema procedimental. Se diseñaron las funciones —necesarias para SIFT— que realizan operaciones entre imágenes: suma, substracción, re-dimensionamiento, conversión a escala de grises y la convolución con algún filtro. La figura 4.4 ejemplifica esto.

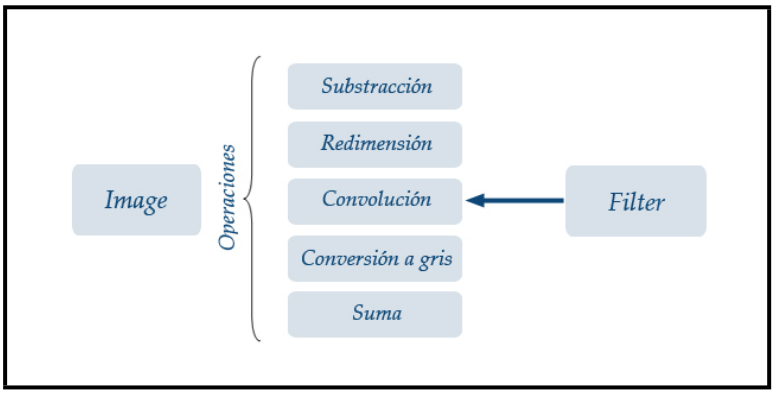


Figura 4.4: Clase *Image* y sus operaciones

Es importante mencionar que para acceder al *buffer* de la imagen, puede hacerse via el buffer lineal (pixeles consecutivos y, en caso de ser en color, entrelazados) o bien, mediante un apuntador a arreglo bidimensional (arreglo rengón-columna) para un manejo más sencillo.

4.3.2. Clase *Filter*

Se trata de una clase que permite definir filtros y utilizarlos sobre imágenes donde la operación básica es la de convolución con un *buffer* de datos. Su diseño es orientado a objetos usando herencia y polimorfismo. Esto es, existe una clase base *Filter* donde se definen el núcleo del filtro, su tamaño, identificador del tipo de filtro y un método abstracto para efectuar la convolución.

Luego, se han creado 4 *especializaciones* de filtros: gaussiano, mediana, Sobel y de caja (*box*). Cada uno de ellos cuenta con sus métodos y atributos necesarios para construir el núcleo requerido y definir su comportamiento en la operación de convolución.

4.4. Implementación del algoritmo SIFT

Los tipos de datos anteriormente enunciados, permiten programar el algoritmo elegido. Tal vertiente ha sido desarrollada intentando cubrir ambos paradigmas: estructurado y orientado a objetos. En realidad, la versión orientada a objetos es un *wrapper*

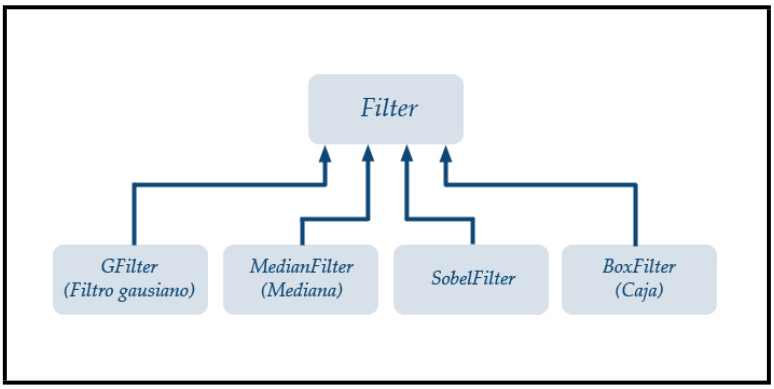


Figura 4.5: Clase *Filter* y los filtros diseñados

de las funciones creadas para lograr el encapsulamiento propio de los datos en un esquema *OO*.

El diseño se hizo utilizando una aproximación *top-down*, o de las funciones de más alto nivel a las de más bajo nivel y necesarias para la implementación de las primeras. Se decidió efectuar la división mostrada:

- **Carga de la imagen de interés.** Primero es necesario cargar la imagen a la que se desea transformar usando SIFT.
- **Convertir a escala de grises.** En caso de que la imagen sea a color, es necesario convertir la imagen a escala de grises en *buffers* de datos flotantes, pues es así como opera SIFT.
- **Reservar espacio para la representación *espacio de la escala*.** Antes de comenzar con la ejecución del algoritmo, debe reservarse la memoria para almacenar cada uno de ellos: el de la imagen y el de las *DoG*.
- **Construir filtros.** Los filtros gaussianos, requeridos para generar la representación *espacio de la escala* deben ser construidos antes de comenzar a identificar puntos candidatos.
- **Detectar candidatos.** Una vez que se cuenta con la memoria requerida para operar, se efectúa la detección de puntos candidatos.
- **Elegir puntos de interés.** Se realiza un refinamiento sobre los candidatos para así determinar cuales, entre ellos, son puntos de interés.

- **Calcular descriptor.** Para cada punto de interés se debe(n)obtener el(los) descriptor(es) del mismo.

Enseguida se explica cada uno de los módulos *superiores* y a las subrutinas usadas por ellos, pero primero se presenta una especie de resumen del algoritmo.

4.4.1. SIFT en breve

A continuación se presenta una breve serie de pasos que describen la operación del algoritmo SIFT, mediante ellos se entiende más fácilmente su operación y permite entender las secciones y apartados que se presentan en lo que resta del capítulo. Dada una imagen de entrada (ya sea adquirida en el instante o almacenada digitalmente):

1. Convertir la imagen a escala de grises usando la expresión 4.1.
2. Detectar los candidatos a puntos de interés (secciones 4.4.4 y 4.4.5).
   a) Construcción de los filtros gaussianos. Su desviación estándar está especificada por la ecuación 4.4.
   b) Convoluciones sucesivas con filtros gaussianos para construir la representación *espacio de la escala* (expr. 3.8).
   c) Calcular las diferencias de gaussianas a partir de la representación creada por medio de la expresión 3.9.
   d) Encontrar los extremos en la función DoG a partir de 2.6.
3. Definir los puntos de interés.
   a) Efectuar la interpolación del punto de interés como se establece en 3.13, 3.14 y 3.15.
   b) Filtrar las respuestas a bordes por medio de la matriz Hessiana como se especifica de la ecuación 3.17 a 3.19.
4. Calcular los descriptores para los *keypoints* encontrados.
   a) Asignar orientación como se indica en la ecuación 3.20 y la sección 3.2.3.
   b) Obtener el descriptor por medio del proceso descrito en las secciones 3.2.4, 4.4.8 y en 4.4.9.

4.4.2. Carga de la imagen de interés

Esta es la primer parte del proceso para obtener la transformación SIFT de una imagen y, utilizando el código desarrollado, puede hacerse de la siguiente manera:

```
Image* myImage = NULL;
myImage = new Image("../ImLibrary/Seattle640x480.raw", 640, 480, 3);
```

La instancia `myImage` de la clase `Image` contiene los datos, pixel a pixel de la imagen y sus atributos (ancho, alto y planos de color). Todas las funciones se basan en objetos o arreglos de la clase `Image`, por lo que su correcta creación es de particular interés.

Existen diversos métodos para cargar o crear una imagen. Puede hacerse desde un archivo, especificando sus dimensiones, pasando un *buffer* de datos como parámetros (ya sea de bytes o tipo `float`) o bien otra imagen.

4.4.3. Conversión a escala de grises

Inmediato a la carga de una imagen, esta debe ser convertida a escala de grises, pues es uno de los requerimientos del algoritmo. Para ello puede usarse el siguiente ejemplo de código:

```
//// First image loading
Image* grayImage = NULL;
grayImage = new Image(myImage->getWidth(), myImage->getHeight(), 1);
cout << "Converting to gray " << endl;
Convert2Gray(grayImage, myImage);
```

Para determinar el nivel de gris de un pixel a partir de su representación en espacio de color RGB, se utiliza la fórmula de la luminancia [MB05], expresada por la ecuación 4.1:

$$grayLevel = 0.2126R + 0.7152G + 0.0722B \tag{4.1}$$

R, G y B indican los níveles de cada uno de los planos de color para cada pixel en la imagen. Lo anterior se realiza ya que es uno de los métodos más probados, populares y que entregan buenos resultados. La imagen con la que se operará a partir

de ahora es la nueva instancia, `grayImage`, y la imagen inicial puede ser liberada o conservada para posibles operaciones futuras con ella.

4.4.4. Representación *espacio de la escala*

Una vez convertida la imagen a gris, debe ser reservado el espacio de memoria necesario para alojar las representaciones *espacio de la escala* de la imagen y sus respectivas *DoG*. La función `SIFTBuildScaleSpace` se encarga de ello y se utiliza del siguiente modo:

```
imMatrix myDoGs = NULL;
imMatrix mySS = NULL;
mySS    = SIFTBuildScaleSpace( 3, 6, grayImage->getWidth(), grayImage->getHeight(),
                               doUpSampling );
myDoGs  = SIFTBuildScaleSpace( 3, 5,grayImage->getWidth(), grayImage->getHeight(),
                               doUpSampling );
```

El tipo `imMatrix` es un apuntador a un arreglo bidimensional de `Image` (`Image***`) en donde se definen, por ejemplo, para `mySS`, un arreglo *espacio de la escala* de 3 octavas y 6 escalas. El tercer y cuarto parámetro corresponden a las dimensiones de la imagen original y el último es un booleano que indica si se debe o no redimensionar al doble la imagen de entrada.

Para redimensionar una imagen, en lugar de utilizar el método propuesto por Lowe (de generar la siguiente imagen submuestreando cada dos pixeles) se implementó uno basado en el algoritmo de trazo de líneas de Bresenham ([Dr.02]) para tener un esquema más general y dimensionar la imagen en un intervalo de 0.5 a 2, y no sólo los extremos.

4.4.5. Construcción de filtros

Antes de comenzar con la detección de candidatos a puntos de interés o *keypoints*, es necesario construir los filtros gaussianos que se emplearan en el cómputo de la representación *espacio de la escala*. Para ello se ha definido la función, que se utiliza como se muestra:

```
float** fGSigmas = NULL;
GFilter*** gFilters = NULL;
GaussianFilterAllocator(&gFilters, &fGSigmas, 3, 3, 1.6f);
```

La variable `fGSigmas` contendrá un arreglo bidimensional con las desviaciones estándar de las gaussianas a utilizar. Luego, el apuntador a arreglo bidimensional `gFilters` servirá para almacenar los filtros, creados con las gaussianas pasadas en `fGSigmas`. La función llama 5 parámetros: donde colocar los filtros, las sigmas, el número de octavas, el número de *picos* (especificados como el número de escalas deseadas menos tres, $s-3$,o las *DoG* necesarias para buscar puntos candidatos al menos una vez) y el valor de la sigma inicial, σ_i.

Basándose en la propiedad de una gaussiana donde la convolución entre dos de ellas se define por, [Now04]:

$$G(\mathbf{x},\sigma_1) * G(\mathbf{x},\sigma_2) = G\left(\mathbf{x}, \sqrt{\sigma_1^2+\sigma_2^2}\right) \tag{4.2}$$

entonces, una ecuación que iterativamente produce la σ del siguiente nivel usando la aproximación de Lowe en [Low04] es...

$$G\left(\mathbf{x},k^{p+1}\right) = G(\mathbf{x},k^p) * G(\mathbf{x},\sigma) \tag{4.3}$$

y despejando a σ sustituyendo la ecuación 4.2 en 4.3 se tiene que...

$$\begin{aligned} k^{p+1} &= \sqrt{(k^p)^2+\sigma^2} \\ \sigma^2 &= \left(k^{p+1}\right)^2 - (k^p)^2 \\ \sigma &= \sqrt{(k^{p+1})^2-(k^p)^2} = \sqrt{k^{2p+2}-k^{2p}} \\ \sigma &= \sqrt{k^{2p}(k^2-1)} = k^p\sqrt{k^2-1} \end{aligned} \tag{4.4}$$

De esta manera, usando la ecuación 4.4 y la escala actual, se determina la σ con que producir el siguiente nivel o escala en la representación *espacio de la escala* de la imagen. En el código mostrado a continuación, *w* es el k^p actual, *p* es incrementado en uno en cada iteración y *kTerm* es la constante $\sqrt{k^2-1}$ de la ecuación 4.4. Finalmente, este proceso se repite para cada octava.

```
kBase = SToK(peakLevels);
for( octave = 0; octave < octaves; octave++)
{
```

```
  w = fInitSigma;
  kTerm = sqrt(pow(kBase, 2.0) - 1.0);
  for (scale = 1 ; scale < nScales; scale++ )
  {
    fSigmas[octave][scale] = (float)(w*kTerm);
    gFilters[octave][scale] = new GFilter(fSigmas[octave][scale]);
    w *= kBase;
  }
  if( octave < octaves-1 )
  {
    fSigmas[octave+1][0] = fSigmas[octave][nScales-1];
    gFilters[octave+1][0] = new GFilter(fSigmas[octave][nScales-1]);
  }
}
```

La función `SToK` convierte el número de niveles deseados en la constante k que representará la k base para la construcción del resto de los filtros. Está basada en la técnica de [Low04] donde dice que debe calcularse por: $k = 2^{1/peaks}$ y *peaks* no es más que el número de *DoG* que se requieran y se cálcula a partir del número de escalas como $peaks = escalas - 3$.

4.4.6. Deteccción de candidatos

Este paso permitirá encontrar ubicaciones que sean candidatos a puntos de interés. Es necesario contar previamente con la memoria reservada de las representaciones *espacio de la escala* y la imagen en escala de grises. Después, basta con escribir la instrucción:

```
Descriptors theDescriptors;
SIFTDetectKeyCandidates( &theDescriptors, grayImage, mySS, myDoGs, doUpSampling,
                         3, 3, fGSigmas, gFilters);
```

Los parámetros que recibe son: una lista donde almacenar los puntos candidatos —`theDescriptors`—, la imagen a operar en escala de grises —`grayImage`—, la representación *espacio de la escala* de la imagen y de las *DoG* —`mySS` y `myDoGs`—, un booleano indicando si se duplica el tamaño de la imagen original —`doUpSampling`—, el número de octavas y de *picos* (*peaks*), las desviaciones estándar y filtros creados —`fGSigmas` y `gFilters`—.

Internamente, esta función construye la representación *espacio de la escala* de `grayImage`, tomando en consideración si debe o no ser duplicada en tamaño la imagen entrante. Lo anterior es realizado por un procedimiento llamado `Resize`, que utiliza el algoritmo descrito en [Dr.02]. Posteriormente son utilizadas dos funciones más, la primera se encarga de hacer las convoluciones con los filtros gaussianos —`SIFTBuildGaussianMaps`— y la otra de construir las *DoG* a partir de los mapas gaussianos recién generados —`SIFTBuildDoGMaps`—. La figura 4.6 ilustra el proceso.

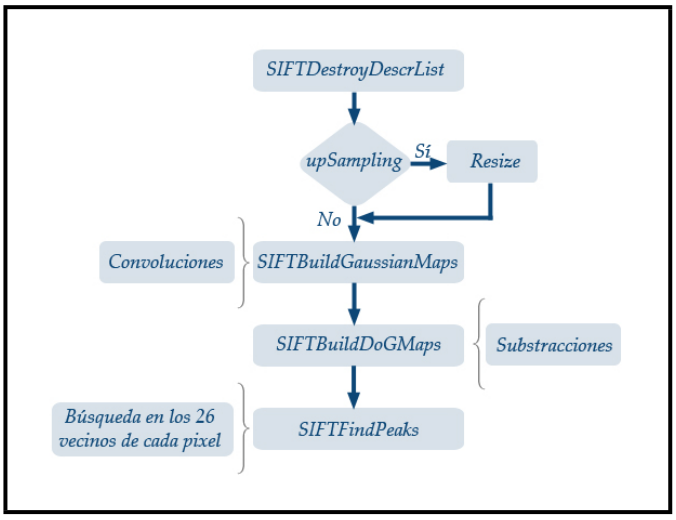


Figura 4.6: Detección de puntos candidatos

La convolución es una operación definida y ejecutada según el tipo de filtro con el que se opere y está entre las operaciones definidas sobre imágenes, figura 4.4. La operación es efectuada por el método del filtro en cuestión y la función `Convolve` únicamente se encarga de pasar los buffers de datos al método de la clase.

Finalmente, es usada `SIFTFindPeaks` que se encarga de buscar los puntos candidatos entre *DoGs* consecutivas, haciendo las comparaciones de cada pixel, con sus 26 vecinos (8 en la escala actual, 9 en la previa y 9 en la siguiente), figura 2.4.

4.4.7. Elección de puntos de interés

Tal y como se describió en el capítulo anterior, después de encontrar candidatos a puntos de interés, debe definirse cuales son aptos para considerarse *keypoints*. En la sección 3.2.2 se revisaron los criterios de elección. En la implementación desarrollada,

se han programado los dos posibles refinamientos, pudiendo el usuario elegir si quiere utilizar alguno en específico o ambos, que es lo normal. La función que realiza las operaciones (ver diagrama de bloques en la figura 4.7) de refinamiento es:

```
SIFTRefine(&theDescriptors, myDoGs, mySS, fGSigmas);
```

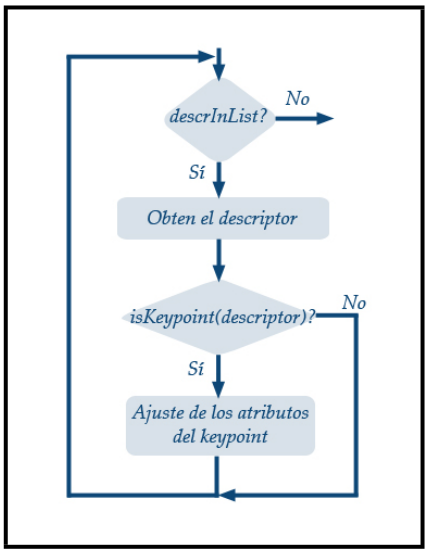


Figura 4.7: Refinamiento de los puntos candidatos

y toma como parámetros la lista de descriptores, donde se encuentran los candidatos, las *DoG* y las desviaciones estándar de los filtros creados. En la figura 4.7, la decisión `isKeyPoint(descriptor)?` es en realidad una función que toma como parámetro el descriptor actual y le aplica dos funciones más: `keyTestInterp` y `keyTestHessian`. El par de funciones mencionadas, hace las pruebas especificadas en la sección 3.2.2.

Si se requiere ejecutar sólo la interpolación o la eliminación de respuesta a bordes para los candidatos, están las funciones:

```
int SIFTExtremum( Descriptors* Extrema, imMatrix theDoGs, float** fSigmas );
int SIFTHessian( Descriptors* Extrema, imMatrix theDoGs, float r );
```

En `SIFTExtremum` se ha utilizado la propuesta de Lowe en [Low04] de obtener las derivadas de las ecuaciones 3.12 y 3.15 mediante restas entre vecinos. Para reutilizar el código de los filtros, se opera con filtros Sobel en ambas direcciones y se hizo

una extensión para derivadas con respecto de la escala. Vista la representación *espacio de la escala* como un arreglo en 3D, compuesto por x, y y σ, el filtro sobel no podría operar, por ello la extensión creada. Luego, el sistema de ecuaciones de la ecuación 3.12 es resuelto con costo mínimo; simplemente se trata de la multiplicación de una matriz por un vector columna.

Por último, la matriz hessiana, empleada para la eliminación de respuesta a bordes es obtenida directamente haciendo uso de un filtro Sobel para calcular las derivadas.

4.4.8. Cálculo del descriptor

El descriptor es obtenido usando la función abajo mostrada. Toma la lista de puntos de interés *refinada* y la representación *espacio de la escala* de la imagen de entrada. Por medio de la lista, extrae para cada *keypoint* su escala y coordenada de pixel. Luego, con la escala se busca en el arreglo de imágenes para obtener el gradiente de las vecindades que se requieren, tal y como se explicó en la sección 3.2.4.

```
SIFTComputeDescriptors(&theDescriptors, mySS);
```

Existe una función adicional con la que es posible refinar y obtener el descriptor: `SIFTRefineAndComputeDesc`. Recibe los mismo parámetros que la enunciada en la sección 4.4.7 pero en la lista de descriptores regresa los que *sobrevivieron* al proceso de refinamiento y fueron calculados. Se trata de un procedimiento bastante simple ya que, en realidad, quien está encargado del cálculo es cada elemento en la lista de descriptores. Por ello, a continuación se presenta la estructura de las clases involucradas en el cómputo aquí descrito.

4.4.9. Clases *Keypoint* y *KeyDescriptor*

El par de clases *Keypoint* y *KeyDescriptor* son las encargadas de definir los descriptores a utilizar. La primer clase funciona a manera de *contenedor*, siendo sus atributos la ubicación del punto de interés o candidato en la imagen (x, y) y la ubicación de la imagen en su representación *espacio de la escala* (octava y escala σ). Los métodos que provee son de acceso a datos, conservando así la característica de ser un contenedor.

Por su parte, *KeyDescriptor* es una extensión de *KeyPoint*. La base funge como contenedor de los datos requeridos, mientras que la hija se encarga de operar con ellos para producir la descripción del punto de interés requerido. La figura 4.8 ilustra lo anterior.

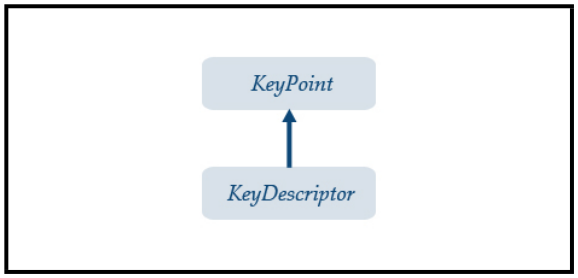


Figura 4.8: Clases *KeyDescriptor* y *KeyPoint*

Retomando la segunda clase, ella es la encargada de construir por completo al descriptor utilizando las técnicas definidas en la sección 3.2.4. Sus atributos más importantes son el histograma de direcciones de 36 posiciones, el valor del *pico* máximo y su posición en el histograma para definir la orientación dominante. Luego están las orientaciones encontradas —recuérdese que se toman todas aquellas superiores al 80 % del pico máximo— que definen el número de descriptores que habrá de construirse para este punto de interés y, obviamente, el(los) vector(es) descriptor(es) construido(s).

El cálculo del ancho de las ventanas de las gaussianas está validado por las consideraciones de [TV98] mencionadas en la sección 3.1.1. Las orientaciones alternas son ajustadas mediante una interpolación de los valores de sus picos en el histograma y sus adyacentes a una parábola, véase la ecuación 4.5.

$$\begin{bmatrix} y_{i-1} \\ y_i \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} x_{i-1}^2 & x_{i-1} & 1 \\ x_i^2 & x_i & 1 \\ x_{i+1}^2 & x_{i+1} & c \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \Rightarrow \overline{b} = \mathbf{A}\mathrm{x} \Rightarrow \overline{\mathrm{x}} = \mathbf{A}^{-1}\overline{b} \tag{4.5}$$

Ya que se trata de un histograma y está espaciado discretamente (figura 4.9), la matriz $\mathbf{A}^{-1}$ en 4.5 se especifica como:

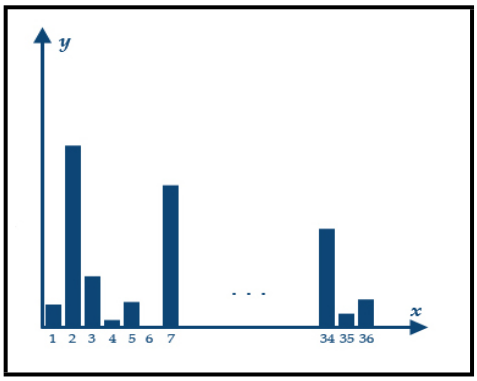


Figura 4.9: Ajuste de picos del histograma a una parábola

$$\mathbf{A}^{-1} = \begin{bmatrix} 0.5 & -1.0 & 0.5 \\ -i - 0.5 & 2i & -i + 0.5 \\ 0.5i\,(i+1) & 1 - i^2 & 0.5i\,(i+1) \end{bmatrix} \tag{4.6}$$

para cada pico con índice i en el histograma. Luego, los parámetros a, b, c de $\overline{\mathrm{x}}$ en la ecuación 4.5 se obtienen directamente, siendo:

$$\overline{\mathrm{x}} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0.5y\,(i-1) - y\,(i) + 0.5y\,(i+1) \\ (-i - 0.5)\,y\,(i-1) + 2iy\,(i) + (-i + 0.5)\,y\,(i+1) \\ 0.5i\,(i+1)\,y\,(i-1) + (1 - i^2)\,y\,(i) + 0.5i\,(i+1)\,y\,(i+1) \end{bmatrix} \tag{4.7}$$

Y finalmente, el máximo de la parábola es obtenido calculando los puntos crítocos, donde su primer derivada es igual a cero $2ax+b = 0$. Después, por medio de su segunda derivada, se verifica que se trate de un máximo por medio de la concavidad, es decir, que el signo del punto crítico en cuestión, evaluado en el punto sea menor que cero. Todo lo anterior para $y = ax^2 + bx + c$. Como se acaba de explicar, el sistema no ofrece mayor complejidad y la obtención de los parámetros de la parábola con que se ajusta son inmediatos. Mediante el máximo se define la orientación verdadera del punto de interés.

El cálculo del descriptor, se desglosa en tres fases, implementados por métodos privados de la clase *KeyDescriptor*:

1. `precomputeWindow`. Calcula los gradientes usados por las ventanas definidas en la sección 3.2.4 y coloca los datos en *buffers* temporales para las magnitudes y para las direcciones.

2. `computeHistogram`. Obtiene el histograma de 36 direcciones y también las orientaciones dominantes.

3. `computeDirs`. Para cada una de las orientaciones calculadas en la fase anterior, se calcula el descriptor definido en la sección 3.2.4. En esta rutina es donde cada pixel en la vecindad de 16 pixeles, es rotado con respecto del ángulo analizado y, posteriormente, se va llenando el vector de 128 dimensiones.

4.5. Reconocimiento de características

Los descriptores de una imagen son el primer paso para reconocer objetos o escenas en otras imágenes. Para ello, es necesario definir métodos propios para encontrar correspondencias entre colecciones de descriptores. Lowe en [Low04] propone un método simple pero efectivo fundamentado en la distancia euclidiana entre dos vectores.

Otros métodos que contribuyen al reconocimiento mediante técnicas basadas en distancia, son aquellos que utilizan árboles para acomodar los elementos de las colecciones. A continuación se expone al primero de ellos y la teoría de como funcionan una vertiente de árboles: los *Kd-Trees*.

4.5.1. Distancia euclidiana

En [Low04] se describe que la mejor correspondencia para un punto de interés, es encontrar a su *vecino más cercano* en la colección de puntos contra la que se pretende reconocer y, se define como el punto con menor distancia euclidiana.

Sin embargo, con un enfoque tan simplista, muchos puntos no tendrán pareja o serán equívocos puesto que varios provendrán del *fondo* o de regiones *recortadas* en la escena, por lo que es necesario encontrar una forma de descartarlos. Usar un umbral de distancia es una opción, sin embargo no es la mejor, ya que algunos descriptores serán mas discriminantes que otros.

La alternativa de comparar la distancia del segundo vecino más cercano con la del primero, se comporta bien, ya que (para tener un *buen* reconocimiento) la

cercanía del vecino correcto más próximo debe ser significativamente menor que la del incorrecto más cercano. En el caso de las correspondencias incorrectas, habrá más de ellas con distancias similares debido a la alta dimensionalidad de la colección y espacio de características.

La figura 4.10, tomada de [Low04], muestra los valores de la medida recién descrita para imágenes con datos reales. La función de densidad de probabilidad para correspondencias correctas e incorrectas se ilustra en términos de la relación de los vecinos más cercano y el segundo más próximo para cada punto de interés. Aquellos *empates* donde el vecino más cercano fue correcto, tienen una PDF que se centra en una proporción menor a la de los incorrectos. Se observa que cuando se utiliza una proporción mayor a 0.8, se elimina el 90 % de falsas correspondencias descartando menos del 5 % de las correctas.

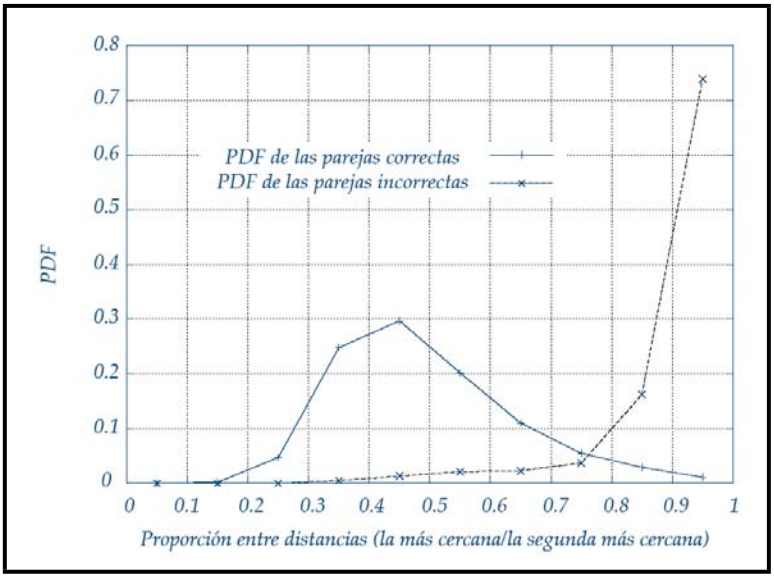


Figura 4.10: Probabilidad de encontrar correspondencias mediante distancia euclidiana reportada por Lowe en [Low04]

En el software desarrollado, se diseñó una función para efectuar esta operación y está definida por:

```
int matches = SIFTCompareDescriptors(theDescriptors, simpleDescrs);
```

que recibe como parámetros dos listas de descriptores. Otra función de utilidad es

`SIFTMatchAndDraw` que toma como atributos a las instancias de un par de imágenes previamente cargadas, y el par de listas que contienen los puntos de interés de cada imagen. Luego, la función genera una nueva imagen, combinación de ambas, donde dibuja líneas entre los pares de correspondencias encontrados. La figura 4.11 es un ejemplo del resultado usando el desarrollo de este trabajo.

Para la figura 4.11 se detectaron 1199 puntos de interés y 1593 descriptores para la imagen sin rotar y 888 puntos con 1131 descriptores para la imagen rotada. La rotación de la imagen original fue de 30 grados en el plano, recortando la imagen producida a la resolución inicial de 640x480 pixeles. La prueba de reconocimiento arrojó 296 correspondencias, siendo 4 de ellas erróneas. Lo anterior conduce a una taza de reconocimiento del 26.17 % de puntos de interés y al 98.65 % de pares correctos.

4.5.2. *Kd-Trees*

Los datos espaciales consisten de puntos, líneas, regiones, superficies y volúmenes. La mayoría de las estructuras de datos que se usan para representar datos espaciales son jerárquicas y basadas en el principio de descomposición recursiva; similar a una estrategia *divide y vencerás*.

Las estructuras jerárquicas son de utilidad por su capacidad de centrarse en los subconjuntos de interés de los datos representados. Tal capacidad ofrece una representación eficiente y mejora los tiempos de ejecución para inserción y búsqueda. A pesar de que los datos espaciales pueden ser contenidos en otras estructuras de datos, las jerárquicas son atractivas por su claridad conceptual y facilidad de implementación. Además, algunas de ellas definen un índice de gran utilidad en aplicaciones que involucran bases de datos espaciales. Ejemplos de este tipo de estructuras son los *quadtrees* (en 2D) y *octrees* (en 3D).

Los árboles *K–dimensionales* o llanamente, por su nombre en inglés, *Kd–Trees* son una estructura de datos utilizada para almacenar un conjunto finito de puntos pertenecientes a un espacio de K dimensiones [Moo91]. Esta se obtiene descomponiendo jerárquicamente el espacio en un número relativamente pequeño de celdas, de manera que ninguna de ellas contenga muchos elementos. Con ellos se consigue un rápido acceso a cualquier objeto a partir de su posición; sólo se necesita recorrer la jerarquía hasta llegar a la celda que contiene el elemento buscado y, finalmente, se revisa en los objetos de la celda para identificar al deseado [Ski98].

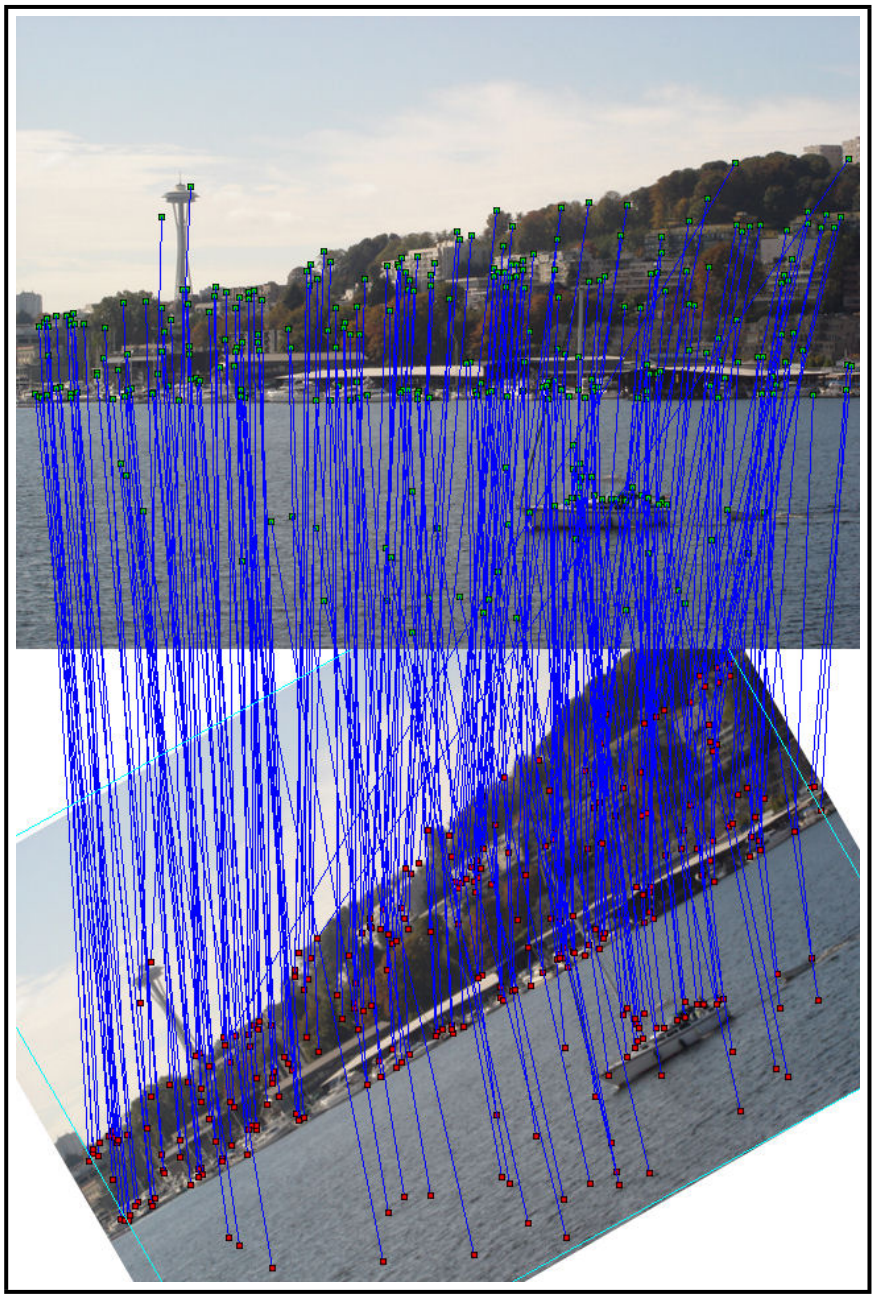


Figura 4.11: Ejemplo de salida para el reconocimiento entre un par de imágenes usando las técnicas desarrolladas en el presente trabajo

En [Low04], se mencionan a los *Kd-trees* como opción para las tareas de reconocimiento, sin embargo se establece que no proveen de su aceleración en búsquedas, ya mencionada, cuando se tienen más de 10 dimensiones. Entonces se menciona otra técnica también basada en árboles multidimensionales llamada *Best-Bin-First* o simplemente BBF. En realidad usa una búsqueda modificada con respecto de la de un *Kd-tree* de modo que los espacios en el espacio de características sean buscados en el orden de su distancia más cercana a partir de la localidad de inicio.

4.6. Transformación entre un par de imágenes

Existen diversas técnicas para determinar la relación geométrica entre parejas de puntos de interés pertenecientes a los conjuntos extraídos de dos o más imágenes.

En el apéndice C se exponen las formalidades de las transformaciones afines. Por ahora basta saber que una transformación afín puede descomponerse en dos: una transformación lineal y una traslación. Por ello, para un punto contenido en I_1 y definido por sus coordenadas de imagen (x, y), su transformación o proyección en I_2 dada por (u, v) se define como:

$$\begin{gathered}[x, y] \xrightarrow{T} [u, v] \\ \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}\end{gathered} \tag{4.8}$$

O bien, en su forma matricial y usando coordenadas homogéneas. . .

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} m_1 & m_2 & tx \\ m_3 & m_4 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4.9}$$

La ecuación 4.9 define 6 incógnitas y 2 ecuaciones, con lo que no se puede encontrar una solución única para la pareja de puntos mapeados. Sin embargo, si se cuenta con 3 pares de puntos en I_1 que mapeen a I_2, entonces se tendrán 6 ecuaciones con 6 incógnitas y es posible encontrar una solución única. Entonces, el sistema a resolver, en forma matricial, está definido por:

$$\begin{bmatrix} x_1 & y_1 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_1 & y_1 & 0 & 1 \\ x_2 & y_2 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_2 & y_2 & 0 & 1 \\ x_3 & y_3 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_3 & y_3 & 0 & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_x \\ t_y \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ v_1 \\ v_2 \\ u_3 \\ v_3 \end{bmatrix} \tag{4.10}$$

Si se denota al sistema por $A\overline{x} = b$, la incógnita $\overline{x}$ se determina resolviendo la ecuación matricial de forma que $\overline{x} = A^{-1}b$. La ecuación 4.11 es la solución al sistema planteado.

$$\overline{x} = \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_x \\ t_y \end{bmatrix} = \frac{1}{r} \begin{bmatrix} y_1(u_3 - u_2) + y_2(u_1 - u_3) + y_3(u_2 - u_1) \\ x_1(u_2 - u_3) + x_2(u_3 - u_1) + x_3(u_1 - u_2) \\ y_1(v_3 - v_2) + y_2(v_1 - v_3) + y_3(v_2 - v_1) \\ x_1(v_2 - v_3) + x_2(v_3 - v_1) + x_3(v_1 - v_2) \\ u_1(x_2y_3 - x_3y_2) + u_2(y_1x_3 - x_1y_3) + u_3(x_1y_2 - y_1x_2) \\ v_1(x_2y_3 - x_3y_2) + v_2(y_1x_3 - x_1y_3) + v_3(x_1y_2 - y_1x_2) \end{bmatrix} \tag{4.11}$$

$$r = (x_2y_3 - x_3y_2) + (y_1x_3 - x_1y_3) + (x_1y_2 - y_1x_2)$$

Una vez calculada la homografía a partir del conjunto de 6 puntos, es necesario evaluar su calidad, pues los puntos pueden tener falsas correspondencias y transformar erróneamente los puntos de la imagen. Para probar la calidad de la homografía se usa como métrica el error de transferencia simétrico, definido en [ZH03] y empleado en [OLL04] para definir la mejor homografía a partir de una colección de ellas y así encontrar *outliers*.

Si $\mathbf{x} \leftrightarrow \mathbf{x}'$ es la correspondencia entre un punto y H la homografía, tal que $\mathbf{x}' = H\mathbf{x}$, entonces, el error de transferencia simétrico se define por:

$$d^2_{transferencia} = d\left(\mathbf{x}, H^{-1}\mathbf{x}'\right)^2 + d\left(\mathbf{x}', H\mathbf{x}\right)^2 \tag{4.12}$$

en donde $d\left(\mathbf{x}, H^{-1}\mathbf{x}'\right)$ representa la distancia entre $\mathbf{x}$ y $H^{-1}\mathbf{x}'$. La media del error para una homografía puede encontrarse como:

$$\mu = \frac{1}{N}\sum_{i=1}^{N}\left\{d^2_{transferencia}\right\}_i \tag{4.13}$$

Luego, de 4.9 se sabe que la homografía, H, está definida por la matriz:

$$H = \begin{bmatrix} m_1 & m_2 & tx \\ m_3 & m_4 & ty \\ 0 & 0 & 1 \end{bmatrix} \tag{4.14}$$

y es necesario invertirla para encontrar la transformación inversa. La siguiente expresión refleja el cálculo de la matriz inversa H^{-1}.

$$H^{-1} = \begin{bmatrix} m_4/\left(m_1m_4 - m_3m_2\right) & -m_2/\left(m_1m_4 - m_3m_2\right) & \left(m_2t_y - t_xm_4\right)/\left(m_1m_4 - m_3m_2\right) \\ -m_3/\left(m_1m_4 - m_3m_2\right) & m_1/\left(m_1m_4 - m_3m_2\right) & -\left(m_1t_y - t_xm_3\right)/\left(m_1m_4 - m_3m_2\right) \\ 0 & 0 & 1 \end{bmatrix} \tag{4.15}$$

Es importante mencionar que en el presente trabajo no se están eliminando los *outliers* de las listas de descriptores, solamente se encuentra la mejor homografía para definir la transformación existente entre la imagen referencia y la que se analiza.

En el código desarrollado, esta parte corre a cargo de un par de funciones: `SIFTComputeTransform` y otra subrutina `AffineHomography2D`. Este par recibe por parámetros las dos listas de descriptores, previamente calculadas y donde están incluidas los descriptores que tuvieron correspondencia para el par de imágenes en cuestión. Luego, toma una terna de números aleatorios para referenciar a los índices de los descriptores en cada una de sus listas y con ellos resuelve la ecuación 4.11.

Enseguida, para la homografía procesada, se utiliza la ecuación 4.13 para calcular el *error de transferencia simétrico* (además de la ecuación 4.15 para obtener la homografía inversa) para cada uno de los descriptores en las listas. Este proceso es repetido un cierto número de veces, que por ahora se ha definido en 25. Finalmente, se elige aquella homografía con menor error para todos los puntos en las listas de descriptores.

4.7. SIFT orientado a objetos

En la sección B se muestra el código necesario para ejecutar SIFT sobre dos imágenes y empatarlas. Se trata de una secuencia muy larga de instrucciones que, si bien funcional, no resulta ser práctca. Es por ello que se creo la versión orientada a objetos, en para evitar que el programador/usuario tenga que llevar tanto control sobre los tipos de datos y variables creadas.

La concepción orientada a objetos de SIFT, utiliza un par de clases base: *SIFT_Properties* y *ScaleSpaceProperties*, que fungen a manera de contenedores de las propiedades de ejecución del algoritmo SIFT. En sus atributos lleva instancias de las clases: *SIFTScaleSpace* y *Matching*.

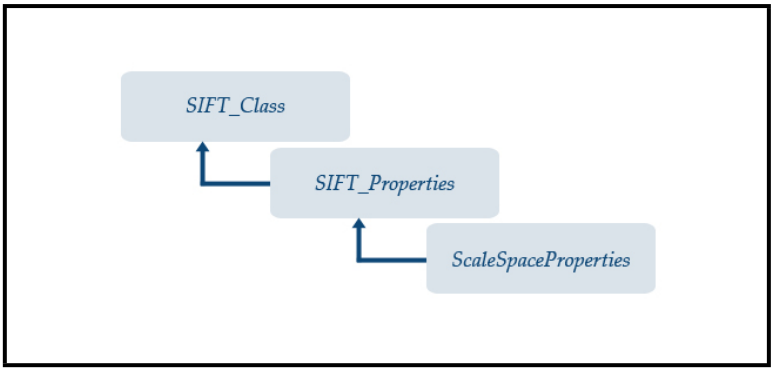


Figura 4.12: Clase *SIFT* que implementa al algoritmo SIFT

La figura 4.12 muestra el esquema de herencia utilizado. Se decidió agrupar de tal modo a las propiedades, ya que los atributos son específicos a ciertas fases de cómputo, como lo es la representación *espacio de la escala* y lo que está tras de ella, así como propiedades particulares de la ejecución del algoritmo.

Los atributos de *ScaleSpaceProperties* corresponden al número de octavas, escalas, el valor de la *escala* inicial, las dimensiones de la imagen y un booleano que determina si es necesario duplicar el tamaño de la imagen original o no. Todos los parámetros anteriores están involucrados en la configuración y creación de la representación *espacio de la escala* de la imagen de entrada.

Por su parte, *SIFT_Properties* contiene la descripción de los atributos de *ScaleSpaceProperties* (a causa de la herencia),y los de ejecución de SIFT. En el caso de este trabajo, se han definido como parámetros de ejecución a las diferentes vertientes

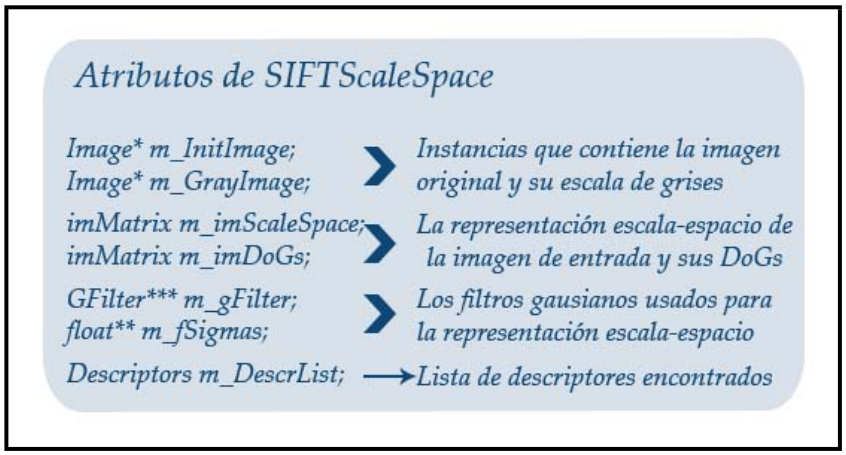


Figura 4.13: Clase *SIFTScaleSpace*

que existen para utilizar SIFT. A pesar de que el algoritmo detecta, refina y produce descriptores de puntos de interés, el usuario podría sólo requerir la detección sin filtrado, detección y filtrado, o tal vez solamente el cálculo de descriptores a partir de una colección de datos suministrada, etc. Por ello, la característica principal de los atributos de la clase en discusión es de definir las etapas por las que pasará una imagen de entrada.

Luego, la clase *SIFTScaleSpace*, figura 4.13, tiene como objetivo crear y contener a la representación *espacio de la escala* de una imagen. Se construyó de tal modo que se encapsularan los datos principales —imágenes convolucionadas y *DoGs*—, permitiendo acceder fácil y libremente a sus datos individuales (en el caso de las listas, a las imágenes que almacenan y en el caso de las últimas, a sus pixeles) e incluso a los atributos de configuración. En la figura se observan los atributos principales de la clase, que almacenan los datos que fundamentan la ejecución del algoritmo.

Finalmente, la clase *SIFT* permite que el usuario defina instancias especificando los atributos previamente mencionados: generación de *espacio de la escala*, etapas de ejecución e imagen a transformar. El método principal de la misma es `Run()` y ejecuta la operación SIFT acorde a los parámetros internos. Una vez finalizado el cómputo, se puede acceder a los datos calculados: imágenes convolucionadas, *DoGs*, descriptores encontrados y calculados, etc. La figura 4.14 ilustra los procesos llamados cuando se invoca el método `Run` y como se ven afectadas las instancias usadas de `SIFTScaleSpace` y `Matching`.

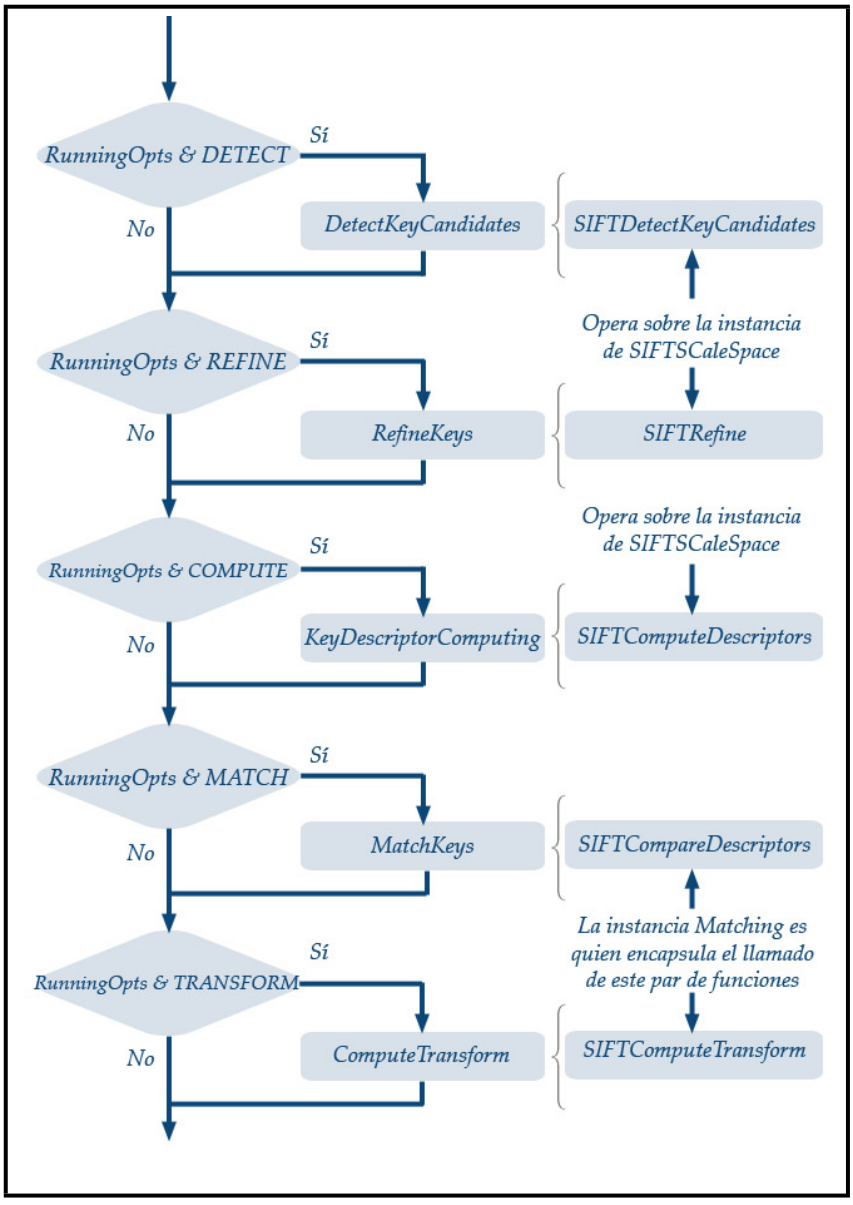


Figura 4.14: Diagrama de ejecución usando el método *Run* de la clase *SIFT*

4.8. Implementación multihilos de SIFT

Como se ha expuesto en el capítulo anterior y en el presente, algunos de los cálculos que debe realizar *SIFT* son costosos. Si bien, el hecho de redimensionar las imágenes puede hacerse mediante algoritmos rápidos y que la convolución de una imagen con gaussianas es de las operaciones que llegan a ser más eficaces, el tamaño de la imagen en sí es la principal limitante de rendimiento.

Por muy rápidos que sean los algoritmos empleados, la cota de rendimiento/velocidad estará determinada por el tamaño de las imágenes con que se opere, ya que en todo caso debe *barrerse* por completo la información contenida en la imagen, es decir se debe hacer un barrido pixel a pixel.

Por tal motivo se decidió elaborar una versión multihilos del algoritmo, de modo que se reduzca el tiempo de cómputo y por lo tanto, se incremente el rendimiento de las aplicaciones donde se use el algoritmo. En la sección 3.2 se desglosan las tareas involucradas en el cómputo de la transformación *SIFT*. Si se estudian a detalle, se llegará a la conclusión de que pueden separarse en fases bien definidas y que son mutuamente excluyentes, y por lo tanto, pueden paralelizarse.

Dentro de los posibles esquemas de paralelismo entre procesos, se ha decidido implementar una vertiente de *pipelining* o *línea de ensamblado*. El término fue tomado de su simil con las líneas de ensamblado de las fábricas, donde cada obrero hace una tarea determinada en cierto tiempo (qué es el del proceso que consume mayor tiempo en completarse) y el producto que toma es la salida de una etapa anterior efectuada por otro obrero. El esquema de desarrollo para el algoritmo utiliza 5 etapas, como se comentó al inicio, tres propias y las dos últimas para el reconocimiento y transformación del objeto:

1. **Detección de puntos candidatos**. Se trata de la primer etapa y es donde se construye la representación *espacio de la escala* para la imagen actual, la pila de *DoG* y se encuentran las localidades candidatas para puntos de interés.

2. **Refinamiento de puntos candidatos**. Se toman los puntos encontrados en la etapa anterior y, valiéndonse también de las *DoG* y la representación *espacio de la escala*, se refinan. De esta forma se producen los *keypoints* o puntos de interés.

3. **Cálculo del descriptor**. Para cada punto de interés determinado en la fase

número 2, se encuentra el descriptor. Nuevamente se hace uso de la representación *espacio de la escala* para producir el descriptor del punto.

4. ***Matching* o empate contra otra imagen**. Si bien no es parte del algoritmo, para los fines del trabajo es apropiado incluirlo pues se pretende hacer reconocimiento de objetos y un posterior rastreo del mismo.

5. **Cálculo de la transformación entre el par de imágenes**. Tampoco forma parte de la transformada SIFT, sin embargo se decidió incluirlo para así satisfacer el objetivo de hacer rastreo de un objeto o área en una imagen.

La figura 4.15 ilustra el proceso. A medida que transcurre el tiempo t, se van llenando las etapas del *pipeline*. En un inicio, t_0, todas las etapas están vacías y el proceso comienza en t_1, con la ejecución de la etapa 1 para la primer imagen asignada como parámetro I_1. En el tiempo que le tome a la etapa 1, t_1 hacer los cálculos correspondientes, las demás etapas no tendran datos con que procesar y solamente equivalen a tiempo muerto.

Una vez terminado el proceso de la etapa 1, entonces se prosigue a ejecutar el siguiente nivel en la línea de ensamblado. Para ello, en t_2 se comienza nuevamente todo el *ensamblado*, la etapa 1 toma como parámetro la siguiente imagen asignada, I_2, y la etapa 2 al producto del cómputo realizado en la etapa 1 de t_1. En la figura 4.15, para t_5 se ha llenado la línea y por lo tanto, los tiempos muertos ya no existen, teniendo máxima eficiencia en cuanto a *producción* se refiere.

Si se sigue el diagrama mostrado en la figura 4.15, se puede observar que, en el mejor de los casos, se puede incrementar el rendimiento del proceso en un $(n-1)/n\,\%$, cuando todas las etapas tienen el mismo tiempo de ejecución individual, y n es el número de etapas. Sin embargo, cuando las etapas difieren en los tiempos de ejecución, la reducción real dependerá de la etapa más tardada.

Supóngase un proceso donde el tiempo de ejecución sin paralelismo es de 1.30min, y que contiene 5 fases bien definidas que son mutuamente excluyentes entre ellas. La etapa 3 tarda 10s en ejecutarse, la 4 un total de 30s, y el resto 12.5s. Entonces, cuando se implemente el *pipeline* se tendrá una mejora de 60s o del 66.66 % sobre el tiempo anterior de 1.30min, ya que la etapa que consume mayor tiempo es la 4 con 30s, que representa 1/3 parte del total. Sin embargo, la etapa 3 tarda sólo 10s, por lo que estará *muerta* 20s *esperando* que termine la etapa 4. Lo mismo sucederá con el resto de

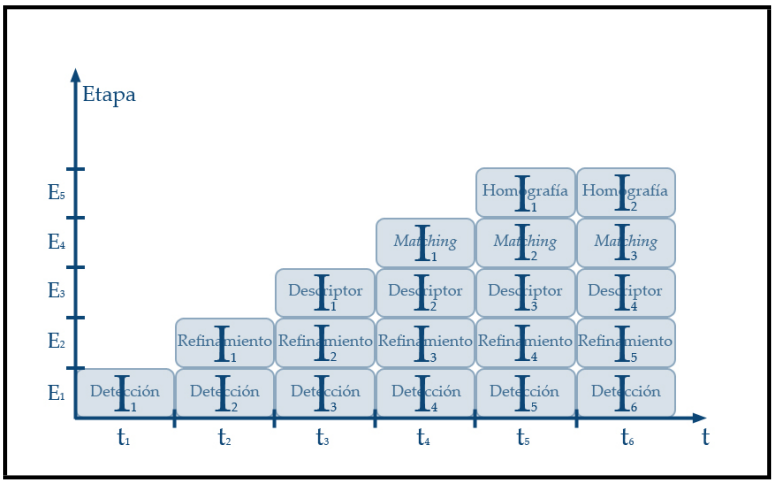


Figura 4.15: Esquema de *pipeline* utilizado

las etapas. Por lo tanto, existen tiempos perdidos de procesamiento, que no importan mucho cuando hay una ganancia razonable en el tiempo total de procesamiento.

Para la implementación desarrollada, se utilizaron las bibliotecas de programación *pthreads*. Se trata de una versión POSIX, multiplataforma para programación con múltiples procesos. Esto se hizo para que, de requerirse, los problemas de portabilidad entre sistemas *Windows* y *Unix* sean reducidos al máximo. El funcionamiento se basa en tres clases principales.

La primera de ellas es *GenThread*, que inicia un hilo de proceso de acuerdo con la sobrecarga de un método de la clase, *Run()*. Con ella se pueden crear nuevos procesos que, cuando se invocan, son ejecutados en su propio proceso y finalizan normalmente. Para el caso de la implementación *pipeline*, una segunda clase —*GenStage*— hereda de *GenThread*, añadiendo algunos atributos propios del control (como señalizaciones de paro y actividad) además de proveer un par de métodos para la asignación del método/función a ejecutar y los parámetros que recibe.

Luego, está la clase superior y que define un proceso en *pipeline*: *Pipelined-Proc*, véase la figura 4.16. Esta clase tiene los atributos necesarios para definir el número de etapas que contendrá, las instancias de cada una de ellas, los medios necesarios para accederles y, fundamentalmente, un proceso adicional a las n etapas: el proceso capataz, por definirlo de algún modo. Este proceso es el encargado de dictaminar el orden de ejecución de las etapas y los argumentos con que funcionarán. Además, es

el encargado de revisar el estado de cada una de las etapas, determinando así cuando han finalizado todas su ejecución y cuando se pueden asignar los nuevos argumentos para cada una de ellas. Este administrador es responsabilidad del programador, pues de él dependerá el correcto funcionamiento del proceso.

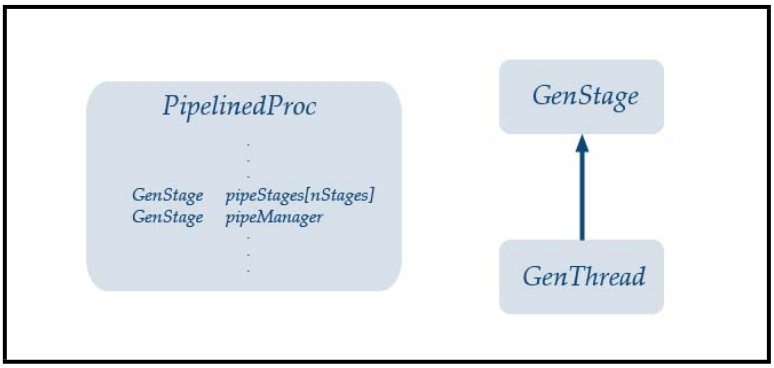


Figura 4.16: Clases multihilos

Como ya se mencionó, para llevar SIFT a un esquema de multiprocesos se dividió en 5 etapas. La clase *SIFT_Mtd* es la encargada de tal implementación. La concepción de la clase proviene de tener diferentes representaciones *espacio de la escala*, en realidad, una por cada etapa. De la enumeración de etapas, presentada al inicio de la presente sección, se observa que, para la operación de cada etapa, esta requiere de la representación *espacio de la escala*, de la pila de *DoG* o bien, de ambas. Por lo tanto, tener una representación con su pila de *DoG* por etapa es la mejor solución, es costoso en términos de memoria, pero para rendimiento es la mejor alternativa.

Para ejemplificar, considérese que se desea obtener la transformada SIFT para una imagen con resolución de 640x480 pixeles con 3 planos de color RGB. Supóngase también que deben hacerse 6 escalas y 3 octavas haciendo *upsampling* para la primera. Entonces, eso conduce a un total de 6 imágenes en cada nivel de la pirámide de la representación *espacio de la escala* y a que la pila de *DoG* sea de 5 imágenes. Finalmente, por cada nivel en la pirámide se tienen 11 imágenes y ya que es de 3 niveles, entonces se almacenan un total de 33 imágenes. En el primer nivel, debido al *upsampling* las imágenes tienen el doble de resolución (1280x960 pixeles), en el segundo es la resolución original de 640x480 pixeles y el tercero la mitad de la resolución original, 320x240 pixeles.

La transformada SIFT indica que deben usarse números reales para la repre-

sentación de las imágenes. Por lo tanto un pixel se almacena en un tipo `float`, en el caso de C/C++ y es de 4 bytes. Volviendo al ejemplo, el primer nivel de la pirámide tiene 11 imágenes de 1280x960 pixeles, en términos de memoria equivalen a 54.067MB. El segundo nivel utilizará 13.516MB y el último 3.379MB. El total de memoria requerida para la representación *espacio de la escala* de una imagen de 640x480 es de 70.963MB.

De forma general, la ecuación 4.16 expresa la cantidad de memoria empleada, M, por la representación *espacio de la escala* para una imagen de wxh pixeles, s escalas y o octavas.

$$M = wh\,(2s-1)\,(\texttt{sizeof(float)})\sum_{i=k}^{o-1+k} 2^{-2i} \quad \text{con} \quad k = \begin{cases} -1 & \text{si hay } \textit{upsampling} \\ 0 & \text{en otro caso} \end{cases} \tag{4.16}$$

En el caso del ejemplo dado, tal vez no se trate de una cantidad de memoria enorme, pero, si se ha mencionado que lo más conveniente es crear una representación *espacio de la escala* como dato para cada etapa, entonces el gasto de memoria ya no está sólo en función de los parámetros mencionados, sino también del número de etapas. Entonces, para 5 fases en un proceso, el gasto, sólo del espacio necesario para almacenar los datos aquí mencionados, sería de 354.816MB, que es bastante RAM. Sin embargo, para fines de experimentación, la velocidad que se pretende que tengan las aplicaciones de visión por computadora y la capacidad en memoria RAM de los equipos actuales, vale la pena experimentar y analizar los resultados.

La figura 4.17 es una variante de la 4.15. En esta nueva versión se han incluído las n instancias de representación *espacio de la escala* para las n etapas y n imágenes en el *pipeline*. Como se observa, cuando se analiza la primer imagen, ésta es operada y su representación colocada en SS_1. A medida que se avanza en el cálculo de la transformada SIFT, las diversas etapas van tomando el producto de sus etapas inmediatas anteriores. El hecho de utilizar n representaciones *espacio de la escala* permite que los datos en operación sean mutuamente excluyentes y estén apropiadamente encapsulados, de forma que se acceda fácilmente a las propiedades necesarias.

Si bien, cada representación *espacio de la escala* es para una imagen, ¿cómo pueden asignarse nuevas imágenes para su procesamiento? La clase *SIFT_Mtd* provee de un método para colocar un apuntador a una localidad de memoria donde se estará actualizando la imagen. De esta forma, si se requiere analizar las imágenes

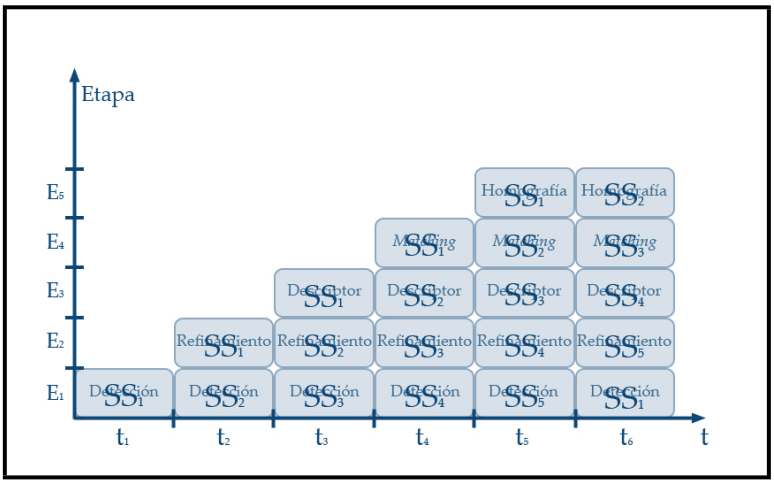


Figura 4.17: Etapas de *pipeline* con su respectiva representación *espacio de la escala*

provenientes de un flujo de video, entonces basta con asignar a *SIFT_Mtd* la localidad de memoria del *buffer* donde se está adquiriendo y actualizando la imagen.

En la figura 4.18 se muestran los principales atributos de la clase. Como ya se mencionó, se necesitan de $n+1$ instancias de *SIFTScaleSpace* así como de *Matching*. Auxiliarmente, para cuando se dibujan los descriptores o las correspondencias entre un par de imágenes al ejecutar en múltiples procesos, se lleva también un arreglo de $n+1$ *buffers* para contener las imágenes procesadas.

Figura 4.18: Clase *SIFT_Mtd*

Finalmente, ya que se trata de un proceso de 5 etapas, el producto final se

obtiene en la etapa 5, una vez que se obtuvo la transformada SIFT de la imagen y (en caso requerido) se efectuó el *matching* contra otra imagen. Entonces, el producto de la etapa 5 es almacenado hasta la sexta ejecución en el proceso, pues de lo contrario los resultados pueden perderse, no se sabría a quien pertenecen o se trataría de acceder a datos que aun no han sido calculados o están incompletos.

4.9. *Kit* de pruebas para tiempo real

El *software* desarrollado hasta este punto permite hacer pruebas con imágenes fijas, pero, ¿qué hay de las secuencias adquiridas por dispositivos de video? Para ello se ha desarrollado un *kit* usando C# y *wrappers* para las clases SIFT y SIFT_Mtd para .NET por medio de la especificación C++/CLI, que permite diseñar clases que cumplen con el paradigma de .NET y código administrado. El código .NET creado tiene tres módulos principales, que son desglosados en las siguientes secciones:

1. **Adquisición de video.**
2. **Manejador de SIFT.**
3. **Interfaz de pruebas.**

4.9.1. Adquisición de video para .NET

Las clases definidas en la sección 4.2 fueron incorporadas dentro de un paquete de C++/CLI en donde se definió una nueva clase: `DShowMan`, que usa directamente tipos de datos administrados y código *unsafe* o *no seguro*, como han sido clasificados todo tipo de apuntadores a memoria.

Entonces, la clase provee de los métodos necesarios para acceder a las propiedades involucradas en la adquisición de video. El más importante de ellos es la instancia de la clase *no administrada* `CCustomPresentation` que se encarga de iniciar el entorno de *DirectShow* y un hilo donde adquiere continuamente las imágenes provenientes del video.

Por otro lado, `CCustomPresentation` permite acceder al *buffer* en memoria con la imagen adquirida y hacer su mapeo como textura de *Direct3D*. El dispositivo de *render* es definido por el nivel más alto, es decir la interfaz de usuario, *GUI*, definida

en la sección 4.9.3. Se trata de un objeto administrado y no se tiene referencia al apuntador, que es necesario para desarrollar con código no administrado, como es el caso de C++. Por lo tanto, revisando la documentación, se encontró que hay una manera de acceder al apuntador y es mediante un *hack* distribuido p or la misma gente de Microsoft:

```
/// m_d3dDevice is a C++ Direct3D object and theDev is the managed resource
IntPtr p = theDev->GetObjectByValue(0xd2b543af);  /// Microsoft's hack
if( p.ToPointer() != NULL )
{
  m_d3dDevice = (LPDIRECT3DDEVICE9)p.ToPointer(); /// Get the managed resource
  m_d3dDevice->AddRef();
}
```

Ya con el recurso de *Direct3D*, es posible hacer el mapeo de la textura desde la instancia usada de `CCustomPresentation` y no entrar en conflictos con la lectura del *buffer* desde aplicaciones administradas o de C#. Por ello, la clase `DShowMan` ofrece un método para mapear la textura cada que el render principal lo solicite: `MapVidTex` es la encargada de la tarea.

La figura 4.19 ilustra los atributos de la clase en cuestión. Haciendo una revisión de los métodos de la misma, se tiene que ofrecen el control necesario para adquirir video, sin embargo, el uso adecuado es tarea del programador. Es por ello que, para el caso de C#, se ha provisto de otra clase especial: `DShowManager` que se encarga de la correcta iniciación y uso de `DShowMan`.

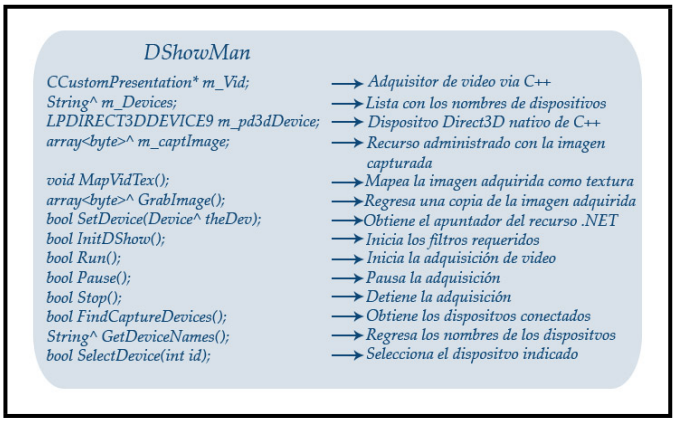


Figura 4.19: Clase *DShowMan*

Para tener una mejor visión de esta última clase, en la figura 4.20 se presenta un diagrama para iniciar a `DShowMan`. Como se puede observar en ella, lo primero es crear la instancia de `DShowManager` pasando como argumento el dispositivo de despliegue. Luego, se solicita la lista de dispositivos conectados al sistema y se elige alguno de los presentes. El identificador es pasado al método `DShowMan.SelectDevice` y, si lo pudo hallar, entonces se invoca al método iniciador de `DShowMan`, `InitDShow`. Por último, se comienza a adquirir el video de la fuente seleccionada usando el método `DShowMan.Run`.

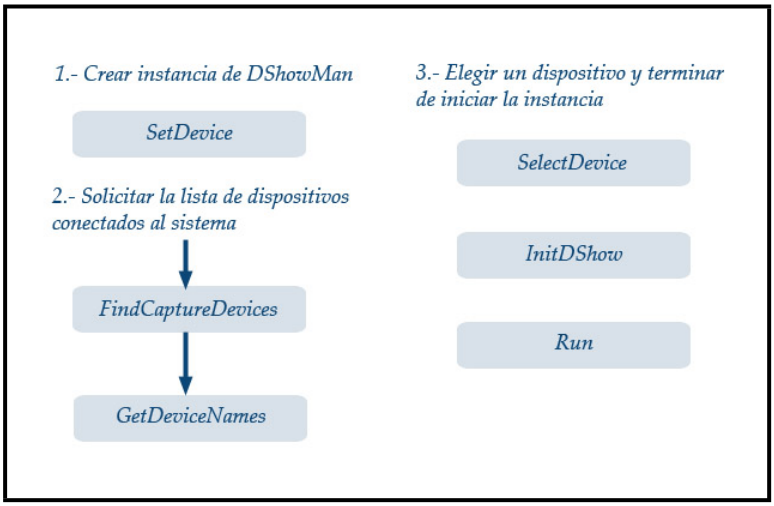


Figura 4.20: Iniciación de *DShowMan*

Con tal procedimiento, `DShowMan` queda configurado y listo para ser utilizado. Mediante los métodos `DShowMan.Pause` y `DShowMan.Stop` puede pausarse y detenerse el video a voluntad. En la sección 4.9.3, se ahondará en la utilidad de la clase `DShowManager`, de C#, pues tiene una relación íntima con la interfaz desarrollada.

4.9.2. Manejador de SIFT para .NET

El *wrapper* para .NET de las clases `SIFT` y `SIFT_Mtd` está definido por la clase administrada `SIFTNet`. Es una *envoltura* simple de las anteriores, pues generaliza el uso de ambas, teniendo una instancia de ambas. De esta manera, sólo se configura si se desea que la ejecución sea multihilos o no y pasar el resto de los parámetros, que son los mismos que para las clases individuales.

Es así que todo queda encapsulado de modo que la ejecución depende exclusivamente de la clase que haya sido iniciada. En el caso de que se haya especificado que se requiere de procesamiento en múltiples hilos, entonces habrá que pasar los parámetros necesarios de ejecución. El más importante de ellos es el apuntador al *buffer* que contiene la imagen a transformar con SIFT. Sin embargo, tratándose de ambientes administrados por .NET, no es posible usar apuntadores a menos que se especifique un bloque *unsafe*. A pesar de la flexibilidad que se tiene al poder usar tales bloques, rompe con la filosofía del código administrado.

Si se adquieren imágenes de una cámara, esto ser hará mediante la clase `DShowMan`, y el *buffer* de datos que utiliza sí está referenciado mediante un apuntador. Entonces es por ello que esta clase, `SIFTNet`, utiliza el método `SetImagePointer` que recibe por parámetro una instancia de `DShowMan`. Y, finalmente, puede tomar la referencia de la imagen que se adquiere.

Cuando se requiere ejecución en un sólo hilo, no se tiene ningún problema y solo hay que configurar la clase con los parámetros adecuados.

4.9.3. Interfaz de pruebas

La interfaz para realizar pruebas se construyó usando la plataforma .NET y C#. Cuatro clases fueron construidas para darle soporte a la *GUI*, dos de ellas ya fueron comentadas y corresponden a la versión C# de la implementación de SIFT —*SIFT_Sharp* que utiliza el *wrapper* de C++ para .NET, *SIFTNet*— y a la captura de video —*DShowManager* que usa el *wrapper DShowMan*—. Las dos últimas se encargan de las tareas relacionadas con el *set up* de las ventanas y los dispositivos de render mediante *Direct3D*: *D3DManager* y *D3DVidPlane*. En la figura 4.22 se muestran las clases utilizadas en la *GUI*, llamada *mkSIFT*.

D3DManager es un objeto global que inicia el objeto principal de *Direct3D* y posteriormente administra los dispositivos generados en su contexto. Luego, *D3DVidPlane* es una clase que envuelve el manejo y administración de *ventanas* (cualquier control de *Windows*, para fines prácticos, es una ventana) para usarlas como dispositivo de *render*; en el caso de la interfaz de pruebas, para mostrar el video proveniente del *stream* de video, la imagen adquirida y procesada como referencia, y los *frames* procesados. Véase la figura 4.21.

Finalmente, la *GUI* está implementada por la clase *mkSIFT*. Esta clase es la

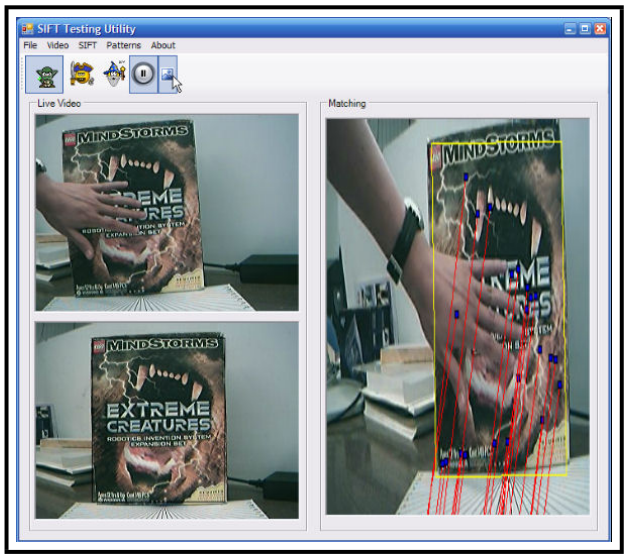


Figura 4.21: Muestra de la interfaz desarrollada

administradora de todos los eventos que ocurren y pueden *dispararse*:

- Video (Procesos ejecutados por la instancia de la clase *DShowManager*)
  - Selección del dispositivo de adquisición. La *GUI dispara* un evento que inicia el proceso de determinar que dispositivos están conectados al sistema.
  - Inicio de la captura.
  - Pausa, paro y reinicio en la captura.
  - Adquisición de un *frame* capturado. Se trata de una tarea en combinación con una instancia de *D3DVidPlane*, ya que *DShowManager* captura el cuadro y *D3DVidPlane* lo dibuja en su dispositivo.
- Elección de una región de interés (*ROI*) en el cuadro adquirido. Otra tarea conjunta de la *mkSIFT* y *D3DVidPlane*, quien finalmente es la que determina, pinta y obtiene la *ROI*.
- SIFT (Instancias de *SIFT_Sharp*). Se utilizan dos instancias, una configurada para ser *single threaded* y otra que es *multithreaded*.
  - Generación de descriptores sobre la *ROI*. Se ejecuta SIFT mediante la instancia que ejecuta un sólo hilo para procesar la *ROI* y determinar los descriptores.

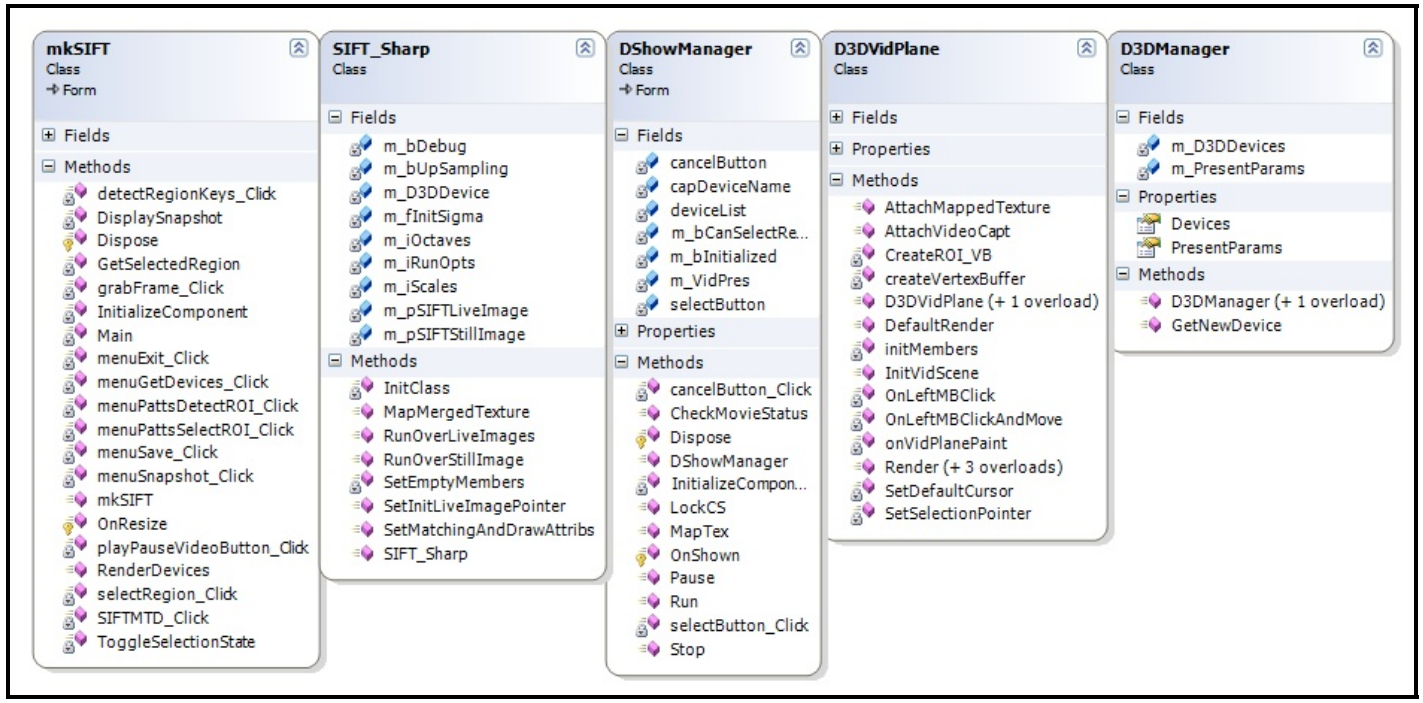


Figura 4.22: Clases involucradas en el funcionamiento de la interfaz de usuario

- Procesamiento en tiempo real del video adquirido. La instancia *multihilos* calcula SIFT sobre cada *frame* del video y hace los procesos de *matching* contra los datos computados de la *ROI*.

- *Render* (Procesos ejecutados por las instancias de la clase *D3DVidPlane*)
  - Video adquirido. Dibuja el *buffer* con la imagen obtenida del *stream* de video.
  - *Frame* obtenido. Mantiene el despliegue del cuadro adquirido y la *ROI*, si es que hay algo seleccionado.
  - Imagen procesada por SIFT. Muestra las correspondencias encontradas entre la imagen de referencia (*ROI*) y el cuadro procesado.

Capítulo 5

Pruebas de *software*

Debido a que el *software* elaborado está compuesto por diversos módulos, las pruebas realizadas se dividieron en las siguientes secciones:

1. **Imágenes *fijas*.** Con ellas se probó que las rutinas encargadas de procesar el algoritmo SIFT, en su más bajo nivel —enfoque procedimental en *C*—, funcionaran correctamente. Para ello es necesario crear cada uno de los atributos de forma independiente y es tarea del programador. El conjunto de prueba se específica en la sección 5.1.

2. **Secuencia de imágenes.** Se alimentaron varias imágenes a una instancia de la clase SIFT que contiene los atributos necesarios para la ejecución del algoritmo y prueba las correspondencias y rastreo del objeto determinado. La sección 5.2 define las pruebas y criterios utilizados. Básicamente es la misma prueba que la anterior, sólo que usando el paradigma *OO* y con medición del tiempo de procesamiento, para compararlo con los resultados del siguiente punto.

3. **Secuencia de imágenes en *multithreading*.** Con esta prueba se pretende determinar cuanto tiempo tarda la ejecución del total de las imágenes suministradas en la sección anterior operando en *pipeline*.

4. **Análisis del video en tiempo real.** Por medio de la aplicación construida en C#, se probará el reconocimiento y rastreo de objetos en las imágenes adquiridas mediante una *webcam*.

5.1. Procesamiento con imágenes fijas en C

Las pruebas realizadas en esta fase son relativamente simples. Para ellas se utilizaron 3 diferentes objetos: una caja a la que se denominará *Box Extreme*, una taza y un par de muñecos de peluche. Además, se hicieron pruebas con una fotografía del mundo real tomada en la ciudad de Seattle. Las pruebas se pueden catalogar en:

1. *Matching* y transformación de la imagen de Seattle bajo rotaciones planas.

2. Rastreo de una zona en la imagen de Seattle bajo rotaciones.

3. Rotaciones y cambios de escala para *Box Extreme*.

4. Rotaciones y cambios de escala para la taza.

5. Un pequeño intento de panorama con la caja *Box Extreme* y la taza.

6. Reconocimiento uno a uno mediante el par de peluches.

5.1.1. Rotaciones de una imagen: *Vista de Seattle*

Se tomo la imagen en su posición original, y mediante un programa de edición de imágenes, se fue rotando en intervalos de 15° en sentido anti–horario. La prueba consiste en encontrar la transformada que mapee correctamente las rotaciones en las imágenes. La imagen de referencia tiene un tamaño de 640x480 pixeles y 3 canales de color. Las subsecuentes operaciones fueron hechas con imágenes del mismo tamaño que la de referencia, salvo aquellas rotadas 90 y 270°, que son de 480x640 pixeles, también con 3 canales de color. En el cuadro 5.1 se han colocado los resultados obtenidos en la ejecución de SIFT sobre las imágenes.

ángulo	Candidatos		Keypoints		Descriptores		Matches		t total
[grados]	Número	$t[s]$	Número	$t[s]$	Número	$t[s]$	Número	$t[s]$	$t[s]$
0 °	11158	2.203	1199	0.110	1593	0.454	1199	3.546	6.375
15 °	8520	2.547	959	0.094	1239	0.360	324	2.843	5.875
30 °	7855	2.453	888	0.094	1131	0.343	296	2.531	5.453
45 °	7060	3.078	870	0.125	1073	0.563	206	3.344	7.141
60 °	7509	2.218	823	0.079	1049	0.296	239	2.704	5.328
75 °	7297	3.812	769	0.125	957	0.469	189	3.000	7.453
90 °	11152	2.000	1257	0.156	1607	0.469	541	3.609	6.265
105 °	7646	2.235	822	70.078	1033	0.282	188	2.156	4.781
120 °	7991	2.218	868	0.094	1076	0.297	187	2.328	4.953
135 °	7443	2.234	910	0.078	1143	0.328	183	2.406	5.062
150 °	8087	2.235	941	0.078	1216	0.359	199	3.547	6.235
165 °	8424	2.188	921	0.093	1227	0.329	233	2.547	5.188
180 °	11134	2.656	1191	0.156	1530	0.516	706	5.109	8.500
195 °	8518	2.562	960	0.094	1219	0.375	243	2.844	5.891
210 °	7845	2.453	877	0.093	1103	0.329	229	3.234	6.125
225 °	7057	2.609	868	0.079	1079	0.328	177	2.516	5.578
240 °	7511	2.594	829	0.093	1034	0.375	199	3.110	6.187
255 °	7292	4.546	768	0.125	962	0.562	206	3.125	8.484
270 °	11127	2.078	1252	0.125	1583	0.453	628	4.953	7.640
285 °	7642	2.281	815	0.094	1026	0.469	196	3.110	5.984
300 °	7979	2.250	858	0.078	1109	0.297	223	2.844	5.516
315 °	7445	2.234	909	0.078	1161	0.360	195	2.562	5.266
330 °	8102	2.265	933	0.078	1195	0.375	271	3.750	6.500
345 °	8431	2.219	936	0.094	1228	0.343	297	2.610	5.281
Promedio	8343	2.507	934	0.100	1191	0.389	315	3.097	6.128

Cuadro 5.1: Rotaciones de [0, 345] ° para la foto de Seattle

De este primer experimento se observa que son detectados un número importante de candidatos, en promedio 8343. El siguiente paso consiste en el refinamiento de ellos (interpolación a su verdadera localidad y eliminación de respuesta a bordes, sección 3.2.2), en donde se producen 934 en promedio. Para las consideraciones de Lowe acerca de este valor, en su artículo [Low04] indica que deben encontrarse alrededor de 1500 puntos, cifra que en ciertas ocasiones es alcanzada y, en lo que al promedio refiere, cumple con una proporción del 62.27 %. En la columna de descriptores, se tiene que se calculan cerca de 27.51 % más del número de *keypoints* encontrados.

De acuerdo con Lowe, la asignación de múltiples orientaciones ocurriría para cerca del 15 % de los puntos. En la actual medida sólo se conoce el número

final y la consideración se ha realizado a partir de la hipótesis de que se tendría una orientación adicional por punto de interés. Sin embargo, el porcentaje reflejado de 27.51 %, podría acercarse más al 15 % reportado, si se hace una evaluaci'on minuciosa de los *keypoints* que generan más de una orientación.

La taza de reconocimiento obtenida es del 33.72 %, un número bajo para los resultados reportados por Lowe. Sin embargo, para los fines de reconocimiento en el conjunto de pruebas, y el uso de una transformación afín en 2D para representar las homografías, conduce a que se necesiten tan sólo 3 correspondencias entre pares de puntos para poder calcularla. No obstante, no se cumple con la proporción esperada y, además, se han encontrado falsas correspondencias. No se midieron las falsas correspondencias puesto que el conjunto de puntos de interés encontrados es alto y no se cuenta con un previo reconocimiento de los mismos para efectos de futuras pruebas. En las imágenes se pueden observar algunos de ellos, pero no se tiene una medida cuantitativa.

El tiempo de cómputo es alto para fines de aplicaciones de tiempo real: 6.128*s* simplemente es demasiado. Sin embargo, si se hace un submuestreo de la imagen original, disminuyendo la resolución de la misma, el tiempo mejorará. En la sección 5.3 se exponen los resultados de trabajar con imágenes de la mitad de tamaño (320x240x3).

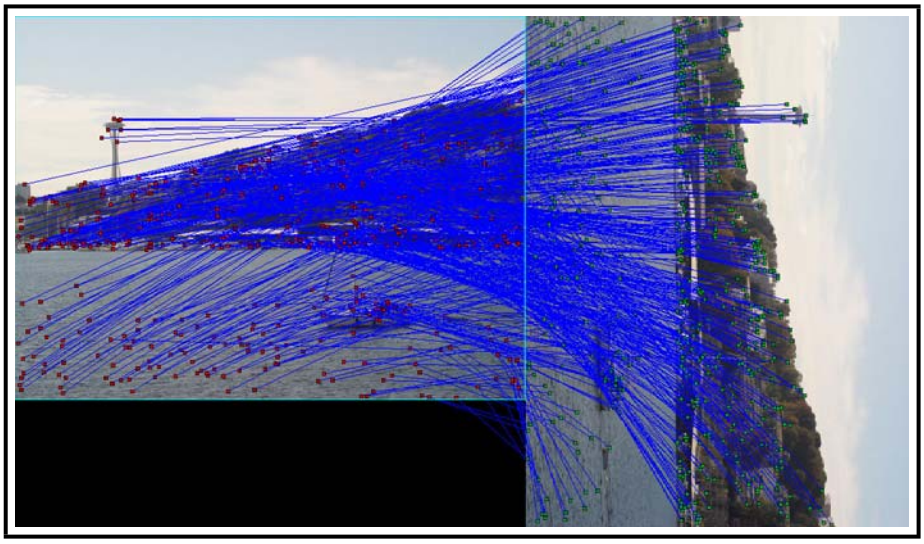


Figura 5.1: Muestra de salida para una rotación de 90°

ángulo	Homografía real				Homografía calculada				Error			
[grados]	m_{11}	m_{12}	m_{21}	m_{22}	m_{11}	m_{12}	m_{21}	m_{22}	m_{11}	m_{12}	m_{21}	m_{22}
0°	1.000	0.000	0.000	1.000	1.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000
15°	0.966	0.259	-0.259	0.966	0.963	0.184	-0.257	1.000	0.003	0.075	0.002	0.034
30°	0.866	0.500	-0.500	0.866	0.866	0.353	-0.499	0.971	0.000	0.147	0.001	0.105
45°	0.707	0.707	-0.707	0.707	0.605	0.790	-0.675	0.683	0.102	0.083	0.032	0.024
60°	0.500	0.866	-0.866	0.500	0.492	0.832	-0.867	0.543	0.008	0.034	0.001	0.043
75°	0.259	0.966	-0.966	0.259	0.263	0.974	-0.959	0.272	0.004	0.008	0.007	0.013
90°	0.000	1.000	-1.000	0.000	0.000	1.000	-1.000	0.000	0.000	0.000	0.000	0.000
105°	-0.259	0.966	-0.966	-0.259	0.039	1.490	-1.451	-1.137	0.298	0.524	0.485	0.878
120°	-0.500	0.866	-0.866	-0.500	-0.612	1.111	-1.037	-0.112	0.112	0.245	0.171	0.388
135°	-0.707	0.707	-0.707	-0.707	-0.433	0.749	0.493	-0.674	0.274	0.042	1.201	0.033
150°	-0.866	0.500	-0.500	-0.866	-1.673	1.071	-0.223	-1.053	0.807	0.571	0.277	0.187
165°	-0.966	0.259	-0.259	-0.966	-0.730	0.453	-0.047	-0.795	0.236	0.194	0.212	0.171
180°	-1.000	0.000	0.000	-1.000	-1.001	-0.009	0.000	-1.000	0.001	0.009	0.000	0.000
195°	-0.966	-0.259	0.259	-0.966	-0.995	-0.385	0.207	-1.220	0.029	0.127	0.052	0.254
210°	-0.866	-0.500	0.500	-0.866	-1.386	1.848	0.726	-1.912	0.520	2.348	0.226	1.046
225°	-0.707	-0.707	0.707	-0.707	-0.678	-0.627	0.217	-1.993	0.029	0.080	0.490	1.286
240°	-0.500	-0.866	0.866	-0.500	-0.345	-0.496	0.581	-1.209	0.155	0.370	0.285	0.709
255°	-0.259	-0.966	0.966	-0.259	-0.348	-0.684	3.286	-7.767	0.089	0.282	2.320	7.508
270°	0.000	-1.000	1.000	0.000	0.038	-1.160	1.000	0.000	0.038	0.160	0.000	0.000
285°	0.259	-0.966	0.966	0.259	0.648	-0.730	1.706	0.706	0.389	0.236	0.740	0.447
300°	0.500	-0.866	0.866	0.500	0.527	-0.659	0.889	0.693	0.027	0.207	0.023	0.193
315°	0.707	-0.707	0.707	0.707	0.697	-0.921	0.718	0.983	0.010	0.214	0.011	0.276
330°	0.866	-0.500	0.500	0.866	0.882	-0.481	0.479	0.837	0.016	0.019	0.021	0.029
345°	0.966	-0.259	0.259	0.966	0.979	-0.225	0.258	0.960	0.013	0.033	0.001	0.006

Cuadro 5.2: Comparación entre la homografía real y la calculada

En las imágenes producidas (apéndice D) se muestra que, en la mayoría de los casos, se obtuvo una homografía apropiada para la rotación especificada. El cuadro 5.2 especifica la homografía que se debía obtener, la calculada y el error entre ellas.

Como bien puede observarse, el error en algunos casos llega a ser alto. La homografía, para fines de rotación y sin cambios de escala, está especificada por una matriz de 2x2 que define la rotación alrededor de un eje perpendicular al plano y, por lo tanto, sus componentes son senos y cosenos. El rango del par de funciones trigonométricas está en [0,1] y, en los resultados arrojados del cómputo interno de la homografía, hay valores que claramente están fuera del rango.

La razón de que ello es la forma en la que se obtiene la transformación. Para un par de imágenes con *n keypoints* en correspondencia ($n > 3$), se toman 3 parejas con un proceso aleatorio, se calcula la homografía y luego se calcula el error de transferencia simétrico —sección 4.6—. El proceso se repite en varias ocasiones (para estas pruebas se realizó 50 veces) y se toma aquella homografía que haya presentado el menor error. Sin embargo, el proceso aleatorio puede tomar como parejas de puntos a falsos *matches* y, si esto ocurre un numero considerable de veces, la homografía calculada será de mala calidad. Esto sucede en algunos de los casos aquí reportados.

5.1.2. Rastreo de una zona en la imagen de Seattle bajo rotaciones

Se realizaron las mismas pruebas que en la sección anterior, sólo que con una selección de una región de la imagen. La figura 5.2 la muestra. Se detectaron 2995 candidatos, produciéndose 387 puntos de interés y 512 descriptores. La hipótesis de este experimento es que la probabilidad de encontrar falsos matches, debe reducirse, ya que los descriptores utilizados corresponden a una sección más pequeña de la imagen y por lo tanto son más locales.

Figura 5.2: Sección de la imagen de Seattle usada como referencia

El número de correspondencias promedio fue de 106, lo que conduce a una taza de reconocimiento del 27.40 %. Si se compara con la obtenida en la sección anterior, la presente es menor en aproximadamente 6 %. La hipótesis presentada

ángulo	Homografía real				Homografía calculada				Error			
[grados]	m_{11}	m_{12}	m_{21}	m_{22}	m_{11}	m_{12}	m_{21}	m_{22}	m_{11}	m_{12}	m_{21}	m_{22}
0°	1.000	0.000	0.000	1.000	1.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000
15°	0.966	0.259	-0.259	0.966	1.064	0.231	-0.247	0.962	0.098	0.028	0.012	0.003
30°	0.866	0.500	-0.500	0.866	1.031	0.717	-0.205	1.242	0.165	0.217	0.295	0.376
45°	0.707	0.707	-0.707	0.707	0.684	1.023	-0.684	0.692	0.023	0.315	0.023	0.015
60°	0.500	0.866	-0.866	0.500	1.417	3.667	-0.833	0.667	0.917	2.801	0.033	0.167
75°	0.259	0.966	-0.966	0.259	0.827	1.025	-0.269	0.181	0.569	0.059	0.697	0.078
90°	0.000	1.000	-1.000	0.000	0.000	1.000	-1.000	0.000	0.000	0.000	0.000	0.000
105°	-0.259	0.966	-0.966	-0.259	-0.012	1.627	-0.929	-0.155	0.246	0.661	0.036	0.104
120°	-0.500	0.866	-0.866	-0.500	-0.470	0.910	0.066	0.846	0.030	0.044	0.932	1.346
135°	-0.707	0.707	-0.707	-0.707	-0.692	1.180	-0.709	-0.849	0.015	0.473	0.002	0.142
150°	-0.866	0.500	-0.500	-0.866	-0.917	0.616	-0.416	-1.081	0.051	0.116	0.084	0.215
165°	-0.966	0.259	-0.259	-0.966	-0.951	0.242	-0.224	-1.030	0.015	0.017	0.035	0.065
180°	-1.000	0.000	0.000	-1.000	-1.000	0.000	0.000	-1.000	0.000	0.000	0.000	0.000
195°	-0.966	-0.259	0.259	-0.966	-1.198	-0.524	0.046	-1.215	0.232	0.266	0.213	0.249
210°	-0.866	-0.500	0.500	-0.866	1.648	0.893	-0.732	-1.550	2.514	1.393	1.232	0.684
225°	-0.707	-0.707	0.707	-0.707	-0.878	-0.872	-0.006	-1.435	0.171	0.165	0.714	0.728
240°	-0.500	-0.866	0.866	-0.500	-0.500	-1.000	0.871	-1.066	0.000	0.134	0.005	0.566
255°	-0.259	-0.966	0.966	-0.259	-0.394	-1.119	0.813	-0.438	0.135	0.153	0.153	0.179
270°	0.000	-1.000	1.000	0.000	0.250	-1.400	1.000	0.000	0.250	0.400	0.000	0.000
285°	0.259	-0.966	0.966	0.259	1.047	-0.683	1.367	0.402	0.788	0.283	0.401	0.143
300°	0.500	-0.866	0.866	0.500	0.377	-1.508	0.869	0.525	0.123	0.642	0.003	0.025
315°	0.707	-0.707	0.707	0.707	1.062	-0.850	0.528	0.778	0.355	0.143	0.179	0.071
330°	0.866	-0.500	0.500	0.866	0.939	-0.528	0.473	0.876	0.073	0.028	0.027	0.010
345°	0.966	-0.259	0.259	0.966	1.061	-0.017	0.210	1.000	0.095	0.242	0.049	0.034

Cuadro 5.3: Comparación entre la homografía real y la calculada de la zona elegida de la imagen de Seattle

pareciera derrumbarse, sin embargo, falta ver los cálculos en lo que a materia de homografías se refiere. Para ello, véase el cuadro 5.3.

Los errores obtenidos entre las homografías encontradas de esta prueba y la de la sección 5.1.1 son parecidos y no se puede definir categóricamente cual es mejor. Sin embargo, para fines de la hipótesis hecha al inicio y basándose más en un análisis cualitativo del reconocimiento y rastreo mediante las imágenes producidas, la región buscada está mejor delimitada en cada imagen y alineada con respecto de la rotación que se hizo. Por ejemplo, la imagen 5.3 muestra una buena detección y alineación respecto de la rotación. No obstante, los errores de los puntos registrados generan un

cierto escalamiento, que provoca que el recuadro delimitante de la región de interés se vea más grande. En el apéndice D está la colección de imágenes generadas que muestran el rastreo en 2D de la región definida por la imagen de referencia.

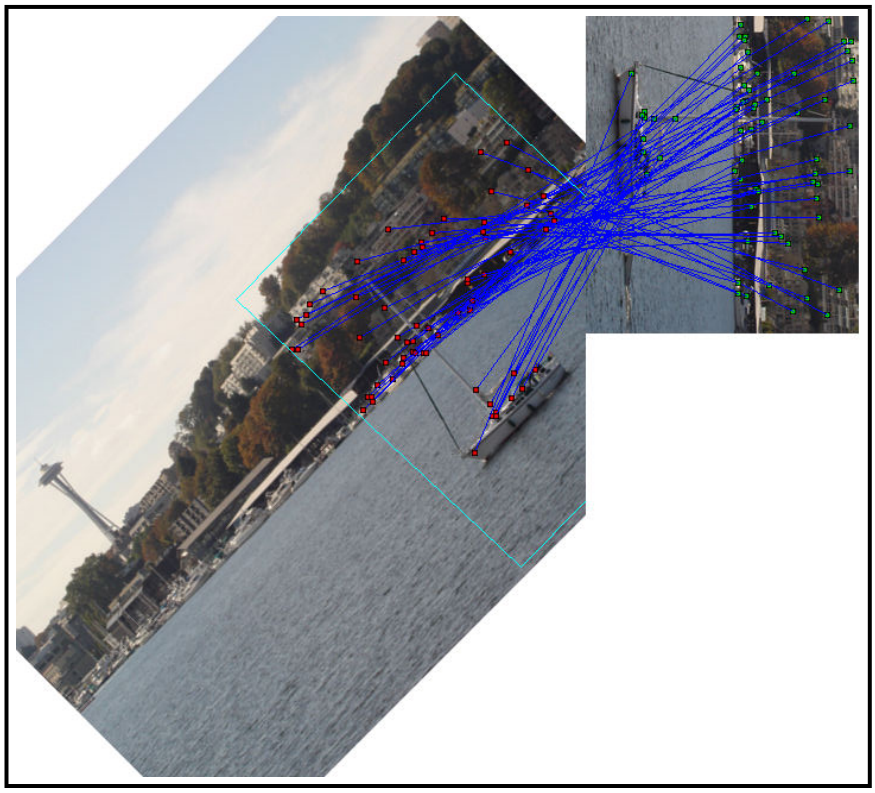


Figura 5.3: Detección para una rotación de 135° usando una sección de la imagen

5.1.3. Rotaciones y cambios de escala en el sujeto *Box Extreme*

Si bien las pruebas anteriores ya evaluaron de alguna manera al algoritmo y denotaron sus características de reconocimiento y rastreo, no se pueden catalogar de contundentes puesto que se trata de una proyección que no se puede manipular. No se pueden hacer movimientos en 3D ni tampoco hacer cambios de escala que sean realistas. Si se escala la imagen, su tamaño cambia, pero la *información* extra, proveniente del fondo de la imagen no cambia. Ello es un factor que puede ser determinante en tareas reales y es por ello que se muestran aquí algunas prubas hechas.

Lo primero fue probar el algoritmo bajo rotaciones en un rango de -70 a 70° sin alterar la escala. El cuadro 5.4 muestra los resultados obtenidos. La imagen de referencia produjo 7326 candidatos, 1126 *keypoints* y 1478 descriptores. El porcentaje de correspondencias fue del 12.26 %, una cifra muy baja que, sin embargo, por la alta densidad de puntos de interés encontrado (en promedio 1057), ofrece buenos resultados para rotaciones en el plano.

ángulo	Candidatos		Keypoints		Descriptores		Matches		t total
[grados]	Número	$t[s]$	Número	$t[s]$	Número	$t[s]$	Número	$t[s]$	$t[s]$
90 °	7632	2.016	1028	0.078	1354	0.375	112	2.671	5.172
70 °	10227	2.031	1000	0.110	1296	0.359	116	2.547	5.063
60 °	10611	2.140	1068	0.094	1392	0.406	115	3.859	6.515
50 °	10402	2.109	1097	0.109	1420	0.391	111	3.922	6.562
40 °	9736	2.125	1049	0.093	1363	0.391	129	3.672	6.296
30 °	10202	2.109	983	0.093	1295	0.375	140	3.329	5.937
20 °	11588	2.110	1024	0.109	1350	0.375	154	3.406	6.016
-10 °	11335	2.110	1113	0.125	1512	0.422	269	4.078	6.766
-20 °	11618	2.110	1114	0.110	1450	0.406	185	3.375	6.032
-30 °	10454	2.125	1074	0.110	1375	0.406	179	3.375	6.407
-40 °	9441	2.125	1056	0.093	1379	0.391	93	3.781	6.406
-50 °	9367	2.110	1057	0.093	1386	0.391	96	3.391	6.016
-60 °	9863	2.125	1079	0.109	1403	0.406	97	3.907	6.562
Promedio	10190	2.103	1057	0.102	1383	0.392	138	3.514	6.135

Cuadro 5.4: Mediciones de rotación para el sujeto de pruebas *Box Extreme*

La siguiente prueba fue determinar el comportamiento bajo rotaciones en profundidad. La homografía que se calcula es 2D, por lo que para este experimento no tiene sentido. Sin embargo, las imágenes producidas, en términos de las correspondencias encontradas, ofrecen una idea de que tan correctas son. En la imagen de referencia se obtuvieron 7955 candidatos, 1130 *keypoints* y 1469 descriptores. La tasa de reconocimiento promedio se colocó en 15.70 %. Un ejemplo con tres imágenes es el de la figura 5.4.

Para las rotaciones mostradas —ángulos de -20, 0 y 20°—, se observa que a pesar de contar con pocos *matches*, la detección y rastreo no parecen estar tan mal, ni tampoco que muchos de ellos sean falsos. En [Low04] se especifica que las rotaciones en profundidad han mostrado ser detectadas no más allá de los 40 o 50°. En las imágenes del apéndice D se aprecia como a medida que va aumentando el ángulo de rotación, la detección comienza a ser más pobre e incorrecta.

(a) Rotación de -20 ° (b) Sin rotación (c) Rotación de 20 °

Figura 5.4: *Collage* de tres imágenes rotadas de *Box Extreme*

ángulo	Candidatos		Keypoints		Descriptores		Matches		t total
[grados]	Número	$t[s]$	Número	$t[s]$	Número	$t[s]$	Número	$t[s]$	$t[s]$
-70 °	6890	2.046	743	0.079	976	0.250	17	1.797	4.187
-60 °	8420	2.156	1022	0.094	1341	0.375	12	2.719	5.359
-50 °	9918	2.078	1230	0.094	1591	0.453	15	3.391	6.032
-40 °	11132	3.218	1352	0.219	1758	0.813	20	4.828	9.078
-30 °	11529	2.094	1328	0.110	1676	0.469	69	3.531	6.219
-20 °	11705	2.156	1307	0.110	1681	0.562	147	4.891	7.750
-10 °	12407	2.250	1309	0.125	1677	0.484	251	3.438	6.328
0 °	11687	2.235	1170	0.125	1517	0.422	1517	3.266	6.094
10 °	11506	2.140	1181	0.110	1554	0.453	245	3.218	5.937
20 °	10629	2.063	1095	0.109	1456	0.406	141	2.875	5.469
30 °	10742	2.860	1320	0.203	1703	0.859	94	5.125	9.094
40 °	10487	2.125	1312	0.110	1711	0.484	68	4.250	7.001
50 °	10062	2.125	1308	0.110	1706	0.484	36	5.156	7.891
60 °	9859	2.062	1273	0.094	1688	0.484	24	4.422	7.078
70 °	9027	2.063	1021	0.094	1337	0.359	14	2.610	5.141
Promedio	10400	2.244	1198	0.119	1558	0.490	178	3.701	6.577

Cuadro 5.5: Mediciones para rotación en profundidad para el sujeto de pruebas *Box Extreme*

Como se ha explicado en la sección 3.2, el descriptor de SIFT es invariante bajo rotaciones y escalas. Hasta ahora, para diferentes ángulos de rotación los descriptores calculados lo han comprobado, aunque con una efectividad baja. El caso de la escala se probó haciendo tres casos: un alejamiento y un acercamiento, con respecto de la posición inicial de 10cm y la pose inicial. ¿Por qué se evalúan tan pocos? La razón es que se probaron distancias mayores, pero el reconocimiento se perdía, produciendo pocas correspondencias e incluso ninguna.

El cambio de escala equivale una variación uniforme del 15 % en el tamaño del sujeto analizado y, para la imagen de referencia, se obtuvieron 6496 candidatos, 1026 puntos de interés y 1367 descriptores. Con la imagen en su pose inicial, se tuvo una correspondencia de 100 % de los puntos, disminuyendo al 8.47 % y 5.16 % (87 y 53 correspondencias respectivamente). Ver figura 5.5.

Respecto de los errores de escala aquí comentados, se tienen dos hipótesis. La primera consiste en que los puntos encontrados en diferentes octavas del proceso de detección no sean repetibles. Esto es, que si en la octava O_i se encontró el pixel (x, y), en la siguiente octava O_{i+1} no se detecte el pixel $(x/2, y/2)$.

(a) Alejamiento de 10cm (b) Posición original (c) Acercamiento de 10cm

Figura 5.5: Cambios de escala para *Box Extreme*

La situación anterior conduciría a encontrar falsos *matches* que impedirían un reconocimiento preciso. Otra posibilidad, respecto de la localidad que se asigna al punto de interés, es que se esté efectuando una interpolación errónea del mismo y entonces sea recorrido a posiciones que no son repetibles.

La otra hipótesis es que el descriptor este siendo mal obtenido respecto de la octava en donde fue encontrado el *keypoint*. Como se ha comentado en la sección 3.2.3, el descriptor depende de una ventana (cuyo tamaño, a su vez, depende de la escala, σ, del punto de interés) donde se opera con sus gradientes, y de una ventana *fija* de 16x16 pixeles.

Entonces, en las útimas octavas, el descriptor puede dejar de ser local, ya que la vecindad cada vez se va haciendo más y más grande, y la imagen procesada más pequeña en un factor de dos. La característica de dejar de ser local, conduciría a que el vector de histogramas varié con respecto de aquellas en donde sí se define un vecindario local al punto de interés, y entonces *la distancia mínima entre los descriptores* de dos localidad similares sea considerable.

5.1.4. Rotaciones y cambios de escala para la taza

El experimento anterior sirvió para conocer la respuesta del algoritmo implementado ante cambios de escala y rotaciones para el sujeto de pruebas *Box Extreme*. El caso aquí presentado resulta ser una curiosidad con respecto de la implementación hecha. Esta taza tiene una leyenda al centro que es completamente simétrica y el *fondo* (la misma taza) es completamente uniforme.

Entonces, lo que se pretende aquí demostrar es que existen equivocaciones en el empate de puntos debido a tal simetría. La taza se ilustra en la figura 5.6. Como puede observarse, la imagen central es un patrón perfectamente simétrico. Entonces, ¿qué diferencía a los gradiente de la sección superior en la frase *LEGORRETA* y su vecindad, respecto de la que se tiene a la izquierda, derecha y abajo? La hipótesis que se puede hacer es que, a pesar de que Lowe dice que —durante el cómputo del descriptor— la vecindad debe ser rotada respecto de la orientación analizada, tal vecindario conservará sus características y por lo tanto, el descriptor será muy parecido. Sin embargo, ¿qué ocurre cuando se encuentran correspondencias de la imagen referencia con una imagen que la contiene, sin cambios de escala ni rotaciones y simplemente, es tal cual la misma imagen sólo que con fondo?

Figura 5.6: Taza usada en las pruebas

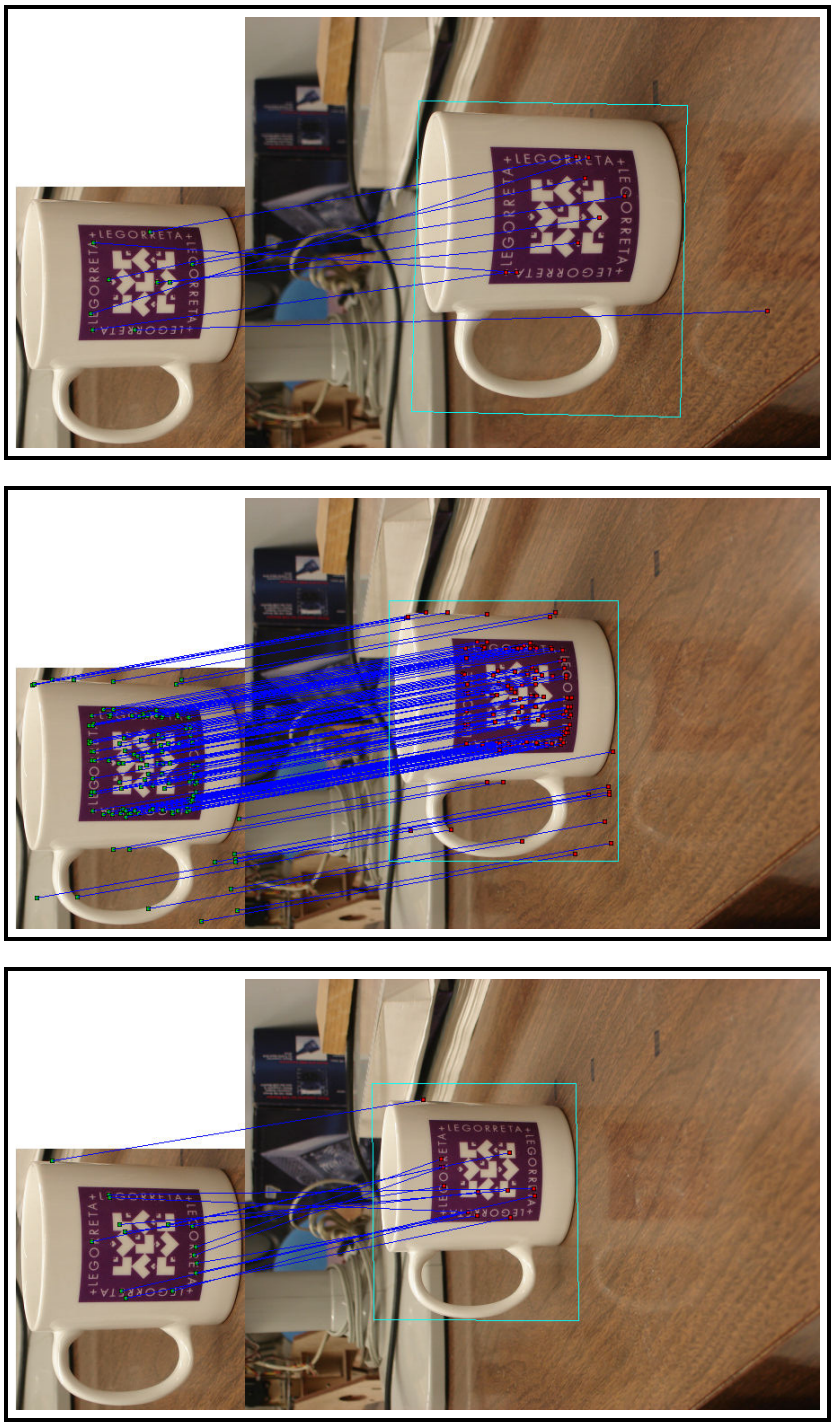
En este caso los descriptores son *altamente distinguibles*, pues son exactamente los mismos para ambas figuras. Por lo tanto, la distancia entre un par de ellos es mínima, de hecho técnicamente debe ser cero. Por eso se encuentran correspondencias exactas, figura 5.7(b).

Para las pruebas de escala (incrementos de $10cm$, equivalentes a un 15 % de aumento o de disminución en tamaño) y el caso de acercamiento se encontraron sólo 9 correspondencias de 207 puntos de interés que contiene la imagen referencia.

Cuando se alejó el objeto, el número creció a 15 *matches*, siendo aun una cifra muy baja (menor al 10 %). Sin embargo, el proceso del cómputo de la homografía fue satidfactorio ya que, como puede observarse en la figura 5.7, la región fue perfectamente encontrada. En las figuras también se alcanzan a notar algunas falsas correspondencias entre pares de puntos que ilustran la hipótesis mencionada.

En el caso de las rotaciones, se utilizarón los ángulos $\theta = \{-10 : -50, -90\}^\circ$ y en promedio se obtuvieron 12 correspondencias de 123 puntos de interés en la imagen de referencia; aproximadamente el 12 %. La figura 5.8 es un ejemplo claro de errores en el empate de los puntos y que da soporte a la hipótesis hecha.

Para comenzar, la imagen muestra un mal cómputo en la homografía, de forma que la región de interés está deformada. Luego, en el caso de las correspondencias, existen puntos que están correctos y otros equivocados. Sin embargo, no parecen ser aleatorias on cualquier otra localidad, sino que se ubican exactamente en puntos de simetría.



(a) Alejamiento de 10*cm*

(b) Posición original

(c) Acercamiento de 10*cm*

Figura 5.7: Cambios de escala para la taza

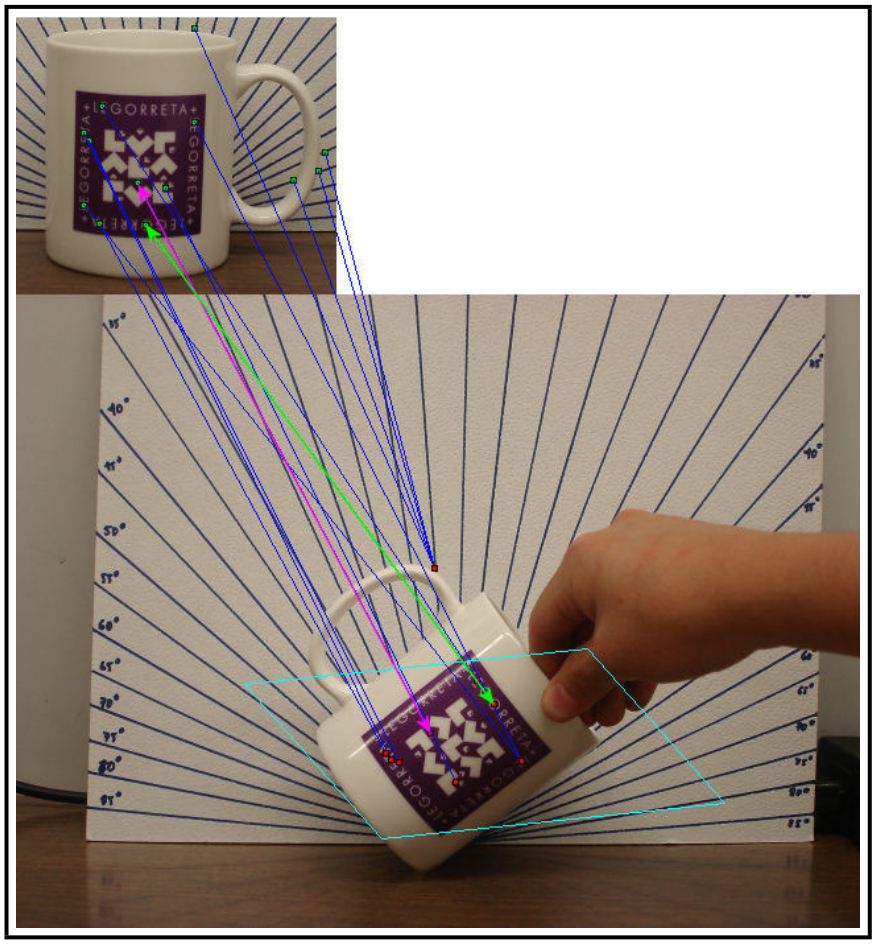


Figura 5.8: Rotación de 50° para la taza. Las flechas indican los puntos equivocados.

Por ejemplo, la flecha verde *confunde* la **O** del texto superior con la **O** del inferior. Lo mismo sucede con un pequeño recuadro que unen las flechas magenta. En el apéndice D se han colocado el resto de las imágenes generadas.

5.1.5. Panorama con la caja *Box Extreme* y la taza.

Dentro de las aplicaciones posibles, 2D, de los algoritmos para extracción de características naturales, está la realización de mosaicos o panoramas con imágenes. El programa *Autopano*, [Now04], es un ejemplo de ello. Entonces, esta sección consta de una pequeña prueba para determinar si pueden extraerse características que puedan utilizarse en el cómputo de este tipo de imágenes.

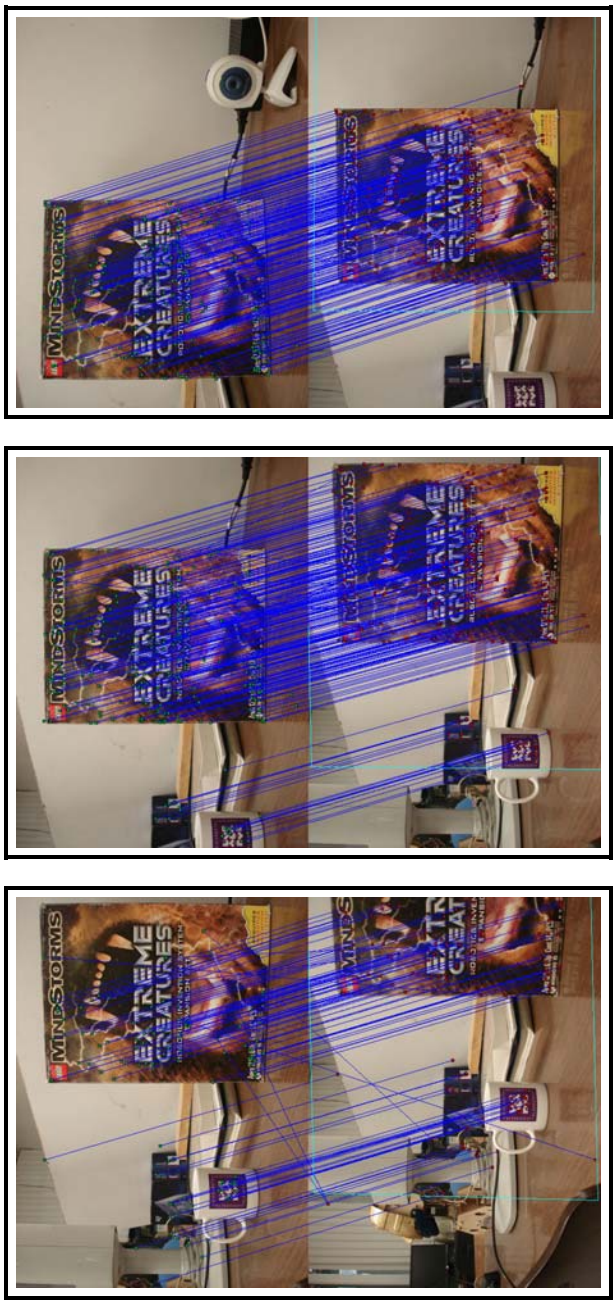
Se configuro una escena formada por los sujetos *Box Extreme* y la taza. Luego, se tomaron sólo cuatro fotografías para ver el efecto. La secuencia es mostrada a continuación en la figura 5.9. Para efectos numéricos, entre la imagen **1–2** se obtuvieron 265 *matches*, para la imagen **2–3** 297 y finalmente para **3–4** 138 correspondencias. Indivudualmente, las imágenes produjeron 918, 1018, 1031 y 761 puntos de interés.

5.1.6. Reconocimiento de un par de peluches

Las pruebas anteriores han trabajado sobre objetos con geometrías muy definidas. La taza y la caja puede considerarse como cajas, si se llevan a un enfoque muy simplista y definido por los borde que pueden observarse cuando se le mira. Otra forma sería definir la taza como un cilindro, que es más correcto. En este tipo de objetos las *esquinas* son abundantes y pueden ser explotados por diferentes algoritmos y sistemas de visión por computadora.

La imagen de Seattle ha sido un buena opción para probar el algoritmo, sin embargo, sólo se cuenta con una imagen para realizar las pruebas. Por ello se decidió utilizar un par de objetos muy comunes en el mundo real. En casas donde hay niños, seguro hay muñecos de peluche y es con un par de ellos que se ha hecho esta sección. Por lo regular son objetos que no tienen una geometría muy definida y que su contaste varía dependiendo de las condiciones de iluminación del ambiente y como incide la luz en ellos. El material con el que son manufacturados ya juega un papel fundamental en los matices que pueden tomar.

La figura 5.10 ilustra el par de objetos utilizados. Esta imagen produce 4635 candidatos, 410 puntos de interés y 530 descriptores. El cuadro 5.6 contiene los



(a) *Matches* para las imágenes 3 y 4 (b) *Matches* para las imágenes 2 y 3 (c) *Matches* para las imágenes 1 y 2

Figura 5.9: Correspondencias para la posterior generación de un panorama

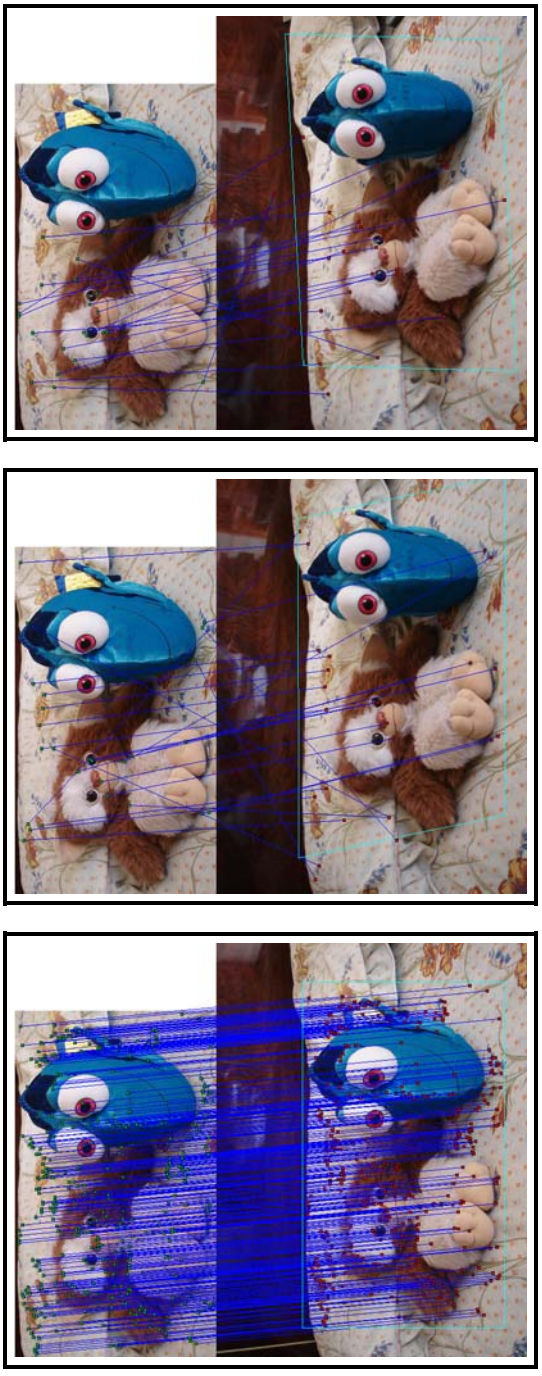
resultados de los experimentos. Se tratan de 6 imágenes que fueron tomadas desde diferentes puntos de vista y sin medición en el ambiente de rotaciones ni escala, por lo que no pueden ser cuantificables.

Figura 5.10: Par de muñecos de peluches utilizados

	Candidatos	*Keypoints*	Descriptores	*Matches*
Frame 1	7454	632	818	390
Frame 2	5372	471	607	26
Frame 3	5190	432	558	21
Frame 4	5231	445	549	20
Frame 5	5181	417	508	8
Frame 6	7072	562	700	28
Promedio	5917	493	623	82

Cuadro 5.6: Resultados en detección para el par de peluches utilizados

Las figuras 5.11 y 5.12 muestran las correspondencias encontradas y el rastreo efectuado. De ellas,se puede determinar que ya resulta ser una tarea más compleja y que el sistema, en el estado actual en el que se encuentra, no es capaz de resolver tal tipo de problemas: se encuentran pocas correspondencias, varias de ellas son incorrectas y en algunos casos no se puede obtener una homografía.

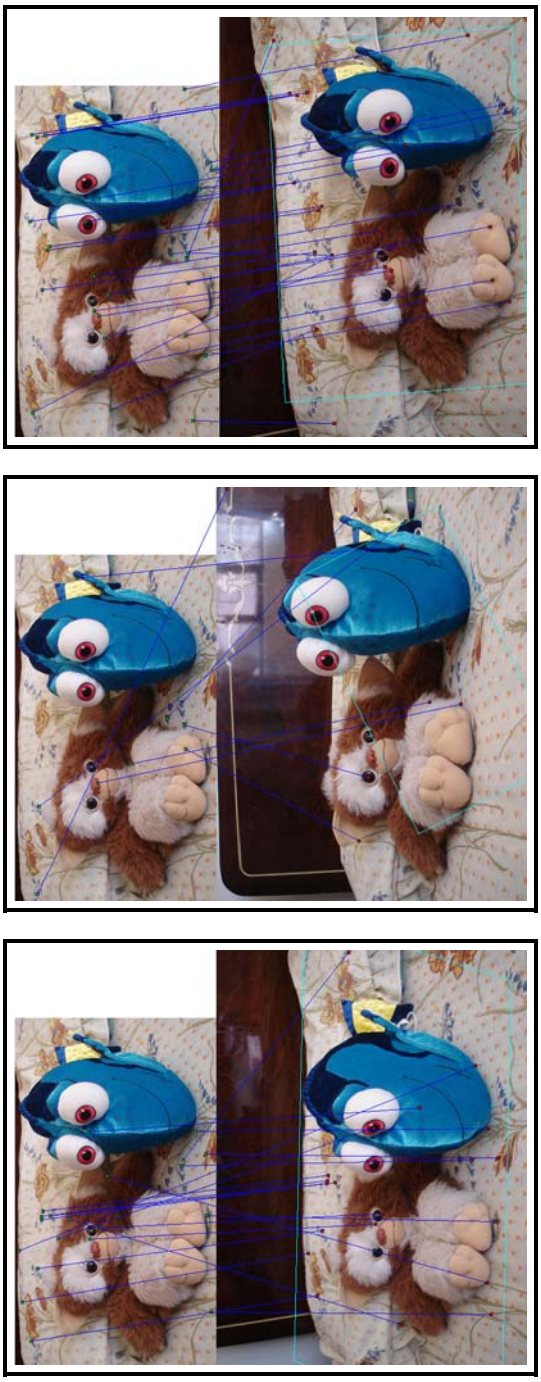


(a) Imagen 1 en la secuencia de 6 imágenes

(b) Imagen 2 en la secuencia de 6 imágenes

(c) Imagen 3 en la secuencia de 6 imágenes

Figura 5.11: Correspondencias para el par de peluches (primera parte)



(a) Imagen 4 en la secuencia de 6 imágenes

(b) Imagen 5 en la secuencia de 6 imágenes

(c) Imagen 6 en la secuencia de 6 imágenes

Figura 5.12: Correspondencias para el par de peluches (segunda parte)

5.2. Clase SIFT y su versión multihilos

Las pruebas a la clase *SIFT* y a *SIFT_Mtd* se hicieron con dos conjunto de 300 imágenes de 320x240 y ya convertidas a escala de gris. El primero de ellos es una secuencia de tomas de *Box Extreme* y el segundo de otra caja utilizada: *Box Logitech* y que no tiene un patrón definido. A continuación se presentan y se discuten los resultados obtenidos mediante las gráficas y cuadros realizados.

Las gráficas están estructuradas de la siguiente manera: primero se muestra el comportamiento de los puntos candidatos, puntos de interés y descriptores encontrados. Luego están las correspondencias que se encuentran y finalmente el tiempo promedio de ejecución de cada fase. En total son 6 gráficas que describen el comportamiento mencionado, 3 para el primer objeto y tres para el otro.

Los cuadros hechos agrupan los atributos calculados para cada objeto. Se realizaron 10 ejecuciones y se obtuvieron datos promedios e individuales a cada una de ellas. Finalmente, se efectuaron 10 ejecuciones para determinar el tiempo de la clase multihilos y compararlo con la clase *SIFT*.

La gráfica 5.13 muestra el comportamiento de los descriptores, puntos de interés y descriptores encontrados. Como se puede observar, el proceso de filtrado resulta ser bastante crítico, pues reduce a los puntos detectados para designarse *keypoints*. En el cuadro 5.7 se tiene que, en promedio, se reducen en un 81.45 % cuando se efectua *up sampling* en la imagen de entrada. Sin embargo, cuando se hace el mismo proceso y no hay muestreo, la tasa de eliminación es un tanto menor: 76.21 %. El número de descriptores generados es de aproximadamente 2:1, 1.77 para ser más exactos.

El número de puntos de interés en correspondencia con otros estuvo alrededor de 39 pares. Esto sí resulta ser un dato interesante, pues el número casi no varía con respecto de si se hace o no *up sampling*. En teoría, hacerlo repercutiría en términos de mayor estabilidad y repetibilidad de las características encontradas. Sin embargo no resulta serlo así. Nuevamente surge la hipótesis de un error en el cálculo del descriptor, o tal vez no se comporte bien una métrica de *matching* como la de distancia euclidiana. En la figura 5.14 se puede apreciar lo comentado con mayor claridad. Mientras que el número de puntos de interés encontrados mantiene su relación de 1.77:1, las correspondencias se mantienen casi iguales, de hecho, las líneas que las representan están casi completamente encimadas.

Box Extreme					
Candidatos		*Keypoints*		Descriptores	
Con *up sampling*	Sin *up sampling*	Con *up sampling*	Sin *up sampling*	Con *up sampling*	Sin *up sampling*
2502	1072	464.1	254.8	593.3	334.9
Matches		Detección [*s*]		Filtrado [*s*]	
Con *up sampling*	Sin *up sampling*	Con *up sampling*	Sin *up sampling*	Con *up sampling*	Sin *up sampling*
38.43	35.03	0.6193	0.1425	0.03106	0.01282
Cómputo del descriptor [*s*]		*Matching* [*s*]		Cómputo de la homografía [*s*]	
Con *up sampling*	Sin *up sampling*	Con *up sampling*	Sin *up sampling*	Con *up sampling*	Sin *up sampling*
0.2034	0.1067	0.4656	0.1017	0.001658	0.001244
Homografías encontradas		Tiempo promedio del cuadro [*s*]			
Con *up sampling*	Sin *up sampling*	Con *up sampling*	Sin *up sampling*		
300/300	300/300	1.334	0.3673		

Cuadro 5.7: Atributos promedios para la secuencia de 300 imágenes y 10 ejecuciones del sujeto de prueba *Box Extreme*

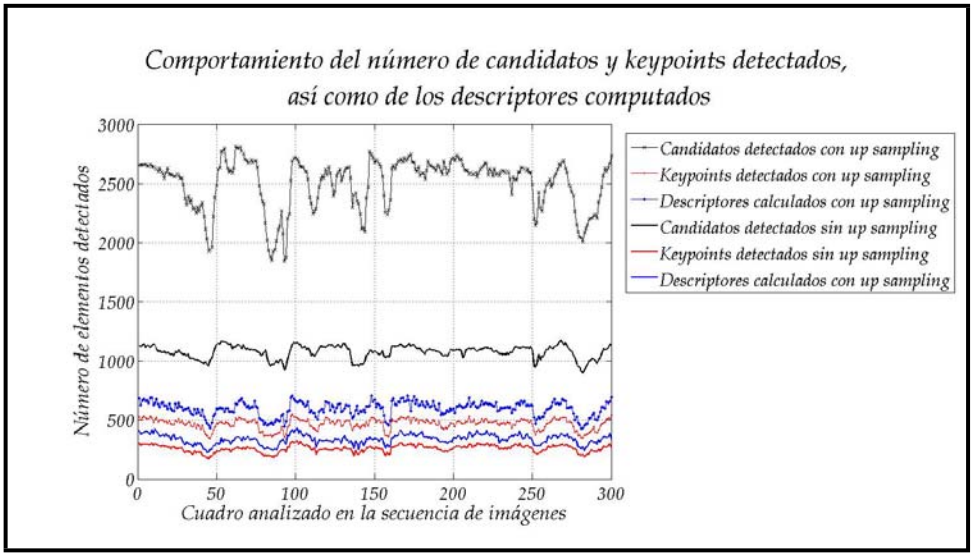


Figura 5.13: Comportamiento en la detección y cálculo de *keypoints*, así como del número de descriptores para el sujeto de prueba *Box Extreme*

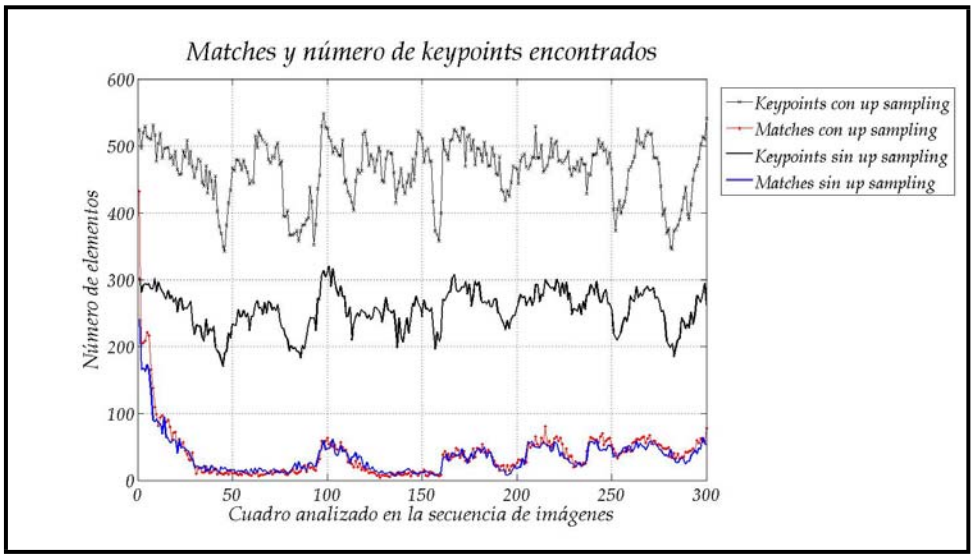


Figura 5.14: Número de *matches* y *keypoints* por *frame* para el sujeto de prueba *Box Extreme*

En cuanto al tiempo de ejecución para cada una de las etapas involucradas en el algoritmo, las figuras 5.15 y 5.16 nos muestran el comportamiento que tienen. Cuando no se hace ningún tipo de muestreo, el rendimiento resulta ser mejor en casi 3.63 % para las categorías mostradas en las gráficas. De hecho, en la gráfica de la figura 5.16 es muy claro como se van acercando los tiempos de procesamiento de detección de candidatos, cálculo de descriptores y obtención de correspondencias.

Las pruebas para la caja *Box Logitech* tuvieron un comportamiento similar a las ya mostradas. La diferencia principal radica en el número de puntos de interés encontrados, y por lo tanto de la tasa de reconocimiento que se tiene. Fuera de ello, el comportamiento es bastante similar y fácilmente verificable al observar las figuras 5.17 y 5.18. Nuevamente las correspondencias tienden a juntarse y ser aproximadamente el mismo número cuando se hace o no *up sampling*.

El promedio de puntos de interés estuvo en 286 cuando se duplica el tamaño de la imagen inicial y de menos de la mitad cuando no se hace, 109. La proporción de *keypoints* detectados respecto de *Box Extreme* es también de casi de 2:1. El cuadro 5.8 contiene los datos promedio encontrados y es en donde puede verse el análisis cuantitativo.

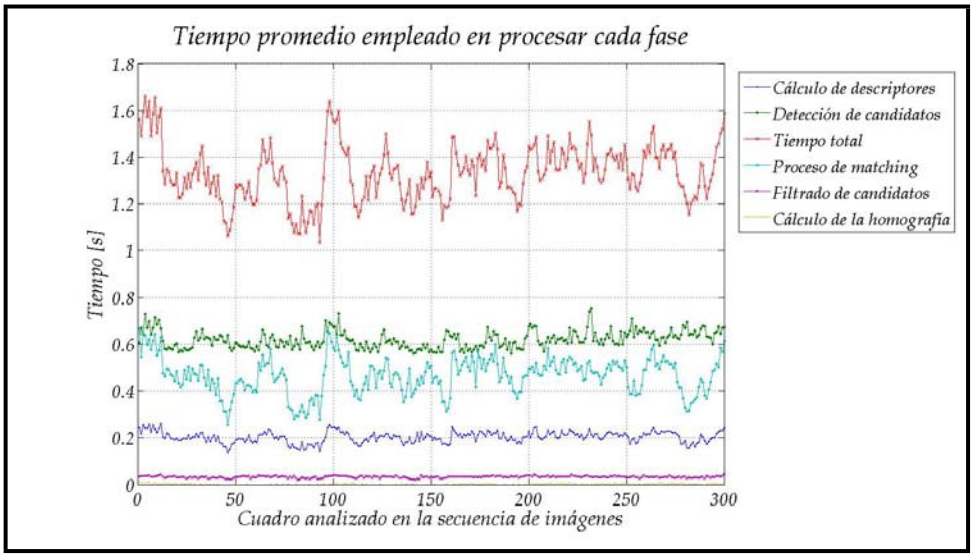


Figura 5.15: Tiempo empleado en calcular cada una de las fases de SIFT para el sujeto de prueba *Box Extreme*, (*up sampling* activo)

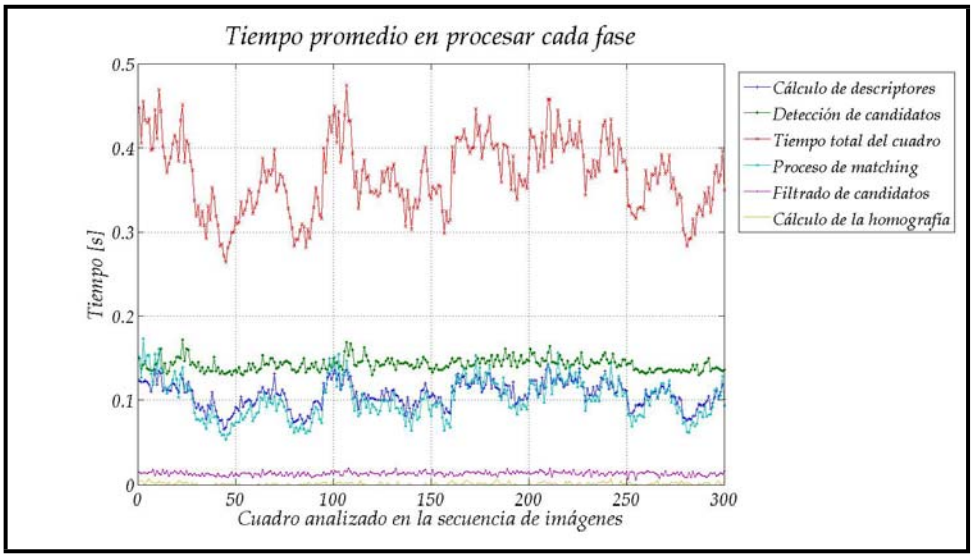


Figura 5.16: Tiempo empleado en calcular cada una de las fases de SIFT para el sujeto de prueba *Box Extreme*, (*up sampling* inactivo)

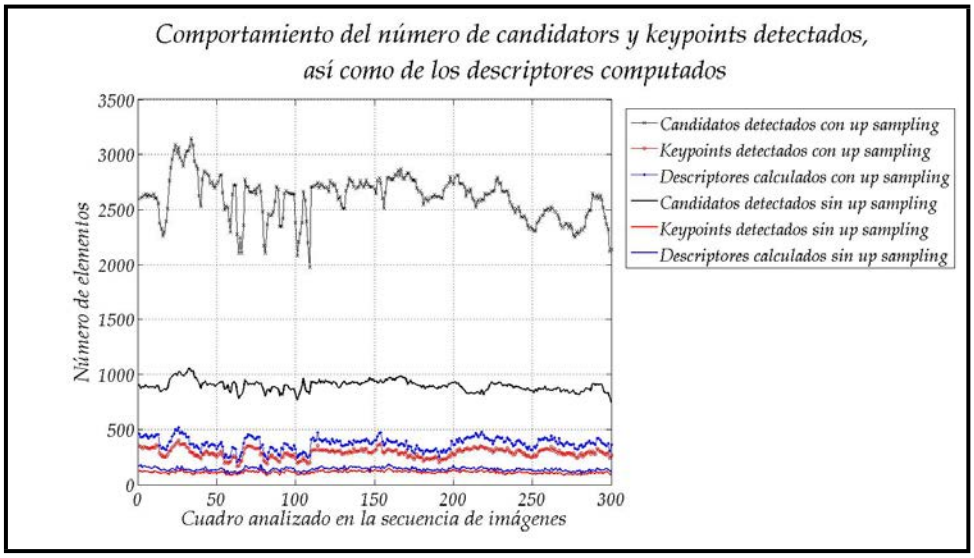


Figura 5.17: Comportamiento en la detección y cálculo de *keypoints*, así como del número de descriptores para el sujeto de prueba *Box Logitech*

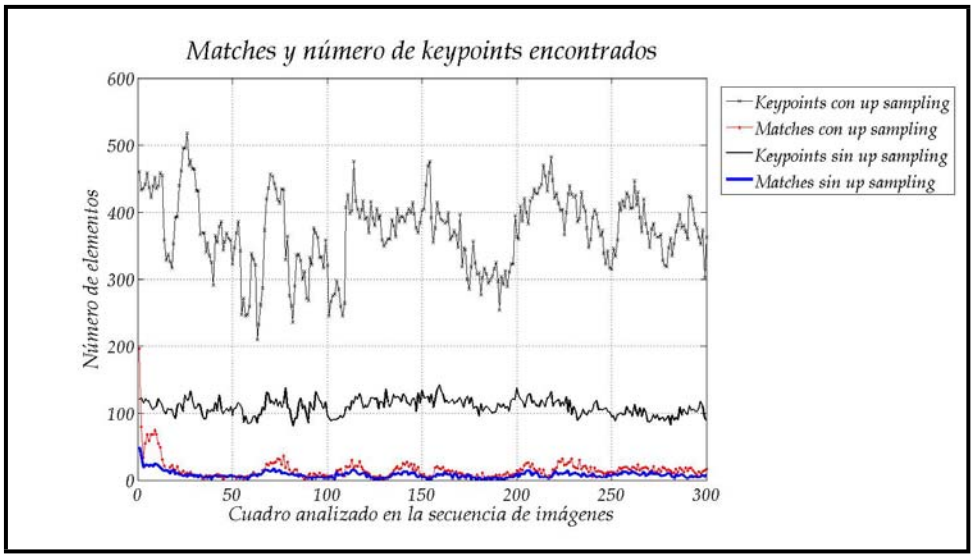


Figura 5.18: Número de *matches* y *keypoints* por *frame* para el sujeto de prueba *Box Logitech*

Box Logitech					
Candidatos		*Keypoints*		Descriptores	
Con *up sampling*	Sin *up sampling*	Con *up sampling*	Sin *up sampling*	Con *up sampling*	Sin *up sampling*
2610	897.9	286.7	108.8	371.8	137.4
Matches		Detección [*s*]		Filtrado [*s*]	
Con *up sampling*	Sin *up sampling*	Con *up sampling*	Sin *up sampling*	Con *up sampling*	Sin *up sampling*
14.75	6.96	0.5661	0.1474	0.03055	0.01046
Cómputo del descriptor [*s*]		*Matching* [*s*]		Cómputo de la homografía [*s*]	
Con *up sampling*	Sin *up sampling*	Con *up sampling*	Sin *up sampling*	Con *up sampling*	Sin *up sampling*
0.1185	0.04621	0.09042	0.009804	0.0006997	0.0002957
Homografías encontradas		Tiempo promedio del cuadro [*s*]			
Con *up sampling*	Sin *up sampling*	Con *up sampling*	Sin *up sampling*		
269/300	216/300	0.8083	0.2257		

Cuadro 5.8: Atributos promedios para la secuencia de 300 imágenes y 10 ejecuciones del sujeto de prueba *Box Logitech*

Por causa del bajo número de *matches*, la homografía (sea correcta o errónea) no se pudo calcular en todos los casos. Solamente se detecto 269 veces cuando hubo *up sampling* y 216 cuando no se hizo.

Los tiempos de procesamiento de este objeto fueron pequeños (ver cuadro 5.8 y figuras 5.19 y 5.20). A partir de los descriptores encontrados puede explicarse esta situación. Si se observa el cuadro, la detección es quien más tiempo emplea y es justificado con base en el *barrido* que se debe hacer sobre las imágenes de las octavas. Luego, si en el proceso de filtrado o refinamiento, son eliminados muchos candidatos, los descriptores a computar también serán menores con respecto del número de candidatos detectados.

Finalmente, los descriptores calculados son empleados en el *matching*. Como se trata de un proceso basado en distancia euclidiana y búsqueda del vecino y el segundo vecino más cercanos, el algoritmo que lo implementa es de orden cuadrático con respecto del número de elementos que haya en las listas de descriptores. En conclusión: a mayor número de descriptores, mayor tiempo de procesamiento.

La prueba del proceso multihilos se hizo con base en los experimentos ya descritos y sólo se midió el tiempo de ejecución, puesto que los parámetros encontrados de candidatos, puntos de interés, descriptores, etc., son constantes y se detectan siempre los mismos. Entonces, para fines de pruebas, lo que importa es medir el tiempo en las ejecuciones *single* y *multi threaded*.

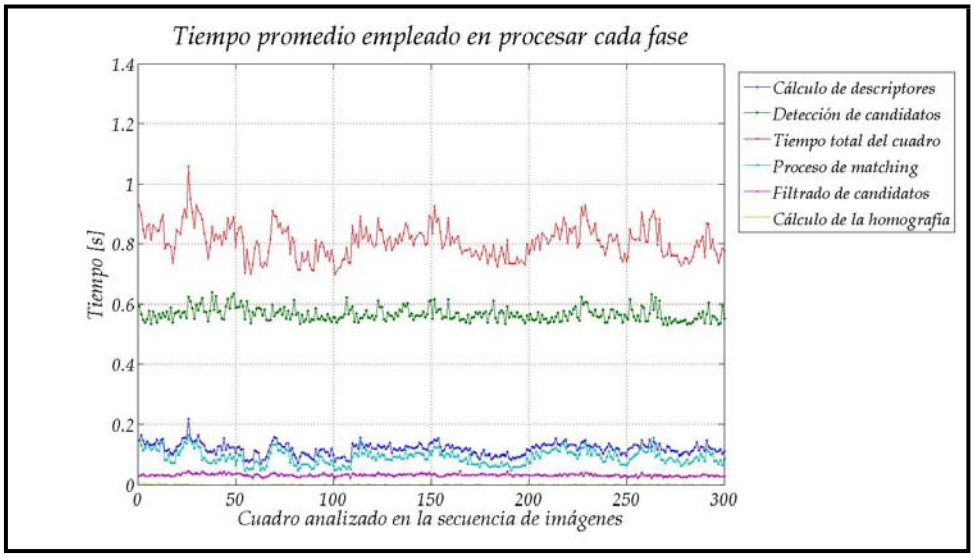


Figura 5.19: Tiempo empleado en calcular cada una de las fases de SIFT para el sujeto de prueba *Box Logitech*, (*up sampling* activo)

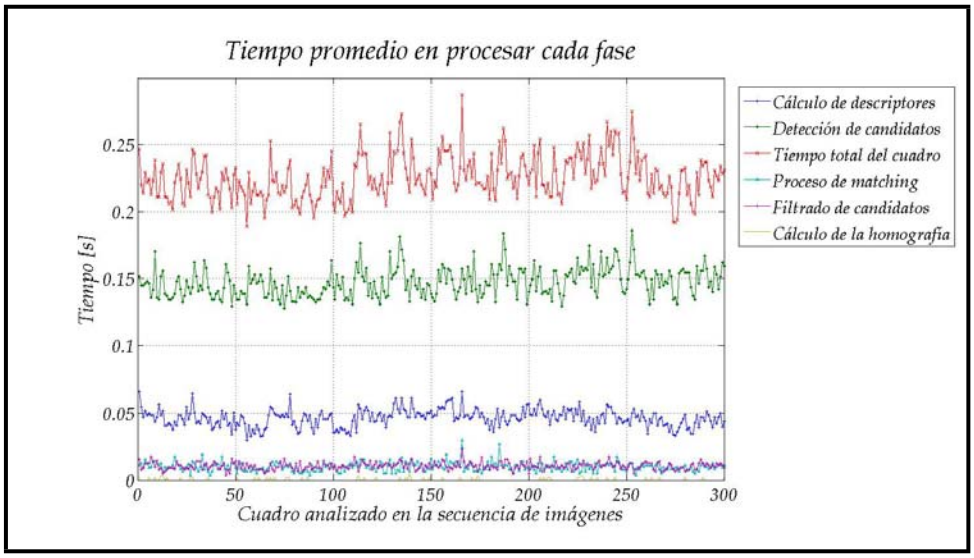


Figura 5.20: Tiempo empleado en calcular cada una de las fases de SIFT para el sujeto de prueba *Box Logitech*, (*up sampling* inactivo)

El cuadro 5.9 muestra los resultados obtenidos mediante el uso de la clase *SIFT*, mientras que el 5.10 los de *SIFT_Mtd*. En los datos contenidos, se puede observar una mejora de unas cuantas décimas de segundo y no de la disminución en tiempo *teórica* que se pretendía alcanzar. Esto es ocasionado por el control interno que debe llevar la clase en lo que se refiere al manejo de *buffers* contenedores de imágenes, especializados por representación *espacio de la escala*, véase la sección 4.8. La copia de la imagen es indispensable para el correcto funcionamiento y ésta consume tiempo.

	Box Extreme		*Box Logitech*	
En [*s*]	*Up sampling*	*No up sampling*	*Up sampling*	*No up sampling*
Ejecución 1	420.375	116.875	257.016	97.812
Ejecución 2	403.422	104.218	252.438	67.609
Ejecución 3	404.859	106.547	250.093	64.266
Ejecución 4	384.531	105.438	245.781	66.515
Ejecución 5	440.938	107.375	240.062	62.516
Ejecución 6	386.953	102.844	245.844	60.063
Ejecución 7	374.688	105.953	240.515	66.468
Ejecución 8	363.391	143.953	241.203	74.375
Ejecución 9	447.187	110.984	234.204	60.766
Ejecución 10	405.871	112.984	235.281	66.125
Promedio [*s*]	403.2215	111.7171	244.2437	68.6515
FPS	0.74400	2.68535	1.22828	4.3698

Cuadro 5.9: Tiempos de ejecución en versión *single threaded*

Teóricamente, el procesamiento de la imagen debería de tardar el máximo de las etapas. Si se analiza esto con respecto de los datos de los cuadros 5.7 y 5.8, se encuentra que los máximos corresponden a la detección de candidatos, con tiempos de 0.6193*s* y 0.1425*s* para la caja *Box Extreme* y de 0.5661*s* y 0.1474*s* para *Box Logitech*.

Entoces, de acuerdo al funcionamiento del un *pipeline*, el tiempo por *frame* debería ser el ya mencionado una vez llena la línea de producción y, para el total de los 300 cuadros, se tendrían tiempos de 185.79*s* y 42.75*s* para *Box Extreme*, y 169.83*s* y 44.22*s* para *Box Logitech*.

Los resultados de las tablas 5.9 y 5.10 muestran que las mediciones promedio estuvieron alejadas en 107 % y 195 % para la caja *Box Extreme* y para *Box Logitech* en un 80 % y 85.88 % respectivamente. Pero si todo ha sido desarrollado con base en las observaciones teóricas y técnicas, ¿por qué no se obtienen los resultados esperados?

	Box Extreme		*Box Logitech*	
En [*s*]	*Up sampling*	*No up sampling*	*Up sampling*	*No up sampling*
Ejecución 1	315.245	82.718	213.032	51.609
Ejecución 2	323.895	82.938	209.025	51.500
Ejecución 3	318.520	83.984	211.032	51.563
Ejecución 4	324.312	83.125	211.765	51.657
Ejecución 5	323.657	84.000	214.937	51.547
Ejecución 6	312.645	82.844	211.063	51.468
Ejecución 7	325.100	84.031	216.031	51.641
Ejecución 8	322.078	83.234	211.172	51.500
Ejecución 9	327.529	83.250	210.564	51.593
Ejecución 10	326.547	83.750	209.828	51.456
Promedio [*s*]	321.953	83.387	211.845	51.553
FPS	0.932	3.598	1.416	5.819

Cuadro 5.10: Tiempos de ejecución en versión *multi threaded*

Primero, es importante notar que, respecto de las medidas de la clase sin *pipeline, SIFT*, sí hay una ganancia en rendimiento, para el sujeto *Box Extreme* se tienen ganancias del 25 % y de 33.98 % en el número de cuadros por segundo, que sigue aun siendo una cifra baja. Para el otro objeto de pruebas, la mejora fue del 15.28 % y del 33.16 %. Aun con la pequeña variación, visualmente se trata de un cambio sustancial, pues no es lo mismo ver 4 cuadros en un segundo, que 6.

Retomando la pregunta formulada en el párrafo anterior, el máximo (obtenido en la fase de detección) es relativamente cercano al tiempo que se necesita para evaluar las correspondencias. Ambas etapas duran casi lo mismo en ejecución y tienen cálculos que son particularmente demandantes de procesador. Además, cuando se utiliza programación con múltiples procesos, la sincronía e inicio de la ejecución no se puede garantizar a ser precisa.

Pueden iniciarse dos procesos uno tras otro y, dependiendo de la carga de procesador y factores del sistema operativo, su inicio real puede ser síncrono e incluso pueden presentarse pequeños o grandes defasamientos. Sin embargo, a juicio personal, el problema principal proviene de que los tiempos de ejecución de las gráficas 5.15, 5.16, 5.19, 5.20, cuyos promedios aparecen en los cuadros 5.7 y 5.8, no conllevan la operación de copia de la imagen en cada representación *espacio de la escala* para el inicio de la detección de candidatos por *frame*.

Otra hipótesis que he considerado tiene que ver directamente con la arquitectura de los procesadores y de como cargan procesos generados por otro proceso. El equipo en el que se probó cuenta con una procesador *Pentium IV* con tecnología *HT* (*Hyper Threading*). Se trata de una solución de Intel para lograr procesamiento multihilos simultáneo (*SMT o Simultaneus Multithreading*). En sus especificaciones se hacen dos notas importantes:

1. Es difícil conocer el beneficio que se puede tener cuando se utilizan múltiples hilos, ya que el rendimiento depende directamente de la carga actual de los diferentes procesos para el(los) procesador(es).

2. Existen casos donde el rendimiento puede verse afectado y sobretodo cuando la carga en los hilos no es apropiada. Un ejemplo son los cálculos científicos que involucran operaciones de punto flotante.

Sin embargo, también se han determinado mejoras en rendimiento de hasta un 30 % con respecto de procesos que se ejecutan en un sólo hilo. Entonces, la hipótesis tiene como raiz ello: la ganancia que se ha comentado produce un aumento en la velocidad de procedimiento que, en el peor caso es de 15.28 % y en el mejor de 33.98 %.

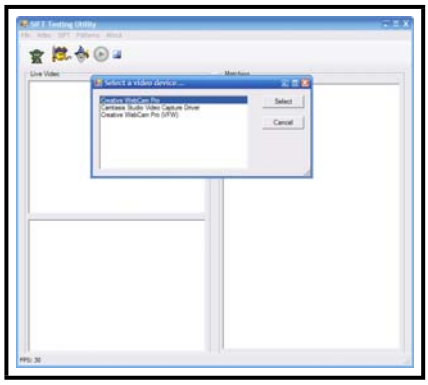
También es importante considerar que las operaciones involucradas en el cálculo de la transformada SIFT requieren de aritmética de punto flotante. Siendo tales operaciones costosas, entonces el rendimiento se verá detrimentado. Otro punto a considerar radica en el hecho de sature al procesador. Cuando se ejecuta un sólo proceso, el procesador le atiende y le otorga el tiempo y recursos necesarios. Cuando se tiene un proceso que crea más procesos, entonces cada uno de ellos debe de ser ejecutado y no interferir entre ellos.

Pero, ¿qué pasa cuando ambos procesos requieren de utilizar cierta parte del procesador? En el caso de los procesos del *software* construido, al menos la primera, tercera y cuarta etapa necesitan hacer cálculos matemáticos intensivos y de punto flotante: cálculo de gaussianas, convoluciones, funciones tirgonométricas, raíces cuadradas, etc. En las primeras fases del procesamiento, antes de que el *pipeline* se llene, el rendimiento es bueno, pero una vez lleno, existe competencia proveniente del resto de los procesos. De este modo, es posible ver que utilizar múltiples procesos es benéfico, pero antes deben hacerse varias consideraciones, inclusive ejecutar el programa en uno, y múltiples procesadores para verificar que el *pipeline* está funcionando.

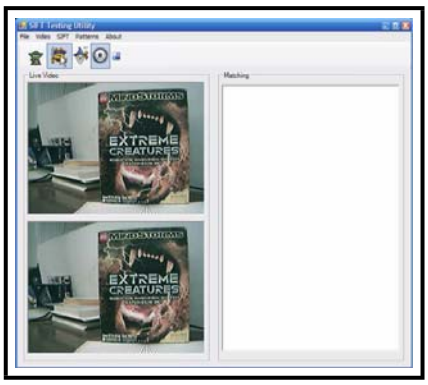
5.3. Análisis en tiempo real

Las pruebas hechas a la interfaz permiten sólo dar un juicio cualitativo más que cuantitativo. En ella no se tienen datos respecto de las características calculadas en cada *frame* procesado sino la representación de la homografía en la imagen analizada, así como de las correspondencias encontradas.

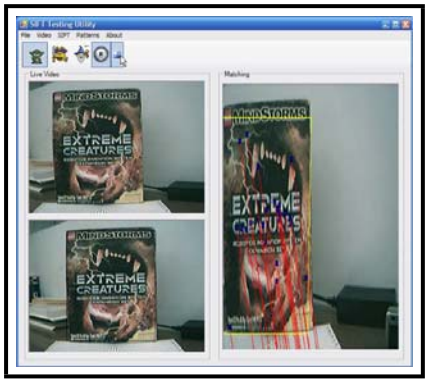
Se trabajó con dos secuencias que han sido capturadas en video. La primera es de la caja *Box Extreme* y tiene una duración de 6.5*min*. El otro video es de la caja *Box Logitech* con duración de 3.16*min*. En ellos se aprecia como se selecciona un dispositivo de video, comienza la reproducción de la adquisición (fig. 5.21(a)), se adquiere un cuadro (fig. 5.21(b)) y se pone en marcha la detección (fig. 5.21(c)).



(a) Selección de un dispositivo de video



(b) Captura de un *frame*



(c) En funcionamiento

Figura 5.21: Interfaz de pruebas para tiempo real

Conclusiones y trabajo futuro

En esta tesis se ha tratado el problema de hacer reconocimiento de objetos y rastreo de los mismos a partir de características naturales, inherentes a sus imágenes capturadas.

Por medio del *software* construido, y los diversos enfoques con que ha sido tratado, se concluye que el objetivo de construir una biblioteca de programación, basada en la extracción de características naturales ha sido cumplida. Esto se logró mediante la implementación del algoritmo SIFT de David Lowe.

Además, no se utilizó código ajeno, sino que la representación de imágenes y filtros fue desarrollada por completo para no tener dependencias externas. Cómo se mencionó también en la introducción, se diseñaron diferentes aproximaciones a la resolución del problema: una procedimental y una orientada a objetos. La *OO* tiene como finalidad hacer más simple el uso del algoritmo y no tener que estar detallando tantos atributos como la creación de la representación *espacio de la escala*, la conversión a escala de grises de la imagen, entre otras.

Basándose en el modelo *OO*, se diseñó una versión *multihilos* para encauzar el procesamiento de imágenes que se adquieren en una localidad de memoria ya determinada y no estar configurando para cada ejecución al objeto. De esta manera se provee de simplicidad en el uso, además de un rendimiento *mejor* debido al *pipeline* implementado.

Otra característica es la versión para .NET que se creó. Considero que es importante pues C# es un lenguaje que está siendo usado ampliamente y que ofrece mucha flexibilidad al programador. Tal es así que se configuró una aplicación de pruebas en tiempo real que captura video, hace el *render* en pantalla del mismo, y permite ver los cuadros procesados mediante el algoritmo SIFT.

Si bien, con esto se cubre el objetivo principal del trabajo, en cuestión de resultados entregados existe aun trabajo por hacer. En el capítulo 5 se han mostrado las evaluaciones realizadas al sistema. En ellas se observa que, a pesar de obtener buenos resultados, aun es necesario continuar con el desarrollo y actualización de las rutinas diseñadas. Primero hay que utilizar las mismas técnicas que utiliza Lowe, como BBF para el *matching*, *clustering* con la transformada de Hough para encontrar conjuntos de *keypoints* e inclusive métodos en ensamblador optimizados para el cálculo de convoluciones y así agilizar aun más el proceso.

El algoritmo descrito por Lowe ha sido seguido al pie de la letra e implementado tal cual sus observaciones y consideraciones experimentales, entonces ¿por qué razón sucede ello? Con base en los resultados obtenidos se observa que no hay invariancia a escala o, mejor dicho, es muy baja la invariancia que se puede obtener a partir de una imagen de referencia.

Las pruebas muestran que para distancias mayores a 10*cm* (variación en escala de aproximadamente 15 %) el reconocimiento se pierde. La hipótesis de la falla se fundamenta en la capacidad para encontrar o determinar a los puntos de interés en cada octava y escala procesada. Si las localidades no conforman un patrón repetible, entonces el reconocimiento será definitivamente pobre. El error puede estar en la detección de puntos o en el refinamiento que se les hace para producir los *keypoints*.

En el caso de invariancia a rotaciones planas, el descriptor computado se comportó bien. De las pruebas hechas, casi no falló ni en las realizadas a la fotografía de Seattle ni a la de la caja *Box Extreme*. Cuando se hicieron las de rotación en profundidad, la falla fue notoria de inmediato para rotaciones mayores a los 50°, tal y como es reportado por Lowe. Por otro lado, cuando las correspondencias entre pares de puntos fueron escasas (como en el caso de la caja *Box Logitech*), la homografía descriptora de la transformación resultó ser errónea. El proceso de su obtención es en parte culpable, pues elige aleatoriamente ternas de puntos y con ellos opera. Si en lugar de esto, se utilizara un proceso basado en probabilidad o, si se trata de búsquedas exhaustivas, del total de combinaciones para los *matches* encontrados. Aun así, no se puede garantizar

una mejor eficacia debido al método de empate utilizado. El uso de métodos más robustos como los *K-d Trees*, BBF o árboles de clasificación, seguro producirían mejora en el desempeño de *matching*.

Un efecto infortunado que se encontró, es el de la equivocación del descriptor cuando se tienen regiones completamente simétricas. El ejemplo de la taza mostró como el descriptor puede ser confundido con otros de regiones iguales. De hecho, como se comentó, el cálculo del descriptor se basa en una ventana de 16x16 pixeles y, para imágenes con simetría, tales regiones serían muy parecidas. Por lo tanto, el descriptor de una región, debería ser el mismo pero bajo una rotación de x grados.

La versión *multithreaded* del algoritmo resultó no ser tan eficiente como se había planteado en un inicio, al menos con respecto de los resultados de la clase *SIFT*. Cómo se comentó en el mismo capítulo 5, el problema puede estar en las copias que se tienen que hacer, para mantener los datos correctos de cada fase de ejecución. Tal operación no está comprendida en la ejecución de la clase *single threaded* y en ello puede estar la pérdida de velocidad en procesamiento.

El *kit* de pruebas hecho en C# no resultó ser complicado en su manufactura, lo que remite a un diseño decente de sus capas inferiores, es decir el código C/C++ encapsulado y portado a código administrado para .NET.

Cómo contribuciones del trabajo de tesis, considero que están:

- La creación de una biblioteca para la adquisición de video desde C++ o C# por medio de *Direct Show*, y que es completamente independiente del resto del código.
- Las clases diseñadas para el manejo de imágenes y las funciones de ciertas operaciones entre ellas.
- Los filtros elaborados y usados en el cómputo de las convoluciones y diferenciación de una imagen.
- El desarrollo de las funciones utilizadas para calcular la transformada SIFT.
- La clase que encapsula a las funciones de SIFT, para un manejo más simple en modos *single* y *multithreaded*.
- Las clases administradas para usarse desde C# de SIFT.

Además, considero importante que el esquema de encapsulado de las funciones, permite trabajar de manera transparente en los níveles superiores, ya que el trabajo de correcciones y actualizaciones procederá siempre de esa capa inferior.

En resumen, de acuerdo con los resultados obtenidos y el trabajo realizado, se concluye que se cumplieron los objetivos aunque es necesario trabajar con mayor énfasis para solucionar los errores cometidos. Es de aquí de donde se parte para determinar el trabajo futuro.

1. Es conveniente contactar a alguna persona bajo el cargo de David Lowe para intercambiar puntos de vista y detalles de la implementación que permitan entender y corregir las fallas detectadas.

2. Con base en las correcciones, elaborar nuevas métricas para determinar el nuevo estado de la biblioteca de programación

3. Hacer pruebas exhaustivas del tiempo de ejecución para cada etapa en la versión *multihilos* para encontrar donde se consume tiempo extra con respecto de los parámetros teóricos.

4. Revisar y replantear los algoritmos de bajo nivel: muestreo de la imagen para hacer *up* y *down sampling*, convoluciones y filtrado para cada uno de los filtros realizados.

5. Incorporar nuevas técnicas para encontrar correspondencias entre pares de puntos de una manera eficiente y con mejor precisión.

6. Implementar la obtención de transformaciones más complejas (proyectivas o afines en 3D) para una mejor descripción y rastreo del movimiento del objeto.

Finalmente, para darle mayor continuidad, habría que vincularlo con alguna tarea real de robótica o de realidad aumentada, completando los módulos requeridos para cada una de ellas, y medir el desempeño que tiene en la aplicación. Además, sería interesante probar la biblioteca para la tarea de creación de mosaicos o panoramas y elaborar una métrica con respecto de Autopano, [Now04], una aplicación dedicada a esos fines y también basada en el algoritmo SIFT.

Apéndice A

Derivadas de una imagen

El contraste en la función de imagen $I(x, y)$ puede presentarse en cualquier dirección. Del cálculo, se sabe que el cambio máximo ocurre a lo largo de la dirección del gradiente de la función, $[\partial f/\partial x, \partial f/\partial y]$.

Una forma de estimar la derivada parcial en el punto $I(x, y)$ sobre x, es calcular. . .

$$\frac{I(x+1, y) - I(x-1, y)}{2}$$

que es el cambio de intensidad entre los vecinos izquierdo y derecho del pixel **x** y divido entre $\Delta x = 2$ pixeles. Debido a que los pixeles puede tener ruido y que el borde puede cruzar en el arreglo de pixeles en cualquier ángulo, es conveniente utilizar 3 diferentes estimaciones en la vecindad del pixel **x**, [SS00]:

$$\frac{\partial f}{\partial x} \equiv f_x \approx \frac{1}{3}\left[\frac{I(x+1,y)-I(x-1,y)}{2} + \frac{I(x+1,y-1)-I(x-1,y-1)}{2} + \frac{I(x+1,y+1)-I(x-1,y+1)}{2}\right] \tag{A.1}$$

Con lo que se obtiene una medida del contraste en la dirección x que es ponderada con las filas anterior y siguiente en dirección y. El contraste en la dirección y se estima de forma similar.

$$\frac{\partial f}{\partial y} \equiv f_y \approx \frac{1}{3}\left[\frac{I(x,y+1)-I(x,y-1)}{2} + \frac{I(x-1,y+1)-I(x-1,y-1)}{2} + \frac{I(x+1,y+1)-I(x+1,y-1)}{2}\right] \tag{A.2}$$

A menudo se descarta la división entre para evitar mayor tiempo de cómputo. Entonces, el gradiente de la imagen se obtiene aplicando los operadores anteriores sobre la vecindad de 8 pixeles del punto **x**.

Para el cálculo de derivadas de orden superior, se toman las derivadas de la vecindad de la localidad **x** en la primera dirección de interés, y con esos datos, se deriva sobre la segunda dirección. Es decir, si se requiere obtener la segunda derivada de la imagen con respecto de x (I_{xx}), en la localidad **x**, primero se diferencían sus 8 vecinos usando la expresión A.1, y con los resultados de cada vecino se vuelve a diferencias con la misma expresión.

En el caso de una derivada cruzada se hace lo mismo. Supóngase que se necesita I_{xy}. Primero se obtienen la derivada en x para cada uno de los 8 vecinos del pixel **x** usando la expresión A.1. Luego, con A.2, se obtiene la derivada en y y se finaliza.

De este modo se pueden obtener las derivadas —de cualquier orden— para una localidad **x** en la imagen.

Apéndice B

Ejemplo de uso de SIFT en C

En el capítulo 4 se presentaron al conjunto de herramientas diseñadas pero no se indicó la forma en la que deben ser usadas. El siguiente bloque de código produce el resultado mostrado en la figura 4.11:

```
#include "../Source/Image/ImageMan.h"          ///Image class
#include "../Source/Image/ImLowLevelOps.h"     // Low level image Ops
#include "../Source/CV/Filter.h"               // Filter class template definition
#include "../Source/SIFT/SIFTFuns.h"           // SIFT routines

using namespace std;                           // STD is used
using namespace ImageRelated;                  // A namespace contains image declaration

extern float** fGSigmas;                       // Needed for storing scale values

void main()
{
   // First, define the two operating images and their gray conversion
   Image*    myImage = NULL, simpleImage = NULL;
   Image*    grayImage = NULL, rotGrayIm = NULL;
```

```
DWORD init, final;              // Used for time measures
GFilter*** gFilters = NULL;     // The gaussian filters created

// We need two descriptor lists: reference and comparing descriptors
Descriptors theDescriptors, simpleDescrs;

// Two more lists containig the matched descriptors
Descriptors matchedDescrs1, matchedDescrs2;

// Four image matrices: 2 scale-space, and 2 DoGs
imMatrix myOctaves = NULL, myDoGs = NULL;
imMatrix rotDoGs = NULL, rotIms = NULL;

// For testing purposes, set up sampling on
bool doUpSampling = true;

// Allocate memory for gaussian filters: 3 octaves, 6 scales, initial sigma = 1.6
GaussianFilterAllocator(&gFilters, &fGSigmas, 3, 3, 1.6f);

// Reference image loading and gray converting
myImage = new Image("../ImLibrary/Seattle640x480.raw", 640, 480, 3);
grayImage = new Image(myImage->getWidth(), myImage->getHeight(), 1);
Convert2Gray(grayImage, myImage);

// Allocate memory for image's scale-space and DoGs: 3 octaves, 6 scales
myOctaves = SIFTBuildScaleSpace( 3, 6, grayImage->getWidth(), grayImage->getHeight(),
                                 doUpSampling );
myDoGs    = SIFTBuildScaleSpace( 3, 5, grayImage->getWidth(), grayImage->getHeight(),
                                 doUpSampling );

// Detect candidates for the first image
SIFTDetectKeyCandidates(&theDescriptors, grayImage, myOctaves, myDoGs, doUpSampling,
                        3, 3, fGSigmas, gFilters);

// Compute the actual keypoints
SIFTRefineAndComputeDesc( &theDescriptors, myDoGs, myOctaves, fGSigmas,
                          iNKeyDescriptors );

// Comparing image loading and gray converting
simpleImage = new Image("../ImLibrary/SeattleRot30_640x480x3.raw", 640, 480, 3);
rotGrayIm = new Image(simpleImage->getWidth(), simpleImage->getHeight(), 1);
Convert2Gray(rotGrayIm, simpleImage);

// Allocate memory for image's scale-space and DoGs: 3 octaves, 6 scales
rotIms   = SIFTBuildScaleSpace( 3, 6, rotGrayIm->getWidth(), rotGrayIm->getHeight(),
                                doUpSampling );
rotDoGs = SIFTBuildScaleSpace( 3, 5, rotGrayIm->getWidth(), rotGrayIm->getHeight(),
                                doUpSampling );

// Detect candidates for the first image
SIFTDetectKeyCandidates(&simpleDescrs, rotGrayIm, rotIms, rotDoGs, doUpSampling, 3, 3,
                        fGSigmas, gFilters);
```

```
// Compute the actual keypoints
SIFTRefineAndComputeDesc( &simpleDescrs, rotDoGs, rotIms, fGSigmas, iNKeyDescriptors );

// Get the image's buffer
ImTy* buff = myImage->getBuffer();
ImTy* buff2 = simpleImage->getBuffer();

// Get memory for two temporary buffers
BYTE* byteBuff = new BYTE[myImage->getBuffSize()];
BYTE* byteBuff2 = new BYTE[simpleImage->getBuffSize()];

// For saving an image, compute the merged size
int mergedW = myImage->getWidth()>simpleImage->getWidth()?
              myImage->getWidth():simpleImage->getWidth();
int mergedH = myImage->getHeight()+simpleImage->getHeight();
int mergedD = myImage->getDepth()>simpleImage->getDepth()?
              myImage->getDepth():simpleImage->getDepth();

// Get memory for the merged image
BYTE* mergedBuff = new BYTE[mergedW*mergedH*mergedD];

// Copy buffers contents
for(int m = 0; m < myImage->getBuffSize(); m++ )
   byteBuff[m] = (BYTE)(255*buff[m]);

for(int m = 0; m < simpleImage->getBuffSize(); m++ )
   byteBuff2[m] = (BYTE)(255*buff2[m]);

// Find Matches
SIFTMatchAndDrawInBuffers(byteBuff, myImage->getWidth(), myImage->getHeight(),
                          myImage->getDepth(), theDescriptors, byteBuff2,
                          simpleImage->getWidth(), simpleImage->getHeight(),
                          simpleImage->getDepth(), simpleDescrs, mergedBuff,
                          &matchedDescrs1, &matchedDescrs2);

// Just takes the merged buffer and puts it in an image for writing to a file
Image* testIm = new Image(mergedBuff, mergedW, mergedH, mergedD);
testIm->WriteIm("test.raw", RAW);
delete testIm;

// Clean memory operations
if( byteBuff ) delete byteBuff;
if( byteBuff2 ) delete byteBuff2;

SIFTDestroyScaleSpace(myOctaves, 3, 6);
SIFTDestroyScaleSpace(myDoGs,  3, 5);

if(myImage)   {   delete myImage;       myImage = NULL; }
if(grayImage)  {  delete grayImage; grayImage= NULL; }
SIFTDestroyDescrList(&theDescriptors);
```

```
    SIFTDestroyScaleSpace(rotIms,  3, 6);
    SIFTDestroyScaleSpace(rotDoGs,    3, 5);

    if(simpleImage)   {  delete simpleImage;  simpleImage = NULL; }
    if(rotGrayIm)   { delete rotGrayIm; rotGrayIm = NULL; }
    SIFTDestroyDescrList(&simpleDescrs);

    GaussianFilterDeallocator(&gFilters, &fGSigmas, 3, 3);
}
```

Apéndice C

Transformaciones afines

A menudo suele tratarse la geometría y propiedades de los vectores, pero ¿qué hay de los puntos? El mundo puede concebirse como un espacio de puntos o ubicaciones, entonces ¿cómo pueden relacionarse estos puntos con los vectores que comúnmente se tratan? La intuición dicta la idea de definir un vector a partir de una pareja de puntos, iniciando en uno y yendo hacia el otro.

Formalmente, un espacio afín $\mathcal{A}$ consiste de un conjunto de puntos $\mathcal{P}$ y un espacio vectorial $\mathcal{V}$. La dimensión n de $\mathcal{A}$ es la dimensión de $\mathcal{V}$. Se referirá al conjunto de puntos por $\mathcal{A}.\mathcal{P}$ y al de vectores por $\mathcal{A}.\mathcal{V}$.

La relación entre puntos y vectores está definida por el axioma que define la substracción entre pares de puntos:

i. $\forall P, Q \in \mathcal{A}.\mathcal{P}, \exists$ un vector único $\vec{v} \in \mathcal{A}.\mathcal{V}$ tal que $\vec{v} = P - Q$.

ii. $\forall Q \in \mathcal{A}.\mathcal{P}, \forall \vec{v} \in \mathcal{A}.\mathcal{V}, \exists$ un punto único P tal que $P - Q = \vec{v}$.

iii. $\forall P, Q, R \in \mathcal{A}.\mathcal{P}, (P - Q) + (Q - R) = P - R$

Otro axioma importante es el conocido como de *coordenadas* establece que $\forall P \in \mathcal{A}.\mathcal{P}, 1 \bullet P = P$ y $0 \bullet P = \vec{0}$, que simplemente establece que la multiplicación de 1

por el punto, da el punto en sí mismo y que la multiplicación por 0 resulta en el vector cero en $\mathcal{A}.\mathcal{V}$.

Algunas identidades de utilidad son [SE53]:

i. $Q - Q = \vec{0}$
ii. $R - Q = -(Q - R)$
iii. $\vec{v} + (Q - R) = (Q + \vec{v}) - R$
iv. $Q - (R + \vec{v}) = (Q - R) - \vec{v}$
v. $P = Q + (P - Q)$
vi. $(Q + \vec{v}) - (R + \vec{w}) = (Q - R) + (\vec{v} - \vec{w})$

Transformación afín

Una transformación afín es una transformación linean no singular seguida de una traslación. En forma matricial tiene la siguiente representación:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_x \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{C.1}$$

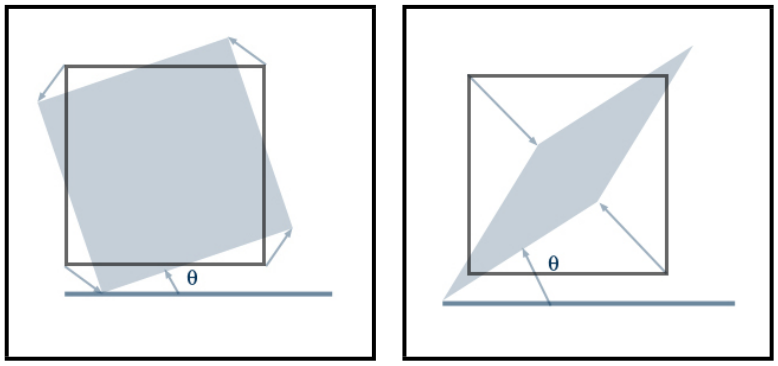
o en su forma de bloque

$$\mathrm{x}' = \mathbf{H_A}\mathrm{x} = \begin{bmatrix} \mathbf{A} & \mathrm{t} \\ 0^T & 1 \end{bmatrix} \mathrm{x} \tag{C.2}$$

donde **A** es una matriz no singular de 2x2. Una transformación afín plana tiene 6 grados de libertad, que corresponden a los 6 elementos de la matriz. La transformación puede ser calculada a partir de 3 puntos en correspondencia. Una forma útil de de entender los efectos geométricos de la componente lineal **A** de una transformación afín, es como la composición de dos transformaciones fundamentales, rotaciones y escalamientos no isotrópicos. La matriz afín **A** puede descomponerse por $\mathbf{A} = \mathbf{R}(\theta)\mathbf{R}(-\phi)\mathbf{D}\mathbf{R}(\phi)$ donde $\mathbf{R}(\theta)$ y $\mathbf{R}(\phi)$ son rotaciones por θ y ϕ respectivamente, y **D** es una matriz diagonal:

$$\mathbf{D} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \mathrm{x} \tag{C.3}$$

La matriz afín **A** es la concatenación de una rotación de ϕ, un escalamiento por λ_1 y λ_2, una rotación de *regreso* por ϕ y la rotación final por θ. La esencia de la *afinidad* radica en el escalamiento no isotrópico a lo largo de los dos ejes ortogonales y con orientación de cierto ángulo. Véase la figura C.1.



(a) Rotación (b) Deformación

Figura C.1: Distorsiones causadas por las transformaciones afines

Como una transformación afín incluye el escalamiento mencionado, las proporciones de longitud y de ángulos no son conservadas, sin embargo, existen tres invariantes:

1. **Rectas paralelas**. Considérense dos rectas paralelas cuya intersección está en el infinito. Bajo transformaciones afines, este punto es mapeado a otro punto en el infinito. Consecuentemente, las líneas paralelas son mapeadas a rectas que aun se intersectan en el infinito, y por lo tanto, son paralelas incluso después de la transformación.

2. **Proporción de las longitudes en segmentos de recta paralelos**. La longitud de escalamiento de un segmento recta, depende solamente del ángulo entre la dirección de la línea y las direcciones de escalamiento. Supóngase que la línea tiene un ángulo α con respecto de la dirección de escalamiento x, entonces la magnitud de escalamiento es $\sqrt{\lambda_1^2 \cos^2 \alpha + \lambda_2^2 \sin^2 \alpha}$. Este escalamiento es común a todas las lineas con la misma dirección.

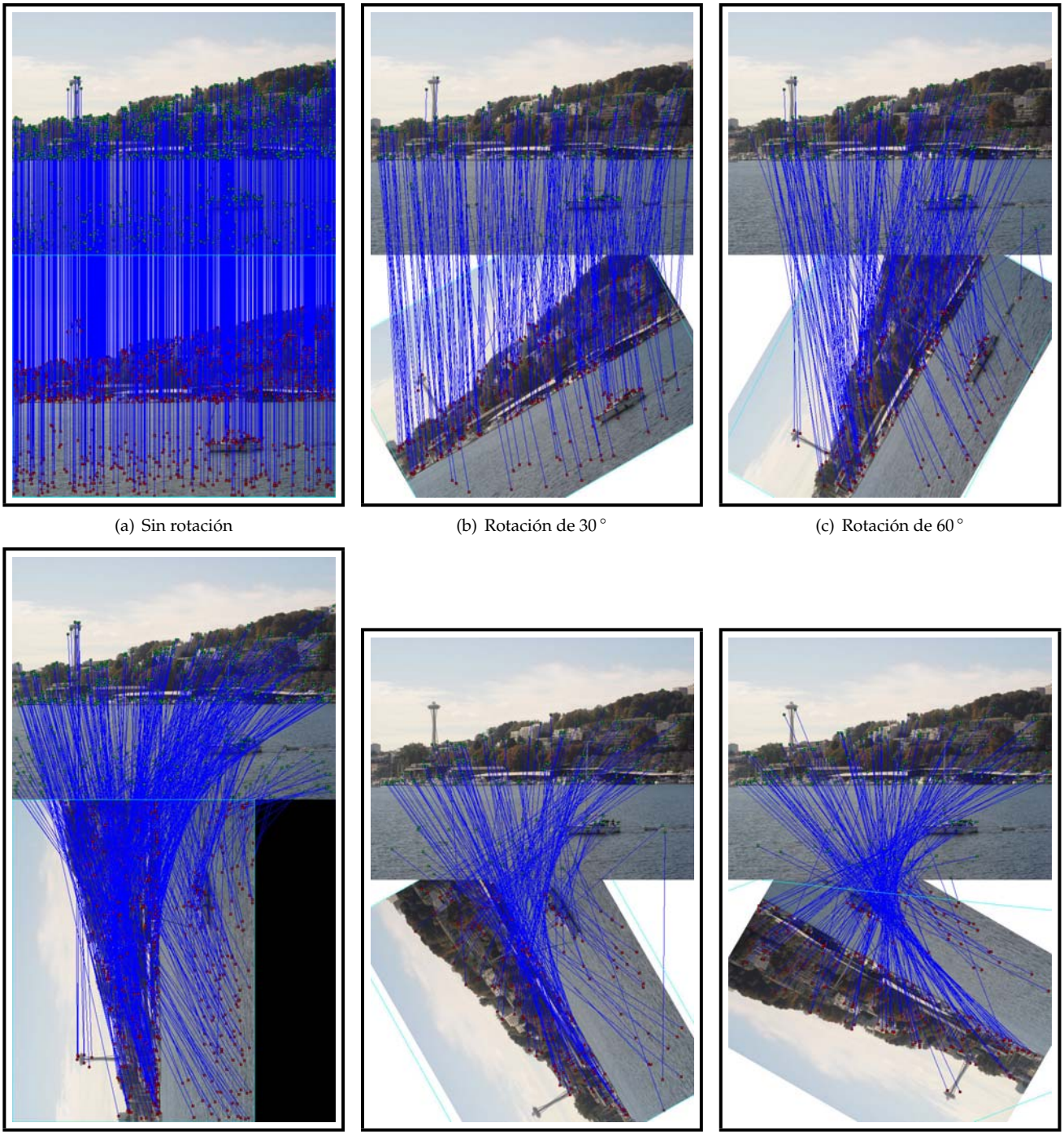
3. **Proporción de áreas**. Esta invariancia puede deducirse directamente a partir de la composición ya expresada. Las rotaciones y las traslaciones no afectan

la figura, de forma que los únicos escalamientos son λ_1 y λ_2. El efecto es que el área se escala por $\lambda_1\lambda_2$ que es igual al determinante de $\mathbf{A}$. Por lo tanto, el área de cualquier forma está escalada por tal cifra y el escalamiento conserva la proporción de áreas.

Apéndice D

Imágenes de prueba

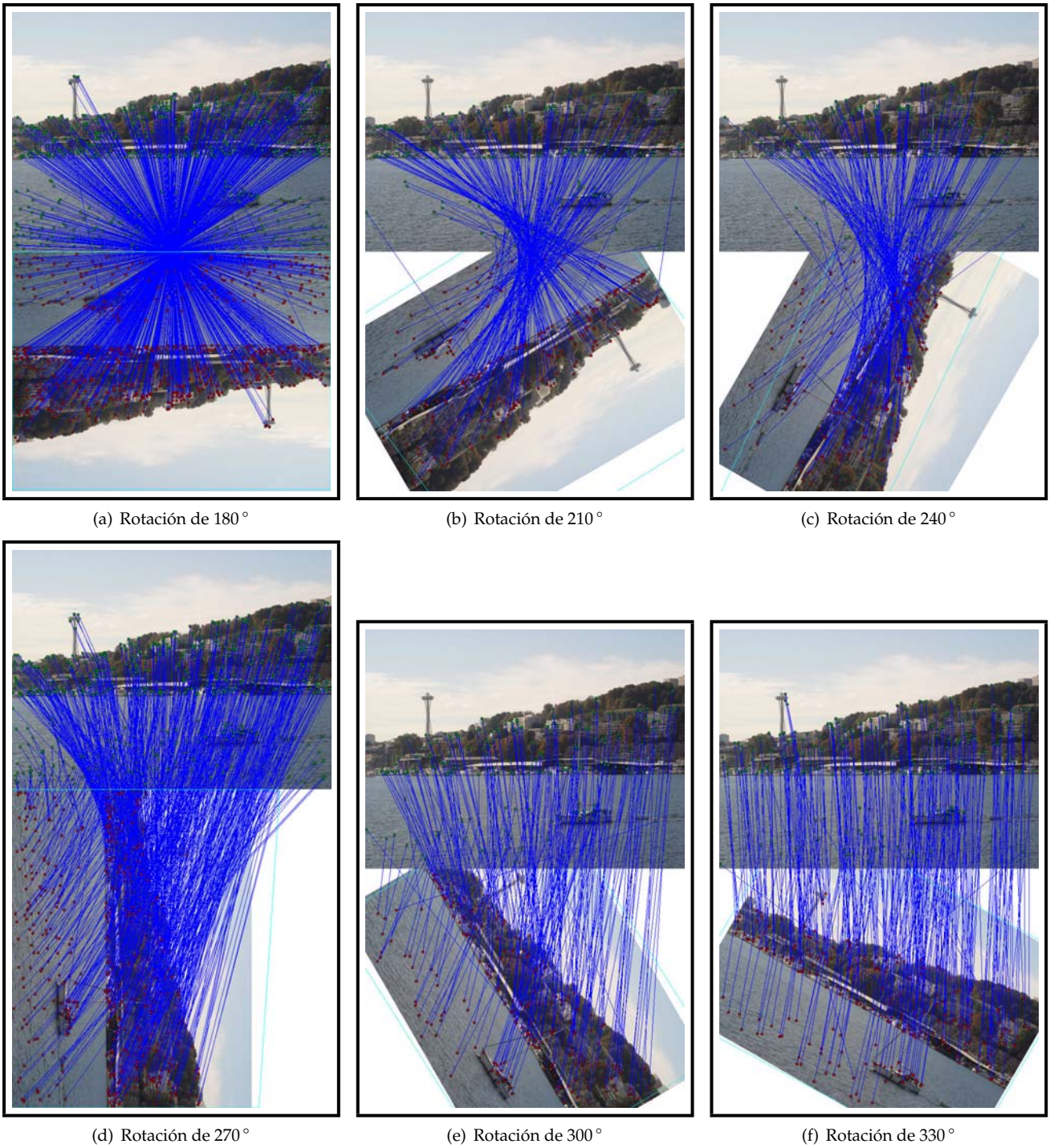
En este apéndice se muestran algunas de las imágenes generadas durante la fase de pruebas. El orden en el que han sido colocadas, corresponde al orden en el que fue presentado cada experimento. Primero se comienza con las rotaciones de la imagen de Seattle y la *ROI* de la misma, luego se presentan aquellas que muestran las pruebas de rotación para la caja *Box Extreme*, pues las pruebas de escala ya fueron incluídas en el texto del capítulo 5. Por último se colocan las imágenes que ilustran las pruebas de rotación de la taza. El par de muñecos de peluche queda descartado al igual que el del panorama, puesto que también fueron mostrados en el capítulo 5.



(a) Sin rotación (b) Rotación de 30 ° (c) Rotación de 60 °

(d) Rotación de 90 ° (e) Rotación de 120 ° (f) Rotación de 150 °

Figura D.1: Rotaciones para la imagen de Seattle (1)



(a) Rotación de 180 °

(b) Rotación de 210 °

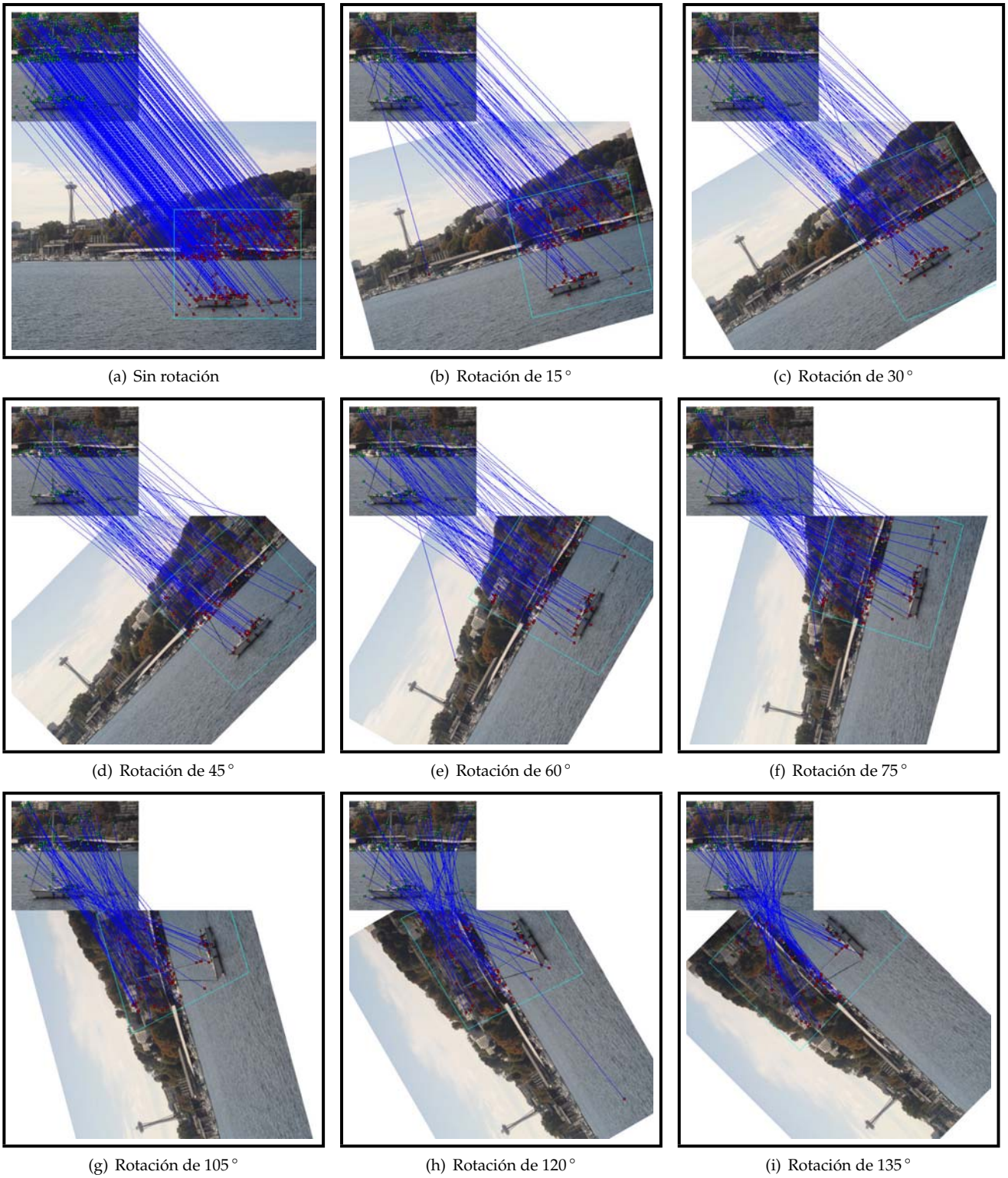
(c) Rotación de 240 °

(d) Rotación de 270 °

(e) Rotación de 300 °

(f) Rotación de 330 °

Figura D.2: Rotaciones para la imagen de Seattle (2)

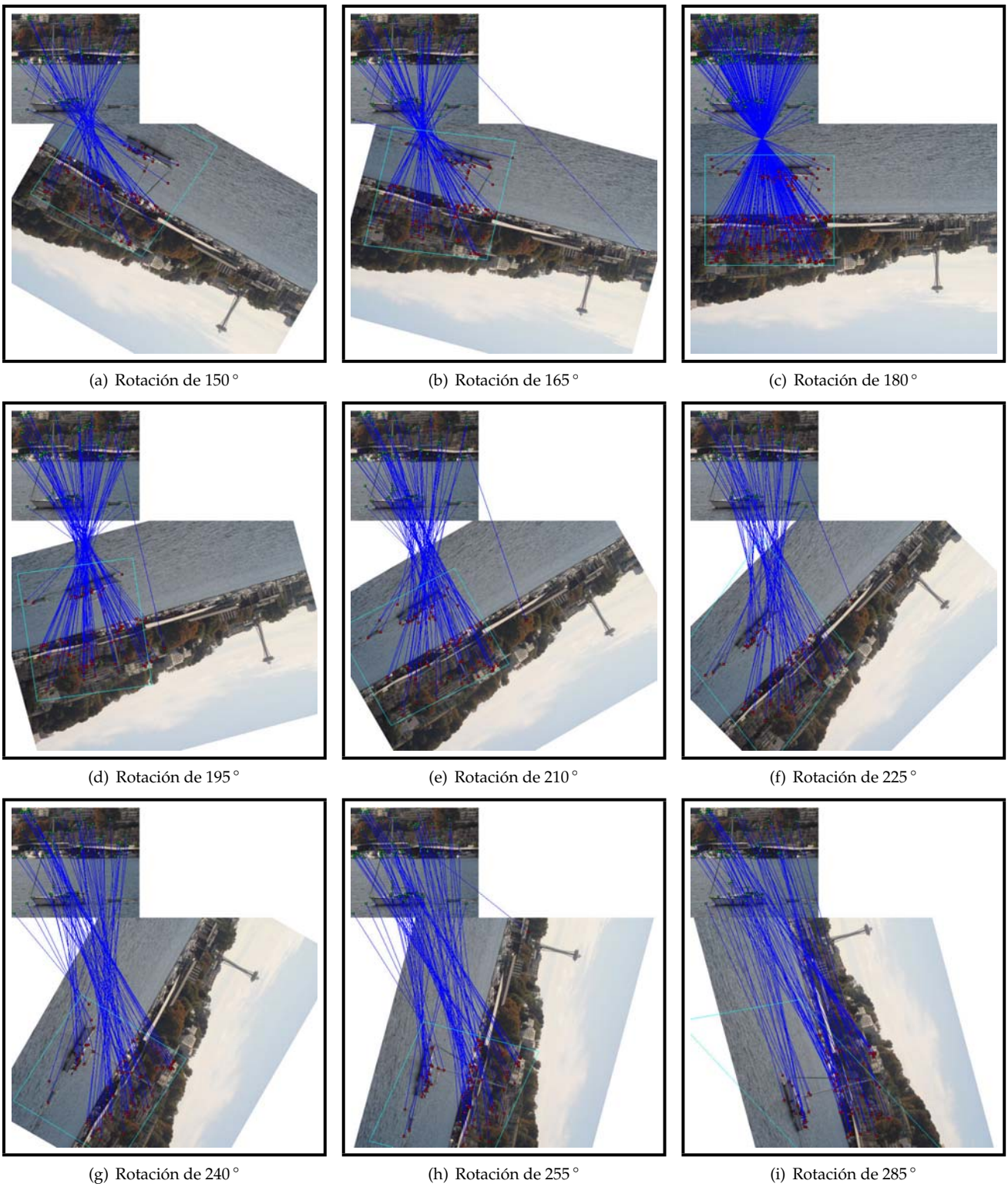


(a) Sin rotación (b) Rotación de 15° (c) Rotación de 30°

(d) Rotación de 45° (e) Rotación de 60° (f) Rotación de 75°

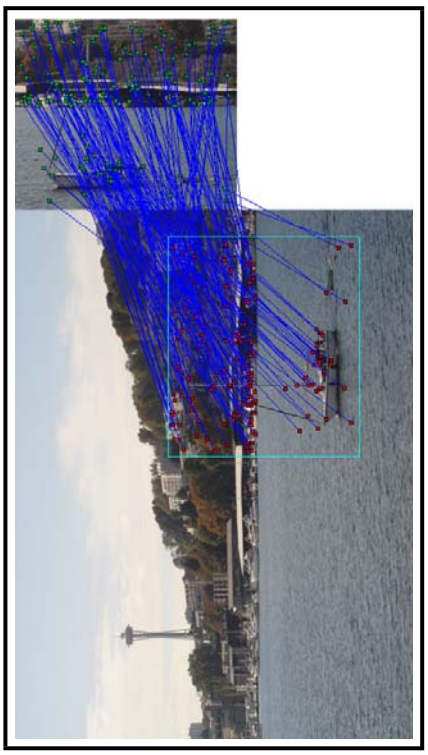
(g) Rotación de 105° (h) Rotación de 120° (i) Rotación de 135°

Figura D.3: Rotaciones para la imagen de Seattle y su *ROI* (1)

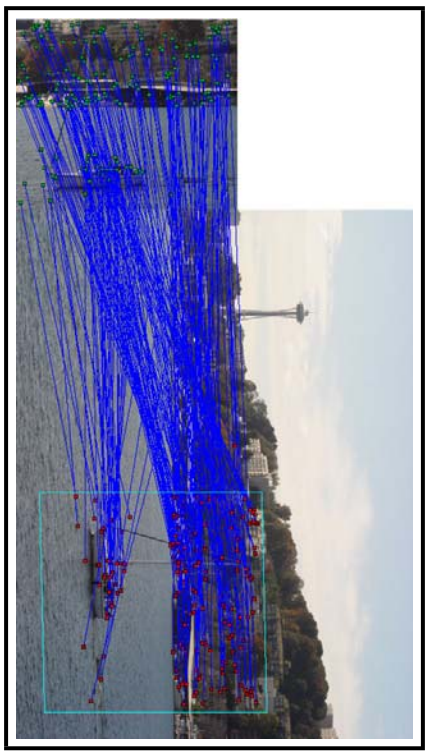


(a) Rotación de 150 °
(b) Rotación de 165 °
(c) Rotación de 180 °
(d) Rotación de 195 °
(e) Rotación de 210 °
(f) Rotación de 225 °
(g) Rotación de 240 °
(h) Rotación de 255 °
(i) Rotación de 285 °

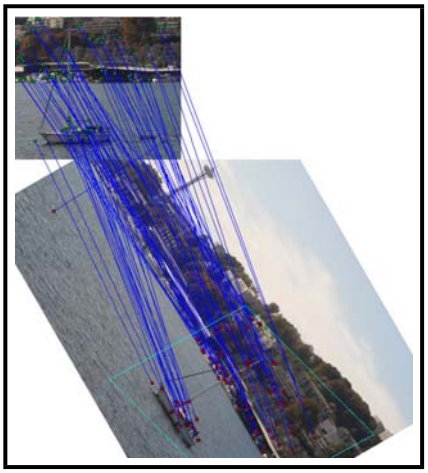
Figura D.4: Rotaciones para la imagen de Seattle y su *ROI* (2)



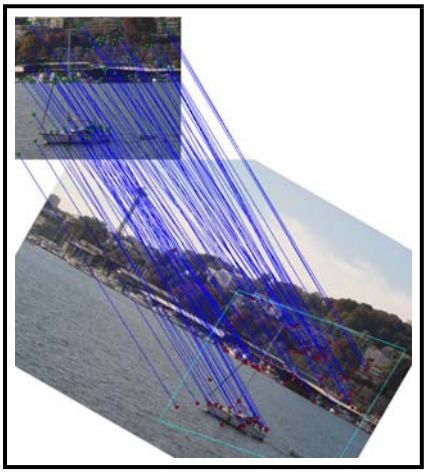
(a) Rotación de 90 °



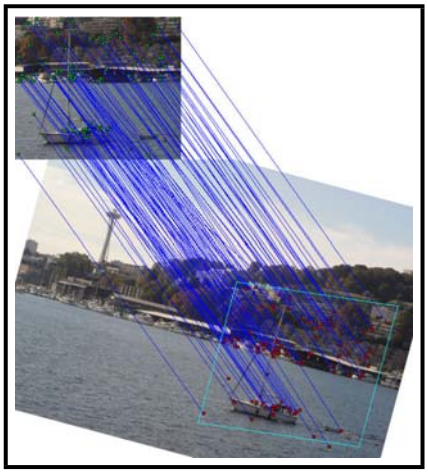
(b) Rotación de 270 °



(c) Rotación de 300 °

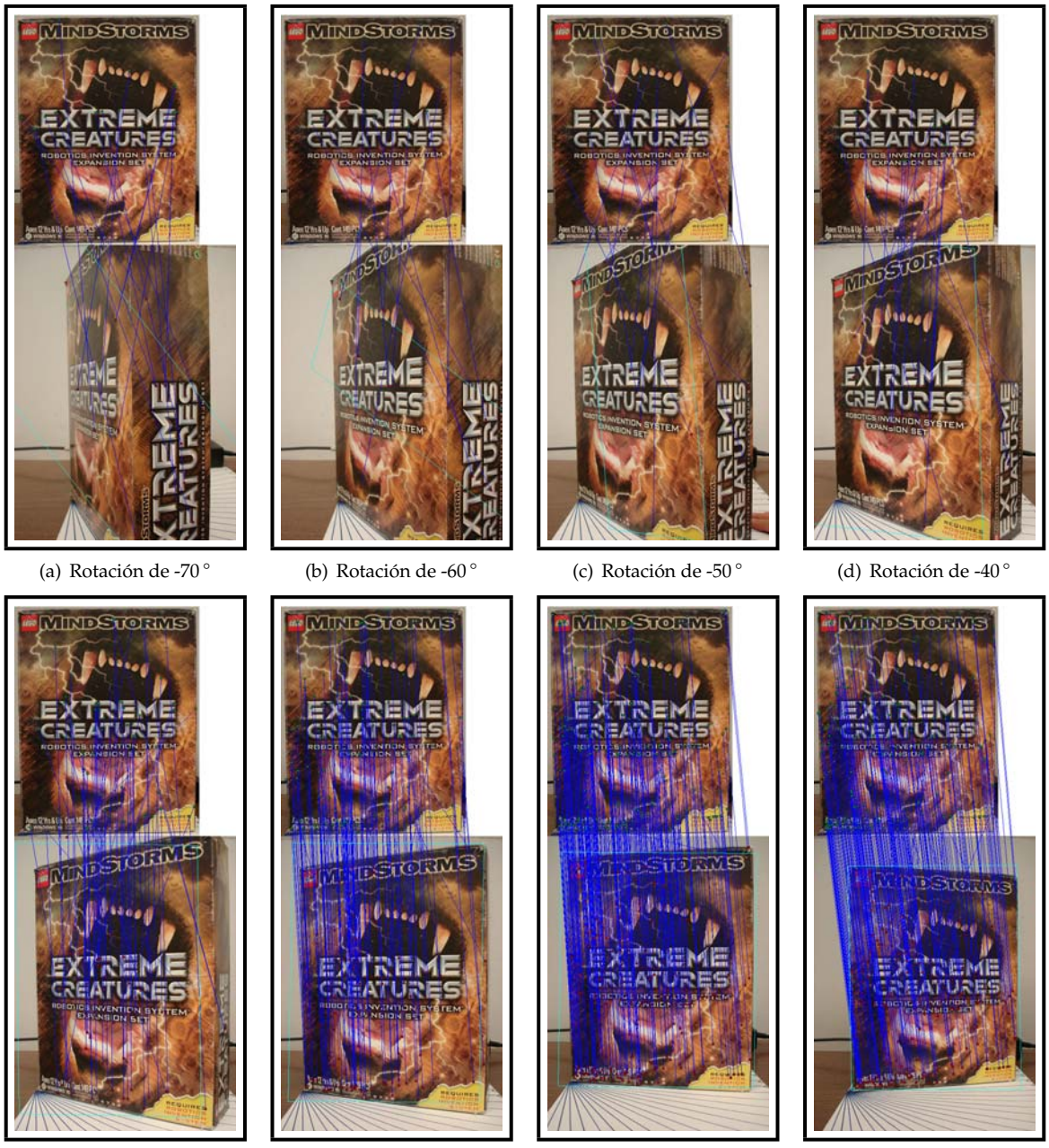


(d) Rotación de 330 °



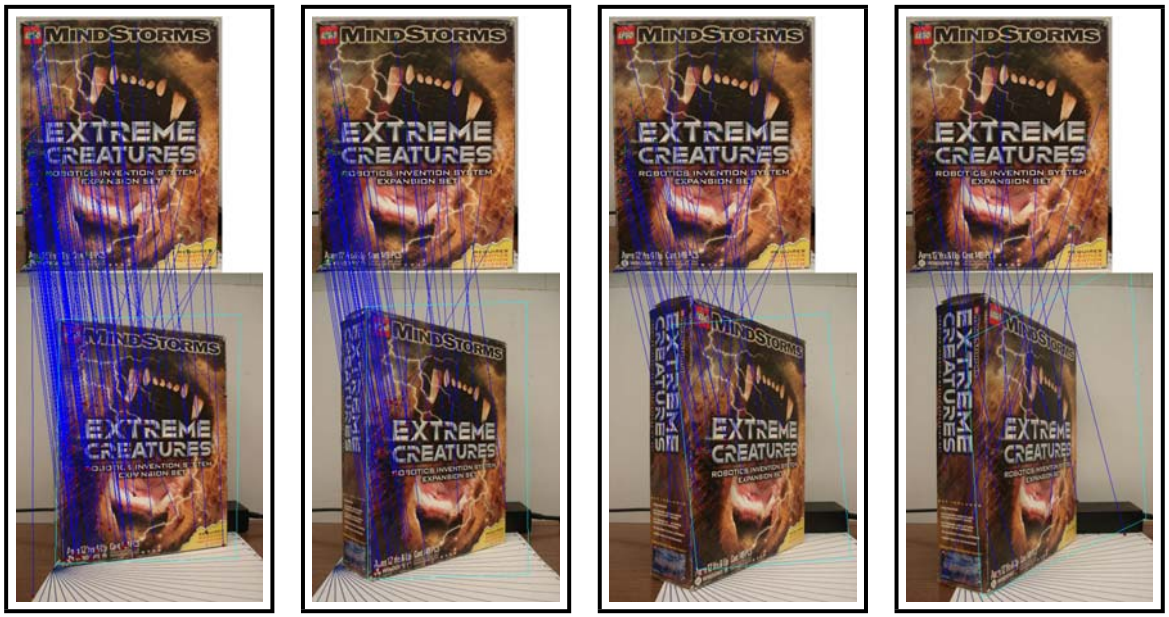
(e) Rotación de 345 °

Figura D.5: Rotaciones para la imagen de Seattle y su *ROI* (3)



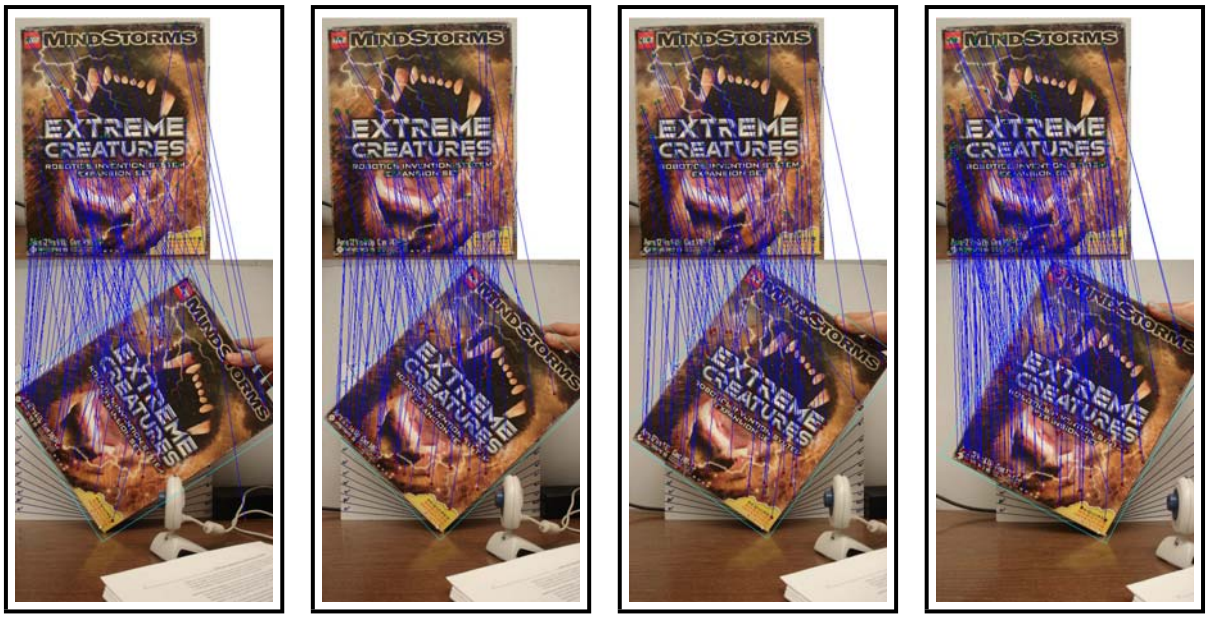
(a) Rotación de -70 °
(b) Rotación de -60 °
(c) Rotación de -50 °
(d) Rotación de -40 °
(e) Rotación de -30 °
(f) Rotación de -20 °
(g) Rotación de -10 °
(h) Rotación de 10 °

Figura D.6: Rotaciones en profundidad para la caja *Box Extreme* (1)



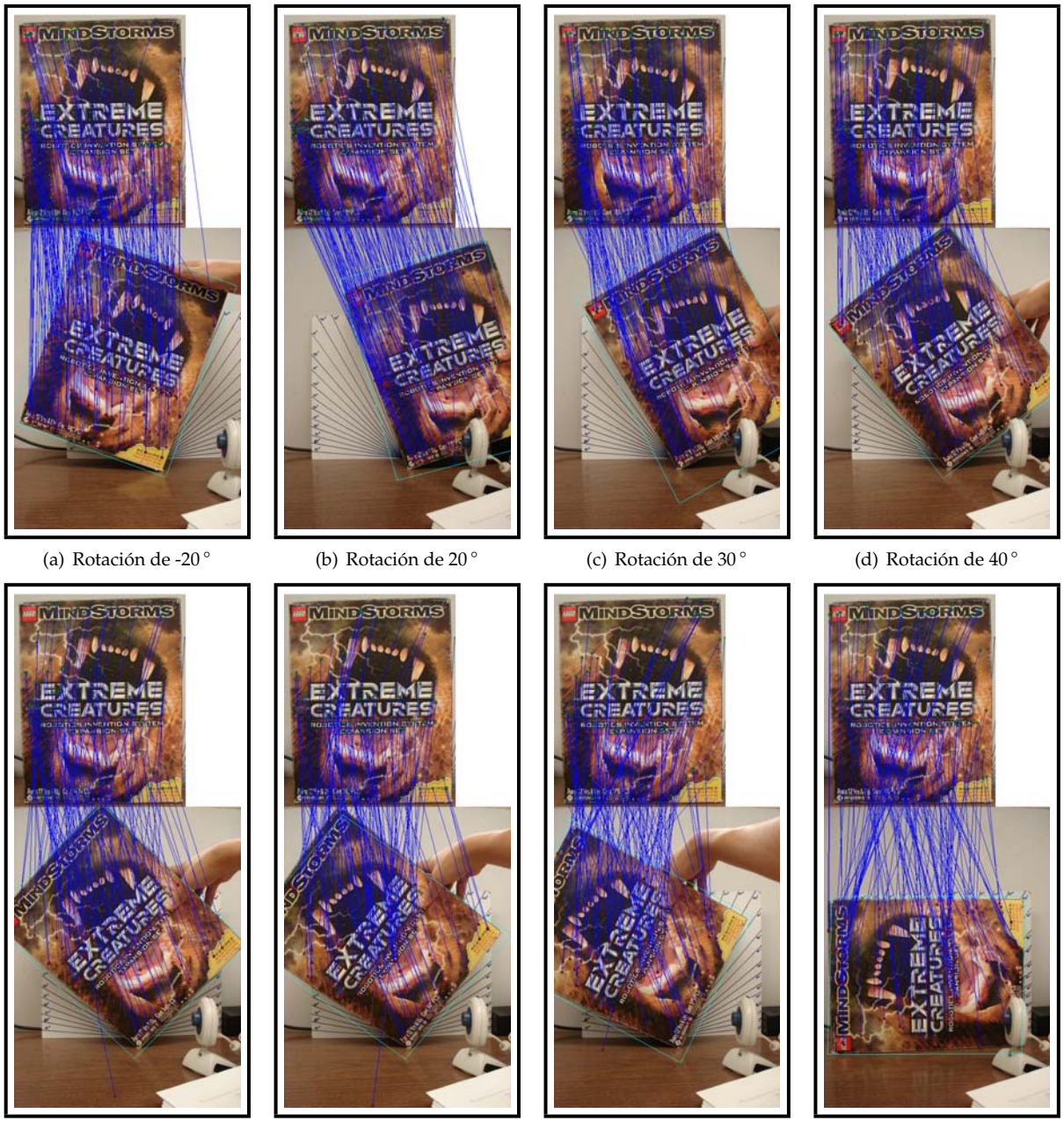
(a) Rotación de 20 ° (b) Rotación de 40 ° (c) Rotación de 50 ° (d) Rotación de 60 °

Figura D.7: Rotaciones en profundidad para la caja *Box Extreme* (2)



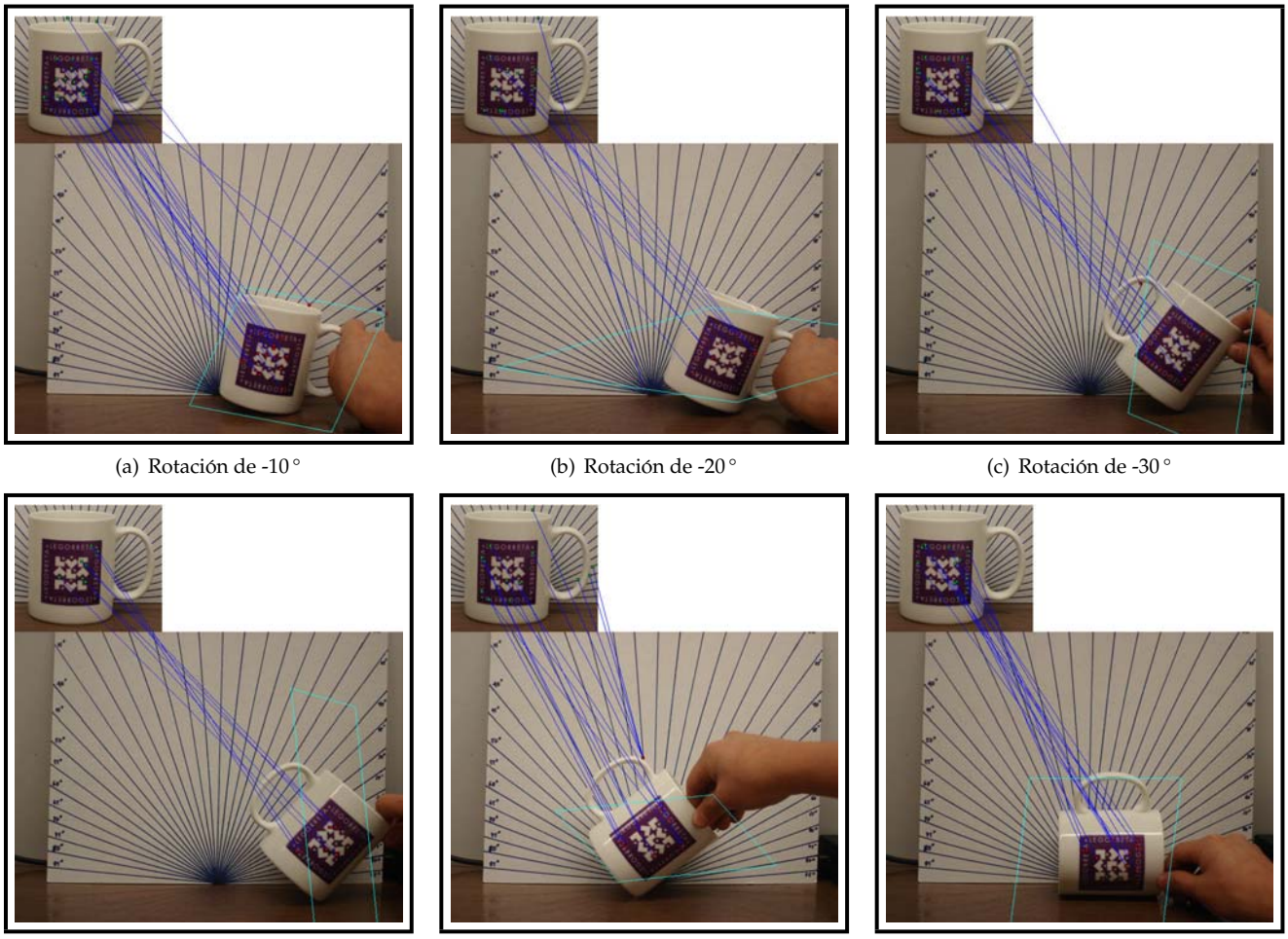
(a) Rotación de -60 ° (b) Rotación de -50 ° (c) Rotación de -40 ° (d) Rotación de -30 °

Figura D.8: Rotaciones en el plano para la caja *Box Extreme* (1)



(a) Rotación de -20 °
(b) Rotación de 20 °
(c) Rotación de 30 °
(d) Rotación de 40 °
(e) Rotación de 50 °
(f) Rotación de 60 °
(g) Rotación de 70 °
(h) Rotación de 90 °

Figura D.9: Rotaciones en el plano para la caja *Box Extreme* (2)



(a) Rotación de -10 °
(b) Rotación de -20 °
(c) Rotación de -30 °
(d) Rotación de -40 °
(e) Rotación de -50 °
(f) Rotación de -90 °

Figura D.10: Rotaciones para la taza

Notación

A continuación se presenta la notación utilizada a lo largo de este trabajo. En el cuadro se presentan operaciones, operadores y señalizaciones que aparecen en las ecuaciones de cada uno de los capítulos.

$\mathbf{x}$, (x, y)	Pixel de la imagen o localidad de interés.
$\overline{\mathbf{x}}$	Vector de incógnitas para un sistema de ecuaciones expresado en forma matricial.
$\vec{x}$	Vector $\vec{x}$ de un espacio vectorial $\mathcal{V}$, en este trabajo se referirá más a los vectores de $\mathbb{R}^2$ y $\mathbb{R}^3$.
$\mathbf{M}$, $\mathbf{A}$, $\mathbf{H}$, etc.	En su mayoría, toda variable/símbolo en mayúsculas y *negritas* refiere a una matriz.
$\mathbf{H}$	Matriz Hessiana, véase la sección 3.1.2 y las ecuaciones 3.6 y 3.16.
H	Homografía de una imagen, sección 4.6 y ecuación 4.14. Hubo de escribirse de este modo, pues la matriz $\mathbf{H}$ define a la matriz Hessiana.
I, $I(\mathbf{x})$, $I(x, y)$	Imagen analizada.
σ	*Escala* de la gaussiana utilizada.
$G(\mathbf{x}, \sigma)$, $G(x, y, \sigma)$	Función gaussiana en 2D.
$L(\mathbf{x}, \sigma)$, $L(x, y, \sigma)$	Convolución de una imagen con un *kernel* gaussiano 2D.
$D(\mathbf{x}, \sigma)$, $D(x, y, \sigma)$	Diferencia de gaussianas: $D(\mathbf{x}, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$.
I_x	Primera derivada de la imagen I con respecto de x.

I_y	Primera derivada de la imagen I con respecto de y.
I_{xx}	Segunda derivada de la imagen I con respecto de x.
I_{yy}	Segunda derivada de la imagen I con respecto de y.
I_{xy}	Segunda derivada de la imagen I con respecto de x y de y.
AR.	Realidad aumentada, *Augmented Reality*.
CV.	Visión por computadora, *Computer Vision*.
DoG.	Diferencia de Gaussianas.
GPS.	Sistema de posicionamiento global, *Global Positioning System*.
LoG.	Laplaciano de Gaussianas.
PDI.	Procesamiento Digital de Imágenes.
RANSAC.	*RANdom SAmple Consensus*. Algoritmo útil para estimar los parámetros de un modelo matemático a partir de un conjunto de datos adquiridos o calculados que contienen errores.
ROI.	Región de interés, *Region Of Interest*.
SIFT.	*Scale Invariant Feature Transformation*. Algoritmo desarrollado por David Lowe para la extracción de características intrínsecas a la imagen que son invariantes a rotación y escala.
SURF.	*Speeded Up Robust Features*. Se trata de otra técnica para la obtención de descriptores locales invariantes a rotación y escala.
VR.	Realidad Virtual, *Virtual Reality*.

Bibliografía

[AB94] R. Azuma and G. Bishop. Improved Static and Dynamic Registration in an Optical See–through HMD. *Proceedings of SIGGRAPH94*, pages 197–203, Julio de 1994.

[Ada04] M. Adams. Top Ten Technologies: #3 Augmented Reality. News Target. Sitio de internet en línea. Disponible en `http://www.newstarget.com/001333.html`. Accedido el 9 de septiembre de 2006, julio de 2004.

[AF01] R. Azuma and S. Feiner. Recent Advances in Augmented Reality. *Computers and Graphics*, Noviembre de 2001.

[Azu97] R. Azuma. A survey of Augmented Reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, Agosto de 1997.

[Bau00] A. Baumberg. Reliable Feature Matching Across Widely Separated Views. *Conference on Computer Vision and Pattern Recognition*, pages 774–781, 2000.

[Bil02] M. Billinghurst. Augmented Reality in Education. New Horizons for Learning, Seattle, WA, Estados Unidos. Sitio de internet en línea. Disponible en `http://www.newhorizons.org`. Accedido el 9 de septiembre de 2006, 2002.

[BW96] D. E. Breen and R. T. Whitaker. Interactive Occlusion and Automatic Object Placement for Augmented Reality. In *Eurographics '96 Proceedings*, pages 11–22, Poitiers, Francia, Agosto de 1996. Elsevier Science Publishers.

[Car07] G. Carrera Mendoza. *Seguimiento de objetos usando características locales invariantes a escala y rotación*. Tesis de Maestría. Posgrado en Ciencia e Ingeniería de la Computación. Ciudad Universitaria, México D.F., marzo de 2007.

[CC02] K. W. Chia and A. Cheok. Online 6 DOF Augmented Reality Registration from Natural Features. In *Proceedings of the IEEE and ACM ISMAR'02*, pages 305–313, Septiembre de 2002.

[CP84] J. L. Crowley and A. C. Parker. A Representation for Shape Based on Peaks and Ridges in the Difference of Low-pass Transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(2):156–170, 1984.

[Dr.02] Dr. Dobb's Journal. *Image Scaling with Bresenham*. Sitio de internet en línea. Disponible en `http://en.wikipedia.org/wiki/Computer_Vision`. Accedido el 5 de agosto de 2006, Abril 10 de 2002.

[EI97] S. Edelman and N. Intrator. *Complex Cells and Object Recognition*. Documento no publicado, Sitio de internet en línea. Disponible en `http://kybele.psych.cornell.edu/~edelman/archive.html`. Accedido el 10 de febrero de 2007, 1997.

[Flo93] L. Florack. *The syntactical Structure of Scalar Images*. Tesis de doctorado, Universidad Utrecht, Noviembre de 1993.

[FM93] S. Feiner and B. MacIntyre. Windows on the World: 2D Windows for 3D Augmented Reality. In *UIST '93: Proceedings of the 6th annual ACM symposium on User interface software and technology*, pages 145–155, New York, NY, Estados Unidos, 1993. ACM Press.

[Fuh98] A. Fuhrmann. Collaborative Visualization in Augmented Reality. *IEEE Computer Graphics and Applications*, 18(4):54–59, julio-agosto de 1998.

[GB06] M. Grabner and H. Bischof. Fast Approximated SIFT. *Asian Conference on Computer Vision*, pages 918–927, 2006.

[GLP94] W. Grimson and T. Lozano-Pérez. An Automatic Registration Method for Frameless Stereotaxy, Image, Guided Surgery and Enhanced Reality Visualization. *IEEE Conf. Computer Vision and Pattern Recognition*, pages 430–436, Junio de 1994.

[HIT00a] HITLab. ARToolKit Coordinate Systems. Sitio de internet en línea. Disponible en `http://www.hitl.washington.edu/artoolkit/documentation/cs.htm`. Accedido el 14 de septiembre de 2006, 2000.

[HIT00b] HITLab. ARToolKit Documentation (Computer Vision Algorithm). Sitio de internet en línea. Disponible en `http://www.hitl.washington.edu/artoolkit/`. Accedido el 14 de septiembre de 2006, 2000.

[HS88] C. J. Harris and M. Stephens. A Combined Corner and Edge Detector. *Proceedings of the 4th Alvey Vision Conference*, pages 147–151, 1988.

[JH97] A. Johnson and M. Hebert. Object Recognition by Matching Oriented Points. *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 684–689, Junio de 1997.

[JH99] A. Johnson and M. Hebert. Using Spin Images for Efficient Object Recognition in Cluttered 3D Scenes. *IEEE Transactions on Patter Analysis and Machine Inteligence*, 21(5):433–449, 1999.

[Jim07] D. Jimison. *Augmented Reality Prototyping for Entertainment*. Entertainment Technology Center, Carnegie Melon University. Sitio de internet en línea. Disponible en `http://campar.in.tum.de/Chair/ResearchGroupCamp`. Accedido el 9 de septiembre de 2006, 2005–2007.

[JJ04] J. Jiménez Juárez. *Visualización de objetos en 3D utilizando Realidad Aumentada y* Shaders *programables*. Tesis de Licenciatura. Ciudad Universitaria, México D.F., junio de 2004.

[JS04] F. Jurie and C. Schmid. Scale-Invariant Shape Features for Recognition of Object Categories. *Computer Vision and Pattern Recognition*, 2:90–96, 2004.

[KB99] H. Kato and M. Billinghurst. Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System. *Proceedings of IWAR*, 1999.

[KB00] H. Kato and M. Billinghurst. ARToolkit version 2.33. HITLab, Seattle, Universidad de Washington, 2000.

[KB01] T. Kadir and M. Brady. Scale, Saliency and Image Description. *International Journal of Computer Vision*, 45(2):83–105, 2001.

[KK97] D. Koller and G. Klinker. Real-time Vision-Based Camera Tracking for Augmented Reality Applications. In Daniel Thalmann, editor, *ACM Symposium on Virtual Reality Software and Technology*, Nueva York, Estados Unidos, 1997. ACM Press.

[Koe84] J. J. Koenderink. The Structure of images. *Biological Cybernetics*, 10(50):363–396, 1984.

[KS05] Y. Ke and R Sukthankar. PCA-SIFT: A more disctinctive representation for local image descriptors. *Computer Vision and Pattern Recognition*, pages 506–513, 2005.

[LF04a] V. Lepetit and P. Fua. *Towards Recognizing Feature Points using Classification Trees*. Technical Report IC/2004/74, Ecole Polytechnique Fédéral de Lausanne (EPFL), Lausanne, Suiza, 2004.

[LF04b] V. Lepetit and P. Fua. Point Matching as a Classification Problem for Fast and Robust Object Pose Estimation. *Conference on Computer Vision and Pattern Recognition*, Junio de 2004.

[LF06] V. Lepetit and P. Fua. Keypoint Recognition using Randomized Trees. Technical report, Ecole Polytechnique Fédéral de Lausanne (EPFL), Lausanne, Suiza, Febrero de 2006.

[Lin94] T. Lindeberg. *Scale-Space Theory in Computer Vision*. Kluwer Academic Publishers, 1994.

[Lin98] T. Lindeberg. Feature Detection with Automatic Scale Selection. *International Journal of Computer Vision*, 30(2):79–116, 1998.

[Low99] D. G. Lowe. Object Recognition from local scale-invariant features. *International Conference on Computer Vision*, pages 1150–1157, 1999.

[Low01] D. G. Lowe. Local Feature View Clustering for 3D Object Recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 682–688, 2001.

[Low04] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[MB05] T. McReynolds and D. Blythe. *Advanced Graphics Programming Using OpenGL*. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. Elsevier, San Francisco, CA, 2005.

[MC02] J. Matas and O. Chum. Robust Wide Baseline Stereo from Maximally Stable Extremal Regions. *British Machine Vision Conference*, pages 384–393, 2002.

[Mik02] K. Mikolajczyk. *Detection of Local Features Invariant to Affine Transformations, Application to Matching and Recognition*. Tesis de doctorado, Institut National Polytechnique de Grenoble, Francia, julio de 2002.

[MK94] P. Milgram and F. Kishino. A Taxonomy of Mixed Reality Visual Displays. *IEICE Transactions on Information systems*, E77-D(12), Diciembre de 1994.

[Moo91] A. Moore. *An Introductory Tutorial on Kd-Trees*. Technical Report No. 209, Computer Laboratory, University of Cambridge, Robotics Insititute, Carnegie Mellon University, Pittsburgh, PA, 1991.

[MS05] K. Mikolakzcyk and C. Schmid. A performance evaluation of local descriptors. *PAMI*, 27:1615–1630, 2005.

[MTD07] MTD. Amire project. Media Technology and Design, Hagenberg College of Engineering, Austria. Sitio de internet en línea. Disponible en `http://mtd.fh-hagenberg.at/depot/graphics/artoolkit/index.html`. Accedido el 9 de septiembre de 2006, 2005–2007.

[MV99] F. Mindru and L. Van Gool. Recognizing Color Patterns Irrespective of Viewpoint and Illumination. *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 1:368–373, 1999.

[MV02] F. Mindru and L. Van Gool. Comparing Intensity Transformations and their Invariants in the context of Color Pattern Recognition. *Proceedings of the 7th European Conference on Computer Vision*, pages 99–112, 2002.

[NA98] R. C. Nelson and Selinger A. Large-scale test of a keyed, appearance-based 3D object recognition system. *Vision Research*, 38(15):2469–2488, 1998.

[NA02] M. S. Nixon and A. S. Aguado. *Feature Extraction and Image Processing*. Newnes, Oxford, Reino Unido, 2002.

[NK06] N. Navab and G. Klinker. *Computer Aided Medical Procedures Laboratory*. Universidad Técnica de Munich. Sitio de internet en línea. Disponible en `http://campar.in.tum.de/Chair/ResearchGroupCamp`. Accedido el 9 de septiembre de 2006, 2003–2006.

[Now04] S. Nowozin. Autopano y libsift. Sitio de internet en línea. Disponible en `http://cs.tu-berlin.de/~nowozin/autopano-sift/`. Accedido el 18 de diciembre de 2005, 2004.

[NY99] U. Neumann and S. You. Natural Feature Tracking for Augmented Reality. *IEEE Transactions on Multimedia*, 1(1):53–64, 1999.

[OLL04] K. Okuma, J. Little, and D. G. Lowe. Automatic Rectification of Long Image Sequences. *Asian Conference on Computer Vision (ACCV)*, Enero de 2004.

[Par97] Y. Park, J.and Cho. Fast Color Fiducial Detection and Dynamic Workspace Extension in Video See–through Self–tracking Augmented Reality. In *Proceedings of the Fifth Pacific Conference on Computer Graphics and Applications*, 1997.

[SE53] P. J. Schneider and D. H. Eberly. *Geometric Tools for Computer Graphics*. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. Elsevier, San Francisco, CA, 20053.

[SH96] A. State and G. Hirota. Superior Agumented Reality Registration by Integrating Landmark and Magnetic Tracking. *Computer Graphics*, 30(Annual Conference Series):429–438, 1996.

[Ski98] S. S. Skiena. *The Algorithm Design Manual*. Springer-Verlag, Nueva York, Estados Unidos, 1998.

[SM97] C. Schmid and R. Mohr. Local Grayvalue Invariants for Image Retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):530–534, 1997.

[Son05] Sony Corporation. *Eye Toy*. Sitio de internet en línea. Disponible en `http://www.eyetoy.com/index.asp`. Accedido el 5 de septiembre de 2006, 2005.

[SS00] L. Shapiro and G. Stockman. *Computer Vision*. Prentice Hall, Estados Unidos, 2000.

[Sti05] M. Stilman. *Augmented Reality for Robot Development and Experimentation*. Technical Report CMU-RI-TR-05-55, Robotics Institute, Carnegie Melon University, noviembre de 2005.

[SZ00] G. Simon and A. Zisserman. Markerless Tracking using Planar Structures in the Scene. In *Proceedings of International Symposim on Agumented Reality*, pages 120–128, Octubre de 2000.

[SZ02] F. Schaffalitzky and A. Zisserman. Multi-view Matching for Unordered Image Sets, or 'How do I organize my holiday snaps?'. *European Conference on Computer Vision*, pages 414–431, 2002.

[TV98] E. Trucco and A. Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall, New Jersey, 1998.

[TV00] T. Tuytelaars and L. Van Gool. Wide Baseline Stereo Based on Local, Affinely Invariant Regions. *British Machine Vision Conference*, pages 412–422, 2000.

[TV06] T. Tuytelaars and L. J. Van Gool. SURF: Speeded Up Robust Features. *9th European Conference on Computer Vision*, I:404–417, 2006.

[UK95] M. Uenohara and T. Kanade. Vision–Based Object Registration for Real–time Image Overlay. *Computers in Biology and Medicine*, 25(2):249–260, 1995.

[US05] Y. Uematsu and H. Saito. Vision Based Registration for Augmented Reality using Multiple-planes in Arbitrary Positon and Pose by moving Uncalibrated Camera. In *Proceedings of Mirage*, pages 99–107, Francia, 2005. INRIA.

[Ved06] A. Vedaldi. sift++. UCLA Vison Lab. Sitio de internet en línea. Disponible en `http://vision.ucla.edu/~vedaldi/code/siftpp/siftpp.html`. Accedido el 20 de enero de 2006, 2006.

[WA92] M. Ward and R. Azuma. A Demonstrated Optical Tracker with Scalable Work Area for Head–Mounted Display Systems. *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, pages 43–52, Abril de 1992.

[Wik07] Wikipedia. *Computer vision*. *Wikipedia, the free encyclopedia*. Sitio de internet en línea. Disponible en `http://en.wikipedia.org/wiki/Computer_Vision`. Accedido el 17 de enero de 2007, Enero de 2007.

[Wit83] A. P. Witkin. Scale-space Filtering. *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 1019–1023, 1983.

[ZF95] Z. Zhang and O. Faugeras. A Robust Technique for Matching Two Uncalibrated Images through the Recovery of the Unknown Epipolar Geometry. *Artificial Intelligence*, 3(78):87–119, 1995.

[ZH03] A. Zisserman and R Hartley. *Multiple View Geometry in computer vision*. Cambridge University Press, Cambridge, UK, 2nd edition, 2003.