Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Departamento de Computación

Sistema de desarrollo para aplicaciones de realidad aumentada

Tesis que presenta

Rosa Atzín Vázquez del Ángel

para obtener el Grado de

Maestro en Ciencias en Computación

Director de la Tesis

Dr. Luis Gerardo de la Fraga

México, D. F. **Octubre 2010**

Resumen

Un sistema de realidad aumentada complementa el mundo real con objetos virtuales coherentes con él, interactuando en tiempo real. La idea es acercarse cada vez más a sistemas en los cuales sea más difícil realizar una separación de los objetos que pertenecen al mundo real y los objetos que pertenecen al mundo virtual.

El uso de la realidad aumentada es interesante porque mejora la percepción del usuario y la interacción con el mundo real, ya que los objetos virtuales pueden servir para desplegar información que no podemos detectar directamente con nuestros sentidos. En la actualidad existen diversos trabajos que utilizan realidad aumentada en diversas áreas, todos ellos hacen uso de video analógico.

El propósito de esta tesis es experimentar con la construcción de una herramienta que permita la creación de sistemas de realidad aumentada con el uso de video digital, proporcionando captura de video, procesamiento de los marcos, reconocimiento de los patrones, calibración de la cámara y desplegado de un objeto virtual en los marcos del video del mundo real.

Para verificar el correcto funcionamiento de la herramienta, además de experimentar con dos diferentes métodos para la calibración de la cámara, se realizaron dos aplicaciones de realidad aumentada:

- Uso de un patrón de círculos en un plano.
- Calibración utilizando círculos concéntricos utilizando un patrón con dicha forma.

Abstract

An augmented reality (AR) system complements the real world with virtual objects, which are consistent with the real one, and both worlds can interact in real time. The goal of AR is a system in which its difficult to make a separation of the objects that belong to the real world and the objects that belong to the virtual world.

The use of augmented reality is interesting because it improves user perception and interaction with the real world, because the virtual objects can be used to display information that we cannot directly detect with our senses.

Currently there are several studies that use augmented reality in many areas, all of them use analog video.

The purpose of this thesis is to experiment by developing a tool, with free software, that allows the creation of augmented reality systems by using digital video, providing video capture, processing video frames, pattern recognition, camera calibration and displaying of virtual objects within the video frames of the real world.

Verify the correct functionality of the system, as well as to experiment with two different kind of camera calibration methods, two applications were developed:

- Augmented reality application by using a pattern of circles in a plane.
- Augmented reality application by using a pattern of concentric circles.

Agradecimientos

A **mi madre Rosa María** por todo el amor y el apoyo que me has brindado, por siempre ser un ejemplo para mi, por ayudarme a levantarme en todas mis caídas, por enseñarme a luchar por más difícil que fuera, por absolutamente todo lo que eres para mi.

A **mi padre José Luis** por enseñarme el valor de trabajar duro para conseguir las cosas que se desean, por siempre ver por nosotros, por todos tus desvelos para que nunca nos faltara nada, por todo lo que me has enseñado.

Al **Dr. Luis Gerardo de la Fraga** por ser el profesor del que más he aprendido, por sus sabios consejos y apoyo durante la realización de esta tesis, por enseñarme el valor de hacer las cosas con pasión y por su infinita paciencia con mi innumerable cantidad de errores. Muchas gracias por todo.

A **mis hermanos Luis y Ale** por todos los buenos ratos que he pasado a su lado, por que todo, incluso las peleas, han sido momentos muy agradables.

A **Gustavo** por todos los momentos en que has estado a mi lado, por que contigo he aprendido muchísimas cosas y por toda la ayuda que me has dado en los momentos difíciles.

A **mi abuelita Altagracia** por todo tu amor, tu ternura, por ser una segunda madre para mi. Muchas gracias abuelita por todo.

Al **Dr. Francisco Rodríguez Henríquez** y al **Dr. Guillermo Morales Luna** por su valioso tiempo al revisar esta tesis, además de sus sabios consejos.

Al **CINVESTAV** por brindarme la oportunidad de estudiar una Maestría, además de todas las herramientas necesarias para la realización de esta tesis.

Al **CONACyT** por el apoyo económico brindado, sin éste me hubiese sido más difícil llevar a cabo este proyecto y a los proyectos 80965 y 60240 del CONACyT.

A **Sofía Reza** por ser tan gran persona, por siempre tratarnos con el cariño de una madre. Gracias Sofi.

Índice general

Índice de figuras

Capítulo 1
Introducción

La Realidad Aumentada (RA) es una variante de los Ambientes Virtuales, también conocidos como Realidad Virtual (RV) [3]. Las tecnologías de RV introducen al usuario en un ambiente sintético. Mientras está en este ambiente, el usuario no puede ver el mundo real alrededor de él. Por el contrario, la RA permite a los usuarios ver el mundo real, con objetos virtuales sobrepuestos.

Existen dos tipos de aplicaciones de RA:

- Las que involucran dispositivos, como lentes, los cuales no requieren que veamos la pantalla de una computadora para percibir el objeto virtual.
- Las que no requieren algún dispositivo ya que el objeto puede visualizarse en una computadora sobre el video del mundo real.

Algunos investigadores definen como un sistema de RA sólo a aquellos que hacen uso de *HMDs* (por sus siglas en inglés *Head-mounted Display*), sin embargo se puede definir un sistema de RA con tres propiedades, lo cual permite que también los sistemas que no hacen uso de estos dispositivos entren dentro de las aplicaciones de RA. Las propiedades son mencionadas a continuación.

- Combinación de objetos virtuales y reales en un ambiente real.
- Interactividad en tiempo real.
- Objetos virtuales coherentes con el mundo real.

Los inicios de la RA fueron marcados por el trabajo de Sutherland [24], en los años 60, dicho trabajo consistió en el primer dispositivo montado en la cabeza, el cual utilizó para mostrar un simple cubo virtual sobre el mundo real, creando así la primer interfaz de RA.

La RA es el campo de la Computación que maneja la combinación del mundo real con objetos tridimensionales, con la idea de acoplarlos de tal manera que en un futuro no se sepa

cuál es el elemento real y cuál fue generado por computadora.

1.1. ¿Por qué RA?

La RA es un tema interesante porque mejora la percepción del usuario y la interacción con el mundo real. Los objetos virtuales pueden servir para desplegar información que no podemos detectar directamente con nuestros sentidos. La información facilitada por la RA puede ayudarnos a realizar tareas del mundo real.

Existe una gran cantidad de aplicaciones de la RA en diversas áreas, como: visualización médica, mantenimiento y reparación, planificación de trayectorias de robots, entretenimiento, navegación de aeronaves militares, entre muchas otras.

1.2. Aplicaciones de la RA

1.2.1. Medicina

Los médicos pueden hacer uso de la RA para la visualización y la práctica de cirugías. Es posible recoger conjuntos de datos 3D de un paciente en tiempo real, con sensores no invasivos como la resonancia magnética, tomografías computadas o imágenes de ultrasonidos. Este conjunto de datos puede ser combinado con una vista del paciente en tiempo real, el efecto que pueden generar es una "visión de rayos X"para los médicos. Esto puede ser muy útil durante una cirugía mínimamente invasiva, reduciendo el trauma de una operación mediante el uso de incisiones pequeñas o inclusive ninguna incisión. La RA puede proporcionar una visión interna sin la necesidad de realizar incisiones.

1.2.2. Fabricación y reparación

Otra categoría de aplicaciones de RA es el ensamblado, mantenimiento y reparación de maquinaria compleja. Las instrucciones pueden ser más fáciles de entender si en lugar de tener manuales con texto e imágenes, se tienen dibujos en 3D sobrepuestos en el equipo, mostrando paso a paso las tareas que se tienen que realizar. Los objetos en 3D pueden incluso ser animados para hacerlo aún más didáctico.

1.2.3. Anotaciones y visualización

La RA puede ser utilizada para realizar anotaciones a objetos y ambientes con información pública o privada. Por ejemplo, algún dispositivo podría dar información sobre el contenido de estantes de libros mientras una persona camina por una biblioteca, sin la necesidad de ver estante por estante el contenido.

1.2.4. Entretenimiento

El introducir RA en los juegos puede ser una gran aportación, ya que agrega muchas posibilidades en la manera en la que se juega. El complementar con objetos virtuales el mundo real hace que se tenga una experiencia mucho más interesante que estar en un mundo que tenga únicamente objetos virtuales. Una cosa más que es de interés al utilizar RA es el poder observar a las personas con quien estamos jugando, en lugar de ver únicamente al personaje que lo representa.

1.2.5. Educación

Sin duda uno de los campos donde más puede funcionar la introducción de las aplicaciones de RA es en la educación, sobre todo para los niños, ya que proporciona una experiencia más vivencial, por lo que la atención y el aprendizaje son mayores. Se puede hacer uso de la RA en diferentes áreas dentro de la educación, sobre todo en temas que sean difíciles de explicar con imágenes, como por ejemplo en el caso de la química, para la explicación de las

reacciones químicas, así como los cambios de hibridación, entre muchos otros temas.

1.3. Estado del arte

Los proyectos de más impacto son las herramientas para el diseño de sistemas de RA, ya que éstos son utilizados por muchos investigadores para modelar aplicaciones. Algunas de las herramientas más significativas son las siguientes:

En el año 1999, se desarrolló *ARToolkit* [13], que es una biblioteca de seguimiento de visión por computadora la cual permite la creación de aplicaciones de RA completas, por medio de la sobreposición de un objeto virtual en una imagen del mundo real. Fue desarrollada por Hirokazu Kato en el *Human Interface Technology Laboratory* de la Universidad de Washington. Actualmente es la biblioteca para construir aplicaciones de RA más popular, es usada por investigadores en todo el mundo para diversos proyectos.

Tiempo después, en el año 2004, basado en *ARToolkit* y con su misma filosofía, M. Fiala desarrolla *ARTag* [7], que es un sistema completo para construir aplicaciones de RA. Hace técnicas de tratamiento digital de imágenes más complejas para la detección de marcas, ofreciendo así una mayor fiabilidad e inmunidad a los cambios de iluminación.

ARTollkitPlus [1] fue desarrollado como parte del *The Handheld Augmented Reality Project*, en la *Graz University of Technology* en Austria. Está construido sobre la base del *ARToolkit*, pero tiene características mejoradas en el reconocimiento de patrones. No está destinado a construir aplicaciones completas de RA, debido a que no lee imágenes desde la cámara, ni proporciona algún módulo para graficación. La detección de patrones se inspira en el *ARTag* y posee mejoras en la inmunidad a los cambios de iluminación con respecto al *ARToolkit* original.

En el año de 2008, se desarrolla en el *Human Interface Technology Laboratory* de la Universidad de Washington, una biblioteca derivada de *ARToolkit* llamada *NYARToolkit*, que no provee captura de video, por lo que no está destinada a la creación de aplicaciones de RA completas.

Todas estas herramientas trabajan con video analógico, por lo cuál la calidad de dicho video es muy baja y no nos permite visualizar unas imágenes bien definidas. El uso de video digital mejoraría mucho la calidad en la aplicación final, es lo que se pretende con este trabajo.

Además de las bibliotecas para la creación de aplicaciones de RA, existen diversas aplicaciones, en diferentes áreas de investigación, algunas de ellas son las siguientes:

Virtual Liver Surgery Planning

Los tumores en el hígado provocan una considerable cifra de muertes en el mundo. Para algunos pacientes que tiene tumor en el hígado, la única cura posible es una cirugía, mientras que para otros pacientes, las técnicas de intervención están disponibles para el control de tumores locales. El tratamiento individual está basado en resultados de laboratorios entre otros estudios, desafortunadamente la correcta identificación representa una tarea difícil. En el año 2004, se propone un sistema virtual [5] para ayudar a la planificación de esta cirugía, en el cual se adquiere la imagen médica y se realiza el desplegado utilizando RA para que se tome la mejor decisión. El sistema incluye herramientas para visualización, medición entre otras cosas que permiten tomar una decisión.

Augmented Reality as a Medium for Cartography

En el año 2005, Reitmayr desarrolló el sistema aumentado de mapas [23], en la *University of Cambridge*. Es una combinación directa de mapas cartográficos como artefactos básicos y RA como interfaz de usuario. Los mapas impresos son desplegados de manera estática mientras que los mapas en computadora soportan información dinámica. Lo que se hizo fue conjuntar los mapas estáticos con información dinámica, desplegando dicha información requerida directamente sobre el mapa físico, mediante herramientas tangibles.

Para la visualización, uno o más mapas pueden ser colocados en una mesa, o en una superficie plana, una cámara es montada sobre la mesa sobre la que están los mapas, dicha cámara registra la información del plano y posteriormente un proyector aumenta los mapas con la información proyectada sobre ellos.

Augmented Reality Scouting for Interactive 3D Reconstruction

Primer prototipo de sistema de reconstrucción interactiva 3D de escenas urbanas [19]. Un *Augmented Reality scout* es una persona equipada con un *ultra-mobile PC*, con una cámara

USB y un receptor de GPS. Esta persona explora el ambiente urbano y determina una secuencia de imágenes en 2D. Estas imágenes son almacenadas y posteriormente se genera el modelo en 3D.

Existen tres reportes técnicos sobre sistemas relacionados a lo que se pretende realizar:

Sistema de Realidad Aumentada [20]

Experimenta con la RA para construir dos sistemas:

1. Un sistema de RA en dos dimensiones en el que la idea es mover un círculo virtual sobre un plano de inclinación
2. Un sistema de RA en tres dimensiones que presenta el movimiento de una pelota virtual dentro de una caja real.

Este sistema utiliza *The GIMP Toolkit* (GTK) para el desplegado de los objetos, por lo que los gráficos son muy simples, utiliza formato VD y Qt para su interfaz.

Medición de respuesta pupilar [14]

Es un sistema que analiza las variaciones de respuestas pupilares a estímulos para contraer y dilatar la pupila. Este sistema obtiene el video con formato VHS(*Video Home System*), es decir, un formato analógico, y lo digitaliza para su procesamiento en la computadora, obteniendo un formato VD. Al igual que el sistema anterior, utiliza Qt para su interfaz.

Sistema para la integración de Entornos Reales y Virtuales mediante Realidad Aumentada [15]

Es un sistema que permite visualizar objetos que dentro del mundo real están ocultos y también cambiar la apariencia de objetos que sí son visibles. Los objetos ocultos que muestra son redes de tubería y redes de cableado, y los otros objetos, incluyen paredes, techos, pisos, puertas y ventanas, esto con la finalidad de realizar diseño de interiores con ayuda de la AR. Este sistema utiliza OpenGL para el modelado de los objetos virtuales, pero a diferencia de los sistemas anteriores, la entrada de video en este sistema está en el USB, por lo que tiene menor velocidad.

Augmented Reality with NYARToolkit, OpenCV and OpenGL [22]

Es una pequeña aplicación realizada para la visualización de un cubo virtual en el ambiente real con AR. Para la obtención del video y de los marcos se utiliza OpenCV, para la creación del objeto utilizan OpenGL, no se especifica qué se utiliza para la creación de la interfaz, ni la entrada de video, pero por la forma en que se obtienen los marcos, se puede deducir que

Trabajo	Biblioteca de Video	Despliegue de Imágenes	Interfaz	Entrada de Video
Sistema de Realidad Aumentada.	Kino	GTK	Qt	Firewire
Medición de respuesta pupilar.	Kino	OpenGL	Qt	Achivos avi
Sistema para la integración de Entornos Reales y Virtuales mediante Realidad Aumentada.	OpenCV	OpenGL	-	USB
Augmented Reality with NYARToolkit, OpenCV and OpenGL	OpenCV	OpenGL	-	USB

Tabla 1.1: Comparación entre cuatro diferentes sistemas de RA.

es por medio del puerto USB. Para el procesamiento de imágenes utilizan *NYARToolkit*.

En la Tabla 1 se puede observar la comparación de los cuatro trabajos anteriores, en cuanto a la biblioteca utilizada para el manejo del video, el desplegado de las imágenes así como la interfaz y la entrada de video.

1.4. Descripción del problema

En la actualidad, el ser humano está expuesto a una gran cantidad de escenarios de los que no tiene información previa, en algunos casos sólo cuenta con la que es percibida por sus sentidos, por esta razón es necesario obtenerla de su entorno por algún otro medio. En muchos de estos escenarios, la información del entorno es necesaria para tomar decisiones lo más rápido posible. Para este fin existen herramientas que permiten la visualización de estos objetos virtuales en el ambiente real por medio de la RA, sin embargo, algunas de ellas presentan carencias en cuanto a la captura del video, en tanto que otros ni siquiera presentan esta posibilidad.

Para atacar las carencias existentes en las actuales herramientas para el desarrollo de aplicaciones de RA, así como de los mismos sistemas realizados, en cuanto a la calidad de video se propone realizar un sistema que abarque desde el procesamiento de los marcos del video, utilizando video digital (VD) porque es un formato mucho más veloz, además de que proporciona una mayor calidad.

Un desafío importante enfrentado es la necesidad de acoplar: captura de video, procesamiento de marcos, procesamiento de imagen y reproducción eficiente de dicho video, sin

pérdida de marcos y sin retrasos. El problema es el uso de VD, ya que se requiere una biblioteca que pueda hacer la captura de cada uno de los marcos que lo conforman, posteriormente se necesita realizar su procesamiento de una manera rápida para realizar la calibración de la cámara y el desplegado del objeto virtual en cada uno de los marcos.

En la Figura 1.1 se observa el proceso que se debe seguir para realizar una aplicación de RA:

- Captura de video. Para obtener un video de muy alta calidad se requiere el uso de VD, el cual debe ser obtenido del puerto *Firewire*.
- Obtención de los marcos. Al ir leyendo el video, se necesita ir tomando cada uno de los marcos, la velocidad del transmisión del VD es de 25 marcos por segundo, por lo cual se deben obtener los 25 marcos para evitar pérdidas en el video.
- Procesar cada marco. Se debe realizar un procesamiento sobre la imagen de cada marco para obtener los patrones que se están utilizando y de esta manera realizar la calibración de la cámara.
- Calibración de la cámara. La calibración de la cámara es muy importante para que los objetos virtuales sean coherentes con el patrón en el mundo real. Al realizar la calibración de la cámara vamos a obtener los parámetros de ésta, los cuales serán aplicados al objeto virtual para lograr que se vea perfectamente ajustado al patrón. Los parámetros que se obtienen son:

  Matriz de rotación

  Vector de translación

  Matriz de proyección.

- Objeto virtual. Finalmente, ya con los parámetros de la cámara se dibuja el objeto en OpenGL.

Para verificar que la herramienta se encuentre trabajando de una manera correcta, se realizaron dos aplicaciones en las cuales se experimentó con diferentes patrones, así como calibración de la cámara.

- Aplicación con patrón plano. Para esta aplicación se utilizó un patrón cuadrado con 16 puntos, 4 filas de 4 cada uno en color rojo. Se realizó la calibración con dichos puntos, y sobre él se dibujó un cuboide con un cilindro girando dentro de él.
- Aplicación con patrón de círculos concéntricos. Para esta aplicación se utilizó un patrón con 2 círculos concéntricos, el más grande de ellos en color rojo y el otro en color blanco, se realizó la extracción de ambas elipses (se extraen elipses ya que en la cámara, un círculo siempre se va a ver como una elipse) y con los parámetros de ellas, la calibración de la cámara. Se dibujó un cilindro sobre el patrón con una barra girando dentro de él.

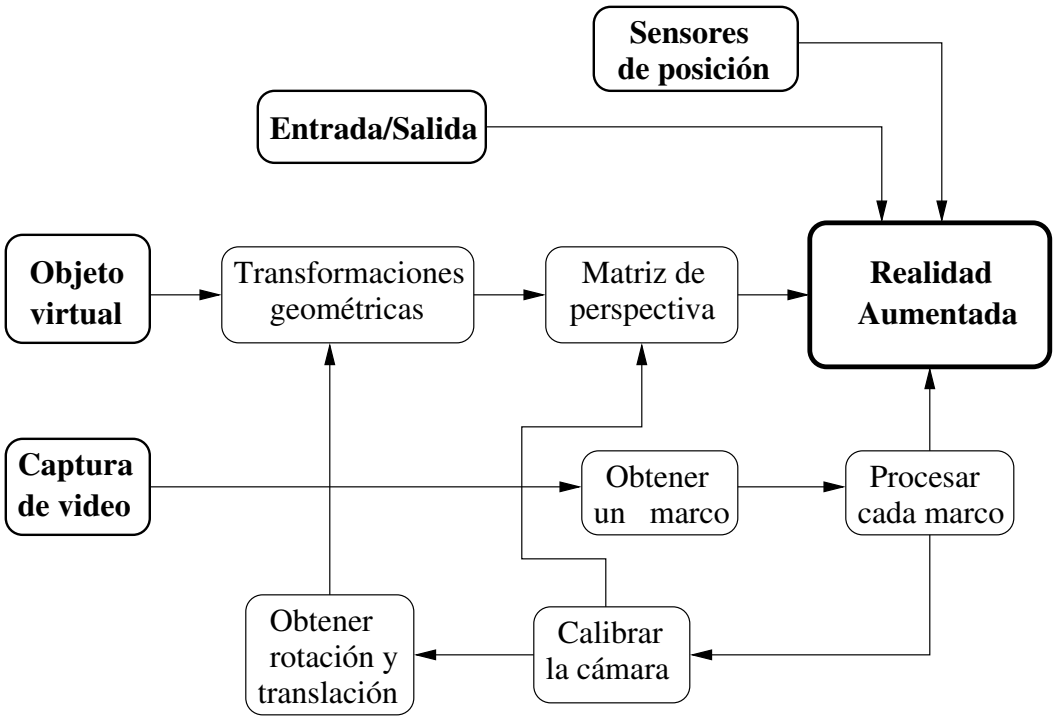


Figura 1.1: Modelo conceptual de la RA

1.5. Organización de la tesis

Esta tesis está organizada en 6 capítulos que describen cada una de las fases del sistema.

- Capítulo 1. Introducción. En este capítulo se define el sistema que se realizó, se describe un diagrama del proceso que involucra un sistema de RA, da una definición general de RA, algunas áreas donde es utilizada y se da una breve descripción de las herramientas más significativas para la creación de aplicaciones de RA, así como de los dos sistemas desarrollados para probar la aplicación. Finalmente la organización de la tesis.

- Capítulo 2. Marco Teórico. En este capítulo se da una introducción a los conceptos, así como herramientas necesarias para entender el desarrollo de esta tesis. Los temas que se tratan son:

  - Video
  - Tratamiento digital de imágenes
  - Calibración de cámara

- Capítulo 3. Sistema de desarrollo. En éste capítulo se describe el procedimiento realizado para la creación del sistema en general, los cambios que se tuvieron que hacer a

algunas bibliotecas en el código fuente de *DVGrab* (para leer marcos de video digital), así como la arquitectura del sistema.

- Capítulo 4. Aplicación 1, autocalibración con un patrón plano. En este capítulo se explica detalladamente la realización del sistema que utiliza el patrón de los círculos en un plano para la calibración de la cámara.
- Capítulo 5. Aplicación 2, autocalibración con un patrón de círculos concéntricos. En este capítulo se explica detalladamente la realización del sistema que utiliza el patrón de los círculos concéntricos para la calibración de la cámara.
- Capítulo 6. Conclusiones. En este capítulo se presentan las conclusiones obtenidas del trabajo de tesis, así como el trabajo a futuro que se puede desarrollar a partir del trabajo realizado.

Capítulo 2
Marco teórico

2.1. Video Digital

El formato de Video Digital(VD) fue introducido por primera vez en 1983 mediante el formato D-1 de Sony, que grababa una señal no comprimida de definición estándar en forma digital, en vez de las formas analógicas que habían sido frecuentes hasta ese entonces. El VD apareció por primera vez en el mercado en el año 1990. Fue creado como un estándar de video que usa como protocolo de transmisión de datos el IEEE 1394 también conocido como *Firewire* e i.Link., que si bien no es parte del formato en sí, está estrechamente ligado a éste. Cualquier equipo DV tanto doméstico como profesional que puede tener conexión de 6 o 4 pines. VD se basa en el algoritmo DTC (Transformada de Coseno Discreta) con una compresión intramarco, es decir, a razón de 5:1.

El flujo de datos de un VD está compuesto por tres niveles en una estructura jerárquica. En la figura 2.1 se presenta la estructura que tiene el formato DV.

Un marco del formato DV está dividido en secuencias múltiples DIF (Formato de Interface Digital). Cada secuencia DIF está compuesta por 150 segmentos de 80 bytes de longitud que componen un bloque DIF, que es la unidad primitiva para todo flujo de VD. Cada bloque DIF de 80 bytes contiene una cabecera ID de 3 bytes que se encargan de especificar el tipo del bloque DIF, así como su posición en la secuencia. Existen cinco tipos de bloques DIF definidos en las cabeceras ID: *Header*, *Subcode*, *Video Auxiliary Information (VAUX)*, *Audio data* y *Video data*.

El formato DV emplea una resolución de 720×576 píxeles con un nivel de color de 24 bits y un estándar de 25 marcos por segundo, aunque puede ser aumentado o disminuido dependiendo de la cámara de video y el programa que almacena el video.

Para video hay dos estándares de imágenes por segundo, los cuales se describen brevemente a continuación.

- NTSC (*National Television System Committee*). Consiste en la ampliación del sistema

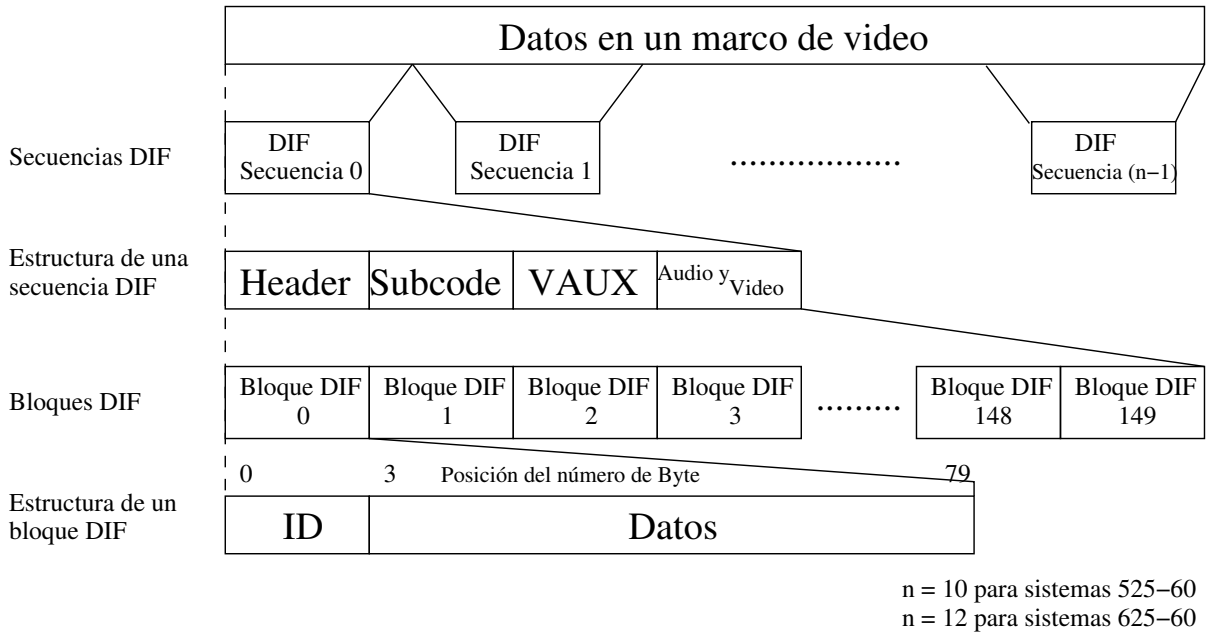


Figura 2.1: Estructura del formato DV

monocromático norteamericano. Este sistema permite la transmisión de cerca de 30 imágenes por segundo

- PAL (*Phase Alternating Line*). Sistema de codificación utilizado en la transmisión de señales de televisión analógica en color en la mayor parte del mundo. Este sistema permite la transmisión de 25 imágenes por segundo.

2.2. Procesamiento digital de imágenes

Desde el inicio de la ciencia, la observación visual ha jugado un papel muy importante. En un principio se utilizaban descripciones verbales, así como dibujos hechos a mano. Con el paso del tiempo fue introducido el uso de fotografías, permitiendo así resultados con una documentación más amplia, permitiendo realizar descripciones con mayor rapidez. Hoy en día, con la tecnología se puede realizar un procesamiento de los datos de las imágenes, obteniendo datos muy exactos sin necesidad de una inspección visual.

El término procesamiento digital de imágenes versa sobre la manipulación y análisis de imágenes por computadora [21]. Es el conjunto de técnicas que se aplica a las imágenes digitales, con el objetivo de mejorar la calidad o facilitar la búsqueda de información.

El procesamiento de imagen puede considerarse como un tipo especial del procesamiento digital en dos dimensiones, el cual se usa para revelar información sobre imágenes y que

involucra hardware, software y soporte teórico.

El procesamiento digital de imágenes juega un papel muy importante en la obtención de datos específicos ya que permite mejorar imágenes digitales para realizar interpretaciones más acertadas sobre su contenido, y así poder tomar decisiones de manera automática.

2.2.1. Segmentación

Diversos estudios psicológicos indican la preferencia de los humanos por realizar agrupaciones de las regiones visuales, según la similitud entre ellas, de ahí surge el origen de la segmentación de imágenes.

El primer paso en cualquier procesamiento de imágenes es la segmentación. Por medio de la segmentación, una imagen será dividida en partes u objetos que la forman. La segmentación terminará cuando se hayan detectado todos los objetos de interés para la aplicación.

La segmentación de las imágenes digitales es una parte muy importante para el procesamiento digital de imágenes, pero a su vez es la parte más difícil de realizar. Existen cuatro grandes grupos dentro de las técnicas de segmentación: Técnicas basadas en los valores de píxel, técnicas basadas en el área, las basadas en bordes y, finalmente, técnicas basadas en la física.

Dentro de las técnicas mencionadas anteriormente, destacan algunas que son mencionadas a continuación.

- Técnicas de umbralización. Basados en valores de píxel, toman decisiones con base en la información de un píxel local. Son muy eficientes cuando los niveles de intensidad de los objetos caen fuera del ángulo recto del fondo de la imagen.
- Métodos basados en esquinas. Basados en los bordes, se centran en la detección de contornos, son muy eficientes cuando todos los bordes están completos.
- Métodos basados en regiones. Basados en el área, particionan la imagen en regiones conectadas, agrupando píxeles vecinos con un nivel de intensidad similar.

2.2.2. Suavizado

Los filtros de suavizado espacial son una de las diversas técnicas enfocadas a mejorar la calidad de una imagen. Su finalidad es reducir el ruido y la borrosidad.

Conocidos también como filtros de promedio, trabajan de una forma muy simple, sustituyen el valor del píxel por el promedio de su vecindario. La vecindad se delimita por una máscara de promediado, la cual se indica en la Figura 2.2. La máscara es colocada sobre cada

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Figura 2.2: Máscara de suavizado

uno de los píxeles, quedando sobre el píxel a modificar, el centro de la imagen y realizando la convolución mediante la ecuación (2.1).

$$R = \frac{1}{9}L = \sum_{i=1}^{9} z_i \tag{2.1}$$

donde L es el resultante de aplicar la máscara a la imagen, la cual debe ser multiplicada por $\frac{1}{9}$ para obtener el promedio, y z_i son los píxeles de la imagen a los cuales se aplica la máscara de suavizado.

En la Figura 2.3 se puede observar un ejemplo del filtro de suavizado aplicado a una imagen.

(a) (b)

Figura 2.3: (a) Imagen de entrada, (b) Imagen con filtro de suavizado

2.2.3. Segmentación por color

La segmentación es una técnica que puede convertir una imagen RGB (*Red, Green, Blue*) a monocromática en una imagen en blanco y negro (segmentación binaria) o a una imagen de un solo color (segmentación por color).

La segmentación por color consiste en extraer únicamente los objetos que pertenezcan al color elegido. Se tiene la imagen en el modelo de colores RGB pero se requiere que dicha imagen sea transformada al modelo HSI (*Hue, Saturation, Intensity*), el cual será explicado más adelante para aplicar esta segmentación.

La segmentación por color consiste en analizar cada uno de los píxeles en sus componentes HSI y tomar únicamente los que se encuentre en el rango del color elegido, realizando sobre este píxel una binarización, es decir, cambiar el color del píxel a negro en caso de que pertenezca al color que se está utilizando y en blanco en caso contrario.

2.2.4. Funciones morfológicas

La descripción básica de la morfología matemática yace en la teoría de conjuntos cuyos primeros trabajos se deben a Minkowski en el año 1897. La continuación de estos trabajos de investigación, bajo impulso y reformulación de Matheron y Serra, se darían posteriormente a conocer bajo la denominación de morfología matemática, como una técnica no lineal de tratamiento de señales.

La palabra morfología comúnmente denota una rama de la biología que estudia la forma y estructura de los animales y plantas [9]. La morfología en el procesamiento de imágenes es una herramienta para la extracción de componentes de una imagen, que son usados en la representación y descripción de la región de una figura, tal como bordes, esqueletos y cubierta convexa. Se utiliza con la finalidad de mejorar los resultados obtenidos por la segmentación.

Las operaciones más comunes son la erosión y la dilatación. Estas operaciones son comúnmente aplicadas sobre imágenes binarias, por lo cual, la dilatación tiende a aumentar el tamaño de los objetos, mientras que la erosión tiende a adelgazar los objetos, o incluso si son muy pequeños los llega a eliminar.

Erosión

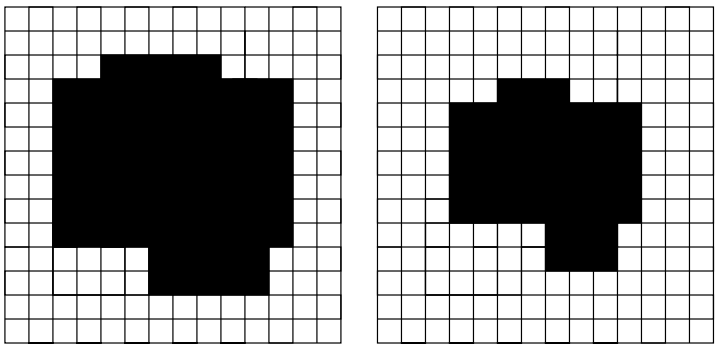
La erosión es el proceso de eliminar todos los puntos que conforman el borde de un objeto, disminuyendo su área. Es de gran utilidad para eliminar pequeños puntos de ruido, debido a que si un objeto no tiene más de dos píxeles de grosor será eliminado con la erosión. La erosión es usada para borrar segmentos pequeños de una imagen que son de poco interés.

La erosión se define generalmente como:

$$E = B \ominus S = \{x, y | S_{x,y} \subseteq B\}$$

Esto es, la imagen binaria E que resulta de erosionar B por S es el conjunto de puntos (x, y) tal que si S es trasladada de modo que su origen se encuentra en el punto (x, y), entonces está completamente contenido dentro de B.

En la Figura 2.4 se puede observar un ejemplo de la erosión aplicada a una imagen pequeña.



(a) (b)

Figura 2.4: (a) Imagen de entrada sin procesamiento, (b) Imagen de (a) con una erosión

Dilatación

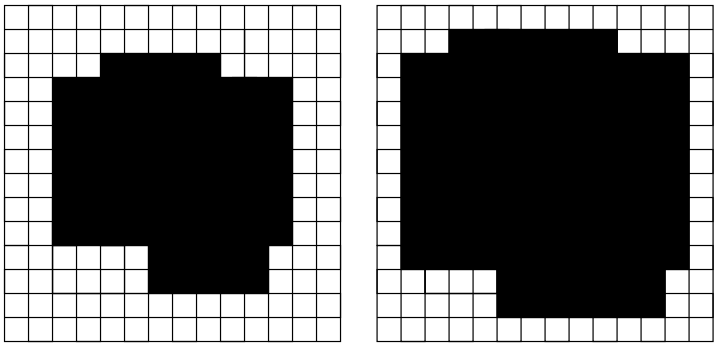
La dilatación es el proceso de incorporar dentro del objeto todos los puntos del fondo de la imagen que lo tocan, aumentando su área. Si se tienen dos objetos separados por menos de tres píxeles, al aplicar una dilatación se conectarán. La dilatación es útil para rellenar huecos en objetos segmentados.

La dilatación se define como:

$$D = B \oplus S = \{x, y | (S_{x,y}) \cap \neq 0\}$$

Esto es, la imagen binaria D que resulta de dilatar B por S es el conjunto de puntos (x, y) tal que si S se traslada de modo que su origen se encuentra en el punto (x, y), entonces su intersección con B es diferente del conjunto vacío.

En la Figura 2.5 se puede observar un ejemplo de la dilatación aplicada a una imagen pequeña.



(a) (b)

Figura 2.5: (a) Imagen de entrada, (b) Imagen dilatada

2.2.5. Detección de bordes

Los contornos caracterizan los límites de un objeto. Un contorno se caracteriza por presentar una transición de claro a oscuro o viceversa. Un método utilizado para la extracción de dichos contornos es Sobel.

Sobel detecta un borde, calculando el gradiente de la imagen en dos direcciones ortogonales. Calcula la intensidad de la imagen en cada punto, dando la dirección del aumento posible más grande de la luz a la obscuridad y el índice del cambio en esa dirección. El resultado demuestra qué tan marcado es el cambio en ese punto y, por lo tanto, qué tan probable es que esa parte de la imagen represente un borde.

El operador Sobel hace uso de dos máscaras, una para los cambios horizontales y una para los cambios verticales. Dichas máscaras se presentan en la Figura 2.6.

−1	0	1
−2	0	2
−1	0	1

(a)

−1	−2	−1
0	0	0
1	2	1

(b)

Figura 2.6: (a) Máscara vertical, (b) Máscara horizontal

El operador Sobel es de gran utilidad, ya que podemos tener un control de los bordes dependiendo qué tan marcados estén. En la Figura 2.7 se puede observar un ejemplo de la aplicación de este operador.

(a) (b)

Figura 2.7: (a) Imagen de entrada, (b) Imagen con la detección de bordes

2.3. Modelos de color

El uso de color en el procesamiento de imágenes es muy utilizado, esto es debido a que un color puede simplificar la identificación, así como la extracción de un objeto en una escena.

El procesamiento de imágenes a color se divide en dos áreas: procesamiento con color real y procesamiento con pseudocolor.

El propósito de un modelo de color es facilitar la especificación de los colores en algún estándar. Un modelo de color es una especificación de un sistema coordenado y un subespacio dentro de este sistema donde cada color es representado por un punto.

El color, como cualquier otro recurso, también tiene su técnica y está sometido a ciertas leyes, y según la aplicación que se desea, se trabaja con distintos modelos de color. Los modelos de color describen los colores que se ven en las imágenes digitales e impresas y el trabajo con ellos.

Los modelos de color permiten no sólo establecer un espacio único común a todos los equipos que forman parte de la cadena de adquisición y reproducción de color, sino que también permiten simular cómo lucirá la imagen y su color en otro dispositivo de la cadena.

Cada modelo de color representa un método diferente de descripción de los colores.

En términos del procesamiento digital de imágenes, los modelos de color más comúnmente utilizados son: RGB (*Red, Green, Blue*), que es utilizado en monitores de color y en video cámaras; CMY (*Cyan, Magenta, Yellow*), que es utilizado en impresoras, y HSI (*Hue, Satu-*

ration, Intensity), el cual corresponde con la forma como los humanos describen e interpretan el color.

2.3.1. Modelo de color RGB

El modelo RGB es de síntesis aditiva del color, o color luz. La forma más directa de operar consiste en utilizar los vectores de los colores rojo, verde y azul en el rango $[0, 1]$. A esta convención se le llama modelo RGB. Este es el modelo de definición de colores en pantalla usado para trabajos digitales.

La pantalla está compuesta por píxeles, donde en realidad cada píxel está formado por un conjunto de tres subpíxeles: uno rojo, uno verde y uno azul, cada uno de los cuales brilla con una determinada intensidad. El monitor produce entonces los puntos de luz, partiendo de tres tubos de rayos catódicos, uno para cada color.

Para indicar con qué proporción mezclamos cada color en pantalla, se asigna un valor a cada uno de los colores primarios, como se puede observar en la Figura 2.8, de manera que el valor 0, por ejemplo, significa que no interviene en la mezcla, y a medida que ese valor aumenta, se entiende que aporta más intensidad a la mezcla.

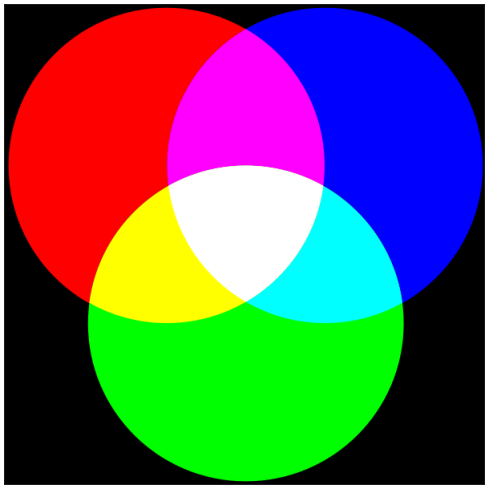


Figura 2.8: Combinación de los componentes RGB

2.3.2. Modelo de color CMY

El modelo RGB realiza un manejo de los colores primarios (rojo, verde y azul), mientras que el modelo CMY maneja los colores secundarios (cian, magenta y amarillo). La mayoría de los dispositivos que depositan pigmentos de color en papel, como las impresoras, reciben datos de entrada en el modelo CMY. o alternativamente pueden realizar la conversión del modelo RGB al modelo CMY.

Para realizar la conversión de RGB a CMY se utiliza la ecuación (2.2)

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2.2}$$

donde:
C = Valor del color cian
M = Valor del color magenta
Y = Valor del color amarillo
R = Valor del color rojo
G = Valor del color verde
B = Valor del color blanco
suponiendo que los valores de los colores se encuentran en el rango $[0, 1]$.

2.3.3. Modelo de color HSI

Los modelos RGB y CMY son ideales para implementaciones en *hardware*, sin embargo no pueden utilizarse para la descripción e interpretación de los colores comparados con la forma en la que los humanos lo hacen, es decir, cuando una persona observa un color, no lo describe indicando sus componentes, el porcentaje de cada uno de los colores primarios, ni siquiera le es posible saber qué colores lo conforman.

Para solucionar el problema anterior, se utiliza el modelo HSI, que define un modelo de color en términos de sus componentes constituyentes. Funciona de manera similar a como los humanos observan los colores.

Cuando un humano observa un color, en lugar de hacer su descripción en base a los porcentajes de los colores primarios, lo hace en base al color puro, la saturación y el brillo. El modelo HSI se encuentra formado por los componentes descritos a continuación.

- Matiz (H). Este componente describe el color puro del objeto, como por ejemplo: rojo, azul, verde, amarillo, etc.
- Saturación (S): Proporciona la medida en la que el color puro ha sido diluido por el color blanco.
- Intensidad (I): Proporciona el nivel del brillo del objeto.

En la Figura 2.9 se puede observar cómo es que funciona cada uno de los valores de HSI.

Para realizar la transformación, inicialmente se utiliza la ecuación (2.3) para obtener los valores r, g, b. Posteriormente se obtiene el valor de h de la ecuación (2.4) y el valor de s de la ecuación (2.5).

$$r = \frac{R}{R+G+B}, \quad g = \frac{G}{R+G+B}, \quad b = \frac{B}{R+G+B} \tag{2.3}$$

$$h = \begin{cases} \cos^{-1}\left[\frac{0.5\cdot[(r-g)+(r-b)]}{[(r-g)^2+(r-b)(g-b)]^{\frac{1}{2}}}\right] & si \quad b \leq g \\ 2\pi - \cos^{-1}\left[\frac{0.5\cdot[(r-g)+(r-b)]}{[(r-g)^2+(r-b)(g-b)]^{\frac{1}{2}}}\right] & si \quad b > g \end{cases} \tag{2.4}$$

$$s = 1 - 3 \cdot min \cdot (r, g, b) \quad s \in [0, 1] \tag{2.5}$$

En el algoritmo 1 se puede observar el procedimiento que se realizó para segmentar la imagen de acuerdo al color elegido. El procedimiento incluye la obtención de los valores de H y de I, y la binarización para tener en la imagen únicamente los objetos del color elegido, todo en blanco y negro.

Algoritmo 1 Algoritmo de conversión a HSI

Entrada: Marco en formato RGB (RGB), ancho del marco ($ancho$), alto del marco ($alto$), ángulos del color buscado ($angulo1$ y $angulo2$)

Salida: Marco binarizado dependiendo del color elegido (hsi)

1: Inicialización de arreglo $hsi[ancho * alto] \rightarrow 0$
2: Inicialización de $n \rightarrow 0$
3: Inicialización de $r \rightarrow 0$
4: Inicialización de $g \rightarrow 0$
5: Inicialización de $b \rightarrow 0$
6: Inicialización de $h \rightarrow 0$
7: Inicialización de $suma \rightarrow 0$
8: **for** $i = 0$ to $i = areatotaldelmarco$ **do**
9: $suma \rightarrow RGB[n] + RGB[n+1] + RGB[n+2]$
10: $r \rightarrow RGB[n++]/suma$
11: $g \rightarrow RGB[n++]/suma$
12: $b \rightarrow RGB[n++]/suma$
13: **if** $r \geq b$ **then**
14: $h \rightarrow cos^{-1}\frac{0.5*(r-g+r-b)}{\sqrt{(r-g)*(r-g)+(r-b)*(g-b)}}$
15: **else**
16: $h \rightarrow 2 * \pi - cos^{-1}\frac{0.5*(r-g+r-b)}{\sqrt{(r-g)*(r-g))+(r-b)*(g-b)}}$
17: **end if**
18: **if** $h > angulo1 \&\& h < angulo2$ **then**
19: $hsi[i] \rightarrow 0$
20: **else**
21: $hsi[i] \rightarrow 255$
22: **end if**
23: **end for**

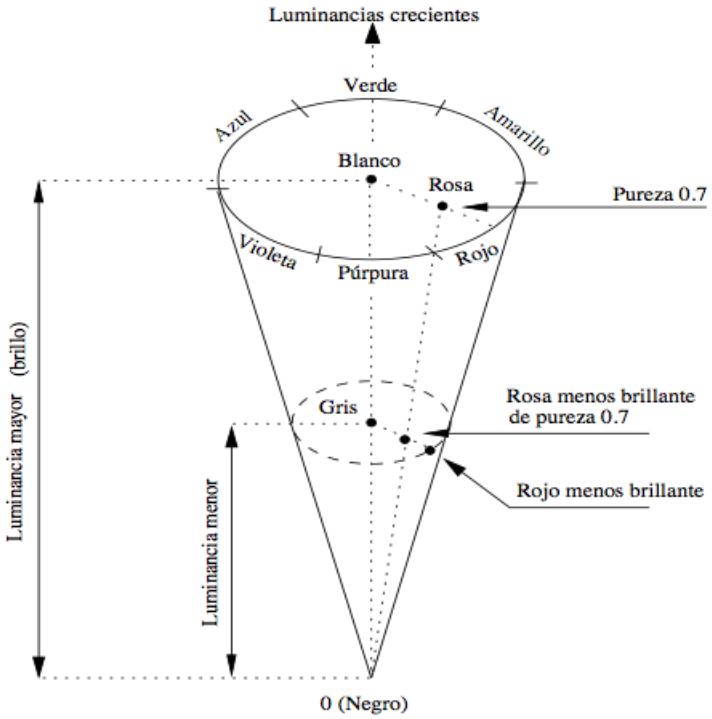


Figura 2.9: Cono de colores del modelo HSI

2.4. Calibración de cámara

En aplicaciones con visión computacional un aspecto importante a considerar es el proceso denominado calibración de cámara, el cual consiste en determinar los parámetros internos de la cámara, tales como distancia focal, factores de distorsión y puntos centrales del plano imagen[17].

El proceso de calibración de cámaras es necesario para poder extraer información métrica a partir de imágenes 2D del mundo 3D. Aunque existen técnicas para inferir esta información desde imágenes captadas por cámaras no calibradas, a través de procesos de calibración de los 3 puntos [10], el proceso de calibración de cámara abre la posibilidad de realizar aplicaciones efectivas en visión tales como RA, reconocimiento, seguimiento y reconstrucción 3D, los cuales se basan en el conocimiento de calibración y pose de la cámara [6].

La idea principal tras los procesos de calibración de cámara es describir el modelo de proyección que relaciona los sistemas de coordenadas que permiten obtener los parámetros de la cámara. En esencia el proceso de calibración de la cámara consiste en determinar la geometría y características internas de la cámara, parámetros intrínsecos y extrínsecos. Estos parámetros normalmente son calculados desde un patrón de calibración que contiene rasgos

fácilmente detectables de manera precisa en la imagen capturada.

Para ubicar objetos en el mundo real, establecemos un marco de referencia, que en el contexto de visión computacional es llamado marco de referencia del mundo. Un objeto en una imagen es medido en términos de coordenadas de píxeles, los cuales están en el marco de referencia de la imagen. El solo conocer la distancia en píxeles entre puntos en una imagen, no nos permite determinar la distancia correspondiente a los mismos puntos del mundo real. Por lo tanto, es necesario establecer las ecuaciones que unen el marco de referencia del mundo con el marco de referencia de la imagen, de manera que se establezca la relación entre los puntos en coordenadas en el espacio 3D y los puntos en coordenadas de imagen 2D. Desafortunadamente, no se puede establecer esta relación directamente, haciéndose necesario establecer un marco de referencia intermedio llamado: marco de referencia de la cámara.

Para encontrar la relación del marco de referencia del mundo y el marco de referencia de la imagen se deben encontrar las ecuaciones que los unan. Al resolver el sistema generado se obtiene la relación buscada, lo cual es equivalente a encontrar las características de la cámara.

Existen diversas técnicas de calibración de cámara. De acuerdo a la literatura [17] estas pueden ser clasificadas en dos grandes categorías.

- Calibración fotogramétrica. Se realiza mediante la observación de patrones cuya geometría en el espacio 3D es conocida con un buen nivel de precisión. Los patrones de calibración normalmente están posicionados en dos o tres planos ortogonales entre ellos. En algunos casos, basta con un único plano, cuya traslación es perfectamente conocida. Este tipo de calibración requiere una configuración elaborada, pero sus resultados son eficientes.
- Autocalibración. Este método se basa en el movimiento de la cámara observando una escena estática, a partir de sus desplazamientos y usando únicamente la información de la imagen. La rigidez de la escena impone en general restricciones sobre los parámetros de la cámara.

2.4.1. Parámetros intrínsecos y extrínsecos

La calibración de cámara se aplica con la finalidad de obtener los parámetros intrínsecos como extrínsecos de la cámara, los cuales se describen a continuación.

- Parámetros extrínsecos. Definen la localización y orientación del marco de referencia de la cámara con respecto al marco de referencia del mundo. Son las rotaciones, así como las traslaciones necesarias para que el objeto sea coherente al patrón con el cual se realiza la calibración.
- Parámetros intrínsecos. Son necesarios para ligar las coordenadas del píxel de un punto de la imagen con las coordenadas correspondientes en el marco de referencia de la

cámara. Se supone que el sistema de imágenes está basado en la cámara obscura, el cual corresponde a tener una proyección en perspectiva, por lo tanto, +

parámetros intrínsecos se refieren a dicha matriz, la cual se le debe aplicar al objeto para ligar los marcos de referencia.

2.4.2. Cálculo de la matriz de proyección K

Se supone que el sistema de imágenes está basado en el modelo de una cámara obscura, la proyección de espacio tridimensional al plano de la imagen bidimensional se puede expresar por la ecuación (2.6)

$$\lambda \mathbf{p} = M\mathbf{P} \tag{2.6}$$

en la cual M es la matriz de transformaciones aplicada a los puntos P que corresponden a los puntos de la imagen en el plano real, para generar $\mathbf{p}$ que denota las coordenadas de los puntos en el plano de la imagen. Es necesario aplicar un escalamiento λ a los puntos $\mathbf{p}$. La matriz M se puede descomponer de la siguiente forma:

$$M = K[R|t] \tag{2.7}$$

La matriz $[R|t]$ es de orden 3×4 que determina la orientación y la posición de la cámara en el espacio 3D, R es la matríz de orden 3×3 que contiene las rotaciones en los ejes x, y y z y t es el vector de traslaciones. K es la matriz de calibración de orden 3×3 que define la proyección en perspectiva de la cámara. La matriz de calibración K, es de la forma (2.8)

$$K = \begin{pmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.8}$$

donde u_0 es el centro de la imagen en el eje x y v_0 es el centro de la imagen en el eje y.

Tenemos también que la homografía H (que es la correspondencia de los puntos del modelo y los puntos del marco capturado), que es una matriz de orden 3×3, va a ser igual a la matriz de transformaciones M, por lo tanto:

$$H = K[\mathbf{r}_1, \mathbf{r}_2, \mathbf{t}] \tag{2.9}$$

Se toman únicamente las dos primeras columnas de la matriz R, que corresponden a los ejes x y y debido a que el patrón es un plano situado en los ejes x, y, por lo que la matriz de rotación en vez de ser de orden 3×3 es de orden 3×2. La ecuación (2.9) se puede descomponer en la ecuación (2.10), siendo $\mathbf{h_1}, \mathbf{h_2}$ y $\mathbf{h_3}$ las columnas que componen la homografía, colocando el escalamiento λ. Es equivalente a la ecuación (2.11), solo que los valores de la matriz de proyección K y de λ se han pasado del otro lado de la ecuación.

$$[\mathbf{h_1}, \mathbf{h_2}, \mathbf{h_3}] = \lambda^{-1} K[\mathbf{r_1}, \mathbf{r_2}, \mathbf{t}] \tag{2.10}$$

$$\lambda K^{-1}[\mathbf{h_1}, \mathbf{h_2}, \mathbf{h_3}] = [\mathbf{r_1}, \mathbf{r_2}, \mathbf{t}] \tag{2.11}$$

Tenemos que los vectores $\mathbf{r_1}$ y $\mathbf{r_2}$ son ortogonales, debido a que son perpendiculares uno del otro, por lo que tenemos la ecuación (2.12)

$$[K^{-1}\mathbf{h_1}]^T[K^{-1}\mathbf{h_2}] = 0 \tag{2.12}$$

$$\mathbf{h_1}^{\mathbf{T}} K^{-T} K^{-1} \mathbf{h_2} = 0 \tag{2.13}$$

Retomando la ecuación (2.8) se realiza el cálculo de K^{-1} que es requerido, dicho cálculo se puede observar en la ecuación (2.14)

$$K^{-1} = \begin{pmatrix} \frac{1}{f} & 0 & \frac{-u_0}{f} \\ 0 & \frac{1}{f} & \frac{-v_0}{f} \\ 0 & 0 & 1 \end{pmatrix} = \frac{1}{f}\begin{pmatrix} 1 & 0 & -u_0 \\ 0 & 1 & -v_0 \\ 0 & 0 & f \end{pmatrix} \tag{2.14}$$

Sustituyendo el valor de K de la ecuación (2.14) en la ecuación (2.13), tenemos:

$$\frac{1}{f^2}(h_{11}h_{21}h_{31}) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -u_0 & -v_0 & f \end{pmatrix} \begin{pmatrix} 1 & 0 & -u_0 \\ 0 & 1 & -v_0 \\ 0 & 0 & f \end{pmatrix} \begin{pmatrix} h_{12} \\ h_{22} \\ h_{32} \end{pmatrix} = 0 \tag{2.15}$$

Resolviendo de la ecuación (2.15) $K^{-T}K^{-1}$ tenemos:

$$\frac{1}{f^2}(h_{11}h_{21}h_{31}) \begin{pmatrix} 1 & 0 & -u_0 \\ 0 & 1 & -v_0 \\ -u_0 & -v_0 & u_0^2 + v_0^2 + f^2 \end{pmatrix} \begin{pmatrix} h_{12} \\ h_{22} \\ h_{32} \end{pmatrix} = 0 \tag{2.16}$$

Para un mejor manejo de la solución a la ecuación (2.16) realizamos la sustitución de la ecuación (2.17).

$$\frac{1}{f^2}(h_{11}h_{21}h_{31}) \begin{pmatrix} 1 & 0 & w_{13} \\ 0 & 1 & w_{23} \\ w_{13} & w_{23} & w_{33} \end{pmatrix} \begin{pmatrix} h_{12} \\ h_{22} \\ h_{32} \end{pmatrix} = 0 \tag{2.17}$$

donde hacemos la sustitución del centro de la imagen y el valor del elemento $(K^{-1}K^{-T})_{33}$ por las ecuaciones (2.18)

$$\begin{aligned} w_{13} &= -u_0 \\ w_{23} &= -v_0 \\ w_{33} &= u_0^2 + v_0^2 + f^2 \end{aligned} \tag{2.18}$$

Resolviendo $\mathbf{h_1}^T K^{-T} K^{-1}$ de la ecuación (2.17) tenemos lo siguiente:

$$[h_{11} + h_{31}w_{13}, h_{21} + h_{31}w_{23}, h_{11}w_{13} + h_{21}w_{23} + h_{31}w_{33}] \begin{pmatrix} h_{12} \\ h_{22} \\ h_{32} \end{pmatrix} = 0$$

y resolviendo la ecuación (2.17) completamente tenemos:

$$(h_{11} + h_{31}w_{13})h_{12} + (h_{21} + h_{31}w_{23})h_{22} + (h_{11}w_{13} + h_{21}w_{23} + h_{31}w_{33})h_{32} = 0$$

despejando w_{33} de la ecuación anterior tenemos la ecuación (2.19)

$$w_{33} = \frac{-(h_{11} + h_{31}w_{13})h_{12} - (h_{21} + h_{31}w_{23})h_{22} - (h_{11}w_{13} + h_{21}w_{23})h_{32}}{h_{32}h_{31}} \tag{2.19}$$

Sustituimos la ecuacion (2.18) en (2.19) y obtenemos lo siguiente:

$$u_0^2 + v_0^2 + f^2 = \frac{-(h_{11} + h_{31}w_{13})h_{12} - (h_{21} + h_{31}w_{23})h_{22} - (h_{11}w_{13} + h_{21}w_{23})h_{32}}{h_{32}h_{31}} \tag{2.20}$$

Lo que estamos buscando es el valor de f, por lo que realizamos el despeje en ecuacion (2.20) de la siguiente manera:

$$f^2 = w_{33} - v_0^2 - v_0^2 \tag{2.21}$$

Es muy importante que el valor de f^2 sea positivo, de lo contrario no se podrá calcular f porque requerimos obtener $\sqrt{f^2}$. Ésta validación es muy importante, ya que en caso de no poder obtener el valor de f, el programa no podrá continuarse ejecutando, y tendrá que esperarse a tener nuevos centroides para hacer nuevamente el cálculo de K.

2.4.3. Cálculo de la matriz de rotación y translación Rt

Una vez que se ha calculado la matriz de proyección K, se requiere hacer el cálculo de la matriz de rotación, así como del vector de translación, para lo cual, retomaremos la ecuación (2.10), y tenemos que:

$$\lambda[\mathbf{r_1}, \mathbf{r_2}, \mathbf{t}] = K^{-1}H = [\hat{\mathbf{r}}_1, \hat{\mathbf{r}}_2, \hat{\mathbf{t}}]$$

Y haciendo uso de la ortogonalidad de los vectores, tenemos:

$$\lambda = |\hat{\mathbf{r}}_1| = |\hat{\mathbf{r}}_2|$$

Una vez que calculamos el valor de λ podemos calcular el valor de $\mathbf{r_1}$, $\mathbf{r_2}$, $\mathbf{r_3}$ y t:

$$\begin{aligned} \mathbf{r_1} &= \frac{\hat{\mathbf{r}}_1}{\lambda} \\ \mathbf{r_2} &= \frac{\hat{\mathbf{r}}_2}{\lambda} \\ \mathbf{t} &= \frac{\hat{\mathbf{t}}}{\lambda} \\ \mathbf{r_3} &= \mathbf{r_1} \times \mathbf{r_2} \end{aligned}$$

Una vez calculados los vectores $\mathbf{r_1}$, $\mathbf{r_2}$ y $\mathbf{r_3}$ se hace la matriz $\mathbf{R}$ la cuál está definida por dichos vectores:

$$R = [\mathbf{r_1}, \mathbf{r_2}, \mathbf{r_3}]$$

Para obtener los ángulos de rotación del objeto, se hizo la descomposición en valores singulares de la matriz R, para ello se utilizó una biblioteca llamada *GSL* [8]. El cálculo de la matriz de rotación se hizo de la siguiente manera:

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

Siendo U y V ortogonales

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

De la matriz R se pueden obtener los ángulos α, β y γ de la siguiente manera:

$$\begin{aligned} \alpha &= \tan^{-1} \frac{r_{32}}{-r_{31}} \\ \beta &= \cos^{-1} r_{33} \cdot \\ \gamma &= \tan^{-1} \frac{r_{23}}{-r_{13}} \cdot \end{aligned}$$

La matriz de rotación está conformada de la siguiente manera:

$$R = R_z(\alpha) \cdot R_y(\beta) \cdot R_z(\gamma)$$

donde cada una de las matrices de rotación se define:

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

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

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

$$R = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Capítulo 3

Sistema de desarrollo

En este capítulo se explica el procedimiento necesario para realizar una aplicación de RA, tal como se muestra en la Figura 3.1. DVGrab maneja dos hilos para guardar los marcos en DV, la aplicación tiene dos hilos más, uno que se encarga de desplegar el objeto cada $\frac{1}{25}$ segundos para evitar la pérdida de marcos en este proceso, mientras que el procesamiento, así como la visualización del objeto virtual se realiza en otro hilo, con la finalidad de que si existe algún pequeño retraso no se vea en el video, sino en el seguimiento del objeto al marco, haciendo casi imperceptible dicho retraso. Para procesar los marcos es necesario que se haya hecho la lectura de uno con anterioridad, para evitar este procesamiento con un buffer vacío, se realiza sólo cuando existe algo en dicho buffer, para controlar esto se utiliza una variable de decisión de qué buffer se utilizará para el procesamiento, la cual se inicializa con un valor diferente a 0 y 1, para no ser tomada en cuenta en este caso.

Todo el procedimiento se explica con más detalle en las secciones siguientes.

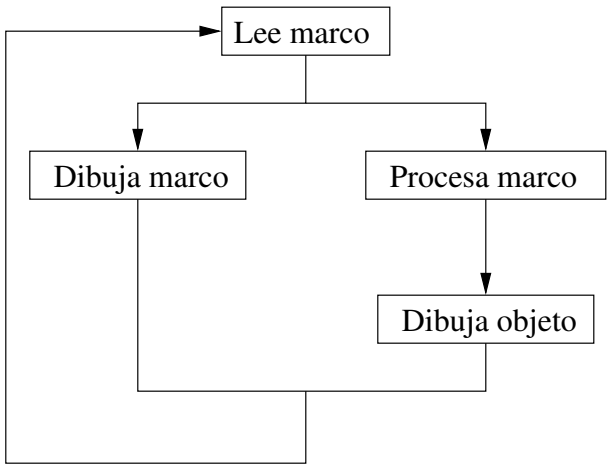


Figura 3.1: Arquitectura lógica general para el sistema de RA

3.1. Extracción del marco

La adquisición del video se realizó con una cámara de VD utilizando como salida el puerto *Firewire* con formato VD, la transmisión de este formato se realiza a una velocidad de 25 marcos por segundo, se requiere extraer cada uno de estos marcos, para lo cual se utilizó *DVGrab* que es una biblioteca para la captura de VD y audio del puerto IEEE1394.

Se utilizó código fuente de la versión 3.5 de *DVGrab*, que fue descargado de la página de *Linux Video Digital* [2]. Para su correcto funcionamiento se requieren las bibliotecas que se mencionan a continuación.

- *libraw1394*. Permite el acceso directo a la información del dispositivo que se encuentra conectado al pierto IEEE1394.
- *libavc1394*, Permite el control de audio y video. Esta biblioteca es utilizada como una interfaz para obtener audio y video del puerto IEEE1394.
- *libdv*. Contiene los códecs necesarios para codificar o decodificar el formato DV.

Las clases utilizadas fueron:

- *Frame*, que contiene la información de audio y video por cada marco de la secuencia de video. Contiene métodos para obtener información de tamaño y contenido de los marcos. Esta clase fue utilizada para obtener los marcos y posteriormente convertirlo a formato RGB. Las funciones de la clase utilizadas se mencionan a continuación.

  GetFrame(). Esta función verifica la posición del buffer de memoria y extrae una instancia de éste. Dicha instancia es de tipo Frame.

  *ExtractRGB(void *rgb)*. *DVGrab* extrae los marcos en formato DV, pero para su uso requerimos que se encuentren en formato RGB, por lo que se utiliza esta función que precisamente hace la decodificación en RGB de los píxeles de la imagen.

  GetHeight(). Extrae la altura en píxeles del marco.

  GetWidth(). Extrae el ancho en píxeles del marco.

  IsComplete(). Verifica que el marco extraído se encuentre completo para tomar la decisión de guardarlo o tomar el siguiente marco.

- *DVgrab*, que hace uso de la clase *Frame* de la cual extrae los marcos del flujo de video a la velocidad de $\frac{1}{25}$ marcos por segundo.
- *IEEE1394Reader*, que realiza la conexión con el puerto IEEE1394 y ejecuta el hilo donde se va a realizar la extracción de cada uno de mis marcos, los cuales serán almacenados en la clase *Frame*.

En la Figura 3.2 se puede observar el funcionamiento de *DVGrab*, para el almacenamiento de los marcos cuenta con una cola de tamaño 50, lo cual equivale a dos segundos de video, en dicha cola se va metiendo el marco extraído y al hacer uso de él es eliminado. En caso de que el tamaño supere los 50 marcos almacenados se ignora el extraído hasta que exista un lugar disponible, cabe mencionar que nunca se dio esta situación. Una vez extraído el marco se realiza la conversión a RGB, tomando en cuenta la condición de cuál será utilizado, lo cual se puede observar en el código presentado más adelante.
Una vez extraído y decodificado cada marco, se almacenó en un *buffer*, del cual se realizó una

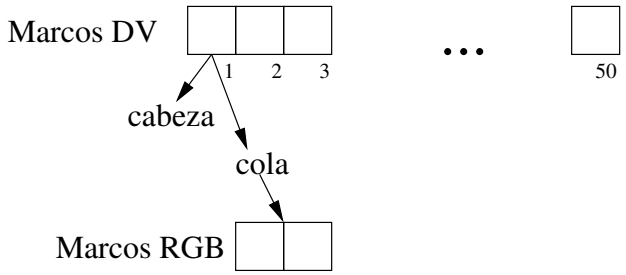


Figura 3.2: Funcionamiento de DVGrab

lectura para su desplegado en pantalla. Para evitar la lectura de marcos incompletos, se utilizaron dos *buffers*, mientras que se almacena en uno, se lee del otro y viceversa.

Todo el código referente a la captura de video se encuentra en la carpeta llamada *dvgrab*, la cual contiene los códigos de la biblioteca con el mismo nombre.

Al código original de *DVGrab* le fueron hechas algunas modificaciones, principalmente para la decodificación y el almacenamiento de los marcos. La función Lectura de marcos se muestra en el Apéndice A.

Al extraer cada uno de los marcos, se hizo la decodificación, almacenándolo en uno de los dos buffers disponibles, pero protegiendo cada uno de estos dos buffers, así como la variable que indica en cual de ellos será almacenado el marco para evitar que se guarde uno sin que el anterior haya sido terminado de guardar.

La forma en la que se protegieron los buffers fue la siguiente: La parte protegida fue tanto el buffer como una variable indicando cual de ellos está libre para guardar. El código Almacenamiento de marcos se muestra en el Apéndice A.

En el algoritmo 2 se puede observar como es que se realiza el almacenamiento de los marcos en dos *buffers*.

Algoritmo 2 Algoritmo para el almacenamiento del marco en uno de los buffers protegidos

Entrada: Marco de video (*marco*)
Salida: Marco almacenado en uno de los buffers ($buffer[0]$ o $buffer[1]$)

```
1: Inicialización de variable protegida count → 0
2: if count == 1 then
3:    Bloqueo de variable protegida
4:    buffer[0] → ConvierteRGB(marco)
5:    count → 0
6:    Desbloqueo de variable protegida
7: else
8:    Bloqueo de variable protegida
9:    buffer[1] → ConvierteRGB(marco)
10:   count → 1
11:   Desbloqueo de variable protegida
12: end if
```

La idea de tener un contador y protegerlo es que cuando el marco ha sido almacenado en uno de los buffers, se cambia el valor del contador (hasta que se ha terminado de escribir) y la siguiente vez que se va a escribir un marco se escribe en el otro buffer, lo cual es controlado por el contador.

Este procedimiento se realiza en un hilo que estará obteniendo cada uno de los 25 marcos por segundo, decodificandolo y guardándolo, para posteriormente ser procesado.

Los marcos del video fueron desplegados en OpenGL utilizando **glDrawPixels()**, la cual dibuja el arreglo de píxels que le indiquemos. Ésta función va a dibujar el marco que no esté siendo procesado, se realiza de esta manera para evitar que tome y dibuje marcos incompletos. La función que dibuja el marco se llama **drawFrame()**.

Para el desplegado de los marcos se emitió una señal cada $\frac{1}{25}$ segundos, con la finalidad de dibujar cada uno de los marcos extraídos, evitando una visualización con retrasos. El código para emitir la señal se muestra en el Apéndice A.

El dibujado de los marcos se realizó con OpenGL. La función que dibuja el marco se encuentra también en la clase **canvas** y se llama **drawFrame()**.

El desplegado del marco se realizó utilizando la función **glDrawPixels()** de OpenGL, se utilizó una condición para tomar la decisión de que buffer dibujar dependiendo del valor de count, se realiza una copia del marco elegido y posteriormente se dibuja dicha copia. El código para dibujado del objeto Se muestra en el Apéndice A.

En el algoritmo 3 se puede observar que el dibujado del objeto se realiza de manera similar a la escritura en uno de los *buffers* para evitar tomar un marco incompleto.

Algoritmo 3 Algoritmo para el dibujado del marco

Entrada: Marco de video (*marco*)
Salida: Marco dibujado cada $\frac{1}{25} segundos$

```
1: if count == 1 then
2:    Bloqueo de variable protegida
3:    copia a arregloDibuja → buffer[1]
4:    Desbloqueo de variable protegida
5: else
6:    Bloqueo de variable protegida
7:    copia a arregloDibuja → buffer[0]
8:    Desbloqueo de variable protegida
9: end if
10: Dibuja arregloDibuja
```

Cuando la variable *count* tiene el valor de 1, el marco extraído se está almacenando en el buffer 0, por lo tanto se copia el buffer 1 para ser dibujado, cuando tiene el valor de 0, el marco extraído se está almacenando en el buffer 1, por lo que se hace la copia del buffer 0 para dibujarla.

Para evitar tomar un marco incompleto para el procesamiento digital de imágenes se realiza una copia del buffer que no está siendo utilizado para escribir, de igual manera que se hace para tomar el marco que será dibujado.

3.2. Procesamiento del marco

El procesamiento se realizó en un hilo, el cual toma uno de los buffers (en el que no se está guardando), mientras que por otro lado, se dibuja cada uno de los marcos en OpenGL, para que sin importar que el procesamiento tarde más de $\frac{1}{25}$ segundos, el video se pueda observar sin retrasos. Sin embargo, el procesamiento realizado es secuencial, ya que se debe terminar uno de los procesos en el marco para continuar con el siguiente. En la Figura 3.3 se puede observar el procesamiento requerido.

El procesamiento del marco se encuentra en la clase **canvas**, la función que lo realiza es **procesaFrameRun()**. Para tomar la decisión de que marco procesar se utilizó la misma condición que fue usada para dibujar el marco, el código para el procesamiento del marco se puede observar en el Apéndice A.

El algoritmo 4 muestra la manera en la que se hace el procesamiento al marco del buffer elegido.

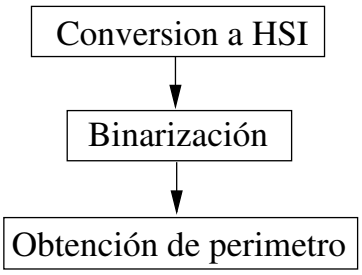


Figura 3.3: Diagrama del procesamiento digital de imágenes aplicado

Algoritmo 4 Algoritmo para el dibujado del marco

Entrada: Marco de video (*marco*)
Salida: Marco procesado

```
1: if count == 1 then
2:    Bloqueo de variable protegida
3:    copia a arregloProcesa → buffer[1]
4:    Desbloqueo de variable protegida
5: else
6:    Bloqueo de variable protegida
7:    copia a arregloProcesa → buffer[0]
8:    Desbloqueo de variable protegida
9: end if
10: Procesa arregloProcesa
11: Dibuja objeto virtual en el marco
```

3.3. Visualización del objeto virtual

Una vez calculadas las matrices de proyección, traslación y rotación, se puede colocar la imagen virtual sobre el marco de video, para que sea posible introducir cualquier objeto a la aplicación, se utilizaron las funciones de OpenGL para aplicar cada una de las matrices al objeto.

- *glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z)*. Esta función sirve para aplicar rotaciones en OpenGL, y la manera en la que funciona es la siguiente: en el primer parámetro va colocado el grado en el que queremos girar el objeto, los siguientes tres parámetros pueden tomar valor de 0 y 1, se colocan con valor 1 si se quiere activar ese eje para la rotación y en 0 si se requiere que en ese eje no se gire el objeto. El segundo parámetro corresponde al eje x, el tercer parámetro corresponde al eje y y el cuarto parámetro corresponde al eje z.

- *glTranslatef(GLfloat x, GLfloat y, GLfloat z)*. Esta función sirve para aplicar traslaciones en OpenGL, y la manera en la que funciona es la siguiente: En el primer parámetro

se coloca la traslación que se le quiere aplicar al eje x, en el segundo parámetro va la traslación para el eje y y en el tercer parámetro la traslación para el eje z.

- *glLoadMatrixf(const GLfloat * m)*. Esta función sirve para cargar la matriz que será aplicada a la imagen virtual, en este caso se utiliza para aplicar la matriz de proyección. La matriz que se carga debe ser de 16 elementos, ya que la matriz de proyección es de 4×4.

Las transformaciones pueden ser aplicadas en diversos modos, para lo cual se utiliza glMatrixMode que establece el modo en el cual se va a trabajar. Los modos que se pueden utilizar son los siguientes:

- GL_MODELVIEW
- GL_PROJECTION
- GL_TEXTURE
- GL_COLOR

Para aplicar las matrices de rotación y traslación, se utiliza GL_MODELVIEW, mientras que para aplicar la matriz de proyección se utiliza GL_PROJECTION.

Una vez aplicadas las correspondientes matrices, se puede colocar cualquier objeto de una manera sencilla, sin tener que realizar modificaciones en el código además del cambio de objeto. Para aplicar los parámetros intrínsecos y extrínsecos al objeto de manera independiente a la creación del objeto se creó la matriz de proyección según la Figura 3.4 en OpenGL de la siguiente manera:

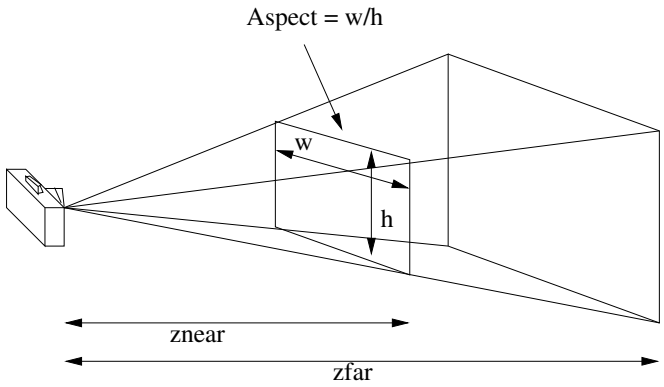


Figura 3.4: Proyección en OpenGL

Para la creación de la matriz de proyección, se utilizó el valor de f calculado para la creación

de la matriz K, la matriz de proyección se creó de la siguiente forma:

$$M = \begin{pmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{zFar+zNear}{zNear-zFar} & \frac{2 \cdot zFar \cdot zNear}{zNear-zFar} \\ 0 & 0 & -1 & 1 \end{pmatrix} \tag{3.1}$$

Para poder visualizar el objeto es necesario realizar una escala por lo menos por k:

$$k = \frac{zNear}{|t_3| - \frac{2}{\sqrt{3}}} \tag{3.2}$$

y el plano de corte trasero es:

$$zFar = \frac{4 \cdot k}{\sqrt{3} + zNear} \tag{3.3}$$

Capítulo 4

Aplicación 1, autocalibración con un patrón plano

En esta sección se describe el procedimiento que se llevó a cabo para la creación de sistemas de RA, utilizando un patrón de 16 círculos en un plano, formando un cuadro de tamaño 4×4, el cual se muestra en la Figura 4.1.

El procedimiento para la creación de esta aplicación se puede observar en la Figura 4.2 de manera muy general.

Como primer paso se extrae cada uno de los marcos que componen el video y se decodifica en el formato RGB para poder ser desplegado en OpenGL. Posteriormente se realiza el procesamiento, el cual consiste en la conversión a HSI, la binarización y la extracción de centroides. Es necesario que los centroides estén ordenados, por lo que se debe realizar un ordenamiento. Posteriormente se debe realizar la calibración, la visualización del objeto y finalmente las ventanas de seguimiento para agilizar el procesamiento.

4.1. Lectura de marco

La lectura del marco se realizó como se describió en el Capítulo 3, con el código modificado de *DVGrab* se realiza la lectura de cada marco, la decodificación a RGB y se almacena en uno de los dos *buffers*, los cuales están protegidos para que escriba intercaladamente en uno o en otro, sin dejar que escriba en uno solo dos veces seguidas para evitar que se encimen marcos.

Para el desplegado se envía una señal cada $\frac{1}{25}$ segundos para que el marco se dibuje en OpenGL, sin importar si ya se terminó o no de procesar. El desplegado del marco se hace independientemente del procesamiento para evitar que se vean retrasos en el video.

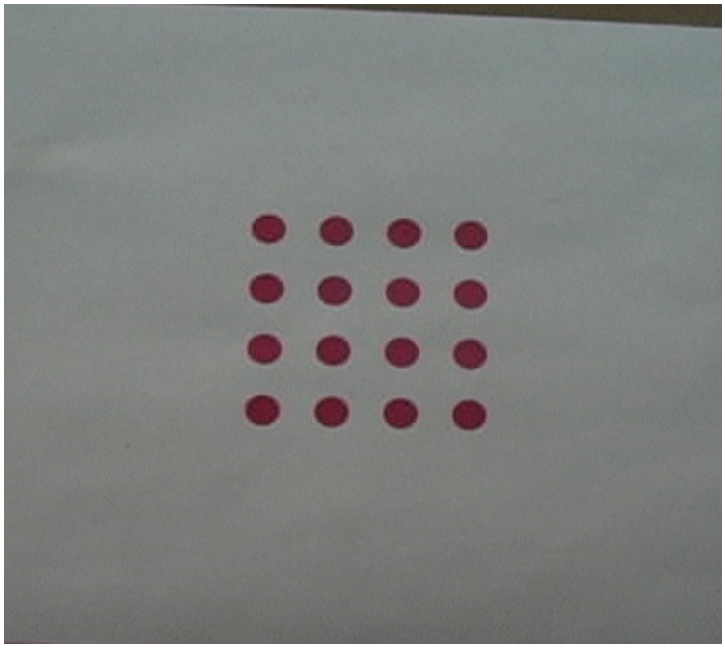


Figura 4.1: Patrón plano

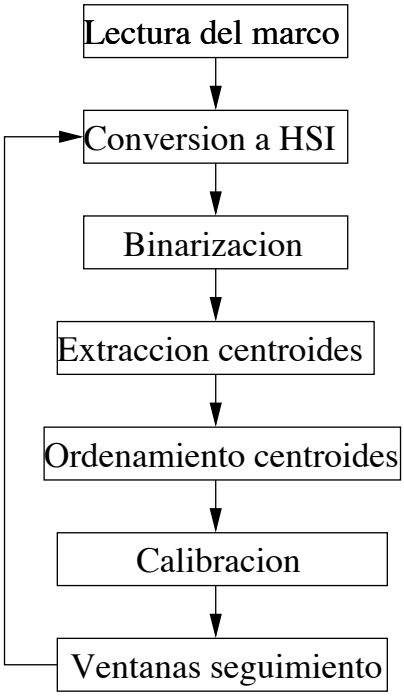


Figura 4.2: Diagrama principal

4.2. Procesamiento del marco

4.2.1. Conversión a HSI

La información de cada píxel que conforma un marco se encuentra en formato RGB (*Red, Green, Blue*) una vez de ser decodificada, como este modelo no funciona para realizar la segmentación por color es necesario hacer la conversión al modelo HSI, de donde únicamente serán utilizadas las componentes H, que nos va a dar el ángulo del color del cual se trata y S, que nos va a dar la pureza del color para evitar tomar un tono de rojo muy diluido.

Se utilizó el color rojo para los patrones debido a que en la escena en la cual se realizaron los experimentos no existen objetos de ese color, además de que es difícil que un objeto de color rojo puro entre a la escena.

El modelo HSI corresponde con la forma en cómo los humanos describen e interpretan el color. El valor de H se encuentra en grados y para el color rojo se tomaron los valores 0^{o} a 10^{o} y 300^{o} a 360^{o}.

Una vez hecha la conversión a HSI se identificaron los píxeles con ángulo perteneciente al color rojo, pero además con una pureza mayor al 20 % para evitar la extracción de objetos en color rojo muy diluido. Los píxeles que cumplieron con ambas condiciones fueron colocados en la imagen almacenada para el procesamiento con el valor perteneciente al color negro (0), y los que no cumplieron con alguna o ninguna de las dos condiciones se colocaron con el valor del color blanco (255). Con esto obtenemos la figura en color blanco y negro con únicamente los objetos de color rojo.

El código utilizado para la conversión a HSI, así como la binarización se presenta en el Apéndice A.

4.2.2. Erosión

Pueden existir pequeños píxeles de ruido en la imagen que sean de color rojo, por lo que es necesario limpiarla aún más, para ello se realizan dos erosiones sobre la imagen resultante de la binarización, para desaparecer los puntos de ruido. No se realiza una sola erosión, ya que esto solo eliminaría los puntos muy pequeños, pero si se quiere eliminar una mayor cantidad de ruido es que se tienen que aplicar dos erosiones.

La función que realiza la erosión se encuentra en la librería *scimagen* [11], el código se puede observar en el Apéndice A.

4.2.3. Extracción de centroides

Con una imagen sin ruido, y con la menor cantidad posible de objetos dentro del marco se puede proceder a la extracción de los centroides que componen el patrón empleado, para ello no es necesario extraer el perímetro de los objetos. Para esto se utiliza *scimagen*, y nos va a regresar los centroides de los 16 círculos que componen el patrón de la Figura 4.1.

Únicamente la primera vez se extraen 16 centroides de una sola imagen, las iteraciones posteriores se busca un centroide de cada ventana de seguimiento, agilizando el procedimiento, ya que la extracción de varios objetos en una sola imagen extrae un objeto y posteriormente lo borra para hacer las demás extracciones.

4.2.4. Ordenamiento de centroides

Los puntos no siempre son obtenidos en el orden en que son requeridos, por lo tanto es necesario realizar un ordenamiento de dichos puntos.

En el algoritmo 5 se puede observar como es que se realiza el ordenamiento de los centroides.
Lo primero que se necesita es la extracción que se hace de todas las figuras que conforman la imagen, y la extracción del centroide de cada una de estas figuras (16 son los centroides requeridos, ya que el patrón tiene 16 puntos). Una vez extraídos los centroides lo que se debe hacer es buscar el punto que se encuentre más arriba de la imagen, el cuál será considerado el primero. Posteriormente se buscan los dos puntos más cercanos, es necesario buscar los dos y no sólo el más cercano, debido a que por la posición del patrón puede ser que el más cercano no sea precisamente el siguiente punto. Una vez que se tienen los dos puntos más cercanos se toma como siguiente el punto que se encuentre más arriba entre los dos seleccionados, v. g. el que tenga una menor y. Se repite lo anterior para obtener el segundo, tercer y cuarto punto.

Una vez que se obtuvieron los cuatro puntos, deben ser borrados, para ello se tiene una variable de control, la cual inicia siendo 0, y una vez que se van guardando los índices dentro del arreglo de valores ordenados, se va colocando dicha variable en 1, lo cual quiere decir que ya no deben ser tomados en cuenta para futuros ordenamientos.

Después de haber borrado los puntos encontrados se repite desde la búsqueda del punto más arriba que es considerado el primero, y la búsqueda de los otros tres puntos. Este procedimiento se repite en total tres veces para obtener los primeros 12 puntos. Los cuatro puntos faltantes ya no se buscarán de la misma manera, debido a que como solo quedan dichos puntos ya no se deben considerar los dos puntos más cercanos, sino únicamente el punto más cercano, de ésta manera se encuentran el segundo, tercer y cuarto puntos.

El ordenamiento anterior funciona bien para patrones sin inclinación o inclinados hacia la derecha, como se muestra en la Figura 4.3 (a), en caso de que el patrón esté inclinado hacia

Algoritmo 5 Ordenamiento de centroides

Entrada: Centroides de los 16 círculos del patrón
Salida: Centroides ordenados

1: Inicializar arreglo $a[16]$
2: inicializar $indice \leftarrow 0$
3: **for** $i = 0$ to $i = 3$ **do**
4: $a[indice] \leftarrow$ indice centroide con menor y
5: peso del centroide con menor $y \leftarrow 1$
6: **for** $j = 0$ to $j = 3$ **do**
7: $p1$ y $p2$ los dos centroides más cercanos de $a[indice]$ con $peso = 0$
8: $indice \leftarrow indice + 1$
9: **if** $p1.y > p2.y$ **then**
10: $a[indice] \leftarrow p2.indice$
11: $p2.peso \leftarrow 1$
12: **else**
13: $a[indice] \leftarrow p1.indice$
14: $p1.peso \leftarrow 1$
15: **end if**
16: **end for**
17: $indice \leftarrow indice + 1$
18: **end for**
19: $a[indice] \leftarrow$ indice centroide con menor y
20: $indice \leftarrow indice + 1$
21: **for** $i = 0$ to $i = 3$ **do**
22: $a[indice] \leftarrow$ centroide más cercano a $a[indice - 1]$ con $peso = 0$
23: **end for**
24: **if** $centroides[a[0]].x >$centroides[a[1]].x **then**
25: Invertir puntos en cada fila
26: **end if**

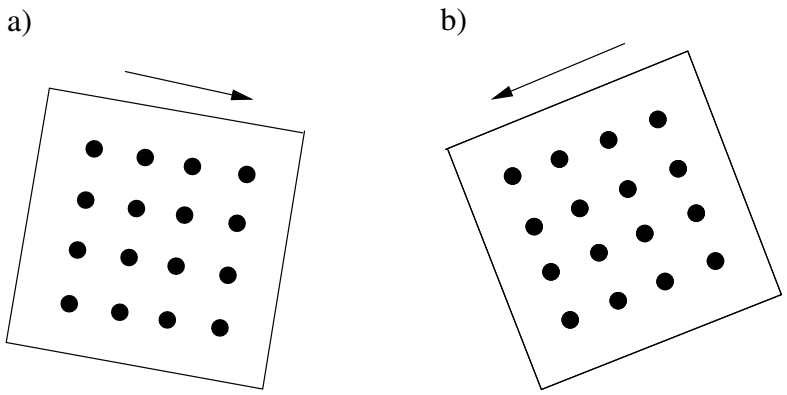


Figura 4.3: Patrón en diferentes inclinaciones: a) hacia la derecha y b) hacia la izquierda

la izquierda, los puntos se ordenarán como se indica en la Figura 4.3 (b), para identificar este caso, con los puntos ya ordenados, se compara el valor de x de los dos primeros puntos, si el segundo punto tiene un valor menor que el primero, entonces se deben invertir los puntos.

4.3. Calibración de la cámara

Una vez que se tienen los 16 centroides ordenados, se realiza el siguiente paso que es la calibración de la cámara, en la Figura 4.4 se puede observar como es que se realiza.

Lo primero que se hace es preguntar si la variable *recalibra* es verdadera o falsa (esta variable empieza siendo verdadera para que la primera vez se haga el cálculo de la matriz de proyección K), en el caso de ser verdadera, se realiza el cálculo de la matriz K (lo cuál se explicará más detalladamente), si no se puede calcular dicha matriz, la variable *recalibra* se coloca como verdadera, para que en la siguiente iteración, con centroides diferentes, se trate de calcular nuevamente esta matriz. Si la matriz K se puede calcular, se realiza lo mismo que si desde un principio *recalibrar* es negativo: Se hace el cálculo de la matriz Rt (de transformaciones geométricas) que contiene a los parámetros extrínsecos de la cámara, que son: matriz de rotación y vector de translación.

Es necesario verificar que los parámetros que se están obteniendo sean correctos, ya que en caso contrario se tendrá que calcular una nueva matriz K. Para realizar esta validación, se tienen 4 puntos de prueba, que son las esquinas del patrón. Las matrices obtenidas se aplican directamente a las coordenadas de cada uno de los puntos de prueba y se calcula el error, v. g. se obtiene la distancia de los puntos de prueba con los puntos originales del patrón y se almacena en la variable *error*.

Para el manejo de *error* se tiene una tolerancia, en caso de que el error supere dicha tolerancia se deberá calcular nuevamente la matriz K, para ello la variable *recalibra* se hace

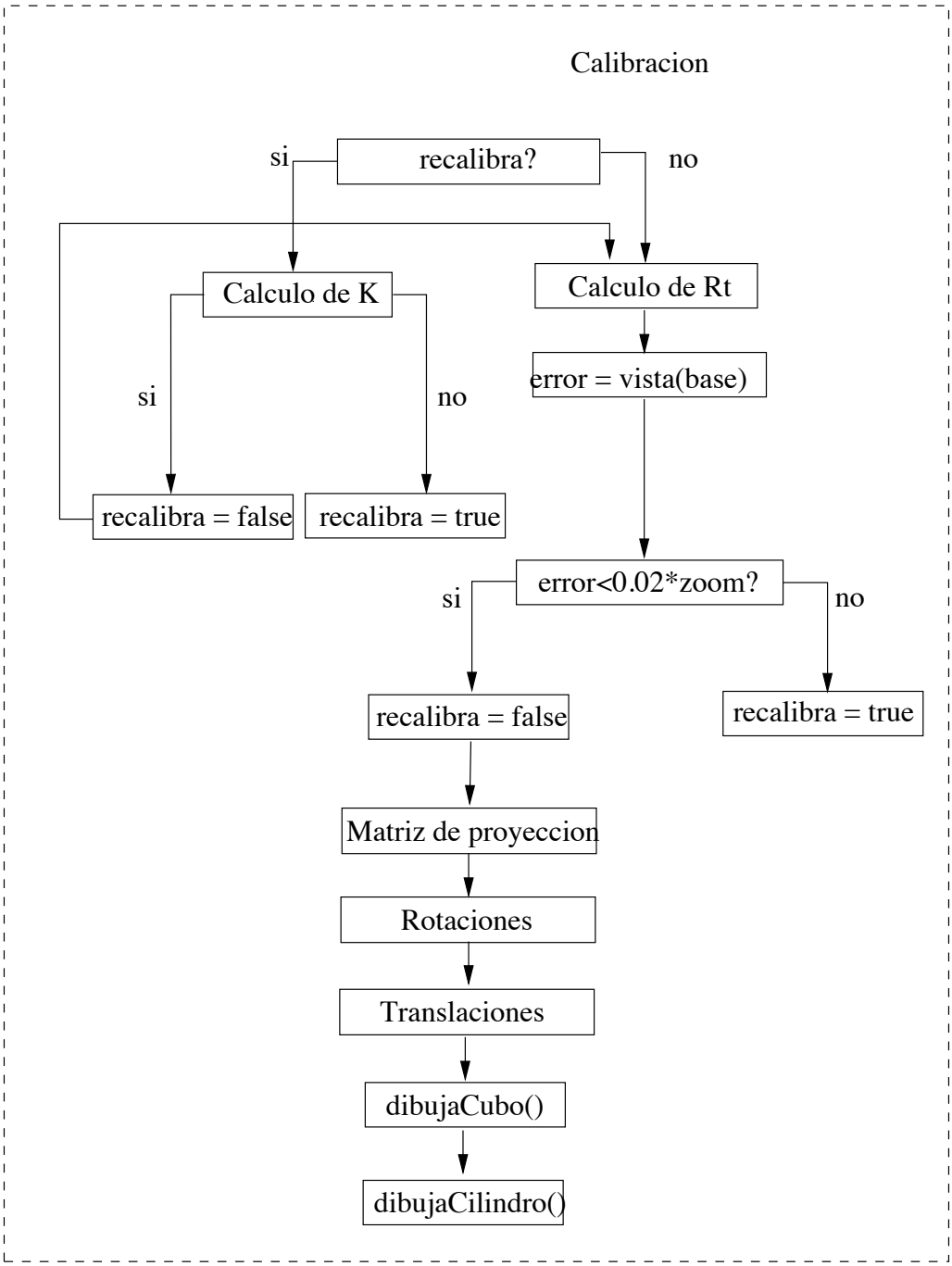


Figura 4.4: Calibración

verdadera, para que en la siguiente iteración se entre directamente al cálculo de la nueva K.

La tolerancia no siempre puede ser la misma, debido a que para una cámara muy alejada, el patrón será muy pequeño, por lo que una tolerancia muy grande puede dar resultados

incoherentes, mientras que para una cámara muy cercana al patrón, éste será de mayor tamaño, y entonces una tolerancia muy pequeña puede provocar que el cálculo de K se de en muchas iteraciones. Para solucionar este problema se hace uso de *zoom*, que es la distancia que hay entre los dos puntos cercanos del patrón. Entonces la tolerancia es multiplicada por la distancia de separación, siendo así más coherente tanto para grandes acercamientos como para una cámara muy alejada.

4.4. Cálculo de la homografía

La homografía es la transformación que cambia de un plano a otro, como se puede observar en la Figura 4.5, el cálculo de la homografía es necesario para obtener los parámetros intrínsecos y extrínsecos de la cámara.

El código para el cálculo de la homografía se puede observar en el Apéndice A.
Una vez calculada la homografía, se calcula el valor de f, para calcular la matriz de proyección K y con esa matriz, realizar el cálculo de la matriz de rotación y la matriz de traslación.

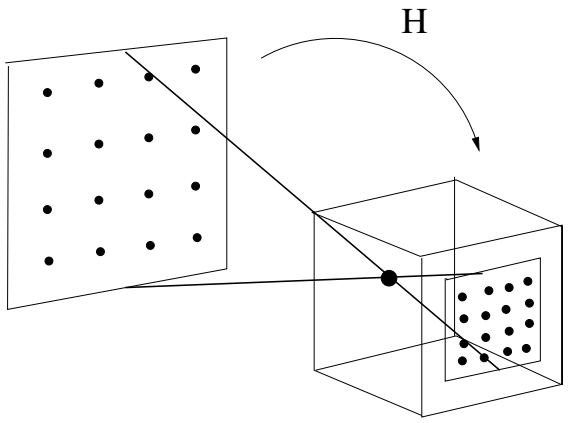


Figura 4.5: Homografía

4.5. Visualización del objeto virtual

Una vez calculados los ángulos de rotación, así como el vector de translación, se pueden aplicar directamente a los objetos de OpenGL con las siguientes funciones:

- glMultMatrix(). Se utiliza para cargar la matriz de proyección que se construye con la matriz K.

- glRotatef(). Se utiliza para realizar las rotaciones con los parámetros obtenidos al objeto virtual.
- glTranslatef(). Se utiliza para realizar las translaciones al objeto virtual.

Lo primero que se aplica es la matriz de proyección, para ello se utiliza la función *glMultMatriz(M)*, y posteriormente se realizan las rotaciones y translaciones.

Los objetos que se dibujan sobre el patrón, fueron creados en dos funciones diferentes:

- Cubo. Se crea un cubo de 3×3 de lado, con centro en $[1.5, 1.5, 1.5]$. Para su creación, se utilizaron las funciones de OpenGL, uniendo los vértices del cubo, los cuales fueron establecidos desde un principio y jamás fueron modificados durante la ejecución del programa
- Cilindro. El cilindro también fue creado utilizando las funciones e OpenGL para unir varios vértices, el cilindro fue creado sin tapas, y de igual manera que el cubo, siempre tuvo el mismo tamaño. El cilíndro trazado tiene como medidas: radio 0.5 y largo 5.0. El centro de éste cilíndro también se encuentra en los puntos $[1.5, 1.5, 1.5]$

En la Figura 4.6 se puede observar como se encuentra dibujado el cubo con el cilíndro en (a) Una vista superior y (b) una vista lateral.

El cilindro que está dentro del cubo, gira $2^{\underline{o}}$ cada marco sobre el eje z, generando así una pequeña animación, lo cual sirve para observar que se pueden realizar animaciones dentro de una aplicación de RA.

La adición de nuevos objetos se logra de una forma muy sencilla, ya que únicamente hay que colocar la función que dibuja el objeto dentro de *paint*, aunque es importante considerar que la calibración se realizó tomando en cuenta un modelo de 0 a 3, es decir, un cubo de dimensión 3 con centro en $[1.5, 1.5, 1.5]$.

En OpenGL se cargan las matrices de la siguiente manera:

```
glMultMatrixf(m);
glScalef(k,k,k);
glTranslatef(t_1, t_2, t_3);
glRotatef(gamma, 0.0, 0.0, 1.0);
glRotatef(beta, 0.0, 1.0, 0.0);
glRotatef(alfa, 0.0, 0.0, 1.0);
cubo();
cilindro(5.0, 0.5, 1.5, 1.5, 1.5, 20);
drawFrame(); //Dibujado del frame
```

La matriz m es modificada al realizar la recalibración, pero es cargada en la función **paintGl()**. La matriz se calcula de la siguiente manera:

```
/** Calculo de matriz m **/

zN = 1.0;
k = zN/(fabs(t_3) - 2.0/sqrt(3.0));
zF = 4*k/(sqrt(3)+zN);
d = zN-zF;
m[0] = f/asp; m[4] = 0.0; m[8] = 0.0;       m[12] = 0.0;
m[1] = 0.0;   m[5] = f;   m[9] = 0.0;       m[13] = 0.0;
m[2] = 0.0;   m[6] = 0.0; m[10] = (zF+zN)/d; m[14] = (2*zF*zN)/d;
m[3] = 0.0;   m[7] = 0.0; m[11] = -1.0;       m[15] = 1.0;
```

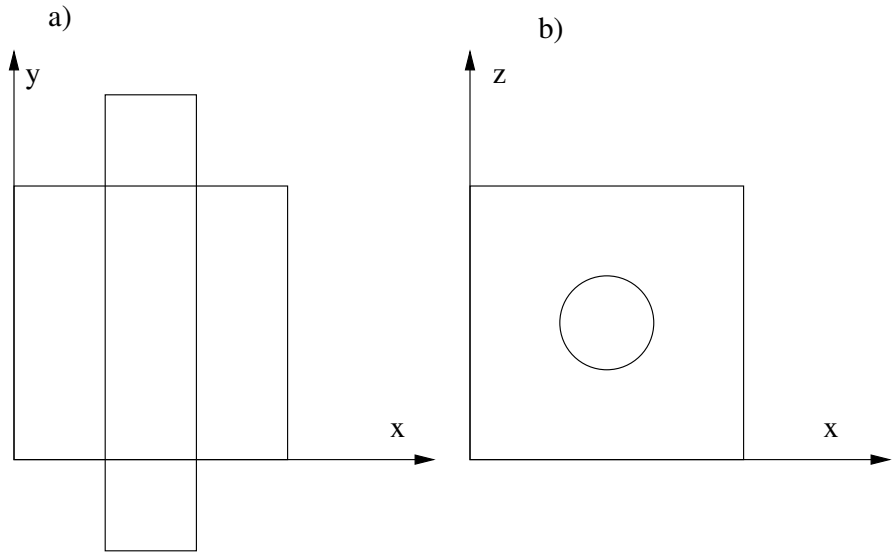


Figura 4.6: a) Objeto visto desde arriba, b) Objeto en una vista lateral

La figura dibujada sobre el patrón fue un cuboide, pero para demostrar que se pueden agregar otros objetos de manera sencilla, se dibujó además un cilindro girando dentro del cubo. La aplicación final se puede observar en la Figura 4.7.

4.5.1. Ventanas de seguimiento

El procesamiento debe ser lo menos tardado posible, esto con la finalidad de evitar la pérdida de información generando una reproducción de video poco eficiente. Para disminuir en lo máximo posible este tiempo de procesamiento, se realiza un seguimiento del objeto, por medio de ventanas.

La manera en la que funcionan las ventanas de seguimiento es la siguiente: Al realizar la extracción de los centroides, las próximas veces únicamente se buscará en un vecindario un poco mayor al lugar donde el centroide fue encontrado, esto para evitar el procesamiento de

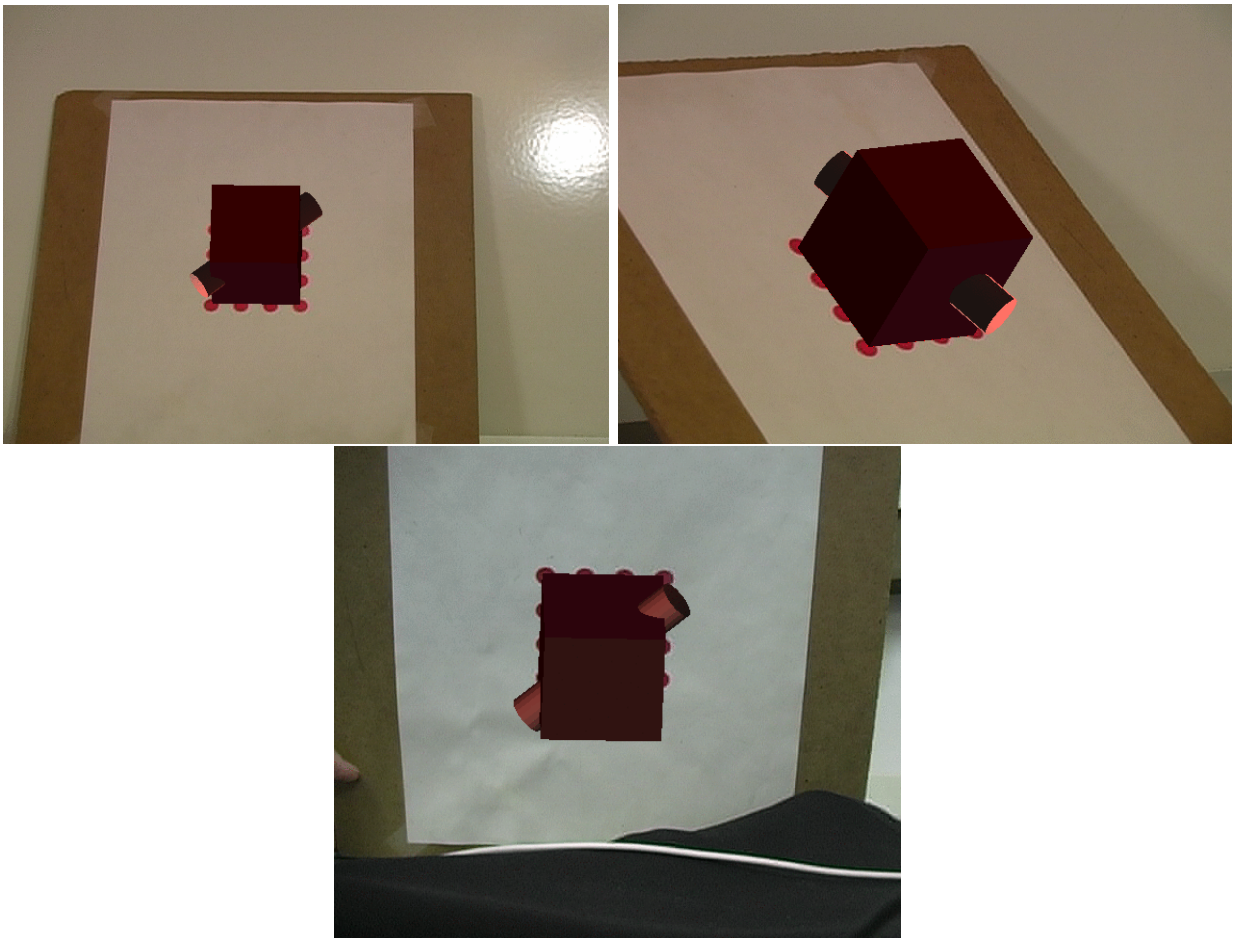


Figura 4.7: Aplicación final desde tres vistas diferentes

toda la imagen, generando así un considerable ahorro de tiempo. Para decidir el tamaño de las ventanas, al momento de estar ordenando los puntos se va a sacar una distancia promedio entre los puntos más cercanos. Esta distancia dará el tamaño de la nueva ventana.

Una vez que se tienen las ventanas de procesamiento, la imagen será procesada únicamente en dichas ventanas, v.g. se hará la lectura completa del marco, pero solo en las ventanas resultantes se realizará la conversión a HSI, la binarización y la búsqueda de los centroides.

El uso de estas ventanas no siempre funciona, debido a que en movimientos muy rápidos o bruscos del patrón, las ventanas resultantes pueden no contener la información necesitada, por lo que si pierde dichas ventanas se tiene que hacer el procesamiento sobre todo el marco para volver a localizar los centroides.

La manera en la que puede saber si se perdieron los centroides de las ventanas, es cuando en alguna de las ventanas no encuentra una imagen. El recortado de dichas ventanas

se realiza por medio de rangos; la primera vez que se procesa el marco, se procesa en el rango de x de 0 a 720, y en el rango de y de 0 a 480, cuando se tienen ventanas, se hace el procesamiento 8 veces pero únicamente en rango de x_1 a x_2 y de y_1 a y_2, teniendo valores diferentes para cada una de las imágenes.

En la Figura 4.8 se puede observar un ejemplo de las ventanas que se están utilizando. en (a) se observa la imagen con las 16 ventanas sobre los círculos, en (b) se observa un aproximado de la distancia media entre los puntos ordenados y en (c) se puede observar una ventana construida con el tamaño de dicha distancia.

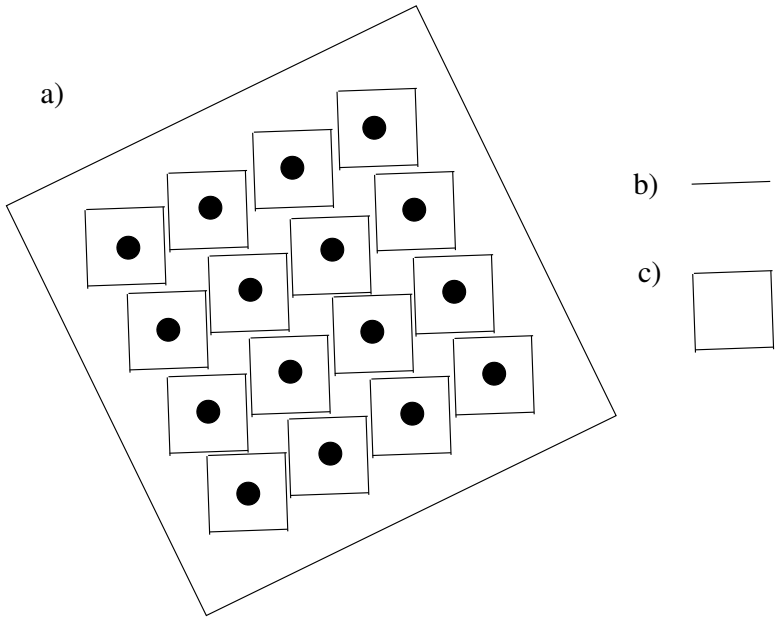


Figura 4.8: a) Patrón con la 16 ventanas dibujadas. b) Tamaño de la distancia promedio entre los puntos ordenados. c) ventana creada con el tamaño de la distancia promedio

Capítulo 5

Aplicación 2, autocalibración con un patrón de círculos concéntricos

En ésta sección se describe el procedimiento que se llevó a cabo para la creación de sistemas de RA, utilizando un patrón de dos círculos concéntricos. El patrón utilizado se muestra en la Figura 5.1.

El procedimiento para la creación de esta aplicación se puede observar en la Figura 5.2 de manera muy general.

Como primer paso se extrae cada uno de los marcos que componen el video y se decodifica en el formato RGB para poder ser desplegado en OpenGL, lo cuál se explica a detalle en la Sec. 5.1. Posteriormente se realiza el procesamiento, que consiste en la conversión a HSI, la binarización y la extracción del perímetro, explicado en la Sec. 5.2. Una vez que se han extraído las siluetas de los cilindros, es necesario realizar un ajuste de dichos cilindros, para lo cual se utilizó un método robusto. Posteriormente se debe realizar la calibración, explicada en la Sec. 5.4, la visualización del objeto, detallada en la Sec. 5.6, y finalmente el método de seguimiento para agilizar el procesamiento, lo cual se explica en la Sec. 5.6.1.

5.1. Lectura de marco

La lectura del marco se realizó como de igual manera que para la aplicación 1, tomando un marco cada $\frac{1}{25} segundo$.

5.2. Procesamiento del marco

5.2.1. Conversión a HSI

El patrón de círculos concéntricos es de color rojo, por ello se debe realizar una segmentación. Al igual que en la aplicación 1, el primer paso una vez que se extrajo el marco fue

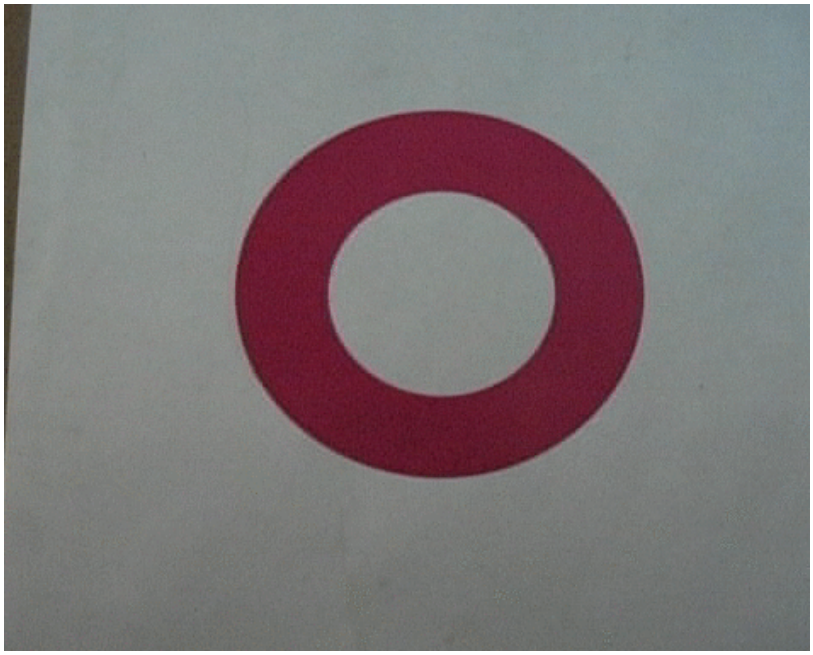


Figura 5.1: Patrón de círculos concéntricos

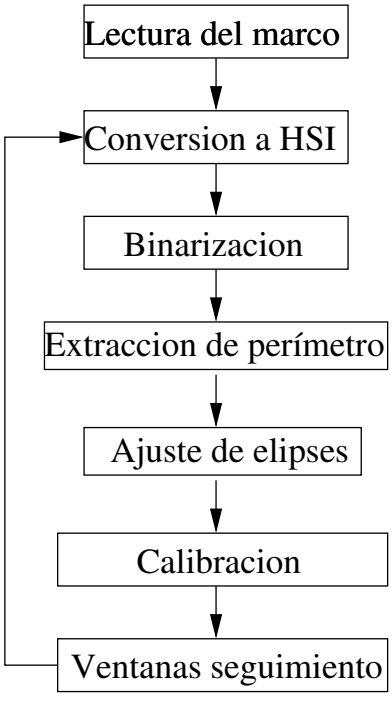


Figura 5.2: Diagrama principal

convertir de formato RGB a formato HSI y realizar la binarización de la imagen para obtener únicamente los objetos en color rojo. Los valores para h usados fueron 0^{o} a 10^{o} y 300^{o} a 360^{o}.

5.2.2. Extracción del perímetro

No es necesaria la extracción de bordes, sino que al tener una imagen ya binarizada, se puede extraer el perímetro de una manera más sencilla: Restando a la imagen original, la imagen erosionada. Esto funciona, ya que al erosionar una imagen se le quita todo el perímetro, entonces al restar a la imagen original la imagen sin perímetro, va a quedar únicamente el perímetro. En la Figura 5.3 se puede observar como es que esto funciona.
Al aplicar el procedimiento anterior a la imagen, queda lista para realizar la extracción de

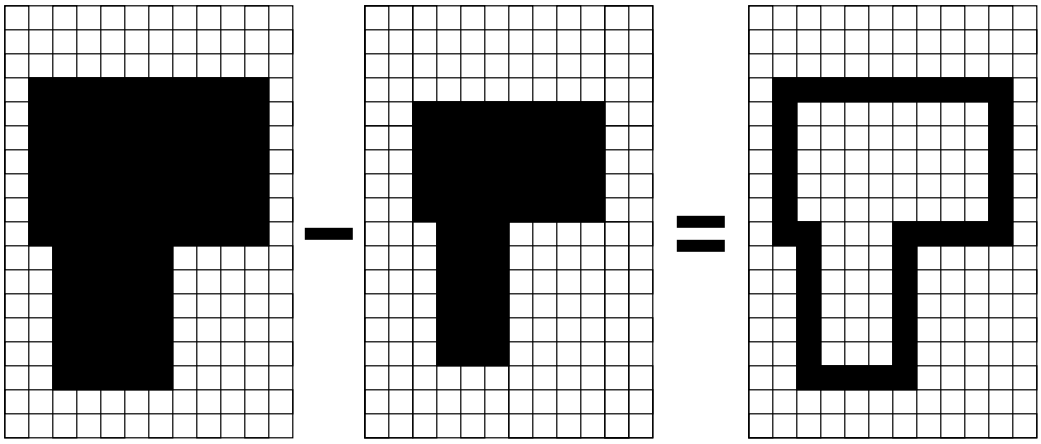


Figura 5.3: Imagen erosionada a imagen original, resultando en el perímetro de la imagen

los parámetros necesarios para realizar la calibración de la cámara.

En el algoritmo 6 se puede observar el procedimiento que se debe seguir para obtener el perímetro de los objetos en un marco de video.

Algoritmo 6 Algoritmo para obtener el perímetro de un marco

Entrada: Marco de video (*marco*), ancho del marco (*ancho*), alto del marco (*alto*)
Salida: Marco erosionado ($perimetro[ancho * alto]$)

1: Inicialización de arreglo $perimetro[ancho * alto] \rightarrow 0$
2: obtención de imagen erosionada $\rightarrow im\_erosionada$
3: **for** $i = 0$ to $i = areatotaldelmarco$ **do**
4: Lectura del renglón del marco $\rightarrow r_i$
5: Lectura del renglón del marco erosionado $\rightarrow re_i$
6: Resta de $r_i - re_i$
7: **end for**

Una vez que se tiene el perímetro de los objetos de interés se puede proceder a la extracción de las características que nos servirán para realizar la calibración de la cámara. Sin embargo, a pesar de que el procesamiento de la imagen haya sido muy reducido y ocupe muy poco tiempo en realizarse, es mejor tener en cuenta un método de seguimiento, para que se realice de una manera muy rápida, y la aplicación final se siga visualizando con la misma calidad sin pérdida de información.

el código para la obtención del perímetro se puede observar en el Apéndice A.

En la Figura 5.4 se puede observar como solo se extrae el perímetro del patrón en color rojo y se ignoran los demás objetos. Adicionalmente del procesamiento realizado, suele ser

Figura 5.4: Extracción de perímetro de un patrón color rojo

importante aplicar una erosión a la imagen, esto con la finalidad de eliminar los pequeños puntos de ruido que puedan haber quedado, pero en este caso no es tan necesario, ya que al realizar la segmentación con el color rojo, es difícil que exista una gran cantidad de ruido en la escena, a menos de que exista un objeto del mismo color. Adicionalmente a las bajas probabilidades de tener un marco con ruido, el algoritmo utilizado para la extracción de las elipses funciona de muy buena manera bajo la presencia de ruido dentro de la imagen.

5.3. Extracción de elipses con evolución diferencial

Para la extracción de las elipses se utilizó un algoritmo de extracción robusta utilizando heurísticas, específicamente utilizando evolución diferencial [4].

Este algoritmo utiliza la evolución diferencial para buscar la elipse que mejor se ajusta a los puntos presentados, es un algoritmo bastante robusto, ya que soporta la presencia de grandes cantidades de ruido.

La única desventaja que tiene es que es un algoritmo pesado, debido a que realiza una

gran cantidad de iteraciones, por ello se aplica únicamente la primera vez, para identificar la posición del patrón, y en los casos en los que se pierde el patrón con la técnica de seguimiento empleada.

El algoritmo del ajuste de elipses busca en toda la imagen la primer elipse evaluando la función con diversos individuos hasta que llega a una condición de paro, en caso de que se estén buscando más elipses, borra los puntos que conforman el primero encontrado, cambiando una variable de peso con la cual ya no serán tomados en cuenta para siguientes iteraciones. En las iteraciones posteriores vuelve a evaluar la función objetivo con diferentes individuos hasta encontrar el óptimo y en caso de requerir la extracción de más elipses, la encontrada vuelve a ser cambiada en sus pesos para ser ignorada en iteraciones posteriores.

Este algoritmo regresa los siguientes parámetros: semieje a, semieje b, coordenadas del centro (x, y) y el ángulo de inclinación.

Para agilizar aún más la búsqueda de las dos elipses, se realizó la búsqueda de la primera en toda la imagen y para la segunda se recortaron los parámetros como el centro y el ángulo de inclinación, ya que al ser círculos concéntricos, estos parámetros deben de ser muy similares.

Adicionalmente se recortaron los parámetros de los semiejes, debido a que este algoritmo siempre encuentra primero la elipse más grande, y en posteriores iteraciones va buscando las elipses que siguen en tamaño.

El ajuste de las elipses utilizando este método es muy efectivo, pero toma un tiempo considerable que provoca que no se pueda apreciar el seguimiento del objeto virtual a una buena velocidad, provocando que se observe un retraso representativo entre el cambio del posición del patrón y el cambio de posición del objeto virtual correspondiente.

El tiempo que se tarda este algoritmo en el ajuste de las dos elipses va de 1.03 a 1.55 segundos, dependiendo del zoom que tenga la cámara o de que tan cerca se encuentre del patrón, el tiempo no es muy grande, pero tomando en cuenta que se reproducen 25 marcos por segundo, se estarían perdiendo entre 26 y 39 marcos.

Para solucionar el problema del tiempo utilizado por este algoritmo, se hace uso de él únicamente para localizar el patrón la primera vez, así como cuando se pierda el patrón del área de búsqueda, para los demás casos se tiene un método de seguimiento que reduce el tiempo de manera muy notoria.

5.4. Calibración de la cámara

Una vez extraídos los dos elipses, se obtienen los parámetros de cada uno de ellos (semieje a, semieje b, centro (x, y) y ángulo de rotación) los cuales son utilizados para realizar la calibración de la cámara.

Para esta aplicación se utilizó un método de calibración con círculos concéntricos. [12]

La diferencia radica en el cálculo de la homografía, ya que una vez calculada, el procedimiento para la obtención de las matrices de rotación, traslación y proyección se realizan de igual manera que en la aplicación 1.

5.5. Cálculo de la homografía

La homografía es la transformación que cambia de un plano a otro, como se puede observar en la Figura 5.5 Para calcular la homografía es necesario obtener la matriz de las elipses

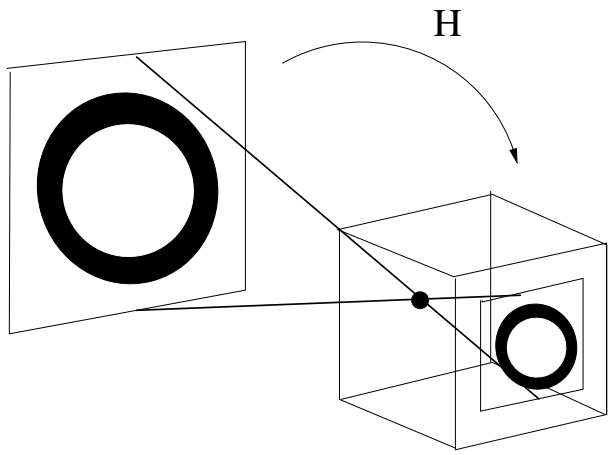


Figura 5.5: Homografía

del patrón, para ello se utilizan las siguientes ecuaciones:

$$
\begin{aligned}
A &= a^2 \cdot \sin\theta^2 + b^2 \cdot \cos\theta^2 \\
B &= \cos\theta \cdot \sin\theta \cdot (b^2 - a^2) \\
C &= b^2 \cdot \cos\theta \cdot (-c_y \cdot \sin\theta - c_x \cdot \cos\theta) - a^2 \cdot \sin\theta \cdot (c_x \cdot \cos\theta - c_y \cdot \cos\theta) \\
D &= b^2 \cdot \sin\theta + a^2 \cdot \cos\theta \\
E &= b^2 \cdot \sin\theta \cdot (-c_y \cdot \sin\theta - c_x \cdot \cos\theta) - a^2 \cdot \cos\theta \cdot (c_x \cdot \cos\theta - c_y \cdot \cos\theta) \\
F &= b^2 \cdot (-c_y \cdot \sin\theta - c_x \cdot \cos\theta)^2 + a^2 \cdot (c_x \cdot \cos\theta - c_y \cdot \cos\theta)^2 - a^2 \cdot b^2
\end{aligned}
$$

donde:

c_x = coordenada x del centro
c_y = coordenada y del centro
a = semieje a
b = semieje b
θ = ángulo de inclinación de la elipse

Una vez calculados los valores anteriores se hace la matriz de la cónica como se indica en la ecuación (5.1)

$$EC = \begin{pmatrix} A & B & C \\ B & D & E \\ C & E & F \end{pmatrix} \tag{5.1}$$

Teniendo como A_1 la matríz de la elipse 1 y como A_2 la matriz de la elipse 2, se calcula la homografía de la siguiente manera:

Se realiza el cálculo de la inversa de A_1 y A_2 y se realiza una multiplicación de $B = A_{2inv} \cdot A_1$. Se obtienen los eigenvectores de B, de los cuales se usa el valor que está en la mediana y el valor que está más alejado de la mediana, cada uno de ellos es multiplicado por $A_{1inv} - A_{2inv}$, el resultado se descompone en sus valores singulares (U S V) y finalmente se multiplica el valor de $U \cdot \sqrt{S}$, del resultado obtenido de usar el valor de la mediana de los eigenvalores se usa la primer columna para la homografía H, siendo la columna 3 de ésta H_3, del resultado de usar el valor más alejado a la media de los eigenvalores se toman las dos primeras columnas para H, que serán H_1 y H_2. A continuación se muestra el código en octave de cómo se realizó lo anterior.

```
% Calculo de inversas
A1i = inv( A1 );
A2i = inv( A2 );
B = A2i * A1;
e = eig( B );

[m, k] = min(abs(median(e)-e));
[U S V] = svd( e(k)*A1i-A2i, 2 );
J1 = U*sqrt(S);

[m, k] = max(abs(median(e)-e));
[U S V] = svd( e(k)*A1i-A2i, 2 );
J = U*sqrt(S);
% Se usan las primeras columnas de J y la primer columna de J1
J(1,3) = J1(1,1);
J(2,3) = J1(2,1);
J(3,3) = J1(3,1);
```

Una vez calculada la homografía, se obtiene el valor de f para calcular la matriz de proyección K y con dicha matriz, calcular las matrices de rotación y traslación como se explica en el Capítulo 3.

5.6. Visualización del objeto virtual

Una vez obtenidos los parámetros intrínsecos y extrínsecos de la cámara, se pueden aplicar directamente a los objetos de OpenGL con las siguientes funciones:

- glMultMatrix().
- glRotatef().
- glTranslatef().

Al igual que en la aplicación 1, para crear la matriz de proyección K se utilizó el valor de f con la siguiente matriz:

$$M = \begin{pmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{zFar+zNear}{zNear-zFar} & \frac{2 \cdot zFar \cdot zNear}{zNear-zFar} \\ 0 & 0 & -1 & 1 \end{pmatrix} \tag{5.2}$$

Escalando el objeto por lo menos por k para poder visualizarlo en la pantalla:

$$k = \frac{zNear}{|t_3| - \frac{2}{\sqrt{3}}} \tag{5.3}$$

y el plano de corte trasero es:

$$zFar = \frac{4 \cdot k}{\sqrt{3} + zNear} \tag{5.4}$$

Se usó la función *glMultMatriz(M)* para aplicar la matríz de proyección, y posteriormente las rotaciones y translaciones.

Los objetos que se dibujan sobre el patrón, fueron creados en dos funciones diferentes:

- Cilindro. Se crea un cilindro de 0.5 de diametro, para ser visualizado en el círculo pequeño del patrón, y de 2 de largo. Para su creación se utilizaron las funciones de OpenGL.
- Barra. Al igual que en la aplicación 1, se colocó una barra girando dentro del cilindro para demostrar que se puede agregar de manera sencilla cualquier objeto. La barra se dibujó uniendo vértices por medio de las funciones de OpenGL.

En la Figura 5.8 se puede observar como se encuentra dibujado el cilindro con la barra (a) Una vista superior y (b) una vista lateral.

En la Figura 5.6 se puede observar un cilindro sobre el patrón de los círculos concéntricos, si se usa un cilindro de diámetro 0.5, éste se posicionará sobre el círculo pequeño, mientras que si se usa un cilindro de diámetro 1, se posicionará sobre el círculo grande como se puede observar en la Figura 5.6.

La adición de la barra dentro del cilindro se hizo con la finalidad de demostrar que se puede agregar de manera sencilla cualquier otro objeto, además de que el objeto virtual puede estar en movimiento. La aplicación con la barra se puede observar en la Figura 5.9. La barra dentro del cilindro gira 2^{o} cada marco sobre el eje z

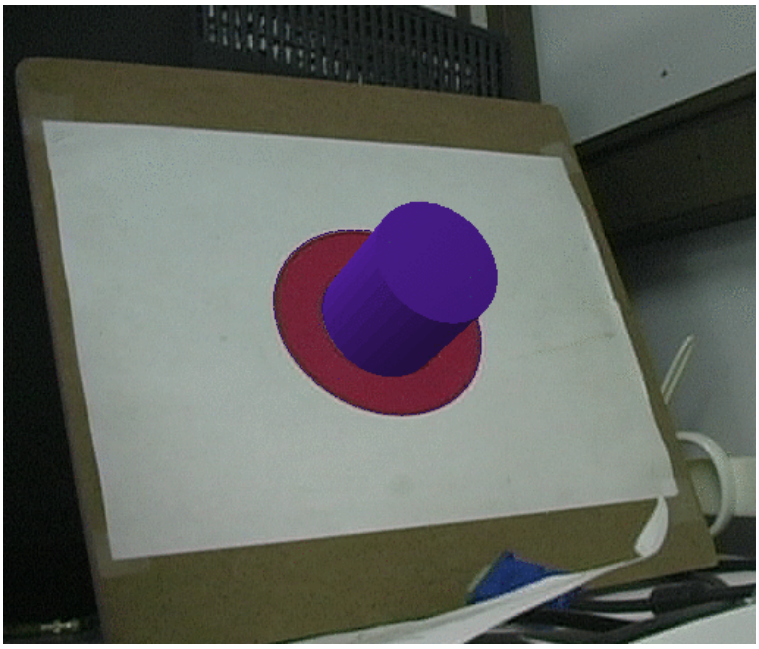


Figura 5.6: Cilindro sobre patrón de círculos concéntricos

La adición de nuevos objetos se logra de una forma muy sencilla, ya que únicamente hay que colocar la función que dibuja el objeto dentro de *paint*, aunque es importante considerar que la calibración se realizó tomando en cuenta un cilindro de diámetro 1, por lo que si dibujamos un cilindro de ese diámetro, se colocará encima del círculo mayor del patrón.

El código para la aplicación de las matrices en OpenGL se presenta en el Apéndice A.

La matriz m es modificada cuando los parámetros de la calibración cambian **paintGl()**. La matriz se calcula de la siguiente manera:

Figura 5.7: Aplicación desde dos vistas diferentes con cilindro de radio 1

```
/** Calculo de matriz m **/

zN = 1.0;
k = zN/(fabs(t_3) - 2.0/sqrt(3.0));
zF = 4*k/(sqrt(3)+zN);
d = zN-zF;
m[0] = f/asp; m[4] = 0.0; m[8] = 0.0;       m[12] = 0.0;
m[1] = 0.0;   m[5] = f;   m[9] = 0.0;       m[13] = 0.0;
m[2] = 0.0;   m[6] = 0.0; m[10] = (zF+zN)/d; m[14] = (2*zF*zN)/d;
m[3] = 0.0;   m[7] = 0.0; m[11] = -1.0;       m[15] = 1.0;
```

La figura dibujada sobre el patrón fue un cuboide, pero para demostrar que se pueden agregar otros objetos de manera sencilla, se dibujó además un cilindro girando dentro del cubo. La aplicación final se puede observar en la Figura 5.9.

5.6.1. Seguimiento

Las elipses son ajustadas utilizando evolución diferencial únicamente la primera vez, esto para saber exactamente la ubicación, posición, así como tamaño de las elipses que conforman el patrón, y de ahí partir para realizar un seguimiento las iteraciones posteriores.

La técnica empleada para el seguimiento de elipses [25] consiste en el seguimiento de líneas ortogonales a cada una de las elipses.

El primer paso para el seguimiento, es la creación de las líneas ortogonales a cada elipse. Para estimar una elipse se requiere de al menos cinco puntos, por ello se realizaron 12 líneas debido a que es una cantidad no muy grande, pero es mayor a los puntos requeridos.

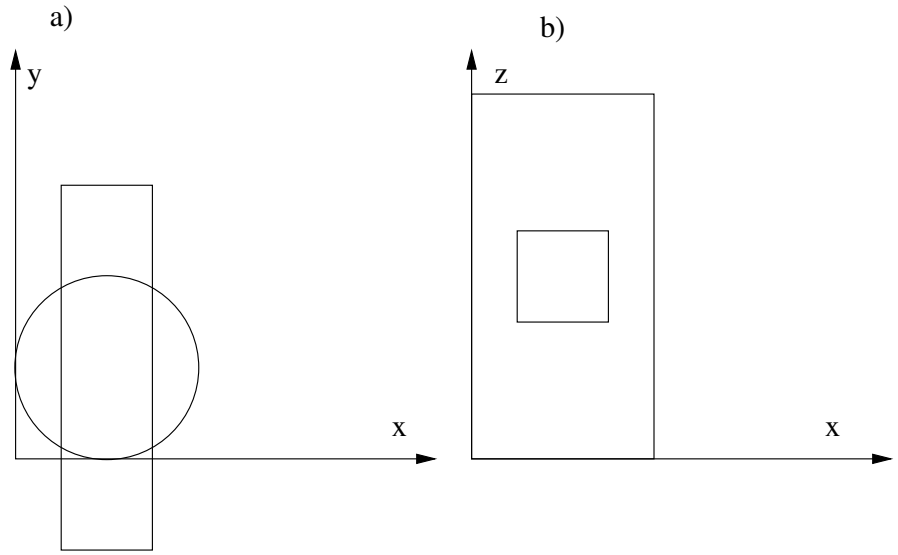


Figura 5.8: a) Objeto visto desde arriba, b) Objeto en una vista lateral

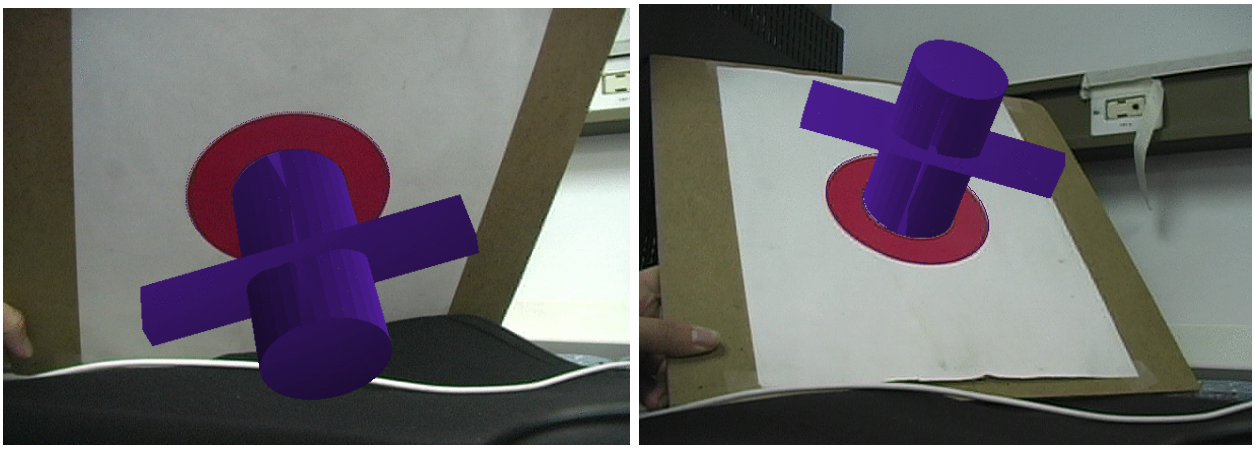


Figura 5.9: Aplicación final desde dos vistas diferentes

Para obtener líneas bien distribuidas, primero se calcularon los puntos en los que las líneas deben ser dibujadas, si 12 lineas serán dibujadas, entonces el ángulo de separación se obtiene con la ecuación (5.5)

$$\alpha = \frac{360}{12} \tag{5.5}$$

Se evalúa 5.6 en la ecuación paramétrica de la elipse (5.8) para obtener los puntos de intersección con cada línea.

$$\alpha \cdot x, \quad x \in [1, 12] \tag{5.6}$$

$$x' = a \cdot \cos(\alpha \cdot x) \cdot \cos\theta - b \cdot \sin(\alpha \cdot x) \cdot \sin\theta \tag{5.7}$$

$$y' = a \cdot \cos(\alpha \cdot x) \cdot \sin\theta - b \cdot \sin(\alpha \cdot x) \cdot \cos\theta \tag{5.8}$$

Una vez obtenidos los puntos de intersección, se crea la línea con las ecuaciones 5.9 y 5.10

$$x = \frac{y'}{\sqrt{y'^2 + x'^2}} \tag{5.9}$$

$$y = \frac{1}{1 + \left(\frac{y'}{x'}\right)^2} \tag{5.10}$$

En la Figura 5.10 se pueden observar las líneas ortogonales a cada una de las elipses del patrón. La manera común para encontrar las intersecciones de las líneas con las elipses sería

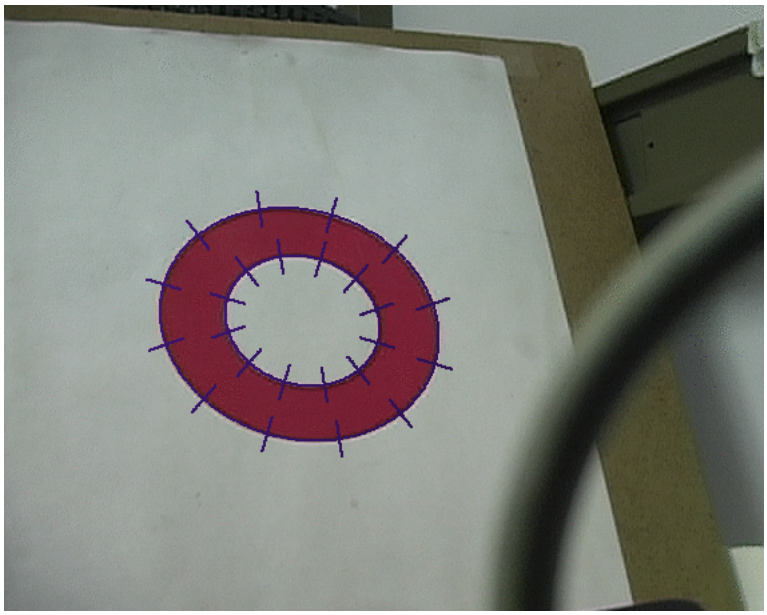


Figura 5.10: Lineas ortogonales a las elipses

aplicar una máscara a la imagen, pero es algo muy tardado debido a que hay que pasarla por toda la imagen. Para agilizar la detección de las intersecciones, al momento de crear un nuevo punto de la línea se revisaron sus ocho vecinos para revisar que a ese punto no perteneciera alguna intersección. En la Figura 5.11 se puede observar cómo se realizó esa búsqueda en los vecinos. Al dibujar un nuevo píxel, se revisa en las coordenadas vecinas a este, si el píxel tiene por lo menos dos vecinos, corresponde a una intersección, de lo contrario no existe intersección.

El problema del uso de este método es que se pueden encontrar tres intersecciones para cada

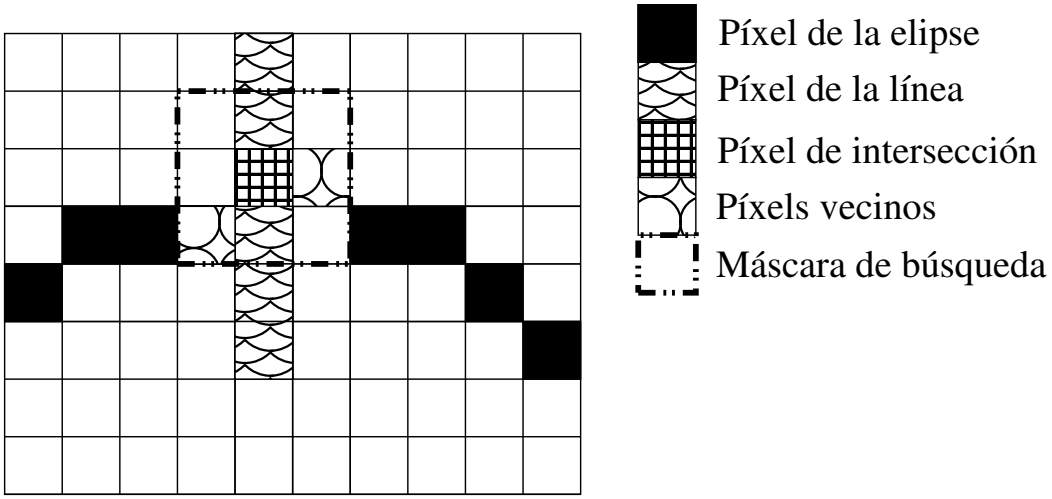


Figura 5.11: Método para búsqueda de intersecciones

línea, en el píxel por debajo al píxel de la intersección, el píxel de la intersección y el píxel por arriba de la intersección. Para solucionar este problema, se revisa que además de los dos vecinos, en la coordenada del punto en la imagen exista un píxel de color negro.

Se deben encontrar por lo menos cinco intersecciones para poder realizar el ajuste, pero en caso de no encontrar las intersecciones mínimas requeridas, se aplica nuevamente el algoritmo de ajuste con evolución diferencial para ubicar nuevamente el patrón.

Una vez encontradas las cinco o más intersecciones se utilizó una versión rápida del método de ajuste directo [16] para la obtención de los parámetros de las nuevas elipses.

5.6.2. Ajuste directo

Una elipse puede ser representada implícitamente por la ecuación general de las cónicas:

$F(\mathbf{a}, \mathbf{v}) = \mathbf{a}^T\mathbf{v} = a'x^2 + b'xy + c'y^2 + d'x + f' + g' = 0$

donde:

$\mathbf{a} = [a', b', c', d', f', g']^T$ y $\mathbf{v} = [x^2, xy, y^2, x, y, 1]^T$

La ecuación va a representar una elipse si $b'^2 - 4a'c' < 0$. Se tiene un problema de optimización, el cual está representado en la ecuación (5.11)

$$g_a(\mathbf{a}) = \sum_{i=1}^{n} (\mathbf{a}^T\mathbf{v}_i)^2 \tag{5.11}$$

sujeta a la restricción de la ecuacion 5.12

$$4a'c' - b'^2 = 1 \tag{5.12}$$

la cual puede ser representada de forma matricial como:

$$\mathbf{a}^T \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{a} = 1 \tag{5.13}$$

El problema de optimización puede ser resuelto mediante un sistema de eigenvectores de la forma:

$$\mathbf{v}^T\mathbf{va} = \lambda C\mathbf{a} \tag{5.14}$$

Una vez resuelto el eigensistema, se obtienen los parámetros $\mathbf{a}$, a los cuales se les debe realizar la transformación en el conjunto de parámetros (a, b, x_c, y_c, ϕ). Para realizar la transformación se utilizaron las siguientes ecuaciones:

$$x_c = \frac{c'd' - b'f'}{b'^2 - a'c'} \tag{5.15}$$

$$y_c = \frac{a'f' - b'd'}{b'^2 - a'c'} \tag{5.16}$$

$$a = \sqrt{\frac{2(a'f'^2 + c'd'^2 + g'b'^2 - 2b'd'f' - a'c'g')}{(b'^2 - a'c')\left[\sqrt{(a'-c')^2 + 4b'^2} - a' - c'\right]}} \tag{5.17}$$

$$b = \sqrt{\frac{2(a'f'^2 + c'd'^2 + g'b'^2 - 2b'd'f' - a'c'g')}{(b'^2 - a'c')\left[-\sqrt{(a'-c')^2 + 4b'^2} - a' - c'\right]}} \tag{5.18}$$

$$\phi = \tan^{-1}\left(\frac{a'-c'}{2b}\right)\left(\frac{90}{\pi}\right) \tag{5.19}$$

Con los nuevos parámetros se realizó una nueva calibración. Como ya se había mencionado, este ajuste únicamente se puede realizar cuando se tienen al menos cinco intersecciones de las líneas de las elipses anteriores y las nuevas elipses, en caso de que no se cumpla con dicha condición se aplica el ajuste con evolución diferencial para ubicar nuevamente el patrón y en iteraciones posteriores se aplica nuevamente este método.

Capítulo 6
Conclusiones

6.1. Conclusiones

El sistema realizado presenta una buena calidad de video, permite cambios en el acercamiento de la cámara y fue totalmente desarrollado con software libre. A pesar de todo el procesamiento que se realiza a cada uno de los marcos, el video se sigue apreciando a una velocidad de 25 marcos por segundo. No hay pérdida de marcos en el desplegado del video, sin embargo hay pérdidas de marcos en su procesamiento, lo cual no es considerable, ya que la pérdida es a lo mucho de tres marcos cuando la cámara está muy cerca del patrón o tiene un zoom grande. En la aplicación del cilindro virtual, el seguimiento del patrón es bastante rápido, debido a que se utiliza un método de seguimiento, y no todo el tiempo se extraen las elipses del patrón para calibrar la cámara en cada marco.

Inicialmente se había optado por el uso de patrones en blanco y negro, pero el procesamiento requerido para la extracción de los parámetros del patrón es mayor, además de que la imagen tras el procesamiento no queda tan limpia, debido a que la presencia de objetos en colores obscuros mete mucho ruido a la escena. El problema de dicha cantidad de ruido fue solucionado con la creación de patrones en color. Esta segmentación nos permite hacer uso de cualquier color, pero lo conveniente es utilizar los que sean poco comunes en la escena donde se va a utilizar la aplicación. Para este trabajo fue elegido el color rojo, debido a que en la escena de trabajo no existía ese color, el cual dió muy buenos resultados.

Las aplicaciones realizadas para probar la funcionalidad de la herramienta fueron dos: una usando círculos concéntricos y otra utilizando círculos en un plano. En ambas aplicaciones se obtuvieron buenos resultados, ya que como se había mencionado con anterioridad, el retraso es casi imperceptible.

Adicionalmente de las aplicaciones realizadas, se experimentó con un patrón cuadriculado, con extracción de líneas, pero este tablero tuvo que ser sustituido por el patrón de los círculos en un plano, debido a que no se pueden detectar líneas con la suficiente precisión en el video digital dada la compresión (y la pérdida de resolución) que presenta.

Finalmente se presenta una herramienta para realizar aplicaciones de RA la cual proporciona desde la captura de video hasta el desplegado de las imágenes, el cual se realiza en OpenGL. No se puede decir que la herramienta está destinada a la creación completa de aplicaciones, debido a que no proporcionamos un módulo propio para graficación; sin embargo la adición de un objeto es muy sencilla, ya que únicamente se coloca en OpenGL y las transformaciones son aplicadas automáticamente.

Este trabajo de tesis produjo los siguientes resultados:

- Una herramienta desarrollada completamente con *software* libre, con el uso de VD y patrones de color, y dos aplicaciones de prueba con diversos métodos de calibración, en ambas (a diferencia de una aplicación con video analógico) se puede realizar un acercamiento y alejamiento de la cámara sin perder calidad en los marcos y por consecuente al realizarlo se sigue apreciando el objeto virtual sobre el patrón.
- Un artículo: Rosa Atzin Vázquez del Ángel y Luis Gerardo de la Fraga. Sistema de realidad aumentada con autocalibración de cámara. Aceptado en el VII Taller-Escuela de Procesamiento de Imágenes PI10 en el Centro de Investigación en Matemáticas A.C. (CIMAT), Guanajuato, Gto. 20-22 de Octubre, 2010.

6.2. Trabajo a futuro

La herramienta realizada tiene algunos deficiencias, que aunque no son muy significativas, pueden ser resueltas. El trabajo que puede ser realizado para mejorar la herramienta es el siguiente:

- Paralelización del procesamiento para que el pequeño retraso que existe se haga aún más pequeño o incluso desaparezca.
- Experimentación con más métodos de calibración para realizar la comparación con las utilizadas y elegir entre ellos el método más exacto.
- Experimentación con más colores que sean menos comunes que el color rojo, con la finalidad de tener una aplicación que funcione al igual en cualquier escenario.
- Un umbral dinámico que se adapte a la escena de acuerdo a las condiciones de iluminación, pero que sin embargo pueda seguir segmentando con color.
- Experimentación con patrones en blanco y negro utilizando el formato HSI.

Bibliografía

[1] **Artoolkitplus**. http://studierstube.icg.tu-graz.ac.at/handheld ar/artoolkitplus.php, June 2009.

[2] **Linux Video Digital**. http://www.kinodv.org/, September 2009.

[3] Ronald T. Azuma. **A survey of Augmented Reality**. *PRESENCE: Teleoperations and Virtual Enviroments*, 6(4):355–385, August 1997.

[4] Luis Gerardo de la Fraga and Gustavo M. López Domínguez. **Robust Fitting of Ellipses with Heuristics**. *WCCI 2010 IEEE World Congress on Computational Intelligence*, pages 3990–3997, July 2010.

[5] et. al E. Sorantin. **The virtual liver surgery planning system**. *European Congress of Radiology: The Matrix, Vienna*, 2004.

[6] E.Trucco and A. Verri. Introductory to techniques for 3-D computer vision. Prentice Hall, 1998.

[7] M. Fiala. **ARTag, An Improved Marker System Based on ARToolkit**. *National Research Council Canada*, Publication Number: NRC: 47419, July 2004.

[8] Free Software Foundation. **The GNU Scientific Library(GSL)**. http://www.gnu.org/software/gsl/, June 2010.

[9] Rafael C. Gonzalez and Richard E. Woods. Digital Image Processing. Prentice Hall, 2nd edition edition, 2001.

[10] R.M. Haralick. *Re-view and analisys of solutions of the three point perspective pose estimation problem. International Journal of Computer Vision*, 13(3):331–356, 1994.

[11] J. Cornejo Herrera, A. Lara López, R. Landa Becerra, and L.G. de la Fraga. **Una biblioteca para procesamiento de imagen: scimagen**. In *VIII Conferencia de Ingeniería Eléctrica*, pages 448–457, septiembre 2002.

[12] Gurdjos P Jun-Sik Kim and Kweon IS. **Geometric and Algebraic Constraints of Projected Concentric Circles and Their Applications to Camera Calibration**. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(4):637–642, April 2005.

[13] H. Kato. **Marker tracking and hmd calibration for a video-based augmented reality conferencing system**. *Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*, pages 85–94, October 1999.

[14] Leonor Borja Martínez. **Medición de respuesta pupilar**. Tesis de maestría en ciencias en ingeniería eléctrica., Cinvestav-IPN, Sección de Computación, Unidad Zacatenco, 18 de Diciembre, 2006.

[15] Alfredo Hernández Moreno and Lino Soberanes Pérez. **Sistema para la integración de entornos Reales y Virtuales mediante Realidad Aumentada**. Ingeniería en sistemas computacionales., ESCOM, Mayo 2008.

[16] M.Stojmenovic and A. Nayak. ***Advances in Image and Video Technology***, volume 4872 of *Lecture Notes in Computer Science*, chapter Direct Ellipse Fitting and Measuring Based on Shape Boundaries, pages 221–235. Springer Berlin / Heidelberg, 2007.

[17] Diego Aracena Pizarro. *Comparación de técnicas de calibración de cámaras digitales. Revista Facultad de Ingeniería - Universidad de Tarapacá*, 13(1):57–67, Abril 2005.

[18] González R.C. and R.E. Woods. Digital Image Processing. Adisson-Wesley, 1993.

[19] B. Reitinger and D. Schmalstieg. **Augmented reality scouting for interactive 3D reconstruction**. *Proceedings of IEEE Virtual Reality*, pages 219–222, 2007.

[20] Norma Irene Serna Rodríguez. **Sistema de Realidad Aumentada**. Tesis de maestría en ciencias en ingeniería eléctrica., Cinvestav-IPN, Unidad Guadalajara, Noviembre 2006.

[21] Azriel Rosenfeld and Avinash C. KaK. Digital Picture Processing. Academic Press, 1982.

[22] Roy. Augmented reality with nyartoolkit, opencv and opengl. http://www.morethantechnical.com/2009/06/28/augmented-reality-with-nyartoolkit-opencv-opengl/, June 2009.

[23] D. Schmalstieg and G. Reitmayr. **Augmented Reality as a Medium for Cartography**. *Multimedia Cartography, 2nd ed.*, pages 267–282, 2008.

[24] I. Sutherland. **A Head-Mounted Three Dimensional Display**. *Fall Joint Computer Conf., Am. Federation of Information Processing Soc. (AFIPS) Conf. Proc. 33*, 9(11):757–764, December 1968.

[25] Markuz Vincze. **Robust tracking of ellipses at frame rate**. *Pattern Recognition*, 34:487–498, 2001.

Apéndice A

Lectura de Marcos

```
/** dvgrab.cc -- DVGrab control class **/


void DVgrab::captureThreadRun()
{
    ...
     ...
   while ( m_reader_active )
   {
      if ( !critical_mass && m_raw_pipe ){
          ...
          ...
          if (count == 1){
             pthread_mutex_lock (&mutexRGB1);
             frame2->ExtractRGB(pixels[0]);
             count=0;
             pthread_mutex_unlock(&mutexRGB1);
          }else{
             pthread_mutex_lock (&mutexRGB2);
             frame2->ExtractRGB(pixels[1]);
             count=1;
             pthread_mutex_unlock(&mutexRGB2);
          }
      }
    ...
    ...
   }
}
```

Almacenamiento de Marcos

```
/* void DVgrab::captureThreadRun() */
if (count == 1){
   pthread_mutex_lock (&mutexRGB1);
      frame2->ExtractRGB(pixels[0]);
      count=0;
   pthread_mutex_unlock(&mutexRGB1);
}else{
   pthread_mutex_lock (&mutexRGB2);
      frame2->ExtractRGB(pixels[1]);
      count=1;
   pthread_mutex_unlock(&mutexRGB2);
}
```

Emitir la señal

```
/** main.c **/
int main (int argc, char **argv){
   QApplication app(argc, argv);
   p = new painter ();
   p->show();
    float fps = 25.0;
    float seconds = 1.0/fps;
   tout_val.it_interval.tv_sec = 0;
   tout_val.it_interval.tv_usec = 0;
   tout_val.it_value.tv_sec = 0;
   tout_val.it_value.tv_usec = seconds*1000000;
   signal (SIGALRM, pinta);
   error = setitimer(ITIMER_REAL, &tout_val,NULL);
   if (error == -1)
      return error;
   int pid = app.exec();
   return pid;
}
void pinta (int i){
   p->updateW();
    error = setitimer( ITIMER_REAL, &tout_val,NULL );
    if ( error == -1 || exitFlag )
      exit (0);
}
```

Dibujado del objeto

```
evento{
   // Se realiza cada 1/25 segundos

   if (count == 1){
      pthread_mutex_lock(&mutexRGB2);
         copyFrame(pixels[1]);
      pthread_mutex_unlock(&mutexRGB2);
   }else{
      pthread_mutex_lock(&mutexRGB1);
         copyFrame(pixels[0]);
      pthread_mutex_unlock(&mutexRGB1);
   }

   glDrawPixels(720, 480, GL_RGB, GL_UNSIGNED_BYTE, pixelsCopy);

}
```

Procesamiento del marco

```
HiloDeProcesamiento{

   if (count == 1){
      pthread_mutex_lock(&mutexRGB2);
         copyFrame(pixels[1]);
      pthread_mutex_unlock(&mutexRGB2);
   }else{
      pthread_mutex_lock(&mutexRGB1);
         copyFrame(pixels[0]);
      pthread_mutex_unlock(&mutexRGB1);
   }

   procesa(pixelsCopy2);

   dibujaObjetoVirtual();

}
```

Conversión a HSI

```
/** canvas.c **/


void canvas::hsi(int contador)
{
    int val, R,G,B;
    float r, g, b, h, s, i, min;
    int n = 0;
    for(int i=0; i<ty; i++){
        for(int j=0; j<tx; j++){
            R = pixels[cont][n++];
            G = pixels[cont][n++];
            B = pixels[cont][n++];
            r = (float)R/(float)(R + G + B);
            g = (float)G/(float)(R + G + B);
            b = (float)B/(float)(R + G + B);
            if (r<=b)
                h = acos((0.5*(r-g+r-b))/sqrt((r-g)*(r-g)+(r-b)*(g-b)↩
                    ));
            else
                h = 2*M_PI - acos((0.5*(r-g+r-b))/sqrt((r-g)*(r-g)+(r↩
                    -b)*(g-b)));
         h *= 180.0/M_PI;
            if (r<g && r<b)
                min = r;
            if (g<r && g<b)
                min = g;
            if (b<g && b<r)
                min = b;
            s = 1.0 - 3*min;1.9
            if ((h > 0.0 && h < 10.0)||(h > 300.0 && h < 360.0) && s↩
                >0.2 )
                psal[i][j] = 255;
         else
                psal[i][j] = 0;
      }
        n += 720*3;
    }
}
```

Erosión

```
/** morphology.c **/

void erosion_3x3( MATRIX *IN, MATRIX *OUT )
{
   int i, j, l, k;
   byte *ptrin, *ptrout;
   int val;

   for( i=0; i<IN->rows; i++ ) {
      ptrout = (byte *) OUT->ptr[i];
      for( j=0; j<IN->cols; j++ ) *ptrout++ = 0;
   }

   for( i=0; i<=IN->rows - 3; i++ ) {
      ptrout = (byte *) OUT->ptr[i+1] + 1;
      for( j=0; j<=IN->cols - 3; j++ ) {
         val = 0xff;
         for( l=0; l<3; l++ ) {
            ptrin = (byte *) IN->ptr[i + l] + j;
            for( k=0; k<3; k++ ) val &= *ptrin++;
         }
         if ( val ) *ptrout = 255;
            ptrout++;
      }
   }
}
```

Cálculo de la Homografía

```
/** calibracion.c **/

void calibracion::homografia(float val[16][2], float H[3][3])
{
   float T1[3][3] = {{0,0,0},{0,0,0},{0,0,0}}, T1i[3][3] = ↩
      {{0,0,0},{0,0,0},{0,0,0}};
   int i=0, j=0, k=0, error=0;
   float valAux[16][2];
   for (i=0; i<16; i++){
      valAux[i][0] = val[i][0];
```

```
    valAux[i][1] = val[i][1];
}

gsl_matrix * A1 = gsl_matrix_alloc (32, 9);
gsl_matrix * V = gsl_matrix_alloc (9, 9);
gsl_vector * S = gsl_vector_alloc (9);
gsl_vector * work = gsl_vector_alloc (9);
float up=0, vp=0, u=0, v=0;
normaliza(valAux, T1);
//  imprime_matriz(T1, "T1");
for (i=0; i<32; i++)
    for (j=0; j<9; j++)
        gsl_matrix_set (A1, i, j, 0.0);
// Hacemos las 16*2 ecuaciones
for (i=0; i<16;i++){
    up = valAux[i][0];
    vp = valAux[i][1];
    u = model1[i][0];
    v = model1[i][1];
    k = 2*i;
    gsl_matrix_set (A1, k, 0, u);
    gsl_matrix_set (A1, k, 1, v);
    gsl_matrix_set (A1, k, 2, 1.0);
    gsl_matrix_set (A1, k, 3, 0.0);
    gsl_matrix_set (A1, k, 4, 0.0);
    gsl_matrix_set (A1, k, 5, 0.0);
    gsl_matrix_set (A1, k, 6, -up*u);
    gsl_matrix_set (A1, k, 7, -up*v);
    gsl_matrix_set (A1, k, 8, -up);

    k++;
    gsl_matrix_set (A1, k, 0, 0.0);
    gsl_matrix_set (A1, k, 1, 0.0);
    gsl_matrix_set (A1, k, 2, 0.0);
    gsl_matrix_set (A1, k, 3, u);
    gsl_matrix_set (A1, k, 4, v);
    gsl_matrix_set (A1, k, 5, 1.0);
    gsl_matrix_set (A1, k, 6, -vp*u);
    gsl_matrix_set (A1, k, 7, -vp*v);
    gsl_matrix_set (A1, k, 8, -vp);
}
error = gsl_linalg_SV_decomp_jacobi (A1, V, S);
//  printf ("Error: %d\n", error);
for (i=0, k=0; i<3; i++)
    for (j=0; j<3; j++)
        H[i][j] = gsl_matrix_get(V, k++, 8);
```

```
   if (H[2][2] < 0.0){
      for (int i=0; i<3; i++)
         for (int j=0; j<3; j++)
            H[i][j] *=-1.0;
   }
   if (H[2][2] > 0){
      for (int i=0; i<3; i++)
         for (int j=0; j<3; j++)
            H[i][j] *= -1.0;
   }

   inversa(T1, T1i); //Calcula la inversa
   multiplica(H, T2, T1); //Multiplica H y T2 y lo guarda en T1
   multiplica(T1i, T1, H); //Multiplica T1i y T1 y lo guarda en H
}
```

Normalización

```
/** calibracion.c **/

/*Normaliza val*/

void calibracion::normaliza(float val[16][2], float T[][3])
{
   float sumx = 0, sumx2 = 0;
   float sumy = 0, sumy2 = 0;
   float mux=0.0f, muy=0.0f, six=0.0f, siy=0.0f;
   int i=0;
   float x=0, y=0;

   for (i=0; i<16; i++){
      x = val[i][0];
      sumx += x;
      sumx2 += x*x;

      y = val[i][1];
      sumy += y;
      sumy2 += y*y;
   }

   mux = sumx/16.0;
   muy = sumy/16.0;
   six = sqrt((float)sumx2/16.0 - mux*mux);
```

```
    siy = sqrt((float)sumy2/16.0 - muy*muy);
    T[0][0] = 1.0/six;    T[0][1] = 0.0f;    T[0][2] = -mux/six;
    T[1][0] = 0.0f;    T[1][1] = 1.0/siy;    T[1][2] = -muy/siy;
    T[2][0] = 0.0f;    T[2][1] = 0.0f;    T[2][2] = 1.0f;

    //Valores normalizados
    for (i=0;i<16;i++){
        val[i][0] = (val[i][0] - mux)/six;
        val[i][1] = (val[i][1] - muy)/siy;
    }

}
```

Obtención de perímetro

```
/** canvas.cpp **/

void canvas::perimetro(MATRIX *MATR, MATRIX *MATR2)
{
    byte *ptrA, *ptrB;
    int m=0;
    erosion_3x3(MATR,MATR2);
    for(int i=0; i<MATR->rows; i++ ) {
        ptrA = (byte *)MATR->ptr[i];
        ptrB = (byte *)MATR2->ptr[i];
        for(int j=0; j<MATR->cols; j++ ){
            if ((*ptrA - *ptrB++)==0)
                *ptrA++ = 0;
            else{
                if ((MATR->cols - j)<5 || (MATR->cols - j)>715 || i <↩
                    5 || i>475)
                    *ptrA++ = 0;
                else
                    *ptrA++ = 255;

            }
        }
    }
}
```

Matrices en OpenGL

```
void canvas::paintGL()
{
    ...
     glMultMatrixf(m);
     glScalef(k,k,k);
     glTranslatef(t_1, t_2, t_3);
     glRotatef(gamma, 0.0, 0.0, 1.0);
     glRotatef(beta, 0.0, 1.0, 0.0);
     glRotatef(alfa, 0.0, 0.0, 1.0);
     cilindro(2.0, 0.25, 1.5, 1.5, 1.5, 20);
    barra();
     drawFrame(); //Dibujado del frame
     ...
}
```