UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

CENTRO DE INFORMÁTICA INDUSTRIAL

SEGUIMIENTO DE POSE DEL OBSERVADOR UTILIZANDO CÁMARA WEB

Tesis presentada para optar por el Título de Ingeniería en Ciencias Informáticas

Autor: **Liudmila Cecilia Rodríguez Ricardo**

Tutor: **MSc.Liudmila Pupo Peña**

Co-tutor: **MSc.Yoander Cabrera Díaz**

La Habana, Junio de 2011

Declaración de autoría

Yo, Liudmila Cecilia Rodríguez Ricardo, declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste, firmo la presente declaración de autoría en La Habana a los _ días del mes de _ del año ____.

_____

Firma del autor

_____

Firma del Tutor MSc. Liudmila Pupo Peña

_____

Firma del Co-Tutor MSc. Yoander Cabrera Díaz

Datos de contacto

Nombre y Apellidos: Liudmila Pupo Peña

Edad: 26 años

Ciudadanía: cubana

Institución: Universidad de las Ciencias Informáticas (UCI)

Título: MSc en Informática Aplicada

Categoría Docente: Instructor

E-mail: lpupo@uci.cu

Graduado de Ingenería en Ciencias Informáticas en la UCI en el año 2007, con 6 años de experiencia en los temas de Gráficos por Computadora y Realidad Virtual. Profesora del Dpto. de Visualización y Realidad Virtual.

Agradecimientos

- A mi familia
- A todos los profesores que me ayudaron de una forma u otra en el desarrollo de la investigación especialmente a mi tutora Liudmila Pupo y el profesor Ernesto Guevara.
- A la UCI, por haberme permitido conocer personas tan especiales como: mis amigas y compañeras Sandra, Irenna, Dayanis, principalmente a Yoana, Libeidy Liusba, que las considero más que mis amigas, mis hermanas blancas como yo les digo. No puedo dejar de mencionar a Adriel, Alexis, Adrián Fonseca Juan Carlos, personas que siempre me apoyaron mucho durante estos años de universidad.

Dedicatoria

Este trabajo está dedicado a mi familia especialmente a mi abuela Virtudes y a mi mamá.

Resumen

El seguimiento de pose del observador es el proceso que se realiza para obtener posición y orientación del punto de vista de la persona que interactúa con un Sistema de Visualización Estereoscópica utilizando una cámara web. Los motores y bibliotecas gráficas que utilizan técnicas de visualización estereoscópica, necesitan conocer la pose [1] del usuario que observa la escena, para mostrar los objetos desde un punto de vista realista en el mundo virtual. Graphics Library Stereo Vision Engine (GLSVe) es una biblioteca gráfica desarrollada en la facultad en conjunto con la Universidad de Oviedo, que permite la creación de sistemas estereoscópicos. El objetivo de este trabajo ha sido incorporarle a GLSVe el seguimiento de pose del observador utilizando cámara web, para lograrlo se realizó un estudio de los métodos y técnicas existentes para la construcción de sistemas de este tipo. Como resultado del estudio se construyó un módulo en C++ con interfaz para C Sharp utilizando la biblioteca Artoolkit, que realiza seguimiento de pose del observador utilizando cámara web y un demo que demuestra la correcta integración con GLSVe.

Palabras clave: GLSVe, observador, seguimiento de pose, visualización estereoscópica

[1] posición y orientación

Índice general

Introducción **1**

Estructura del documento 3

1. Fundamentación Teórica **4**

1.1. Realidad virtual y visualización estereoscópica 4

1.2. Sistemas de seguimiento 7

1.2.1. Técnicas de seguimiento de pose con cámaras de video en tiempo real 10

1.2.2. Técnicas de calibración de cámara 13

1.2.3. Tipos de calibración de cámara 15

1.3. Bibliotecas para el desarrollo de visión por computadora 15

1.3.1. *OpenCv* 16

1.3.2. *ArToolkit* 16

1.3.3. *Touchless* 17

2. Solución Propuesta **19**

2.1. Propuesta del sistema 19

2.2. Calibración de cámara 21

2.3. Descripción del patrón utilizado 23
2.4. Detección y reconocimiento de patrones 24
2.5. Estimación de la posición y orientación del patrón. 26
2.6. Metodologías y herramientas para el desarrollo de software 31
2.6.1. Biblioteca gráfica 31
2.6.2. Lenguajes de programación 31
2.6.3. Framework de desarrollo 32
2.6.4. Entorno integrado de desarrollo utilizado 32
2.6.5. Herramienta de modelado 32
2.6.6. Metodología de software 33

3. Diseño de la solución 34

3.1. Características del sistema 34
3.1.1. Modelo de dominio 34
3.1.2. Personal relacionado con el sistema 35
3.1.3. Requisitos de Software 35
3.1.4. Exploración. Historia de Usuario 37
3.1.5. Planeación del sistema 40
3.2. Construcción de la solución 42
3.2.1. Tarjetas Contenido, Responsabilidad y Colaboración (CRC) . . . 42
3.2.2. Diagrama de componentes 45
3.2.3. Patrones de diseño 46
3.2.4. Estándares de codificación 47
3.3. Desarrollo de las iteraciones 48

3.3.1. Iteración 1 49
3.3.2. Iteración 2 50
3.3.3. Iteración 3 51

4. Análisis de resultados **52**

4.1. Pruebas de aceptación 53

Conclusiones **55**

Recomendaciones **56**

Referencias bibliográficas **57**

Acrónimos **59**

Índice de figuras

1.1. Aplicación en la medicina, juegos y educación 5
1.2. Visualización de los Sistemas Realidad Virtual (SRV) en monitores 2D . 5
1.3. Dispositivos de visualización estereoscópica 5
1.4. Formas de interacción con los Sistemas de Visualización Estereoscópica (SVE) 6
1.5. Vista desde diferentes perspectivas 6
1.6. *Immersion's CyberGlove* 8
1.7. *Ascension Flock of Birds*un dispositivo de *tracking* magnético 9
1.8. Gafas (con led incorporados) utilizadas para el *tracking* infrarrojo 9
1.9. Controles del *Nintendo Revolution* 10
1.10. Técnicas de seguimiento 11

2.1. Composición del sistema propuesto 20
2.2. Modelo geométrico de la cámara 21
2.3. Matriz de parámetros intrínsecos 23
2.4. Patrón de ARtoolKit 23
2.5. Morfología del patrón 24
2.6. Imagen capturada (a), imagen después de la umbralización (b). 25

2.7. Conexión de componentes (c), contornos (d). 25

2.8. Extracción de bordes y esquinas del marcador (e), detección de coordenadas tridimensionales del marcador (f). 26

2.9. Sistemas de coordenadas de la cámara y el marcador 27

2.10. Imagen antes de la normalización y después de la normalización 27

2.11. Vectores de dirección unitaria 29

2.12. Matriz de transformación obtenida 30

3.1. Diagrama de dominio 35

3.2. Diagrama de componentes 46

4.1. Prueba con realidad aumentada 52

4.2. Vista de la aplicación 53

Índice de tablas

3.1. Personal relacionado con el sistema. 35

3.2. Historia de Usuario (HU) activar cámara web. 37

3.3. HU cargar el fichero del patrón. 38

3.4. HU detectar y reconocer un patrón. 38

3.5. HU posiciones en XYZ. 38

3.6. HU orientación del patrón. 39

3.7. HU vector orientación. 39

3.8. HU actualización de posición y orientación. 39

3.9. HU cerrar video. 40

3.10. Esfuerzo de trabajo. 40

3.11. Estimación del tiempo de construcción de cada iteración. 42

3.12. Descripción de la clase Tracking_camara. 43

3.13. Clase Video de la biblioteca ARtoolKit 43

3.14. Clase que contiene las principales funcionalidades de ARtoolKit 44

3.15. Descripción de la clase C_tracking. 44

3.16. Descripción de la biblioteca gráfica. 45

3.17. Descripción de la clase Tracking. 45

3.18. Estimación de tiempo de la iteración1 49

3.19. Estimación de tiempo de la iteración2 50

3.20. Estimación de tiempo de la iteración3 51

4.1. Pruebas de Aceptación realizadas al sistema 54

Introducción

Los SRV brindan una nueva interfaz que ayuda a crear la ilusión de que el usuario está inmerso en un mundo generado por el ordenador, presupone una visualización avanzada de entornos tridimensionales [Tecnopeixe, 2008].

El desarrollo de los motores y bibliotecas gráficas han propiciado el auge de los SRV. Las aplicaciones de estos sistemas se ven reflejadas en diferentes sectores de la sociedad como: la medicina, los video-juegos, la educación y la investigación científica.

En la actualidad las técnicas de visualización estereoscópica están incluidas entre las funcionalidades que brindan los motores gráficos. Estas técnicas contribuyen a mejorar la visualización de aplicaciones tridimensionales (3D) en dispositivos 2D (monitores, televisores, pantallas de proyección), aumentando la sensación de profundidad en estas aplicaciones.

Aunque estas técnicas mejoran la inmersión visual del observador que interactúa con los sistemas estereoscópicos; presentan el inconveniente de que si éste cambia su posición relativa respecto a la pantalla, los objetos de la escena no son actualizados para ser observados desde el nuevo punto de vista.

La forma convencional de solucionar el problema, usada por los motores gráficos, es permitir cambiar la posición y orientación de los objetos a través de la interacción con *mouse* y teclado. De esta forma se obtienen aproximaciones pobres y poco realistas. Una mejor forma sería estimar la posición y orientación del observador respecto a la escena en tiempo real.

El seguimiento de pose es la técnica que permite obtener posición y orientación del observador en una escena mediante una cámara en cada instante de tiempo [2].

Para la creación de aplicaciones estereoscópicas, nuestra universidad cuenta con una biblioteca llamada GLSVe. Esta biblioteca no tiene incorporada funcionalidades que permitan realizar un seguimiento de pose del observador, esto presupone que los usuarios siempre observan los objetos mostrados desde una perspectiva fija, aunque varíe de posición o ángulo de visión. Existe entonces la **necesidad** de incorporar a GLSVe esta funcionalidad.

Por lo tanto el **problema científico** que esta se ha propuesto resolver investigación es ¿Cómo brindar seguimiento de pose del observador a la biblioteca GLSVe?

El **objeto de estudio** de este trabajo son los sistemas de seguimiento y el **campo de acción** estará sobre los sistemas de seguimiento de pose a usuario [3].

El **objetivo** de la investigación es desarrollar un módulo que permita realizar seguimiento de pose del observador para ser incorporado a la biblioteca GLSVe.

Para dar cumplimiento al objetivo propuesto se han definido las siguientes tareas de investigación:

- Investigar los elementos a tener en cuenta en un sistema de seguimiento de pose, para definir el adecuado para un sistema de visión estereoscópica.
- Conciliar diferentes bibliotecas para el trabajo con la cámara y selección de una de ellas.
- Investigar las técnicas de calibración extrínseca e intrínseca de la cámara, para lograr seguimiento de pose del observador de forma más efectiva.
- Elaborar soluciones técnicas que permitan realizar seguimiento de pose del observador para ser incorporado a la biblioteca GLSVe.

[2] Esto se conoce en la literatura científica como *tracking*
[3] Esto se conoce en la literatura científica como *head-tracking*

- Realizar una aplicación de prueba que permita visualizar las nuevas funcionalidades incorporadas a la biblioteca GLSVe.

Con el cumplimiento de las tareas propuestas se podrá dotar a GLSVe de un módulo que permita realizar el seguimiento de la pose del observador. Pudiéndose obtener en las aplicaciones finales que se elaboren, un mayor realismo de las escenas simuladas.

Para mejor compresión del documento, se dividirá en capítulos los cuales contendrán todas las especificaciones de las tareas planteadas.

El **Capítulo 1** “Fundamentación Teórica” se analizaron los sistemas de seguimiento de pose, las técnicas y métodos para realizar seguimiento de pose utilizando cámara web, las bibliotecas de visión por computadora utilizadas para este tipo de seguimiento y la calibración de cámara.

El **Capítulo 2** “Soluciones Técnicas” se expone la propuesta de solución al problema, además de los métodos, técnicas, bibliotecas y herramientas utilizados.

El **Capítulo 3** “Diseño de la solución” se exponen el análisis, diseño y construcción de la solución propuesta.

El **Capítulo 4** “Análisis de resultados” se exponen las pruebas de precisión, aceptación e integración del módulo de seguimiento de pose realizado.

Capítulo 1

Fundamentación Teórica

En este capítulo se realiza una descripción detallada de la situación problémica de la investigación. También se exponen las técnicas y bibliotecas de visión por computadora, que permiten realizar seguimiento de pose del observador utilizando cámara web, en sistemas de visualización estereoscópicos.

1.1. Realidad virtual y visualización estereoscópica

Un SRV, es un sistema de visualización por computadora que se encarga de simular algún fenómeno del mundo real valiéndose de varios dispositivos y recursos para la visualización e interacción. Estos sistemas tienen gran auge y desarrollo en la actualidad y su uso se extiende a diferentes campos de la sociedad como: la medicina, los video-juegos, la educación y la investigación científica.

Los dispositivos en los que se muestran las aplicaciones de Realidad Virtual (RV) son planos (ver figura 1.2), esto dificulta en gran medida que una persona pueda percibir que está observando un objeto u escena 3D. Por esta razón muchas personas no perciben la sensación de inmersión en estos sistemas.

La **inmersión** se logra tratando que la persona que interactúa con la aplicación virtual

Figura 1.1: Aplicación en la medicina, juegos y educación

Figura 1.2: Visualización de los SRV en monitores 2D

se sienta como si estuviera en el mundo real, para esto se actúa sobre algunos sentidos como el oído, el tacto y la vista. Para aumentar la inmersión visual surgen las técnicas de Visualización Estereoscópica (VE).

Los motores y bibliotecas gráficas incluyen técnicas de VE para desarrollar aplicaciones que permitan representar escenas 3D con mayor sensación de profundidad visual y realismo a la vista del observador. Cada una de estas técnicas tiene un formato asociado y cada formato utiliza un dispositivo (ver figura 1.3)para su visualización.

Figura 1.3: Dispositivos de visualización estereoscópica

La **interacción** es la capacidad que tienen los SRV y los SVE de poder introducir datos del mundo real al entorno virtual y poder sentir o percibir sus reacciones. Permitiendo

al espectador no sólo ver o sentir los objetos en el espacio, sino también modificarlos o afectarlos de alguna manera [Tecnopeixe, 2008].

La interacción mejora sustancialmente la percepción espacial, porque el movimiento es uno de los efectos visuales que más contribuye a la ilusión de profundidad [Vincent, 2006]. Existen varias formas de introducir datos en los SVE, las más conocidas a través de dispositivos como el *mouse*, teclado, mandos wii y el movimiento del observador que interactúa con la escena.

Figura 1.4: Formas de interacción con los SVE

¿Porque es necesario el seguimiento de pose del observador en los SVE?

Un aspecto importante para lograr la correcta visualización de los objetos en los SVE es el punto de vista desde el cual la persona los observa. Esto se fundamenta en que cuando miramos un objeto en el mundo real desde diferentes posiciones, se obtienen diferentes vista del objeto o sea la perspectiva del objeto también cambia 1.5.

Figura 1.5: Vista desde diferentes perspectivas

Los motores gráficos desarrollan aplicaciones desde un punto de vista estático, de esta forma, si la persona que está interactuando con la aplicación cambia la posición u orientación de su cabeza, no visualizaría el objeto como si estuviera en el mundo real. Una forma de solucionar el problema sería conocer la posición y orientación de

la cabeza del observador de la escena en tiempo real y así actualizar el modo en que la escena muestra los objetos en cada momento.

La precisión y exactitud del seguimiento de pose del observador es fundamental para lograr una adecuada interacción e inmersión de los usuarios en los sistema de realidad virtual que usen VE, si el punto de vista del observador no se encuentra en la posición que el *tracking* reporta, el usuario verá los objetos distorsionados o escalados (grande, pequeños, comprimidos) causando efectos y sensaciones no deseadas, que conllevan al fracaso de las aplicaciones.

1.2. Sistemas de seguimiento

Diferentes autores definen el *tracking* o seguimiento como:

- La técnica que permite conocer en cada momento la posición y orientación del usuario en el mundo virtual [Tecnopeixe, 2008].
- La localización automática del usuario y de su orientación [Camacho, 2009].
- El proceso de seguir un punto o una serie de puntos cuadro a cuadro en una secuencia de datos que varían al transcurrir el tiempo, para obtener posición y orientación en cada momento [López, 2007].

En esta investigación cuando se hable de seguimiento de pose del observador se estará haciendo referencia al *tracking* de usuario para obtener posición y orientación del punto de vista del usuario que interactúa con un SVE.

Entre las fuentes de datos más comunes que permiten seguir la pose del observador, para obtener información de sus movimientos, se encuentran: los campos electromagnéticos, los sistemas mecánicos, las señales ultrasónicas, los leds infrarrojos y el procesamiento de video:

El ***tracking* ultrasónico** usualmente utiliza frecuencias de sonido alrededor de los 40 kHz. Mediante métodos de triangulación, se obtiene el tiempo y la localización del sonido que viaja desde un emisor a uno o varios receptores. La ventaja de esta técnica se encuentra en la pequeña escala de transmisores que utiliza y la desventaja está en la velocidad a la que viaja el sonido, lo cual convierte en lento el proceso de sincronización de sonido y la actualización de los valores del sistema, además la velocidad puede variar dependiendo del entorno en que se encuentre, lo que provoca que el *tracking* no sea estable [Noris, 2005].

El ***tracking* mecánico** es utilizado fundamentalmente para la captura de ángulos y rangos de movimiento mediante sensores flexibles. La ventaja de estos sistemas consiste en la exactitud y la velocidad con que se actualizan los datos y no son susceptibles a interferencias, pues los sensores son directamente cableados en la máquina. La principal desventaja del seguimiento mecánico es el volumen de trabajo que es limitado por la instalación eléctrica de la propia máquina [Noris, 2005].

Figura 1.6: *Immersion's CyberGlove*

El ***tracking* electromagnético** usa los mismos principios que el seguimiento ultrasónico, envía un campo magnético el cual es inducido dentro de sensores que pueden medir la distancia desde el origen. La dirección del campo magnético generado, permite al sensor extraer información de orientación del impulso registrado. La desventaja de esta técnica consiste en que el tamaño del volumen de trabajo va a estar limitado por la potencia que tenga el campo magnético generado y la interferencia con cuerpos metálicos en la vecindad [Noris, 2005]. La figura 1.7 muestra algunos de

los dispositivos utilizados para generar un campo magnético.

Figura 1.7: *Ascension Flock of Birds*un dispositivo de *tracking* magnético

El ***tracking* infrarrojo** es un tipo de *tracking* de video que resuelve el problema de la iluminación del medio y el ruido en la imagen, pero la desventaja de esta tecnología en comparación con el seguimiento de video estándar es el costo del equipamiento (ver figura 1.8) y la interferencia de cuerpos calientes en el campo de visión de la cámara [Noris, 2005].

Figura 1.8: Gafas (con led incorporados) utilizadas para el *tracking* infrarrojo

El ***tracking* de radio** en la industria del entretenimiento, el *Nintendo Revolution* (ver figura 1.9) es controlado por varios jugadores al mismo tiempo. Dos sensores de radio permiten determinar la posición de los controles mediante métodos de triangulación y al inclinar los sensores que se encuentran dentro de los controles permite medir la orientación.

El ***tracking* de video** es la solución más accesible, pues el equipamiento para realizar el seguimiento está limitado solamente a un video o una cámara de video. Una o más cámaras permiten calcular posición y orientación de un objeto o un patrón proyectado dentro de la imagen. Es la mejor opción para el trabajo con aplicaciones en tiempo

Figura 1.9: Controles del *Nintendo Revolution*

real. Las principales desventajas de este tipo de tracking son: tienden a disminuir con la distancia, las condiciones del medio luminoso y el ruido influyen en la ejecución del seguimiento causando problema al reconocer el objeto se desea seguir [Noris, 2005].

El *tracking* de video tiene gran variedad de usos, algunos son: la interacción humano-computadora, la seguridad y la vigilancia, la comunicación y compresión de vídeo, la realidad aumentada, el control de tráfico y en el cine.

1.2.1. Técnicas de seguimiento de pose con cámaras de video en tiempo real

Siempre que se trabaja con vídeo (tanto en tiempo real como pre-grabado) y se desea saber lo que sucede en la secuencia (movimiento de la cámara o de los objetos), parte del problema se basa en saber hacia dónde se mueven ciertos puntos característicos en un fotograma con respecto a los siguientes o anteriores. En la captura de movimiento en vídeo, debe estimarse el movimiento de ciertos bloques de píxeles.

El proceso para hacer seguimiento de pose del observador mediante cámara consta técnicamente de dos fases: la detección del usuario, para localizarlo dentro de la imagen en la secuencia de video y el reconocimiento, para identificarlo en cada secuencia de video.

Según [Carlos Pérez, 2003a], [Carlos Pérez, 2003b] existen dos vías para realizar el seguimiento de pose del observador: basadas en marcas y las no basadas en marcas, como se muestra en la figura 1.10.

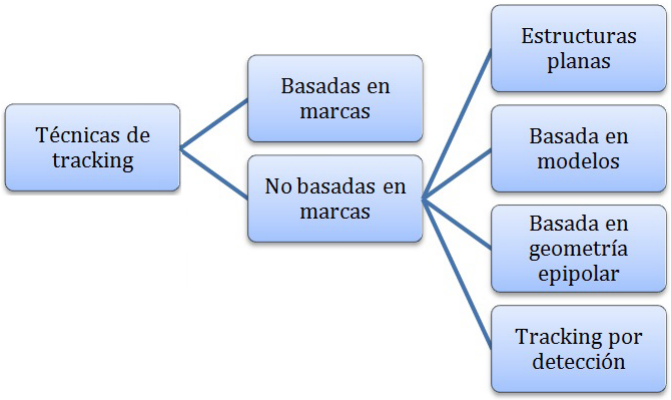

Figura 1.10: Técnicas de seguimiento

Seguimiento basado en marcas

El seguimiento basado en marcas es utilizado en casos en los que el espacio de la escena se encuentra bien definido y las características del entorno donde se van a utilizar son bien conocidas. Los patrones deben ser únicos en el contexto que se vayan a utilizar y difícil de confundir con características y colores que pueda poseer la persona que lo va a utilizar. Una mala selección del patrón, pueden incidir de forma negativa en la precisión de los valores del seguimiento.

Para la detección y reconocimiento de patrones dentro de una imagen se utilizan técnicas de análisis de imágenes y métodos de detección de patrones, que utilizan *Template Matching* y bases de imágenes de conocimiento.

Seguimiento no basado en marcas

Cada una de las técnicas no basadas en marcas emplea diferentes características para realizar el seguimiento. Estas características pueden ser del entorno en que se desarrolla el *tracking* o específicas del objeto a seguir.

El seguimiento basado en análisis de estructuras planas, se refiere a elementos geométricos presentes en la escena, que contengan superficies planas y puntos clave presentes en estas superficies.

El seguimiento basado en modelos como su nombre lo indica, emplean modelos geométricos de tipo CAD o una proyección del objeto, dependiendo del entorno sobre el que se va a ejecutar el seguimiento. Los seguimientos basados en modelos suelen ser robustos, precisos y fáciles de implementar.

El seguimiento basado en características extraídas de la imagen, se basa fundamentalmente en el planteamiento de la siguiente interrogante: ¿por qué hacer seguimiento del objeto entero cuando se puede obtener el mismo resultado haciendo seguimiento solo de las características? Este planteamiento suele ser computacionalmente más eficiente que el basado en modelo, pero es menos robusto [Carlos Pérez, 2003a]. Las dos últimas técnicas (basadas en geometría epipolar y seguimiento por detección) usan diferentes tipos de restricciones geométricas para la realización del seguimiento.

Según la bibliografía consultada [Gutiérrez, 2010] dependiendo de los grados de libertad que se quieran obtener con el sistema, se pueden definir 3 tipo de seguimiento de pose: seguimiento 2D [1], 4D [2] y 3D[3].

En la variante 2D solo se toma en consideración la posición del usuario en el plano imagen, es decir, no tenemos en cuenta la distancia a la cámara. Esta variante presenta la peculiaridad de que una misma traslación del observador es interpretada de distinta forma en función de la distancia de éste a la cámara.

En la variante con 4D se considera la traslación de la cabeza en los 3 ejes de coordenadas, es decir, se añade un nuevo grado de libertad determinado por la distancia de la cabeza a la pantalla y un segundo grado más definido por el movimiento de la cabeza hacia los lados del cuerpo.

[1] 2 grados de libertad, movimiento en los ejes x,y.

[2] Es un término medio que se utiliza para referirse a los 4 grados de libertad, movimiento en los ejes (x,y,z) y rotación en (x,y)

[3] 6 grados de libertad, movimiento y rotación en los ejes x,y,z.

La variante 3D considera la traslación en los 3 ejes, también se considera el ángulo de la cabeza con respecto a los 3 ejes. Esta variante es la más compleja, pero permite reflejar todos los movimientos de la cabeza.

1.2.2. Técnicas de calibración de cámara

Otro de los aspectos importantes cuando se va a implementar un seguimiento de pose utilizando cámara es la calibración, pues tiene gran influencia en la precisión y exactitud del *tracking*. La calibración de cámara comprende dos aspectos distintos. En primer lugar es necesario determinar la relación que existe entre la posición respecto a la cámara de los puntos observados y la imagen que de ellos se obtiene.

La bibliografía [Rodríguez, 2003], clasifica la calibración de cámara en:

Técnicas de optimización lineal. Su principal ventaja es la simplicidad del modelo empleado, que revierte en un algoritmo de cómputo simple y rápido. Mediante un proceso de ajuste por mínimos cuadrados se determina la matriz que relaciona las coordenadas tridimensionales de los puntos de control y las de sus imágenes. En su contra tienen que no son aptos cuando introducimos la distorsión de la lente como un factor a calibrar, limitando por tanto la exactitud de las medidas realizadas, y la dificultad aparejada a la obtención de los parámetros a partir de la matriz calculada.

Técnicas de optimización no lineal. Cuando el modelo empleado para la cámara se aleja del pin-hole básico y se introducen parámetros que reflejan la distorsión causada por la presencia de lentes.

Otras bibliografías como [Garcia, 2007], aportan otras clasificaciones:

Computación lineal vs no lineal:

- Lineal: usan técnicas de resolución de sistemas de ecuaciones lineales, son muy simples de implementar y muy rápidos (como el DLT [4]).

[4] *Direct Linear Transform*, algoritmo creado en 1971 para calibrar cámaras

- No lineal: se basan en el uso de métodos iterativos, como el algoritmo *Gold Standard*. Generalmente se requiere una buena aproximación inicial obtenida de un método lineal. Son mucho más lentos, pero permiten resolver modelos de cámara más complejos (ej: modelar la distorsión) que incluyen un mayor número de parámetros.

Calibración explícita vs implícita:

- Explícita: se obtienen los valores de cada uno de los parámetros que forman el modelo.
- Implícita: se obtienen generalmente matrices de transformación que contienen el conjunto de todos los parámetros. No se puede conocer el valor exacto de algunos parámetros.

Calibración en un paso vs multipaso

- Un solo paso: en cada ciclo del proceso de resolución se actualizan todos los parámetros a la vez.
- Multipaso: en cada fase se obtiene un conjunto distinto de parámetros, usándose aproximaciones en los primeros pasos para aquellos que aún no se hayan calculado y aplicándose los resultados que se van obteniendo en los siguientes pasos.

Patrón en un plano vs múltiples planos

- Todos los puntos del patrón están en el mismo plano. Por tanto, tienen la ventaja de reducir el ruido en las coordenadas del patrón, ya que una de las coordenadas 3D es nula.

- Multipaso y múltiples planos: dentro de este grupo se pueden distinguir dos tipos. Por un lado, aquellos que necesitan conocer la relación entre los planos, generalmente se opta por que formen un diedro, es decir, dos planos que forman un ángulo de 90 grados.

  Por otro lado, aquellos en los que no es necesario conocer la relación entre las posiciones de los planos. Generalmente pueden ser adquisiciones sobre el mismo patrón variando la colocación del mismo o realizando movimientos de la cámara.

1.2.3. Tipos de calibración de cámara

Calibración intrínseca de la cámara, consiste en extraer de la matriz de calibración obtenida de una imagen, las características intrínsecas son: propiedades internas como la distancia focal, centro de proyección, no cambian si se mueve la cámara [Garcia, 2007].

Calibración extrínseca de la cámara consiste en extraer de la matriz de calibración obtenida de una imagen, las propiedades siguientes: vectores de traslación y rotación relativos al movimiento [Garcia, 2007].

1.3. Bibliotecas para el desarrollo de visión por computadora

Conocer e identificar un objeto (incluida su posición relativa al observador), así como recuperar algunas de sus características, es una labor que normalmente recae sobre la disciplina conocida como visión por computadora. La realidad virtual se basa en gran medida en esta disciplina, y si bien la visión no es el único sentido para ser aumentado, sí que es uno de los más importante en lo que atañe a la misma [Camacho, 2009].

A continuación se realiza una descripción de bibliotecas que sirven como base para desarrollar aplicaciones que utilizan técnicas de visión por computadora.

1.3.1. *OpenCv*

OpenCv(Open Source Computer Vision Library)[Bary Gradsky, 2008]. Es una biblioteca de tratamiento de imágenes, destinada principalmente a aplicaciones de visión por computadora en tiempo real. La biblioteca está escrita en C y C++ corre bajo *Linux*, *Windows* y *Mac OS X*. Tiene un activo desarrollo para interfaces como: *Python*, *Ruby*, *Matlab* y otros lenguajes.

OpenCv se caracteriza por implementar una gran variedad de herramientas para la interpretación de la imagen, siendo la *Iplimage* su principal estructura para el tratamiento de las mismas. Es compatible con *Intel Image Processing Library (IPL)* que implementa algunas operaciones en imágenes digitales y permite optimizar las funcionalidades en más de la mitad de las estructuras. *OpenCv* es principalmente una biblioteca que implementa algoritmos para: la calibración de cámara, la detección de rasgos, rastrear (Flujo Óptico), el análisis de la forma (Geometría, Contorno que Procesa), el análisis de movimientos (Plantillas del Movimiento, Estimadores), la reconstrucción 3D (Transformación de vistas), la segmentación de objetos y el reconocimiento (Histograma, etc.).

1.3.2. *ArToolkit*

Es una biblioteca para el desarrollo de aplicaciones de realidad aumentada en tiempo real. Es de libre uso para fines no comerciales y fue desarrollada por el Dr. Hirokazu Kato de la Universidad de Osaka en Japón. El seguimiento del punto de vista del usuario se realiza siguiendo diversos patrones cuadrados. Principalmente incluye funciones de seguimiento y reconocimiento, aparte del render OpenGL para los objetos

sintéticos. Es una biblioteca originalmente en C++, aunque permite convertir código a otros lenguajes como Java o Python.

ArToolkit utiliza técnicas de visión por computadora para calcular la posición y orientación de la cámara respecto a un patrón defino por la biblioteca, permitiendo al programador conocer donde se encuentra exactamente el mismo dentro de la escena.

Entre las funcionalidades que se pueden lograr con *ArToolkit* se encuentran:

- Seguimiento y orientación individual de la cámara.
- Utiliza códigos de seguimiento sencillos y optimizados.
- Detecta patrones cuadrados dentro de una imagen.
- Sencillo código de calibración de la cámara.

Las funcionalidades de *ArToolkit*, además de permitir crear aplicaciones de realidad aumentada, son muy eficiente para desarrollar sistemas de seguimiento basado en marcas.

1.3.3. *Touchless*

Se trata de una biblioteca para la plataforma .NET que permite crear aplicaciones *multi-touch* con una cámara web como interfaz. *Touchless* incluye una aplicación de prueba que se puede descargar y capturar gestos que luego serán reconocidos. Se caracteriza por[wittysparks, 2004]:

- Disponible solo para Microsoft Windows.
- Reconocimiento de gestos humanos a través de cámara web.

Touchless es un experimento no concluido y en pleno desarrollo que aún no es nada sofisticado. Es un motor sencillo que explota técnicas no muy avanzadas de reconocimiento de imágenes para buscar objetos llamativos en las imágenes capturadas por la cámara web y usarlos como marcadores que luego se utilizarán para interactuar con la aplicación. [wittysparks, 2004].

Después de haber analizados las diferentes bibliotecas se puede concluir que la biblioteca *OpenCv* puede ser utilizada para realizar seguimiento basado en características y marcas por la eficiencia de sus algoritmos para el tratamiento y análisis de imágenes. Para el seguimiento basado en marcas es mejor utilizar ARtoolKit, es una biblioteca orientada específicamente a la detección de marcas, ahorra tiempo y tamaño de código en las aplicaciones. Ambas bibliotecas implementan algoritmos para realizar calibración lineal y no lineal de cámara. ARtoolKit además posee un registro amplio de calibraciones de cámara, de donde se pueden obtener los valores de distorsión de lente y matriz intrínseca sin necesidad de realizar el proceso de calibración completo. Touchless no es muy recomendable pues no posee una estructura robusta y posee grandes desventajas con sus homólogas desde el punto de vista funcional.

Capítulo 2

Solución Propuesta

En este capítulo se describe la propuesta de solución para resolver el problema planteado en la investigación, se proponen métodos y algoritmos de detección y reconocimiento de patrones, descripción de la geometría de cámara y las herramientas seleccionadas para el desarrollo de la aplicación.

2.1. Propuesta del sistema

El sistema estará compuesto por una cámara de video situada frente al observador de la escena. La cámara captura los movimiento del observador mediante un patrón situado en el dispositivo que se utilice para visualizar el formato de la escena estereoscópica, como muestra la figura 2.1.

El sistema debe capturar todos los movimientos que pueda realizar el observador de una escena mientras no pierda el contacto visual con la cámara, por lo que se decidió realizar un seguimiento 3D de la pose. El observador para visualizar el formato de las técnicas mostradas en las escenas estereoscópicas utiliza dispositivos de visualización, conociendo donde se encuentran posicionados y como están orientados estos dispositivos en la escena, se pueden estimar la posición y orientación del punto

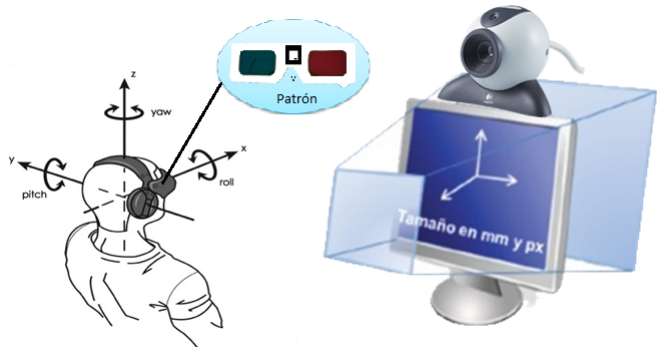

Figura 2.1: Composición del sistema propuesto

de vista del observador. Los dispositivos pueden variar su modelo, tamaño, color, en dependencia de la técnica que se visualice o la fabricación de los mismos, por lo que se descartó la posibilidad de realizar un seguimiento basado en características. Utilizar patrones en los dispositivos permite realizar un seguimiento independientemente de las características que posean los mismos, por lo que se utilizará un seguimiento basado en marcas. La captura de las imágenes en tiempo real se realiza a través de una sola cámara, por este motivo se realiza la calibración para obtener la matriz intrínseca, fundamental para realizar un seguimiento con mayor precisión.

Como el sistema de seguimiento es basado en marcas, se seleccionó la biblioteca ArToolkit[Hirokazu Kato, 2009] a la cual se hizo referencia en el capítulo anterior, es una biblioteca de procesamiento de imágenes en video, con amplias funcionalidades para el trabajo con marcadores.

Actualmente a los SVE creados en la facultad no brindan la funcionalidad de interactuar con los usuarios mediante el seguimiento del movimiento de la cabeza. La GLSVe tiene un sistema de seguimiento que utiliza colores como patrón a seguir, lo que tiende a confundir muchas veces la zona de detección y reconocimiento de la marca, además de los problemas de iluminación característicos de estos sistemas, estos inconvenientes afectan considerablemente la precisión de los valores que se obtienen. En el seguimiento de marcas que se propone se usan patrones cuadrados que mejoran considerablemente la precisión y exactitud del seguimiento. También se

utiliza la técnica de binarización de imagen dado un valor de intensidad de píxel, que mejora sustancialmente los problemas que causa la iluminación del medio en el que se desarrolle el seguimiento.

2.2. Calibración de cámara

La calibración es el primer paso que se realiza cuando se trabaja con una cámara, pues garantiza mayor exactitud en la estimación de los puntos. A continuación se describe la geometría de cámara, para ayudar a entender el origen de los valores de la matriz intrínseca de calibración utilizada en la solución. Esta descripción está basada en: [Garcia, 2007] y [Bary Gradsky, 2008], documentos que pueden ser utilizados para profundizar en el contenido.

Se definen dos modelos de cámaras: aquellas con un centro definido, y las que tienen su centro en el infinito. Respecto a estas últimas, hay una caracterización concreta llamada cámara afín, que se usa como generalización de la proyección paralela (sin perspectiva). La figura 2.2, muestra el modelo geométrico de una cámara con centro de proyección definido (*pin-hole model*).

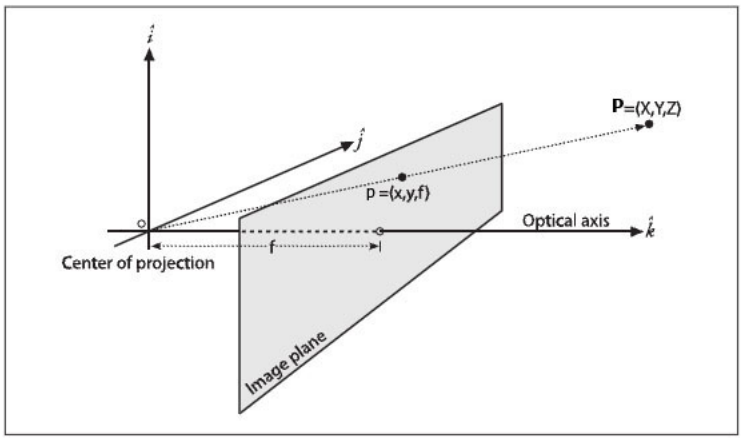

Figura 2.2: Modelo geométrico de la cámara

Donde f es la distancia focal, P la distancia del objeto respecto al centro de proyección

y p es la proyección del objeto en el plano de la imagen. A este modelo se le incluyen dos parámetros nuevos c_x y c_y que se interpretan como el posible desplazamiento que pueda poseer el centro de proyección de la imagen. El resultado es un modelo relativamente simple en el cual el punto P del mundo físico cuyas coordenadas son (X,Y,Z), es proyectado en la pantalla en algún píxel, dado por las siguientes ecuaciones 2.1, 2.2.

$$x_{pantalla} = f_x(\frac{X}{Z}) + c_x \tag{2.1}$$

$$y_{pantalla} = f_y(\frac{Y}{Z}) + c_y \tag{2.2}$$

¿Cómo se puede entonces caracterizar una cámara a partir de una imagen tomada con ella? La relación de un objeto del mundo físico denotado por un mapa de puntos P con coordenadas (X, Y, Z) a puntos en la imagen p=(x, y) es llamado transformaciones proyectivas. Si se dispone de las suficientes correspondencia $X_i \longleftrightarrow x_i$, se puede estimar una matriz de cámara P_{3x4} (ver figura 2.12), de forma que x = PX . Cuando se realizan transformaciones proyectivas, es conveniente utilizar coordenadas homogéneas. Las coordenadas homogéneas asociadas a un punto en un espacio proyectivo de dimensión n, se expresa típicamente como (n + 1) - la dimensión del vector (por ejemplo, x, y, z se convierte en x, y, z, w), debe cumplirse también, que si cualquiera de los dos puntos proporcionales son equivalentes. En este caso, el plano de la imagen representa el espacio proyectivo y tiene dos dimensiones, entonces los puntos en el plano se representaran como un vector tridimensional p=(p_1, p_2, p_3) . Todo esto se formula de forma ideal, suponiendo que la lente de la cámara es perfecta y no presenta distorsión. El resultado final de este proceso es una matriz de 3x4, que contiene las propiedades físicas de la cámara utilizadas posteriormente en el proceso de estimación de la posición y orientación del patrón a seguir en la pantalla.

$$\begin{bmatrix} sf_x & 0 & x_c & 0 \\ 0 & sf_y & y_c & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix} = \mathrm{P} \begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix}$$

Figura 2.3: Matriz de parámetros intrínsecos

La matriz intrínseca se obtuvo utilizando el fichero *camera-para.dat* de la biblioteca ARToolKit.

2.3. Descripción del patrón utilizado

El patrón utilizado es una imagen cuadrada, asimétrica, en blanco y negro, descrita por un cuadro negro que contiene uno blanco cuatro veces más pequeño que él, el cuadro blanco contiene figuras en el centro de color negro, como se muestra en la figura 2.4. El material en que se imprime la plantilla conviene que no sea brillante y el tamaño de la marca debe ser suficiente como para que la cámara capte el dibujo dentro del cuadro blanco y no tan grande como para que no quepa en el ángulo de visión de la cámara. Este mismo tipo de marcador es utilizado también por la biblioteca ARtoolKit.

Figura 2.4: Patrón de ARtoolKit

Las características morfológicas del patrón a seguir están recogidas en un archivo que gestiona la biblioteca ARtoolKit llamado *patt.name_pattern*. En la figura 2.5 puede verse parte del contenido del archivo *patt.* que caracteriza morfológicamente al patrón. La información que brinda puede ser interpretada de la siguiente manera, las cifras indican el nivel de gris de los píxeles del cuadro interior del marcador en un rango de 0 a 255, de modo que 255 representa al blanco puro y 0 al negro puro.

```
234 235 240 233 240 234 240 235 240 237 240 238 240 240 240 232
229 240 240 240 240 240 240 240 240 240 240 240 240 240 240 228
227 240 240 240 240 240 240 240 240 240 240 240 240 240 240 239
240 240 240 240 240 240 240 240 240 240 240 240 240 240 240 240
236 240 240 240 240 240 240 240 240 240 240 240 240 240 240 240
234 240 240 240 240 240 240 240 240 240 240 240 240 240 240 240
236 240 240 240 240 240 240 240 240 240 240 240 240 240 240 240
231 240 240 240 240 240 240 240 240 240 240 240 240 240 240 240
229 240 240 240 240 240 240 240 240 240 240 240 240 240 240 240
225 149 240 240 186 216 225 174 240 240 240 237 238 240 240 240
150 107 238 231  75 208 115 147 238 228 223 226 237 180 226 240
150  62 181 213  62 187 113 169 197  72  29 237 120  50  53 207
149  63  47  78  53 184 113 101 142   5 150 150  45 217 186  83
121  84 220 222  58 180 121  92 128 109 237 124 155 232 161  64
149  71 240 240  76 210  98 109 122 108 240 129  51 119 161 155
149 186 240 240  98 219 135 152 207 191 236 227 152  77 175 209
235 235 240 233 240 234 240 235 240 236 240 238 240 240 240 240
229 240 240 240 240 240 240 240 240 240 240 240 240 240 240 240
227 240 240 240 240 240 240 240 240 240 240 240 240 240 240 240
240 240 240 240 240 240 240 240 240 240 240 240 240 240 240 240
236 240 240 240 240 240 240 240 240 240 240 240 240 240 240 240
234 240 240 240 240 240 240 240 240 240 240 240 240 240 240 240
                              ...
```

Figura 2.5: Morfología del patrón

2.4. Detección y reconocimiento de patrones

En en el desarrollo de esta aplicación se ha necesitado usar algunas funciones externas (de la biblioteca ARToolKit) de análisis de imágenes para detectar los marcadores de los dispositivos de seguimiento en cada *frame* del vídeo de entrada. Estas funciones se basan en un enfoque de detección de esquinas con un algoritmo de estimación orientado a la rapidez.

Lo primero que se realiza es la captura de la imagen(como se muestra en la figura 2.6 inciso (a)) a través de la cámara, la imagen es sometida a un proceso de **umbralización o binarización**(ver figura 2.6 (b)). En este proceso se utiliza un valor de

umbral (de 0 a 255), de forma que los píxeles cuya intensidad supere el valor del umbral definido, son transformados en píxeles de color negro y el resto se transforman en píxeles blancos.

Luego de este proceso la imagen a analizar queda completamente en blanco y negro, esto reducirá el campo de búsqueda y agilizará el proceso de detección. Luego se buscan todas las regiones cuadrada(ver figuras 2.7 (c) y (d)) que se encuentren en la imagen binarizada, todas las regiones cuadradas puede que no sean marcadores, por eso para cada cuadro se captura el patrón que contiene dentro y es comparado constantemente con plantillas de patrones definidas en archivos llamados ***pre-trained***. Cada patrón posee un identificador asociado, después que se detecta el marcador se compara en la secuencia de video el identificador del nuevo patrón encontrado con el anterior. Al finalizar este proceso se obtienen las coordenadas tridimensionales del marcador como se muestra en las imágenes(2.8 (e) y (f)).

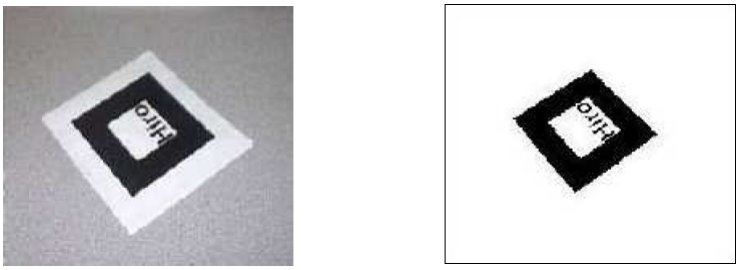

Figura 2.6: Imagen capturada (a), imagen después de la umbralización (b).

Figura 2.7: Conexión de componentes (c), contornos (d).

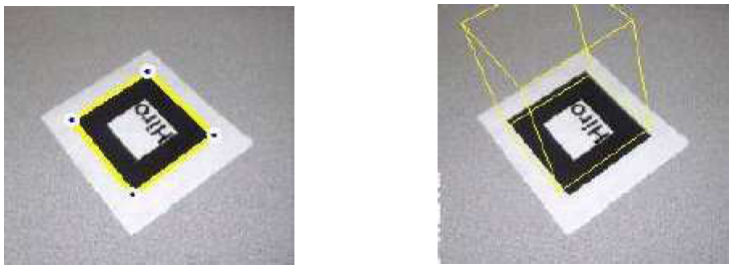

Figura 2.8: Extracción de bordes y esquinas del marcador (e), detección de coordenadas tridimensionales del marcador (f).

2.5. Estimación de la posición y orientación del patrón.

La relación entre las coordenadas del marcador y las coordenadas de la cámara se estima mediante el análisis de la imagen. Conociendo el tamaño del patrón usado para estimar la posición y orientación, se calcula la matriz de transformación de las coordenadas del marcador a las coordenadas de la cámara. Esta matriz de transformación (Tcm) se puede ver representada en las siguientes ecuaciones 2.3 y 2.4.

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} = \begin{pmatrix} V_{11} & V_{12} & V_{13} & W_x \\ V_{21} & V_{22} & V_{23} & W_y \\ V_{31} & V_{32} & V_{33} & W_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{pmatrix} \tag{2.3}$$

$$= \begin{pmatrix} & V_{3x3} & & W_{3x1} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{pmatrix} = Tcm \begin{pmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{pmatrix} \tag{2.4}$$

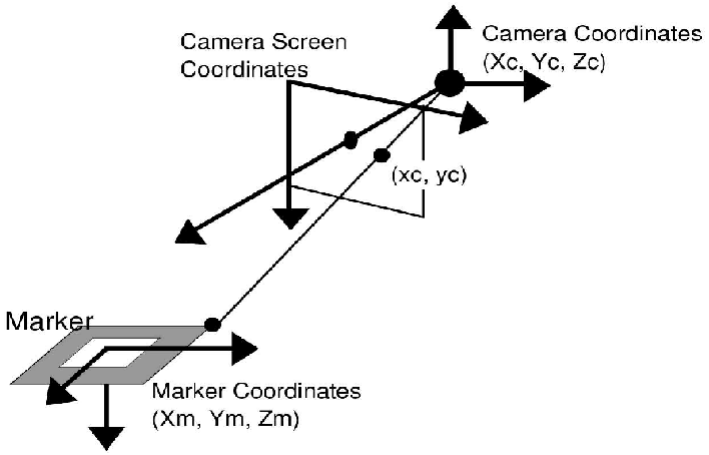

Figura 2.9: Sistemas de coordenadas de la cámara y el marcador

Después de tratar la imagen de entrada con una umbralización en blanco y negro, los cuadriláteros contorneados exteriormente pueden ser definidos por cuatro segmentos de líneas que son extraídos. Los parámetros de estos cuatro segmentos de líneas y las coordenadas de los cuatro vértices donde los segmentos de líneas se intersectan son almacenados por un proceso posterior realizado por ARtoolKit.Los cuadriláteros formados por los segmentos de líneas son normalizados (ver figura 2.10) y la sub-imagen interna al cuadrilátero es comparada por emparejamiento con el patrón que se le ha dado al sistema inicialmente.

Figura 2.10: Imagen antes de la normalización y después de la normalización

Para este proceso de normalización la siguiente ecuación 2.5 representa la transformación de la perspectiva que se usa. Todas las variables en la matriz de transformación se determinan sustituyendo las coordenadas de la pantalla y las coordenadas de los cuatro vértices detectados en el marcador por (x_c, y_c) y (X_m, Y_m) respectiva-

mente. Después el proceso de normalización se lleva a cabo usando esta matriz de transformación.

$$\begin{pmatrix} hx_c \\ hy_c \\ h \end{pmatrix} = \begin{pmatrix} N_{11} & N_{12} & N_{13} \\ N_{21} & N_{22} & N_{23} \\ N_{31} & N_{32} & 1 \end{pmatrix} \begin{pmatrix} X_m \\ Y_m \\ 1 \end{pmatrix} \tag{2.5}$$

Cuando dos lados paralelos de un marcador son proyectados en la imagen, las ecuaciones de estos segmentos lineales en las coordenadas de la pantalla de la cámara (*camera screen*) son las siguientes:

$$\begin{aligned} a_{1x} + b_{1y} + c_1 = 0 \\ a_{2x} + b_{2y} + c_2 = 0 \end{aligned} \tag{2.6}$$

Para el patrón el valor de estos parámetros ha sido ya obtenido en el proceso de determinación de líneas. Dada la matriz de la perspectiva de la proyección P obtenida mediante la calibración de la cámara en la ecuación 2.7, las ecuaciones de los planos que incluyen los dos lados respectivamente pueden ser representadas como la 2.8 en el sistema de coordenadas de la cámara sustituyendo X_C y Y_C en la 2.7 por x y y en las ecuaciones 2.6.

$$P = \begin{pmatrix} P_{11} & P_{12} & P_{13} & 0 \\ 0 & P_{22} & P_{23} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} hx_c \\ hy_c \\ h \\ 1 \end{pmatrix} = P \begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} \tag{2.7}$$

$$\begin{aligned} a_1P_{11}X_c + (a_1P_{12} + b_1P_{22})Y_c + (a_1P_{13} + b_1P_{23} + c_1)Z_c = 0 \\ a_2P_{11}X_c + (a_2P_{12} + b_2P_{22})Y_c + (a_2P_{13} + b_2P_{23} + c_2)Z_c = 0 \end{aligned} \tag{2.8}$$

Dado que los vectores normales a estos planos son n_1 y n_2 respectivamente, el vector de dirección de los dos lados paralelos del cuadrilátero es dado por la salida del

producto n_1 y n_2.

Dados los dos vectores de dirección unitarios obtenidos a partir de los dos pares de lados paralelos del cuadrilátero siendo estos u_1 y u_2, estos vectores deben ser perpendiculares. Sin embargo los errores en el procesamiento de la imagen provocan que los vectores no sean exactamente perpendiculares. Para compensar esto dos vectores de dirección unitarios y esta vez sí perpendiculares son definidos como v_1 y v_2 en le plano que contiene a u_1 y u_2 como se muestra en la siguiente figura(2.11).

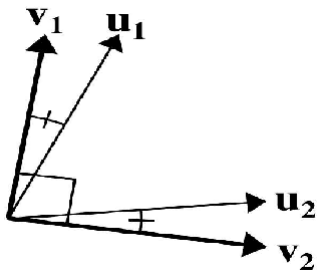

Figura 2.11: Vectores de dirección unitaria

Dado que el vector de dirección unitario perpendicular a v_1 y v_2 es v_3, ya se tiene la componente V_{3x3} de la matriz de trasformación Tcm para pasar de las coordenadas del marcador a las coordenadas de la cámara que sería $[V_1{}^t\ V_2{}^t\ V_3{}^t]$, ver en 2.3.

A partir de la componente de rotación V_{3x3} de la matriz de transformación, se pueden obtener usando las ecuaciones 2.3, 2.4 y 2.7 las coordenadas de los cuatro vértices del marcador en el sistema de coordenadas del marcador, y esas coordenadas en el sistema de coordenadas de la cámara, ocho ecuaciones que contienen los componentes de la transformación W_x W_y W_z son generados y el valor de esas componentes de transformación W_x W_y W_z son obtenidos de esas ecuaciones.

La matriz de transformación encontrada por el método anterior puede tener errores. Sin embargo estos pueden reducirse mediante el siguiente proceso. Las coordenadas del vértice del marcador en el sistema de coordenadas del marcador pueden ser transformadas en coordenadas del sistema de coordenadas de la cámara usando la matriz

de transformación obtenida. Entonces la matriz de transformación se optimiza hasta que la suma de las diferencias entre estas coordenadas transformadas y las coordenadas medidas directamente en la imagen sean mínimas. Aunque hay 6 variables independientes en la matriz de transformación, solo los componentes de la rotación serán optimizados y después la transformación de los componentes es reestimada usando el método mencionado anteriormente. Mediante la iteración de este proceso un número de veces la matriz de transformación es encontrada con más precisión. Es posible llevar a cabo la optimización con todas las seis variables independientes, no obstante el coste computacional sería considerable.

Mediante este proceso la función *arGetTransMat* obtiene la matriz$_{3x4}$ de transformación (como muestra la figura 2.12) entre el sistema de coordenadas del marcador y el sistema de coordenadas de la cámara, es decir la posición y orientación relativa del marcador con respecto a sistema de referencia de la cámara.

$$\begin{bmatrix} V_{11} & V_{12} & V_{13} & W_x \\ V_{21} & V_{22} & V_{23} & W_y \\ V_{31} & V_{32} & V_{33} & W_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 2.12: Matriz de transformación obtenida

2.6. Metodologías y herramientas para el desarrollo de software

2.6.1. Biblioteca gráfica

GLSVe (Graphics Library Stereo Vision Engine), está escrita en C Sharp y programada sobre *OpenGL*. Es diseñada para facilitarle principalmente a investigadores o estudiantes la creación de aplicaciones gráficas y de realidad virtual incorporando técnicas estereoscópicas.

2.6.2. Lenguajes de programación

El C++ es un lenguaje de programación, diseñado a mediados de los años 1980, por Bjarne Stroustrup, como extensión del lenguaje de programación C. Las principales características de C++ son el soporte para programación orientada a objetos y el soporte de plantillas o programación genérica (*templates*). Se puede decir que C++ es un lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos [Oduardo, 2009].

Se utiliza C++ para desarrollar una biblioteca de enlace dinámico (DLL) que realiza el seguimiento de pose del observador que interactúa, con un SVE mediante una cámara web.

Como el objetivo general de esta investigación es integrar a GLSVe la funcionalidad seguimiento de pose del observador, se utilizó el lenguaje de programación C Sharp para desarrollar una interfaz que permitiera incorporar las funcionalidades del módulo hecho en C++ en GLSVe.

2.6.3. Framework de desarrollo

Un *framework* [1] de desarrollo es una infraestructura de software que crea un entorno común para integrar aplicaciones e información compartida dentro de un dominio dado. Es una aplicación semi-completa que contiene componentes estáticos y dinámicos que pueden ser personalizados para obtener aplicaciones de usuario específicas.

Las aplicación se desarrolla en Microsoft. Net, es un *framework* independiente de software, provee un extenso conjunto de soluciones predefinidas para necesidades generales de la programación de aplicaciones, y administra la ejecución de los programas escritos específicamente con la plataforma.

Se utilizaron las funciones de interoperabilidad que brinda el *framework* para importar funciones de C++ a C Sharp.

2.6.4. Entorno integrado de desarrollo utilizado

Un entorno de desarrollo integrado, en inglés *Integrated Development Environment* (IDE), es un programa compuesto por un conjunto de herramientas para un programador. Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

Se utilizó Visual Studio 2008 con el *framework. Net* para desarrollar la interfaz del módulo desarrollado en C Sharp y su integración a la biblioteca GLSVe.

2.6.5. Herramienta de modelado

El Lenguaje Unificado de Modelado(UML) es un lenguaje gráfico para visualizar, especificar, construir y documentar cada una de las partes que comprende el desarrollo de software. Lenguaje Unificado de Modelado (UML) posee formas de modelar

[1] marco de trabajo para el desarrollo de software

conceptos como lo son procesos de negocio y funciones de sistema, además de aspectos concretos como lo son escribir clases en un lenguaje determinado, esquemas de base de datos y componentes de software reusables [Larman, 1999]. Se puede aplicar en el desarrollo de software entregando gran variedad de formas para dar soporte a una metodología de desarrollo de software pero no especifica en sí mismo qué metodología o proceso usar.

Fue seleccionada la edición Community de Visual Paradigm que es gratuita, soporta la versión 2.0 de UML y permite su extensión mediante la conexión de módulos conectables (plug-in) o usando plantillas (templates). Es una herramienta CASE que utiliza UML como lenguaje de modelado. Visual Paradigm Community es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación [Paradim, 2007].

2.6.6. Metodología de software

Extreme Programming (XP) es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo del software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en la retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos pequeños con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico [Beck, 2002]. Por las características anteriormente mencionadas se seleccionó esta metodología.

Capítulo 3

Diseño de la solución

En este capítulo se expone todo lo referente al diseño e implementación de la solución propuesta. Se ilustran los aspectos fundamentales de la ingeniería de software como: requisitos funcionales y no funcionales, historias de usuarios, planificación del desarrollo de la aplicación, descripción de las clases y estándar de codificación.

3.1. Características del sistema

3.1.1. Modelo de dominio

Debido a que no se cuenta con una definición total de los procesos de negocio, se plantea confeccionar un modelo de dominio. Un modelo de dominio es una descripción gráfica del contexto del sistema. Es un modelo conceptual que relaciona gráficamente los términos más importantes que se manejan en el sistema.

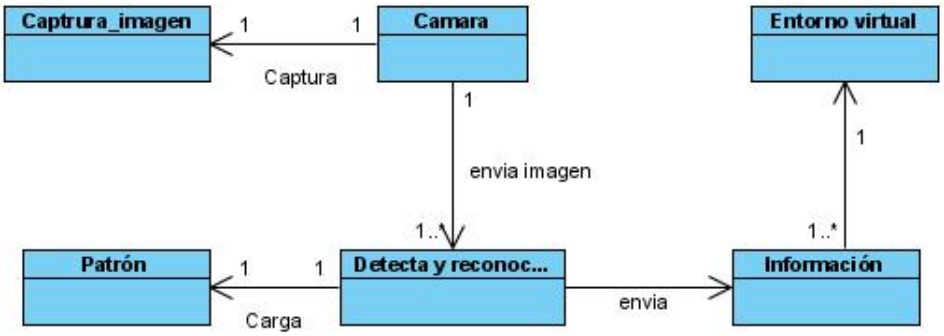

Figura 3.1: Diagrama de dominio

3.1.2. Personal relacionado con el sistema

Se define como persona relacionada con el sistema a aquella que está de una forma u otra vinculada al proceso de desarrollo de la aplicación, y las que interactúan con el mismo.

Personal relacionado con el sistema	Justificación
Desarrollador	Persona encargada de implementar e integrar las funcionalidades de la aplicación.
Usuario	Persona que va a interactuar con la aplicación.

Tabla 3.1: Personal relacionado con el sistema.

3.1.3. Requisitos de Software

Requisitos funcionales:

- Activar la cámara de video
- Cargar fichero del patrón.
- Detectar y reconocer el patrón dentro de la imagen.

- Cerrar video
- Obtener posición en (x,y,z).
- Obtener orientación.
- Obtener Quat [1]
- Actualizar posición y orientación.

Requisitos no funcionales:

Hardware:

- Cámara web
- CPU Pentium 4.
- 512 RAM
- La aceleración gráfica debe estar en correspondencia con aplicación gráfica donde se utilice el sistema.

Software:

- Sistema operativo Window XP / Window 7
- Controlador de la cámara web.

Rendimiento:

- La aplicación debe funcionar en tiempo real.
- La aplicación debe tener un alto nivel de procesamiento y un buen manejo de la memoria de la máquina.

[1] vector de 4 valores que contiene la rotación de un punto, se conoce como quaternion

3.1.4. Exploración. Historia de Usuario

Uno de los artefactos más importantes que genera la metodología XP son las historias de usuario HU. Estas tienen el mismo propósito que los casos de uso y son escritas por el propio cliente, tal y como ven ellos las necesidades del sistema, por tanto son descripciones cortas y escritas en el lenguaje del usuario sin terminología técnica.

Las HU conducen al proceso de creación de las pruebas de aceptación, los cuales servirán para verificar que estas historias se han implementado correctamente. Otra de sus características es que solamente proporcionan los detalles sobre la estimación del riesgo y cuánto tiempo conllevará su implementación.

Durante el análisis de exploración se identificaron 8 HU, cada una de ellas respondiendo a las diferentes funcionalidades solicitadas por el cliente y dando una idea de cómo debe ser su posterior implementación. Estas se describen a continuación:

Historia de Usuario	
No. 1	**Nombre:** Activar cámara
Usuario: usuario	
Programador responsable: Liudmila Cecilia Rodríguez Ricardo.	
Prioridad en el dominio: Alta.	**Nivel de Complejidad:** Alta.
Tiempo de Estimación: 2 semanas.	**Iteración Asignada:** 1
Descripción: Carga un archivo XML, que permite activar una cámara web conectada a la computadora y ajustar los parámetros de calibración.	
Información adicional (Observaciones): La aplicación necesita conocer si existe conectada una cámara web para poder empezar la captura de imágenes.	

Tabla 3.2: HU activar cámara web.

Historia de Usuario	
No. 2	**Nombre:** Cargar fichero del patrón
Usuario: usuario	
Programador responsable: Liudmila Cecilia Rodríguez Ricardo.	
Prioridad en el dominio: Alta.	**Nivel de Complejidad:** Alta.
Tiempo de Estimación: 1 semana.	**Iteración Asignada:** 1
Descripción: Permite cargar el patrón que se va a necesitar en el proceso de detección.	
Información adicional (Observaciones): Para poder empezar la detección de un patrón es necesario cargar los datos del mismo.	

Tabla 3.3: HU cargar el fichero del patrón.

Historia de Usuario	
No. 3	**Nombre:** Detectar y reconocer el patrón dentro de la imagen.
Usuario: usuario.	
Programador responsable: Liudmila Cecilia Rodríguez Ricardo.	
Prioridad en el dominio: Alta.	**Nivel de Complejidad:** Alta.
Tiempo de Estimación: 3 semanas.	**Iteración Asignada:** 1
Descripción: En este proceso mediante métodos de detección se busca el patrón dentro de la imagen capturada del video. Devuelve la una matriz de transformación.	
Información adicional (Observaciones): La aplicación necesita localizar el patrón en la imagen para poder ubicar su posición y orientación.	

Tabla 3.4: HU detectar y reconocer un patrón.

Historia de Usuario	
No. 4	**Nombre**: Obtener posición en (x,y,z).
Usuario: usuario	
Programador responsable: Liudmila Cecilia Rodríguez Ricardo.	
Prioridad en el dominio: Media.	**Nivel de Complejidad:** Media.
Tiempo de Estimación: 2 semanas.	**Iteración Asignada:** 2
Descripción: Obtiene la posición en X,Y,Z del patrón encontrado	
Información adicional (Observaciones): Datos importantes para poder interactuar con los objetos en el mundo virtual.	

Tabla 3.5: HU posiciones en XYZ.

Historia de Usuario	
No. 5	**Nombre:** Obtener matriz de rotación.
Usuario: usuario	
Programador responsable: Liudmila Cecilia Rodríguez Ricardo.	
Prioridad en el dominio: Media.	**Nivel de Complejidad:** Media.
Tiempo de Estimación: 2 semanas.	**Iteración Asignada:** 2
Descripción: Se obtiene la rotación del patrón detectado	
Información adicional (Observaciones): Dato importante para poder interactuar con los objetos en el mundo virtual.	

Tabla 3.6: HU orientación del patrón.

Historia de Usuario	
No. 6	**Nombre:** Obtener Quat
Usuario: usuario	
Programador responsable: Liudmila Cecilia Rodríguez Ricardo.	
Prioridad en el dominio: Media	**Nivel de Complejidad:** Media
Tiempo de Estimación: 1 semanas.	**Iteración Asignada:** 2
Descripción: Se obtiene un vector de 4 posiciones, que contiene la orientación del patrón.	
Información adicional (Observaciones): Dato importante para poder interactuar con los objetos en el mundo virtual.	

Tabla 3.7: HU vector orientación.

Historia de Usuario	
No. 7	**Nombre:** Actualizar posición y orientación en la escena
Usuario: usuario	
Programador responsable: Liudmila Cecilia Rodríguez Ricardo.	
Prioridad en el dominio: media	**Nivel de Complejidad:** Media
Tiempo de Estimación: 1 semanas.	**Iteración Asignada:** 3
Descripción: Actualiza la posición y orientación de los objetos en la escena.	
Información adicional (Observaciones): Se realiza cuando la cabeza del usuario cambia de posición u orientación	

Tabla 3.8: HU actualización de posición y orientación.

Historia de Usuario	
No. 8	**Nombre:** Cerrar video
Usuario: usuario	
Programador responsable: Liudmila Cecilia Rodríguez Ricardo.	
Prioridad en el dominio: Media	**Nivel de Complejidad:** Media
Tiempo de Estimación: 1 semanas.	**Iteración Asignada:** 3
Descripción: Desactiva la cámara de video.	
Información adicional (Observaciones): Cuando el usuario cierre la aplicación desocupa la memoria del sistema.	

Tabla 3.9: HU cerrar video.

3.1.5. Planeación del sistema

En esta fase el cliente establece la prioridad de las historias de usuario y se acuerda el alcance del *release*.

Estimación de esfuerzo por Historia de Usuario

Se realizó una estimación del esfuerzo realizado en cada una de las HU identificadas en la exploración, con el propósito de realizar un buen desarrollo del sistema propuesto.

Historias de usuario	Puntos de estimación
Activar cámara	1 semana
Cargar fichero del patrón	1 semanas
Detección del patrón.	3 semana
Obtener posición en (x,y,z).	1 semanas
Obtener orientación.	1 semanas
Obtener Quat	1 semanas.
Cerrar Video	1 semanas
Actualizar posición y orientación en la escena	1 semanas
Total:	**11 semanas.**

Tabla 3.10: Esfuerzo de trabajo.

Plan de iteraciones

Las HU seleccionadas para cada entrega son desarrolladas y probadas en el ciclo de iteración, de acuerdo al orden establecido.

Iteración 1

Esta iteración tiene propósito de darle cumplimiento a las historias de usuario que se consideraron de mayor importancia(prioridad alta) para el desarrollo de la aplicación. Al finalizar esta iteración se obtuvo una versión de prueba con funcionalidades muy básicas.

Iteración 2

En esta iteración se realizó la implementación de las historias de usuario con prioridad de dominio media y se corrigieron los errores encontrados en la iteración anterior. Al final de esta iteración se le mostró al cliente la segunda versión obtenida, con las nuevas funcionalidades implementadas para ver la aceptación del mismo, y saber si era necesario realizar algún cambio.

Iteración 3

En esta iteración se implementan las restantes historias de usuarios para la obtención del producto final. Después de una revisión de lo realizado en las anteriores iteraciones se obtuvo la versión 1.0 del sistema.

Plan de duración de las iteraciones.

Como parte del ciclo de vida de un proyecto guiado por la metodología de desarrollo de software XP, se crea un plan de duración de cada una de las iteraciones que se llevarán a cabo durante el desarrollo del proyecto. Este plan tiene como finalidad mostrar la duración de cada iteración, así como el orden en que serán implementadas las HU en cada una de las iteraciones.

Iteración	Historias de usuarios	Duración total de las iteraciones
Iteración 1	Activar cámara	6 semanas
	Cargar fichero del patrón	
	Detección del patrón.	
Iteración 2	Obtener posición en (x,y,z).	3 semanas
	Obtener orientación	
	Obtener Quat	
Iteración 3	Cerrar video Actualizar posición y orientación en la escena	2 semanas

Tabla 3.11: Estimación del tiempo de construcción de cada iteración.

3.2. Construcción de la solución

Para el diseño de las aplicaciones, XP no requiere la presentación del sistema mediante diagramas de clases utilizando notación UML, en su lugar se usan otras técnicas como las tarjetas CRC. No obstante el uso de estos diagramas puede aplicarse siempre y cuando influyan en el mejoramiento de la comunicación entre el equipo de desarrollo, no sea un peso su mantenimiento, no sean extensos y se enfoquen en la información importante.

3.2.1. Tarjetas CRC

Para poder diseñar el sistema, se debe cumplir con tres principios: Cargo o Clase, Responsabilidad y Colaboración CRC. Las tarjetas CRC permiten desprenderse del método de trabajo basado en procedimientos y trabajar con una metodología basada en objetos. Debido a la facilidad de uso y entendimiento que propician dichas tarjetas, se ha, decidió utilizarlas para diseñar la aplicación.

Tarjeta CRC	
Clase: Tracking_camara	
Súper Clase: -	
Sub Clase(s): -	
Responsabilidades: • Pintar objeto • Activar cámara web • Cargar patrón • Detectar patrón y reconocer en el video • Obtener posición del patrón en XYZ. • Obtener orientación • Actualizar posición y orientación del observador • Cerrar video	**Colaboraciones:** clase GLSV clase Tracking

Tabla 3.12: Descripción de la clase Tracking_camara.

Tarjeta CRC	
Clase: Video	
Súper Clase: -	
Sub Clase(s): -	
Responsabilidades: • Configurar video. • Capturar imágenes • Parar video	**Colaboraciones:**

Tabla 3.13: Clase Video de la biblioteca ARtoolKit

Tarjeta CRC	
Clase: Ar	
Súper Clase: -	
Sub Clase(s): -	
Responsabilidades: • Estructura para el trabajo con patrones • Cargar ficheros .patt • Métodos para el análisis de imágenes de un video • Contador de tiempo • Calculo de transformación con matrices.	**Colaboraciones: -**

Tabla 3.14: Clase que contiene las principales funcionalidades de ARtoolKit

Tarjeta CRC	
Clase: C_Tracking	
Súper Clase: -	
Sub Clase(s): -	
Responsabilidades: • Activar cámara de video • Cargar fichero de configuración de cámara • Calibración cámara • Cargar patrón • Detectar y reconocer patrón dentro de la imagen. • Obtener posiciones en X,Y,Z • Obtener rotación • Obtener Quat • Cerrar video	**Colaboraciones:** clase AR clase Video

Tabla 3.15: Descripción de la clase C_tracking.

Tarjeta CRC	
Clase: GLSV	
Súper Clase: -	
Sub Clase(s): -	
Responsabilidades: • Contiene métodos y estructuras para pintar y posicionar objetos en la escena.	**Colaboraciones:-**

Tabla 3.16: Descripción de la biblioteca gráfica.

Tarjeta CRC	
Clase: Tracking	
Súper Clase:	
Sub Clase(s): -	
Responsabilidades: • Importar todas las funciones de la clase C_tracking. • Brindar una interfaz que permita utilizar las funciones de C_tracking en C Sharp.	**Colaboraciones:** Clase C_Tracking

Tabla 3.17: Descripción de la clase Tracking.

3.2.2. Diagrama de componentes

Un diagrama de componentes muestra las dependencias lógicas entre componentes software, sean estos componentes: fuentes, binarios o ejecutables, ilustran las piezas del software, controladores embebidos, etc. Los diagramas de componentes se relacionan con los diagramas de clases, ya que un componente normalmente se corresponde con una o más clases, interfaces o colaboraciones pero un diagrama de componentes tiene un nivel más alto de abstracción que un diagrama de clases, usualmente un componente se implementa por una o más clases (u objetos) en tiempo de ejecución.

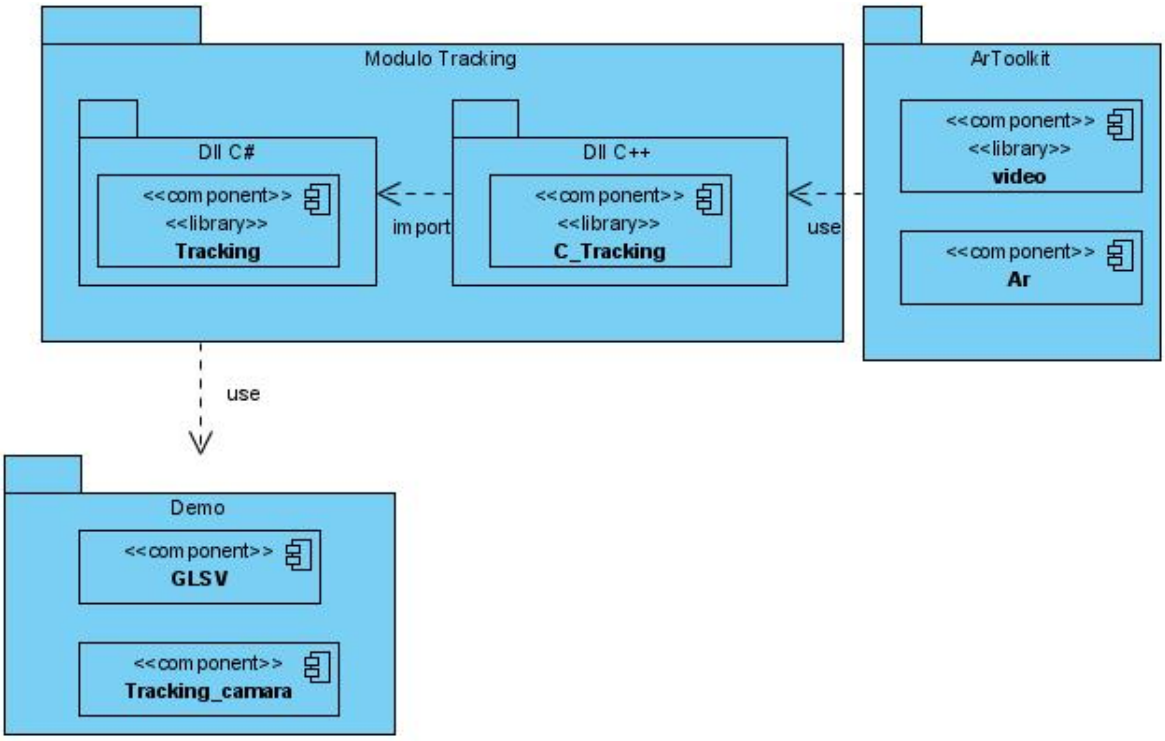

Figura 3.2: Diagrama de componentes

3.2.3. Patrones de diseño

Los patrones de diseño son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Son soluciones basadas en la experiencia. Existen problemas durante el desarrollo de software que se repiten o que son análogos, que responden a cierto patrón. Por lo que es deseable tener una colección de dichos patrones con soluciones óptimas para cada caso a modo de buenas prácticas[Machado, 2010].

- **Encapsulamiento (ocultación de datos):** Se utiliza para definir el nivel de visibilidad que poseen los miembros de las clases y las clases utilizando **private** y **public**.

3.2.4. Estándares de codificación

Las convenciones o estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código.

Los estándares de codificación elevan la mantenibilidad del código, se utilizan como punto de referencia para los programadores, mantienen un estilo de programación y ayudan a mejorar el proceso de codificación, haciéndolo entre otras cosas más eficientes y entendible.

Nomenclatura de clases y métodos

Los nombres de clases, sus constructores y métodos deben tener su primera letra en mayúscula. Los nombres con varias palabras deben comenzar con minúsculas a continuación de la anterior y se utilizará el guión bajo para separarlas. Los nombres deben ser claros y en correspondencia con la función que desempeñan para mejorar la comprensión del código, ejemplo una clase que realiza operaciones aritméticas debe declararse:

```
class Tracking

void Activar_camara();

void Cerrar_video;
```

Nomenclatura de variables

Las variables deben contener solo minúsculas, y ser cortas. Nombres que no sean palabras solo deben ser usadas como iteradores en for(), while(). Ejemplo:

```
ARMarkerInfo* marcador;

while(int i = 0);
```

Identación

Se utiliza el estilo Allman, también conocido como *ANSI style*. Este estilo ubica las

llaves asociadas a una sentencia de control en la siguiente línea, las sentencias son identadas al siguiente nivel de las llaves con 1 o 2 TAB de espacio. Ejemplo

```
if(Vacio())
{
Operacion1();
Operacion2();
}
```

3.3. Desarrollo de las iteraciones

En la exploración se detallaron los HU correspondientes con cada una de las iteraciones a desarrollar, teniendo en cuenta las necesidades requeridas por el cliente. Durante el transcurso de las iteraciones se lleva a cabo una revisión del plan de iteraciones y se modifica en caso de que sea necesario. Como parte de este plan, se descomponen las HU en tareas de programación o de ingeniería, asignando a un equipo de desarrollo (o una persona), responsable de su implementación, aplicando la práctica de la programación en parejas. Estas tareas no tienen que necesariamente ser entendidas por el cliente, pueden ser escritas en lenguaje técnico y son para el uso estricto de los programadores.

Teniendo en cuenta la planificación realizada con anterioridad, se llevó a cabo el desarrollo del sistema en tres iteraciones, obteniéndose como finalidad un producto con todas las restricciones y características deseadas por el cliente. A continuación se detallan cada una de las iteraciones.

3.3.1. Iteración 1

En esta iteración se da cumplimiento a la implementación de las HU que corresponden con los números 1, 2 y 3 consideradas de mayor importancia para el desarrollo de la aplicación, con el fin de obtener una versión del producto con algunas funcionalidades críticas como: Conectar el dispositivo, habilitar la captura de los componentes del mismo.

Historias de Usuario	**Tiempo de Implementación(semanas)**	
	Estimación	**Real**
Activar cámara	2	1
Cargar patrón	1	1
Detección del patrón.	3	4

Tarea de Ingeniería.	
No. De la Tarea: 1	**No. De la HU:** 1
Nombre de la tarea: Activar cámara	
Tipo de Tarea: Desarrollo.	**Puntos estimados:** 2
Fecha inicio: 8/02/2011	**Fecha fin:** 16/02/2011
Programador responsable: Liudmila Cecilia Rodríguez Ricardo	
Descripción: Esta funcionalidad es de vital importancia para el funcionamiento del sistema, debido a que si no se encuentra ningún dispositivo de captura, en este caso sería una cámara web, el sistema no podría funcionar.	

Tarea de Ingeniería.	
No. De la Tarea: 2	**No. De la HU:** 2
Nombre de la tarea: Cargar patrón	
Tipo de Tarea: Desarrollo.	**Puntos estimados:** 1
Fecha inicio: 17/02/2011	**Fecha fin:** 28/02/2011
Programador responsable: Liudmila Cecilia Rodríguez Ricardo	
Descripción: Si no se carga el patrón que se utilizará como marca, no se puede generar el flujo de información que se desea obtener.	

Tarea de Ingeniería.	
No. De la Tarea: 3	**No. De la HU:** 3
Nombre de la tarea: Detección del patrón.	
Tipo de Tarea: Desarrollo.	**Puntos estimados:** 3
Fecha inicio: 2/03/2011	**Fecha fin:** 19/03/2011
Programador responsable: Liudmila Cecilia Rodríguez Ricardo	
Descripción: Esta es la funcionalidad principal del sistema. La misma se encarga de analizar el flujo de información proveniente desde el dispositivo de captura y buscar en cada imagen obtenida el patrón que se haya cargado como marca.	

Tabla 3.18: Estimación de tiempo de la iteración1

3.3.2. Iteración 2

En esta iteración se realizó la implementación de las historias de usuario 4, 5 y 6. Estas historias de usuario brindan la posición y orientación del patrón detectado en la imagen capturada, respecto a la cámara, además se ofrece un vector de 4 posiciones que brinda la posición y orientación para aplicaciones gráficas que utilicen este tipo de estructura.

Historias de Usuario	**Tiempo de Implementación(semanas)**	
	Estimación	**Real**
Obtener posición (x,y,z)	1	1
Obtener orientación	1	1
Obtener Quat	1	1

Tarea de Ingeniería.	
No. De la Tarea: 4	**No. De la HU:** 4
Nombre de la tarea: Obtener (posición x,y,z)	
Tipo de Tarea: Desarrollo.	**Puntos estimados:** 1
Fecha inicio: 20/03/2011	**Fecha fin:** 27/03/2011
Programador responsable: Liudmila Cecilia Rodríguez Ricardo	
Descripción: Esta funcionalidad retorna de forma independiente un valor double con posición respecto a los 3 ejes de coordenadas del patrón.	

Tarea de Ingeniería.	
No. De la Tarea: 5	**No. De la HU:** 5
Nombre de la tarea: Obtener orientación	
Tipo de Tarea: Desarrollo.	**Puntos estimados:** 1
Fecha inicio: 28/03/2011	**Fecha fin:** 18/04/2011
Programador responsable: Liudmila Cecilia Rodríguez Ricardo	
Descripción: Esta funcionalidad retorna una matriz de 3x4 con la orientación del patrón	

Tarea de Ingeniería.	
No. De la Tarea: 6	**No. De la HU:** 6
Nombre de la tarea: Obtener Quat	
Tipo de Tarea: Desarrollo.	**Puntos estimados:** 1
Fecha inicio: 28/03/2011	**Fecha fin:** 18/04/2011
Programador responsable: Liudmila Cecilia Rodríguez Ricardo	
Descripción: Esta funcionalidad retorna en un vector[4] la posición y orientación del patrón	

Tabla 3.19: Estimación de tiempo de la iteración2

3.3.3. Iteración 3

En esta última iteración se implementó las HU 7 y 8, actualiza la posición y orientación del observe en esta iteración se integran todas las funcionalidades realizadas en las iteraciones anteriores, se desactiva la cámara de video.

Historias de Usuario	**Tiempo de Implementación(semanas)**	
	Estimación	**Real**
Cerrar video	1	1
Actualizar posición y orientación de la escena	1	1

Tarea de Ingeniería.	
No. De la Tarea: 7	**No. De la HU:**7
Nombre de la tarea: Actualizar posición y orientación de la escena.	
Tipo de Tarea: Desarrollo.	**Puntos estimados:** 2
Fecha inicio: 10/05/2011	**Fecha fin:** 17/05/2011
Programador responsable: Liudmila Cecilia Rodríguez Ricardo	
Descripción: Esta funcionalidad permite actualizar la posición y orientación de la clase Observer de la GLSVe, cuando el observador realiza un movimiento de la cabeza frente a la cámara.	

Tarea de Ingeniería.	
No. De la Tarea: 8	**No. De la HU:**8
Nombre de la tarea: Cerrar video	
Tipo de Tarea: Desarrollo.	**Puntos estimados:** 2
Fecha inicio: 10/04/2011	**Fecha fin:** 17/05/2011
Programador responsable: Liudmila Cecilia Rodríguez Ricardo	
Descripción: Desactiva la cámara para detener el proceso de seguimiento, por lo generar se utiliza cuando el usuario desea abandonar la escena.	

Tabla 3.20: Estimación de tiempo de la iteración3

Capítulo 4

Análisis de resultados

Para probar la precisión y exactitud del seguimiento se utilizó la realidad aumentada. La prueba consistió dibujar una figura, en la posición y orientación obtenidas con el seguimiento . A continuación se muestran figuras.

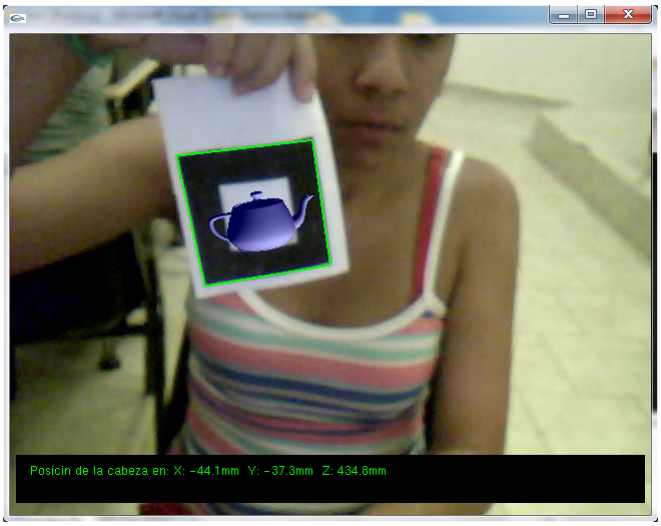

Figura 4.1: Prueba con realidad aumentada

El demo construido utilizando GLSVe demuestra el correcto funcionamiento de la actualización y obtención de los valores de posición y orientación del punto de vista del observador que interactúa con la escena estereoscópica.

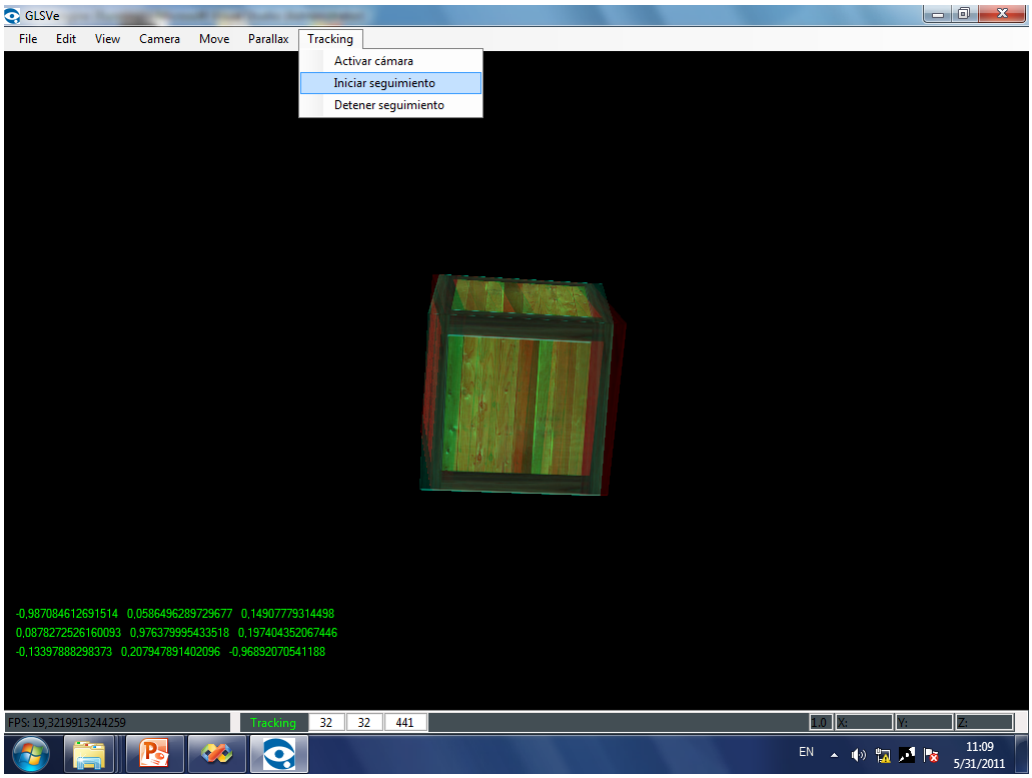

Figura 4.2: Vista de la aplicación

4.1. Pruebas de aceptación

Las pruebas de aceptación son creadas en base a las HU, en cada ciclo de iteración del desarrollo. El cliente debe especificar uno o diversos escenarios para comprobar que una historia de usuario ha sido correctamente implementada.

Las pruebas de aceptación son consideradas como ”pruebas de caja negra”. Los clientes son responsables de verificar que los resultados de estas pruebas sean correctos. Así mismo, en caso de que fallen varias pruebas, deben indicar el orden de prioridad de resolución. Una historia de usuario no se puede considerar terminada hasta tanto pase correctamente todas las pruebas de aceptación.

Caso de Prueba de Aceptación.	
Código: P1_1	**Historia de Usuario:** 1
Nombre: Comprobar la detección del dispositivo de captura conectado.	
Descripción: Evaluar que el sistema cargue el dispositivo correctamente.	
Condiciones de ejecución: El usuario debe comprobar que el dispositivo de captura está conectado e instalado correctamente en la computadora.	
Entrada/Pasos de ejecución: Ejecutar la funcionalidad en la aplicación.	
Resultado esperado: El dispositivo de captura es detectado correctamente.	
Evaluación de la prueba:-	

Caso de Prueba de Aceptación.	
Código: P1_3	**Historia de Usuario:** 3
Nombre: Detección y seguimiento del patrón.	
Descripción: Verifica que el sistema carga el patrón correctamente y lo reconoce en el flujo de video	
Condiciones de ejecución: El cliente debe asegurarse que el dispositivo de captura permanezca conectado al sistema y mantener el patrón visible a la cámara.	
Entrada/Pasos de ejecución: Asegurarse de que la aplicación se sigue ejecutando.	
Resultado esperado: El sistema reconoce el patrón y realiza todos los cálculos pertinentes.	
Evaluación de la prueba:-	

Caso de Prueba de Aceptación.	
Código: P1_4	**Historia de Usuario:** 4
Nombre: Comprobar los valores en la posición (x,y,z)	
Descripción: Verificar que los datos de la posición que se obtiene, es la correcta.	
Condiciones de ejecución: El cliente debe asegurarse que el dispositivo de captura permanezca conectado al sistema y mantener el patrón visible a la cámara.	
Entrada/Pasos de ejecución: El usuario debe mover la cabeza frente la cámara	
Resultado esperado: El sistema obtiene los valores actualizados de la cabeza del usurario	
Evaluación de la prueba:-	

Caso de Prueba de Aceptación.	
Código: P1_5	**Historia de Usuario:** 5
Nombre: Comprobar los valores de la matriz orientación.	
Descripción: Verificar que el sistema cambia los valores correctamente.	
Condiciones de ejecución: El cliente debe asegurarse que el dispositivo de captura permanezca enfocando el patrón que se desea reconocer.	
Entrada/Pasos de ejecución: El usuario debe girar la cabeza a los lados del cuerpo	
Resultado esperado: El sistema obtiene los valores actualizados de la rotación de la cabeza del usurario	
Evaluación de la prueba:-	

Tabla 4.1: Pruebas de Aceptación realizadas al sistema

Conclusiones

En el transcurso de la investigación se ha llegado a las siguientes conclusiones:

- Utilizando cámara web y seguimiento de marcas se ha desarrollo un módulo funcional que permite realizar el seguimiento de pose del observador para ser utilizado por la biblioteca GLSVe.
- Se ha obtenido un módulo genérico que puede ser utilizado en otro tipo de aplicaciones que necesiten conocer la posición y orientación del observador que interactúa con la escena virtual.

Recomendaciones

Se recomienda:

- Integrar el módulo desarrollado a aplicaciones de realidad virtual que no posean interacción de este tipo.
- Estudiar otras vías que permitan realizar seguimiento de pose del observador utilizando otros dispositivos.

Referencias bibliográficas

[Bary Gradsky, 2008] Bary Gradsky, A. K. (2008). *Learning OpenCV*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

[Beck, 2002] Beck, K. (2002). Una explicación de la programación extrema: Aceptar el cambio. Master's thesis, Addison Wesley.

[Camacho, 2009] Camacho, S. P. (2009). Sistema de tracking híbrido modular para realidad aumentada. Master's thesis, Universidad de Castilla-La Mancha.

[Carlos Pérez, 2003a] Carlos Pérez, O. R. (2003a). Dispositivos de visualizacion espacial. II(I).

[Carlos Pérez, 2003b] Carlos Pérez, O. R. (2003b). Geometric considerations for stereoscopic virtual enviroments. I(I).

[Garcia, 2007] Garcia, J. (2007). Integración de objetos sintéticos en imágenes y vídeo real. Master's thesis, Universidad Politécnica de Madrid.

[Gutiérrez, 2010] Gutiérrez, I. H. (2010). Wii-immersion: Aprovechando las capacidades del wiimote para la mejora de la sensación de inmersión. Master's thesis, Universidad de Pompeu Fabra.

[Hirokazu Kato, 2009] Hirokazu Kato, Mark Billinghurst, I. P. (2009). Artoolkit version 2.33. *http://www.hitl.washington.edu/resarch/shared_space/download/*.

[Larman, 1999] Larman, C. (1999). *UML y Patrones. Introducción al análisis y diseño orientado a objetos*. Editorial Prentice Hall Hispanoamericana. S.A. México.

[López, 2007] López, P. (2007). 3d animation compositing. *http:www.pablo-lopez.com*.

[Machado, 2010] Machado, R. (2010). Hablando de patrones de diseño. *http:///reisbel.blogspot.com/*.

[Noris, 2005] Noris, B. (2005). Real-time stereo 3d hand tracking:a six degrees of freedom pointing device. Master's thesis, Dpto de Informática de la universidad de Geneva.

[Oduardo, 2009] Oduardo, I. G. (2009). Arquitectura del sistema de clonación y distribución de imágenes de sistemas operativos. Master's thesis, Universidad de la Ciencias Informáticas.

[Paradim, 2007] Paradim, V. (2007). Visual paradin. *http://www.visual-paradigm.com/product/vpuml/*.

[Rodríguez, 2003] Rodríguez, L. S. (2003). Obtención de mapas de profundidad densos mediante visión activa por movimiento controlado de una cámara.aplicación a tareas de reconocimiento. Master's thesis, Departamento de Ingeniería fotónica, Madrid.

[Tecnopeixe, 2008] Tecnopeixe, P. (2008). Realidad virtual. I(I).

[Vincent, 2006] Vincent, J. (2006). Dispositivos de visualizacion espacial. I(I).

[wittysparks, 2004] wittysparks (2004). Artoolkit. *http://videos.wittysparks.com/id/4290494699/*.

Acrónimos

API Interfaz de Programación de Aplicaciones (traducido de las siglas en inglés API: Application Program Interface).

SRV Sistemas Realidad Virtual

SVE Sistemas de Visualización Estereoscópica

VE Visualización Estereoscópica

RV Realidad Virtual

UML Lenguaje Unificado de Modelado

HU Historia de Usuario

XP Extreme Programming

CRC Contenido, Responsabilidad y Colaboración

TDD Desarrollo Dirigido por Pruebas

GLSVe Graphics Library Stereo Vision Engine

UCI Universidad de las Ciencias Informáticas