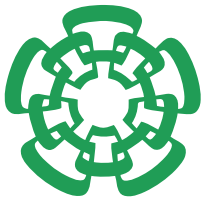


Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Interfaz de lenguaje natural usando Kinect

Tesis que presenta

Sergio Herman Peralta Benhumea

para obtener el Grado de

Maestro en Ciencias en Computación

Director de la Tesis:

Dr. Luis Gerardo de la Fraga

México, Distrito Federal **Septiembre, 2012**

Resumen

Las interfaces de usuario fueron creadas como respuesta a la necesidad de facilitar la interacción con las computadoras. La evolución de las interfaces de usuario nos ha llevado desde las tarjetas perforadas y línea de comandos, hasta las interfaces gráficas, cuyo uso ha sido predominante desde hace más de dos décadas, lo que ha permitido que la computadora y los dispositivos móviles se posicionen como herramientas de la vida cotidiana. La evolución de la computación, además de concentrarse en el desarrollo de hardware y software más poderoso y con mayores prestaciones, ha llevado también a la investigación para el desarrollo de interfaces de usuario más poderosas, fáciles de usar y con la característica de que éstas sean las que se adapten al usuario y no viceversa. Esta búsqueda ha llevado a la investigación de la Interacción Natural (IN) y de las Interfaces Naturales de Usuario (INU). El objetivo de la INU es permitir al usuario interactuar con la computadora como si ésta se tratáse de otro humano, es decir, busca eliminar la necesidad de estar consciente del uso de cualquier dispositivo, al enfocarse en mecanismos basados en habilidades del humano, tales como gesticulaciones o movimientos específicos y voz, que le son familiares al usuario, para ser utilizados como datos de entrada a la interfaz, volviendo más natural la interacción con la computadora.

Gracias al sensor Kinect, la consola Xbox 360 de Microsoft permite la interacción con videojuegos sin más que con movimientos del cuerpo y voz, eliminando el uso del control de la consola, como en el caso del Wii de Nintendo. El Kinect entrega imágenes de profundidad que permiten interpretar el ambiente y estimar el movimiento del usuario con la ayuda de bibliotecas de software.

El propósito de esta tesis es diseñar y experimentar con la construcción de una interfaz natural basada en la interpretación de secuencias de movimientos del usuario a *comandos de interacción* utilizando el sensor Kinect, Qt, diversas bibliotecas de desarrollo y el ambiente de GNU/Linux. Para verificar el funcionamiento de la interfaz, se diseñaron e implementaron tres aplicaciones construidas sobre ésta: (1) un ratón virtual, (2) una navegación en menús y, (3) una interfaz para la manipulación de objetos virtuales.

Palabras claves: Visión por Computadora, Interfaz Natural de Usuario, Interacción Humano-Computadora, Reconocimiento de Comandos, Cámara de Profundidad.

Abstract

User interfaces were created as a response to the need of facilitating computer-human interaction. The evolution of user interfaces has taken us from punch cards and the command line to graphical interfaces, whose usage has been predominant for more than two decades and also has positioned the computers and mobile devices as daily life tools. In addition to focusing on the development of high performance hardware and software, this computing evolution has also led to interface research focused in it's ease of use, and the charateristic that the interface is the one that adapts to the user and not vice versa. This search has led to the research of Natural Interaction (NI), and Natural User Interfaces (NUI). The aim of the NUI is to allow the user to interact with a computer as if it were another human. NUI seeks to eliminate the need to be aware of the use of any device by focusing on mechanisms based on human skills such as specific movements and gestures or voice, which are familiar to the user, for use as input to the interface, becoming a *more natural* interaction with the computer.

With the Kinect sensor, the Microsof Xbox 360 console allows us to interact with video games using nothing but body movements and voice, eliminating the need to use console's controller, as in the case of Nintendo's Wii. Kinect generates depth images that allow us to interpret the environment and estimate the movement of the user using some software libraries.

The purpose of this thesis is to design and experiment by developing a natural interface system based on the translation of user's movements sequences to *interaction commands* using the Kinect sensor, Qt, some free software libraries and the GNU/Linux environment. To verify the correct functionality of the natural interface, three applications were developed on top of the system: (1) a virtual mouse, (2) a menu navigation and (3) a virtual object interface handler.

Keywords: Computer Vision, Natural User Interface, Human-Computer Interaction, Command Recognition, Depth Camera.

Agradecimientos

Al Dr. Luis Gerardo de la Fraga:
Por todos su sabios consejos, motivación y apoyo durante la realización de esta tesis y en sus cursos impartidos, por brindarme la oportunidad de llevar a cabo el presente trabajo de tesis, por enseñarme el valor de analizar los pequenõs detalles de las cosas y por ser el profesor del que más he aprendido durante mi estancia en el CINVESTAV.

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 durante mi estancia en el CINVESTAV, sin el cual me hubiese sido más difícil culminar este trabajo de tesis y al proyecto 168357 de Ciencia Básica 2011 del CONACyT.

Al Dr. Francisco Rodríguez Henríquez y al Dr. José Guadalupe Rodríguez García
Por sus atenciones, consejos y su valioso tiempo en la revisión de esta tesis.

A los profesores del Departamento de Computación
Por brindarme algunos de sus conocimientos y permitirme aprender temas selectos y por brindarme lecciones que me han permitido crecer como computólogo y como persona.

A mi madre Lourdes y a mi tía Blanca
Por su apoyo incondicional tanto en los momentos difiíciles como en los alegres, por creer en mí, por inculcarme valores para formarme como una buena persona y por lo sacrificios que han hecho para permitirme cumplir mis objetivos.

A Laura
Por creer en mí, por toda su comprensión, cariño y apoyo incondicional, por siempre estar cerca y ayudarme a levantarme en los momentos difíciles y por compartir tiempos de profunda alegría.

A mi familia
Por su valioso apoyo y consejos que me han hecho crecer como persona.

A Sofía Reza
Por brindarme siempre su apoyo y motivación durante mi estancia en el CINVESTAV.

A mis amigos y compañeros del CINVESTAV

Índice general

Índice de figuras

Índice de tablas

Capítulo 1
Introducción

1.1. Antecedentes y motivación

La computadora se ha vuelto un instrumento básico para muchas actividades de la vida cotidiana, entre las que destacan la educación, el trabajo y el entretenimiento. Gran parte de este hecho se debe a la facilidad que la *interfaz* de la computadora brinda al usuario para realizar tareas y procesar información.

La evolución de la computación se ha concentrado principalmente en el desarrollo de computadoras cada vez más poderosas, de tamaño cada vez más reducido y con más funcionalidades, mientras se ha continuado la búsqueda por la mejora de la interacción a través de interfaces de usuario más poderosas que resulten más naturales y amigables, que se adapten al usuario y no viceversa. La Interacción Humano-Computadora (IHC) [1] es un área de investigación que se enfoca en el estudio de la interacción entre el humano y la computadora. Del lado del humano se concentra en la mejora de los mecanismos de interacción y aprendizaje y de lado de las computadoras en la mejora de la visualización y procesamiento de la información.

Actualmente la interfaz más utilizada es la Interfaz Gráfica de Usuario (IGU), que sigue el paradigma VIMA[1] (*Ventana*, *Ícono*, *Menú* y dispositivo *Apuntador*), al abstraer la interacción con íconos que representan elementos que el humano está acostumbrado a utilizar cotidianamente, y la metáfora del *escritorio* como contenedor y base de la interacción. Esto ha posicionado a la IGU, junto al ratón y teclado, como la interfaz predominante desde los años 90.

La búsqueda por la mejora de la interacción ha llevado a extender las capacidades de la IGU. Los dispositivos móviles y los videojuegos son los ejemplos más comunes.

Dado que los dispositivos móviles, al contrario de las computadoras convencionales, cuentan con recursos limitados y que el tamaño de la pantalla es reducido, no es posible

[1] *Entrada de Wikipedia de VIMA:* `http://en.wikipedia.org/wiki/WIMP_%28computing%29`

visualizar más de una aplicación a la vez, por lo que se optó por extender la interfaz de usuario con el desarrollo de las interfaces multimodales [2] que combinan diversos mecanismos de entrada (que además de sensado de movimientos mediante acelerómetros, incluyen pantallas multi-táctiles, reconocimiento de gestos usando las cámaras incorporadas y reconocimiento de comandos de voz), con mecanismos de salida (como sonido, vibración y visualización en pantalla). Dicha combinación (Figura 1.1) brinda una experiencia multimodal [3], llevando a un nivel de interacción completamente distinto al que se alcanza en una computadora. Mediante la interfaz multimodal, el usuario puede interactuar con aplicaciones que no se visualizan en la pantalla, pero se están ejecutando. Por ejemplo, es posible cambiar de canción en un reproductor de audio al agitar el dispositivo, mientras que en la pantalla se lee el correo electrónico. Todas estas mejoras en las interfaces han abierto una puerta para desarrollar aplicaciones completamente distintas a las que existían hace pocos años.

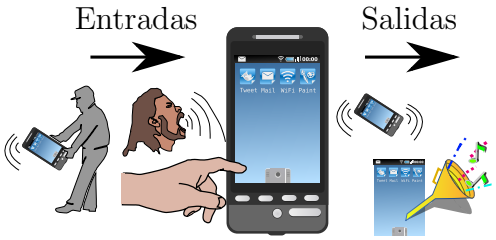


Figura 1.1: Ejemplo de interfaz multimodal

Uno de los factores determinantes en el éxito de un videojuego es la experiencia de juego a través de una interfaz intuitiva y fácil de aprender para el usuario. Los controles fueron evolucionando a lo largo del tiempo (Figura 1.2) para ser más ergonómicos y brindar retroalimentación al usuario de lo que ocurre en el juego (mediante vibración o botones sensibles a la presión). Recientemente, el desarrollo del control de la consola *Wii*[2] de Nintendo y posteriormente, el sistema *sixaxis*[3] de la consola Play Station 3 de Sony, brindaron un nivel de interacción *más natural* con una máquina: poder utilizar los movimientos que se realizan con el control para interactuar con el videojuego, gracias a la incorporación de sistemas de acelerómetros en los controles. El sensor Kinect para la consola Xbox 360 de Microsoft llevó la interacción entre el usuario y el videojuego a un nivel superior en el que no se requieren mandos, al permitir al usuario interactuar con el videojuego mediante movimientos del cuerpo.

La idea de usar el cuerpo como medio de interacción con la computadora y el desarrollo tecnológico dieron pauta a la investigación de la Interacción Natural (IN) [4, 5] y la Interfaz Natural de Usuario (INU) [4, 6, 7], como sucesora de la IGU (ver Figura 1.3). La INU se basa en el uso de habilidades del cuerpo, tales como movimientos y gesticulaciones,

[2] *Entrada de Wikipedia del control de* Wii*:* `http://es.wikipedia.org/wiki/Wii_Remote`

[3] El sistema *sixaxis* significa "seis ejes" haciendo referencia a los seis (six) ejes (axis) de detección de movimiento (3 para movimientos posicionales en el espacio mediante acelerómetros, y 3 para la detección de rotación mediante giroscopios). Mś información en `http://es.wikipedia.org/wiki/DualShock_3_y_Sixaxis`

[4] *Información obtenida de:* `http://en.wikipedia.org/wiki/Gamepad`

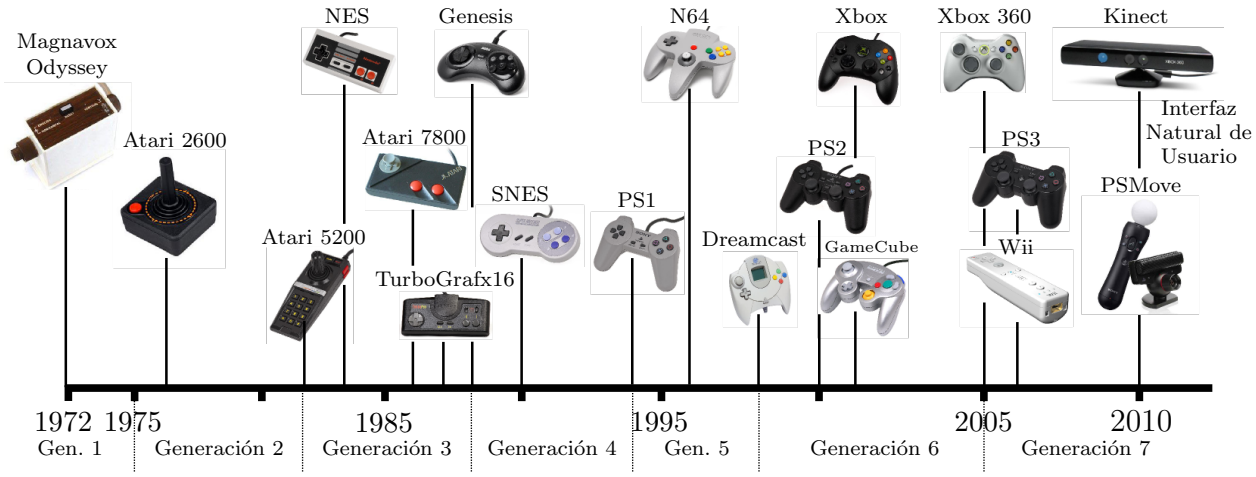


Figura 1.2: Evolución de los controles de videojuegos[4]

para ser utilizadas como medio de interacción con la computadora, eliminando el uso de dispositivos de entrada. Es decir, el objetivo de las INU es lograr que el usuario interactúe con la computadora como si ésta fuese otra persona, sin que el usuario esté consciente de utilizar un dispositivo.

El desarrollo del sensor Kinect ha constituido a la consola Xbox 360 de Microsoft como la primer INU comercial, que a diferencia de *Wii* y *PSMove*, elimina completamente la necesidad de usar el control de la consola al reemplazarlo por el movimiento del cuerpo y comandos de voz. La interacción con el videojuego se logra al visualizar los movimientos del usuario como movimientos de un personaje en la pantalla y la realización de acciones cuando el usuario efectúa un movimiento específico o pronuncia un comando de voz, los cuales son reconocidos por el sistema e interpretados por el videojuego.

Figura 1.3: Evolución de las interfaces de usuario

Figura 1.4: Sensor Kinect

Kinect no sólo renovó el concepto de interacción con un videojuego, sino que ha permitido que el desarrollo de las INU sea fortalecido, gracias al esfuerzo de *OpenNI* [8], *libfreenect* [9] y posteriormente Microsoft [10], para extender el uso del Kinect, de la consola Xbox 360 hacia las computadoras convencionales e incluso arquitecturas ARM.

El uso de Kinect en computadoras convencionales ha tenido una historia corta, como se relata en [9]. El jueves 4 de noviembre del 2010 se empezó a vender Kinect en E.U., el

mismo día Adafruit[5] lanzó una convocatoria[6] para entregar un premio de 1000 dólares a la primer persona que lograra escribir un controlador para Windows o cualquier otro sistema. El mismo día, Microsoft declaró que no permitiría el hackeo de su dispositivo, Adafruit respondió subiendo el premio a 2000 dólares. El sábado un hacker logró controlar el motor y Adafruit elevó el premio a 3000. Para el lunes, el mismo hacker publicó un video, donde presentaba la habilidad de controlar el motor y extraer las imágenes de color y profundidad de Kinect y pedía 10,000 dólares por su código y mantener su desarrollo. El martes Adafruit publicó un registro del flujo de datos entre el Kinect y la consola Xbox, lo cual permitió que más hackers empezaran a analizar la conectividad de Kinect. Finalmente, el miércoles Kinect fue lanzado en Europa y para la noche, otro hacker logró conectarse y acceder a los datos de Kinect mediante una computadora convencional, constituyendo *libfreenect* y ganando el premio.

Mientras se desarrollaba *libfreenect*, un grupo de compañías, entre ellas *PrimeSense* [11] (que desarrolló y vendió a Microsoft el sistema de captura de profundidad de Kinect) y *ASUS* (los desarrolladores del *X-tion Pro*[7]) fundaron *OpenNI* (Open Natural Interaction) [8] como una organización sin fines de lucro, con el objetivo de *"certificar y promover la compatibilidad e interoperabilidad de dispositivos de Interacción Natural, aplicaciones y middleware*[8]*, y acelerar la introducción de aplicaciones de IN al mercado"* y una visión más enfocada al desarrollo de INUs. *OpenNI* provee una biblioteca multiplataforma de código abierto que pretenden ser un estándar para acceder a dispositivos de IN (actualmente soporta Kinect y X-tion Pro). Sin embargo, para acceder a las capacidades de rastreo de movimiento y detección de algunos gestos de movimiento, se requiere el middleware *NITE* [12] (escrito y distribuido en paquetes binarios por *OpenNI*).
Microsoft aceptó a *OpenNI* y el hackeo de libfreenect, replanteó su estrategia alentando a desarrolladores para que experimentaran con el Kinect. Posteriormente, el 16 de junio del 2011, publicó la primera versión de su kit de desarrollo de Kinect [10].

1.2. Estado del arte

Para la realización del presente trabajo de tesis se consultaron diversos artículos relacionados a las interfaces de usuario que se han desarrollado utilizando el sensor Kinect con la finalidad de extender el uso del ratón y teclado para llevar a una forma de interacción más natural con la computadora. Algunos de los artículos revisados, además de controlar una aplicación, se enfocan en el análisis de los datos obtenidos con Kinect, o su integración con con otros dispositivos referentes al área de investigación de cada artículo.

Se optó por clasificar los artículos revisados en tres categorías: (1) control de aplica-

[5] *Adafruit Industries se fundó en 2005 por una ingeniera del MIT con el objetivo de constituir el mejor lugar en línea para aprender sobre productos electrónicos*

[6] *Proyecto Open Kinect:* http://www.adafruit.com/blog/2010/11/04/the-open-kinect-project-the-ok-prize-get-1000-bounty-for-kinect-for-xbox-360-open-source-drivers/

[7] *ASUS X-Tion Pro* es un dispositivo similar a Kinect. `http://www.asus.com/Multimedia/Motion_Sensor/Xtion_PRO/`.

[8] Middleware es un conjunto de bibliotecas que permiten a una aplicación comunicarse y transferir información con otras aplicaciones, hardware, software, redes y sistemas operativos

ciones de escritorio, (2) aplicaciones de gráficos por computadora y, (3) análisis de datos. En la tabla 1.1 se presenta una comparativa sobre el uso de Kinect en cada artículo.

Control de aplicaciones de escritorio
Se consideró en esta categoría a los artículos que presentan trabajos enfocados en reemplazar los dispositivos de entrada a partir de gestos o movimientos para controlar aplicaciones de escritorio.

Tabla 1.1: Tabla comparativa de los trabajos revisados en el estado del arte

Artículo	**Biblioteca de acceso a Kinect**	**Datos usados de Kinect**	**Bibliotecas, aplicaciones y dispositivos extras**
FAAST: The Flexible Action And Articulated Skeleton Toolkit [13]	*OpenNI*, gestos *NITE*	esqueleto	OpenGL, VRPN, emulador de ratón y teclado
Kinemote [14]	*OpenNI*	esqueleto, imágenes de profundidad	OpenGL, emulador de ratón
Web GIS in practice X: a Microsoft Kinect natural user interface for Google Earth navigation [15]	*OpenNI*	esqueleto, imágenes de profundidad	OpenGL, Google Earth[9], Google Street View[10]
Interaction in Augmented Reality Environments Using Kinect [16]	*OpenNI*, gestos *NITE*	esqueleto, imágenes RGB	OpenGL, OGRE[11]
Natural Exploration of 3D Models [17]	*OpenNI*	esqueleto, imágenes de profundidad	OpenGL, entorno multi-táctil
Controller-free exploration of medical image data: experiencing the Kinect [18]	*OpenNI*	esqueleto	OpenGL, OpenCV[12]
Gaming for Upper Extremities Rehabilitation [19]	*OpenNI*	esqueleto	OpenGL, guante SmartGlove, videojuego
Sensor Fusion: Towards a Fully Expressive 3D Music Control Interface [20]	*OpenNI*	esqueleto	OSC, vibráfono[13]

- Nota: Como se explicó anteriormente, para rastrear el esqueleto, *OpenNI* requiere *NITE*, sin embargo en la tabla se especifica *NITE* cuando en el artículo se utilizó la detección de gestos provistos por *NITE*.

[9] *Google Earth.* `http://www.google.com/earth/index.html`
[10] *Google Street View.* `http://maps.google.com/help/maps/streetview/`
[11] *Open Source 3D Graphics Engine. Biblioteca de renderizado 3D*

- **FAAST: The Flexible Action and Articulated Skeleton Toolkit** [13], presenta un middleware construido sobre *OpenNI/NITE* para facilitar la integración de los movimientos del usuario con aplicaciones de realidad virtual en Windows. Incorpora un emulador de teclado y ratón y un servidor de propagación *VRPN* (Red Periférica de Realidad Virtual) del esqueleto sobre la red, lo cual permite el desarrollo de aplicaciones de control remoto. FAAST trata los datos del esqueleto en dos categorías: *esqueletos articulados* y *acciones*. La primera facilita el acceso a los datos del rastreo del esqueleto de *NITE* y los transmite por la red. Las *acciones* agregan más gestos a los reconocidos por *NITE* y además permite la definición de gestos propios, al indicar umbrales angulares y de distancias entre las articulaciones del esqueleto. *E.g.* una acción "click" puede definirse como un ángulo mayor a 100° entre el brazo y antebrazo derecho y una distancia mayor a $30cm$ entre la mano derecha y el hombro derecho.

- **KinEmote** [14], es una aplicación para Windows que permite controlar el ratón con la mano. Está basada en *OpenNI/NITE* y el reconocimiento de gestos con la mano: juntar el dedos índice y pulgar para hacer click izquierdo, cerrar la mano para arrastrar el click y la palma de la mano para deslizarse por las barras de desplazamiento de las aplicaciones.

- **Web GIS in practice X: a Microsoft Kinect natural user interface for Google Earth navigation** [15], presenta *Kinoogle*, una interfaz natural para Google Earth basada en Kinect y *OpenNI/NITE* para interactuar mediante gestos de manos y movimientos del cuerpo. El sistema cuenta con dos formas de interacción, la primera es la manipulación del globo terráqueo con gestos de la mano y movimientos de los brazos para hacer *zoom*, trasladar, rotar, inclinar y reiniciar el globo. La segunda forma de interacción se activa al alcanzar un nivel de *zoom* predeterminado. La interfaz cambia para presentar Google Street View y los gestos se reemplazan por *caminar* (el usuario mueve los brazos secuencialmete adelante y atrás, mientras está de pie), *voltear* (cambiar el ángulo de la cámara al inclinar un hombro hacia adelante o hacia atrás) y *salir* (al extender los brazos hacia adelante).

Control de aplicaciones de gráficos por computadora

El área de investigación de los gráficos por computadora trata de la construcción de escenas virtuales (bi o tridimensionales) a partir de modelos matemáticos que describen los elementos presentes en la escena. La mayoría de los artículos revisados en el estado del arte hacen uso de gráficos por computadora para la visualización de los datos de Kinect, sin embargo, los que se encuentran enfocados en mejorar la interacción en aplicaciones de ésta área son los que se describen a continuación.

- **Interaction in Augmented Reality Environments Using Kinect** [16], presenta un sistema de realidad aumentada para manipular objetos tridimensionales. El sistema se basa en la detección de gestos de movimiento de la mano (mediante *NITE*) para elegir en un menú de entre rotar, trasladar, escalar o cambiar el objeto. Posteriormente se realiza la transformación al objeto a partir del movimiento de las

[12] *Open Source Computer Vision, Biblioteca de funciones de visión por computadora*
[13] *Vibráfono* es un instrumento musical de percusión, similar a un xilófono.

manos del usuario. La escena del visualizador se compone por las imágenes de color tomadas con la cámara RGB de Kinect, el objeto virtual y el menú de opciones.

- **Natural Exploration of 3D Models** [17], presenta una comparativa de un visualizador de objetos tridimensionales en un entorno multi-táctil y un entorno 3D mediante el seguimiento del movimiento y estado de las manos (abierto o cerrado, utilizando la técnica de *cubierta convexa* en la imagen de profundidad de Kinect). El visualizador permite cuatro transformaciones geométricas sobre los objetos tridimensionales (rotación, rotación sobre el eje Z, escalado o *zoom* y paneo o traslación), que son detectadas mediante 4 gestos, que en la versión multi-táctil se detectan con los dedos tocando la superficie y en la versión 3D, al apuntar las manos hacia el Kinect y abrir o cerrar las manos (simulando la acción de tocar la superficie táctil).

- **Controller-free exploration of medical image data: experiencing the Kinect** [18], presenta un sistema de exploración de imágenes médicas áltamente interactivo capaz de ser utilizado en los quirófanos al no requerir un contacto físico, permitiendo al médico visualizar las imágenes del archivo del paciente. El sistema está basado en el rastreo del esqueleto y un mecanismo de transición de estados usando umbrales de distancias relativas al esqueleto (lo que denomiman como "área de activación"). Dependiendo de la distancia entre las manos y el área de activación, se cambia de entre un estado de espera, estabilización (inicialización de detección de gestos) y uno de detección de gestos. Las funcionalidades del sistema (que se acceden durante el estado de detección de gestos) son escalamiento, traslación, rotación, transición, cursor, extracción y borrado de áreas de interés de la imagen.

Análisis de datos y otros dispositivos
Esta categoría considera los artículos que presentan sistemas que utilizan los datos proporcionados por Kinect y otros dispositivos relacionados a su área de investigación.

- **Gaming for Upper Extremities Rehabilitation** [19], presenta un sistema de captura de movimiento de manos utilizando Kinect y un guante (SmartGlove) con sensor de movimiento y presión para extraer parámetros detallados de las extremidades superiores como entradas para un juego. Para probar el sistema se desarrolló un juego en el que el paciente debe atrapar joyas que caen por la pantalla. Los parámetros de movimiento durante el juego pueden ser almacenados para realizar un análisis médico posteriormente.

- **Sensor Fusion: Towards a Fully Expressive 3D Music Control Interface** [20], presenta un sistema para manipular el sonido de un instrumento musical. El sistema toma la posición 3D de las manos mediante Kinect y *OpenNI* y las envía como un de flujo de mensajes bajo el protocolo Open Sound Control (que permite comunicarse con aplicaciones de edición y sintetización de sonido a través de una red). El sistema se probó extendiendo las capacidades de un vibráfono mediante el movimiento vertical y horizontal de las manos. El movimiento horizontal de la mano derecha incrementa o disminuye el tono armónico y el movimiento vertical permite incrementar y decrementar las frecuencias altas. Adicionalmente, el movimiento horizontal de la mano izquierda controla la proporción del filtrado a la señal de sonido y el movimiento vertical controla la tasa de bits de codificación.

1.3. Descripción del problema

Actualmente estamos inmersos en un mundo en el que dependemos de las computadoras para resolver tareas en el trabajo, en la educación, procesar información, entretenimiento, artes, e incluso el control de sistemas en automóviles y casas inteligentes, entre muchas otras. Es esta realidad la que implica la importancia de interfaz con la computadora y su relación con la efectividad de las aplicaciones, sin embargo, a pesar de que la capacidad de procesamiento y prestaciones de las computadoras han crecido a un ritmo acelerado, las interfaces siguen basadas en el uso de del ratón y el teclado desde los años 70's [21].

Como se menciona en [3], la mejor interfaz sería aquella en la que el usuario pueda interactuar con la computadora en la misma forma en la que se comunica con el mundo, siendo altamente efectiva, intuitiva e invisible para él, al no tener que estar consciente del uso de un dispositivo de entrada para realizar una tarea.

En la revisión del estado del arte se encontraron aplicaciones que permiten utilizar el Kinect para controlar aplicaciones de escritorio, manipular objetos virtuales, e incluso combinar la información de Kinect con otros dispositivos para aplicaciones de rehabilitación médica, entre otras. Pero en todas ellas, los desarrolladores partieron de la construcción de un sistema mínimo basado en las bibliotecas de acceso a Kinect, para desarrollar su aplicación. El tiempo que lleva comprender el funcionamiento de las estas bibliotecas puede ser un factor determinante en el desarrollo de aplicaciones.

FAAST [13] es la aplicación con mayor similitud al trabajo desarrollado en esta tesis, sin embargo, está enfocada a reemplazar la entrada de teclado y ratón por gestos basados en umbrales ángulares y de distancia entre las partes del cuerpo, lo cual pudiera resultar complejo al requerir estar consciente de movimientos tan específicos del cuerpo, además de encontrarse disponible sólo para Windows.

Para atacar las carencias encontradas en las aplicaciones de interacción natural revisadas, se propone el desarrollo de una interfaz natural basada en la interacción con la computadora a través de secuencias de movimientos, que funja como base para el desarrollo de aplicaciones de interacción natural en Qt[14] [22] para sistemas Linux. La interfaz funciona bajo un paradigma de *comandos* basados en secuencias de *acciones* (identificación de movimientos simples, que resultan intuitivos y fáciles de replicar al usuario), que actúan directamente sobre los elementos de la misma aplicación, o en el entorno de escritorio (emulando la entrada de ratón), lo que permite interactuar con otras aplicaciones de la computadora. La tesis también comprende el desarrollo de tres aplicaciones de prueba construidas sobre el sistema de interfaz natural.

Gracias a *OpenNI* y *NITE* es posible rastrear el movimiento del usuario mediante Kinect y utilizar dichos datos como más convenga. Para establecer la forma en que se definen las *acciones* para interactuar en la interfaz natural, se tomó en cuenta el estudio de Rauterberg

[14]Qt es una biblioteca multiplataforma basada ampliamente usada para desarrollar aplicaciones con una interfaz gráfica de usuario en C++

y Steiger que clasificaron la interacción entre humanos y objetos como: (1) *Movimientos discretos*, los cuales involucran un solo movimiento para alcanzar un objetivo estático. (2) *Movimientos repetitivos*, se refieren a la repetición de un mismo movimiento a uno o más objetivos, como al martillar un clavo. (3) *Movimientos secuenciales*, involucran movimientos discretos a varios objetivos regular o irregularmente espaciados, como al escribir en el teclado. (4) *Movimientos continuos* en los que se requiere el control muscular, como al conducir un auto, o señalar con la mano. (5) *Posicionamiento estático* al mantener una determinada posición del cuerpo por un tiempo determinado [6].

Uno de los desafíos más importantes enfrentados en el desarrollo de este trabajo de tesis, es que el sistema de interfaz natural debe procesar la información lo más rápido posible para lograr analizar el movimiento del usuario, incluso si éste es rápido. Si la velocidad de procesamiento fuese muy baja, no se podría distinguir el movimiento del usuario, lo cual no haría viable la construcción de la interfaz.

La Figura 1.5 presenta un diagrama general del funcionamiento del sistema. Durante la inicialización se realiza la conexión con Kinect y la configuración de los módulos del sistema. Posteriormente se inicia un ciclo de funcionamiento que se detiene hasta que se indique el cierre del sistema. El ciclo comienza por la extracción del cuadro de Kinect *i.e.* obtener el esqueleto del usuario y las imágenes de color y/o profundidad (si la aplicación lo requiere). A partir del esqueleto obtenido se realiza la detección de acciones al obtener las celdas de la malla en las que se encuentra cada mano (como se explica en el capítulo 3). Si por lo menos una de las manos está en alguna de las celdas se procede a la detección de comandos (buscando si las acciones detectadas permiten una transición que lleve al final de la secuencia de acciones de un comando), si un comando es detectado se procede a enviar un evento Qt notificando la ocurrencia del comando, la aplicación responde ejecutando el método asignado al comando. Opcionalmente se realiza la visualización de las imágenes extraídas de Kinect. Posteriormente se realiza el procesamiento y la visualización de los datos de la aplicación.

El funcionamiento del sistema de interfaz natural está basado en las siguientes consideraciones:

- La velocidad del sistema se encuentra limitada a la velocidad de obtención de datos de Kinect, que oscila en los 30 cuadros por segundo.
- El procesamiento de cada cuadro debe ser lo más rápido y eficiente posible.
- El análisis de los movimientos del usuario en cada cuadro debe ser también rápido y eficiente.
- La velocidad del sistema y de obtención de datos de Kinect, se encuentra también limitada a la capacidad de procesamiento de la computadora en donde se ejecute el sistema.

Si consideramos el caso ideal con la respuesta del sistema a 30 cuadros por segundo, el sistema se encuentra regido por un ciclo de procesamiento cada ≈ 33.3 milisegundos, esto significa que se cuenta con menos de este tiempo para obtener los datos desde Kinect,

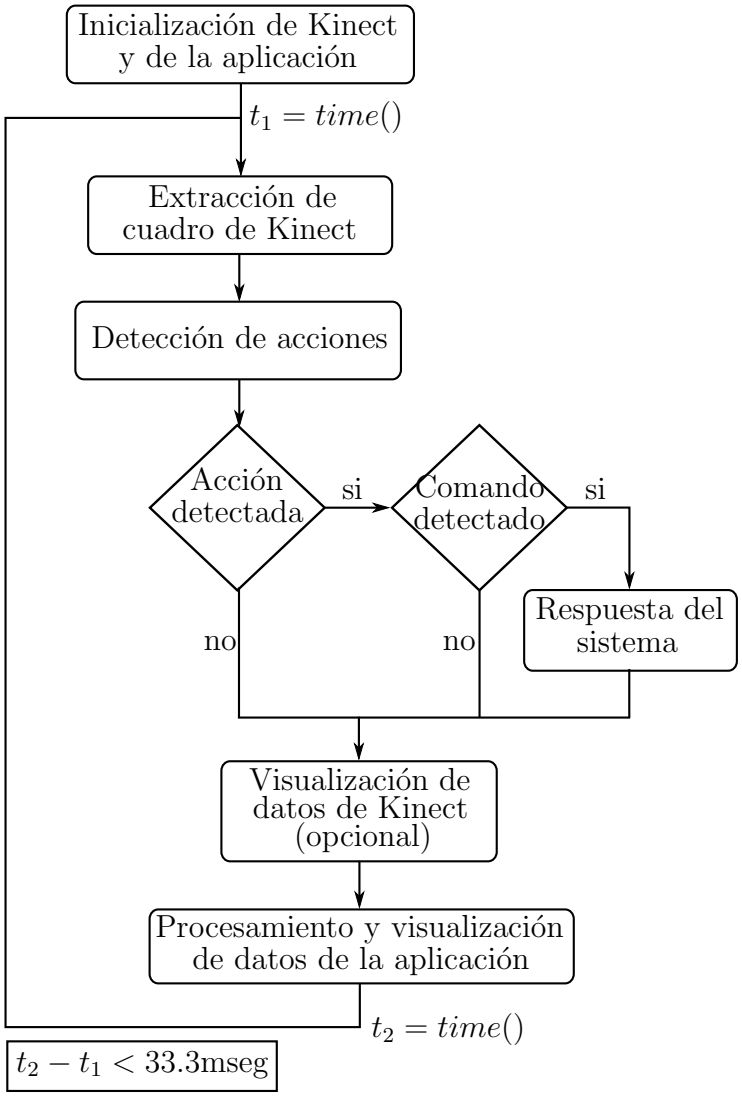


Figura 1.5: Diagrama general del sistema de interfaz natural

analizar el movimiento para detectar si el usuario ha realizado una secuencia de movimientos que desencadenan una orden a en la interfaz, visualizar los datos de Kinect y visualizar la respuesta de la aplicación (por ejemplo, el manipulador de objetos virtuales requiere que se pinte en la pantalla el objeto virtual cuando se esté interactuando con él).

1.4. Objetivos del proyecto

General

Implementar una interfaz natural que obtenga gestos o movimientos del cuerpo mediante el Kinect y los traduzca a comandos para interactuar con la computadora.

Particulares

1. Incorporar al Kinect en una interfaz desarrollada con Qt.

2. Implementar el comportamiento de un ratón con tres botones construida sobre el sistema interfaz natural.

3. Implementar una interfaz para la navegación de menús construida sobre el sistema de interfaz natural.

4. Implementar una serie de comandos para manipular un objeto tridimensional, construida sobre el sistema de interfaz natural.

1.5. Organización de la tesis

La tesis se encuentra organizada en 5 capítulos que describen cada una de las fases del sistema.

- Capítulo 2. Marco Teórico. En este capítulo se da una introducción a los conceptos y las herramientas que fueron necesarias para el desarrollo de la tesis. Los temas a tratar son:
  - Reseña de las interfaces de usuario hasta llegar a las interfaces naturales de usuario.
  - El funcionamiento del sensor Kinect.
  - El funcionamiento de la cámara de profundidad.
  - La reconstrucción de objetos.
  - La composición de objetos virtuales y las transformaciones geométricas. aplicables a éstos, usadas en una de las aplicaciones de prueba.
  - El filtrado para la reducción de ruido de los datos del rastreo del cuerpo.
- Capítulo 3. Sistema de desarrollo. En este capítulo se describe el diseño y desarrollo del sistema de interfaz, los conceptos de acción, malla de acciones y comando, que constituyen la base de la interfaz, las bibliotecas utilizadas para el acceso y visualización de los datos de Kinect, la extracción y visualización de las imágenes de profundidad y de color de las cámaras de Kinect, así como la reproducción de sonido y las pruebas del filtrado de datos.
- Capítulo 4. Aplicaciones. En este capítulo se describen las 3 aplicaciones de prueba que se construyeron sobre la interfaz (ratón Kinect, navegación de menús y manipulador de objetos virtuales), incluyendo la lista de comandos de interacción y el árbol de comandos que permite interactuar en cada una, así como el uso y adaptación de la biblioteca libg3d para cargar objetos virtuales complejos y visualizarlos en un widget de Qt mediante OpenGL. Por último se describe la arquitectura del sistema.
- Capítulo 5. Conclusiones. En este capítulo se presentan las conclusiones que se obtuvieron en el desarrollo de la tesis, así como el trabajo a futuro que se puede realizar para extender el trabajo realizado.

Capítulo 2
Marco Teórico

2.1. Interfaces de usuario

La interfaz de usuario es el espacio en donde se desarrolla la interacción entre el humano y la máquina. El objetivo de las interfaces de usuario es abstraer el funcionamiento de una máquina en una orden (o conjunto de órdenes) para evitarle al usuario la necesidad de saber el funcionamiento de la máquina.

Un ejemplo de una interfaz se presenta en el uso de una grúa hidráulica (Figura 2.1). El operador sabe, por experiencia, que para posicionar el gancho para levantar un objeto debe mover las palancas de dirección y altura del brazo de la grúa. En este ejemplo, la interfaz está constituida por las palancas, éstas ocultan la *cinemática directa* involucrada en el movimiento del brazo de la grúa, que es el cálculo de las rotaciones en las articulaciones y el desplazamiento de los eslabones del brazo para alcanzar el objeto.

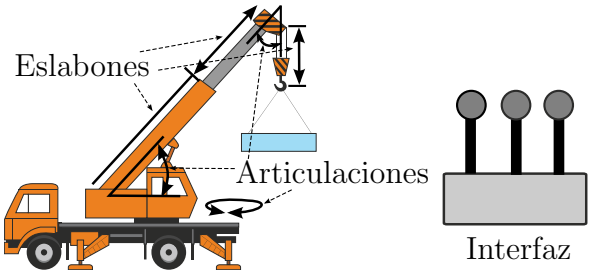
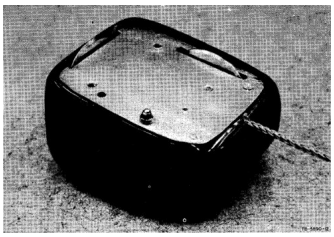


Figura 2.1: Interfaz de una grúa

En esta sección se presenta una breve reseña histórica desde la *Interfaz Gráfica de Usuario (IGU)*, que ha sido la interfaz predominante desde hace más de dos décadas. A continuación se mencionan algunos conceptos y avances tecnológicos que marcaron estas interfaces, los cuales sentaron los principios de la investigación de la *Interacción Natural (IN)*, y las *Interfaces Naturales de Usuario (INU)*.

2.1.1. Interfaces posteriores a la interfaz gráfica de usuario

Hasta hace algunos años la interacción con la interfaz gráfica de la computadora se realizaba únicamente a través del teclado y ratón. Éste último ha tenido cambios y mejoras importantes a lo largo del tiempo. El funcionamiento del ratón ha sido adaptado a las computadoras portátiles y otros tipos de ratón y dispositivos como el *trackball*, *pointtrack*, *touchpad* y recientemente el *magick trackpad* y el señalamiento con la *pantalla táctil* que extienden la funcionalidad del ratón al detectar gestos con los dedos (ver Figura 2.2).

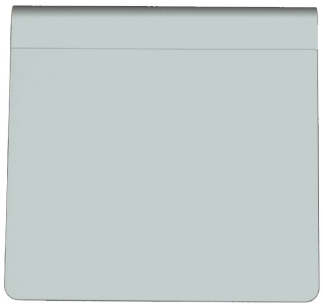


(a) Ratón de Engelbart [23] de 1968, el primer ratón de computadora

(b) Trackball en el TrackMan de Logitech

(c) Trackpoint de IBM

(d) Touchpad



(e) Magic Trackpad de Apple

(f) Pantalla táctil (imagen del iPad de Apple)

Figura 2.2: Dispositivos apuntadores similares al ratón[1]

El desarrollo de las pantallas táctiles y multi-táctiles permitió cambiar la forma en que se interactúa con una computadora, al poder tocar y manipular elementos de la pantalla directamente con los dedos o un lápiz óptico y obtener una retroalimentación inmediata de la computadora. Esta tecnología ha sido ampliamente usada en las *Interfaces Multimodales* [2, 3] y con éstas, se empezó a cambiar la forma de interactuar con computadoras, teléfonos móviles (cuya combinación con estas interfaces impulsó el *cómputo móvil* [24]) y otros dispositivos como las consolas de videojuegos.

Las interfaces multimodales se concentran en la combinación de varios métodos de entrada y de salida para extender la interfaz gráfica de usuario, aumentando la usabilidad[2] y

[1] Imágenes tomadas de: (a) [23], (b) http://upload.wikimedia.org/wikipedia/commons/f/f9/Wireless-trackman-mouse.jpg, (c) http://commons.wikimedia.org/wiki/File:Trackpoint.jpg, (d) http://commons.wikimedia.org/wiki/File:Touchpad.jpg, (e) http://upload.wikimedia.org/wikipedia/commons/0/0e/Apple_Magic_Trackpad.jpg, (f) http://en.wikipedia.org/wiki/File:IPad_2_Smart_Cover_at_unveiling_crop.jpg

[2] *Usabilidad* se refiere a la facilidad que la interfaz brinda al usuario al permitirle distintas formas de realizar una misma tarea, por ejemplo dictar palabras en lugar de escribirlas.

accesibilidad[3] de los sistemas que las utlizan.

El funcionamiento de las interfaces multimodales se explicará a continuación retomando el desarrollo de los dispositivos móviles. Actualmente los teléfonos móviles y tabletas digitales tienen el objetivo de brindar al usuario la capacidad de producir información y acceder a servicios de información en cualquier momento y en cualquier lugar a través de la infraestructura de redes de comunicación, con la principal característica de mantener su portabilidad, es decir, mantener un tamaño reducido. Estamos acostumbrados a que la interacción con una computadora se visualice a través de la pantalla, sin embargo la portabilidad de los dispositivos móviles conlleva una estrecha relación entre el tamaño de la pantalla, la capacidad de procesamiento y la duración de la batería. Esto implica que la interacción con un dispositivo móvil no puede ser igual que con una computadora convencional. Es por esto que los mecanismos de entrada y salida utilizados van más allá del teclado y un ratón, al utilizar una interfaz multimodal que en algunos sistemas operativos móviles incluye voz (reconocimiento de voz como entrada y síntesis de voz como salida), gestos táctiles (mediante pantallas multi-táctiles y plumas ópticas) y movimientos (usando acelerómetros y giroscopios).

La interfaz multimodal ha sido un gran salto en la interacción con las computadoras, sin embargo, su uso requiere estar consciente del uso de dispositivos de entrada para interactuar con la computadora. Estudios recientes en interacción humano computadora se están enfocando en las *Interfaces Naturales de Usuario (INU)*, el tema que trata esta tesis, donde el objetivo es reemplazar los dispositivos de entrada por el reconocimiento de gestos de movimiento y habilidades del cuerpo, con la idea de establecerse como la siguiente generación de interfaces de usuario y la evolución del paradigma WIMP de la IGU. En la siguiente sección del documento se explicará más a detalle el concepto de la INU.

2.2. Interfaz Natural de Usuario

A lo largo de la historia de la computación nosotros nos hemos tenido que adaptar a las interfaces provistas por las computadoras, en lugar de que éstas sean las las que se adapten a nosotros. Esta idea es la que dió pauta a la investigación de la *Interacción Natural (INU)* [4, 5], que estudia la forma en que interactuamos con nuestro ambiente (bajo el contexto de la computación) y las *interfaces naturales de usuario* [3, 5], que buscan sustituir las interfaces de usuario actuales por mecanismos que nos permitan interactuar con la computadora de la misma forma en la que interactuamos diariamente con nuestro entorno, es decir volviendo transparentes los dispositivos de entrada y facilitando el aprendizaje al usuario para interactuar mediante la interfaz.

Interacción se refiere a la capacidad de instigar y guiar comportamientos en base a *estímulos estáticos* o *estímulos dinámicos*, de tal forma que éstos lleven al establecimiento de

[3] *Accesibilidad* se refiere a la cualidad de adaptarse a las necesidades de los usuarios, por ejemplo, un dispositivo móvil puede utilizar síntesis y reconocimiento de voz para brindar servicios a una persona con discapacidad visual.

un diálogo entre la persona, su ambiente y la retroalimentación que éste le brinda a la persona. Un *estímulo estático* se entiende como la indicación para realizar una acción (un botón implica por sí mismo que puede ocurrir algo al presionarlo) y un *estímulo dinámico* se entiende como un estímulo con retroalimentación (como al reproducir un sonido de aceptación al presionar el botón), es decir, transmite a la persona la idea de que lo que ha hecho ha sido entendido por una máquina.

El término *natural* se refiere en este contexto a lo que le es más fácil o común hacer a una persona en su vida diaria. Diariamente interactuamos con otras personas y objetos que nos brindan una retroalimentación inmediata y continua. Un ejemplo simple es el conducir un automovil: es más natural saber por experiencia que se debe utilizar el volante, la palanca de velocidades y los pedales simultáneamente para dar vuelta que, por ejemplo, utilizar los siguientes comandos en para mover la tortuga en el lenguaje Logo[4]:

```
avanzar HASTA_CURVA
derecha 90
avanzar HASTA_CURVA_O_FINAL_DE_RECORRIDO
```

La INU busca la construcción de una interfaz adaptada a nuestras capacidades (como en el ejemplo del automóvil) que permite concentrar nuestra atención en *"realizar una tarea"* en lugar de en *"cómo realizar una tarea"*, *i.e. la interacción se alcanza de forma natural al mejorar la experiencia de usuario con una interfaz que es, o que se vuelve invisible al usuario en un proceso de aprendizaje por repetición.*

Reseña histórica de la INU

La INU tiene una historia de desarrollos y experimentos que se remonta a los años 80's en la búsqueda de cambiar la forma de interactuar con las computadoras. Steve Mann fue uno de los primeros en tratar de cambiar la forma de interactuar con las computadoras mediante estrategias de interfaces de usuario con el mundo real y propuso el nombre de INU para este tipo de interfaces [7]. Entre sus trabajos se encuentra el EyeTap [25] (Figura 2.4), unos lentes que agregan realidad aumentada a lo que ve el usuario, mismo que Mann ha estado perfeccionado desde 1980.

Uno de los primeros sistemas que utilizaron el movimiento del cuerpo y la voz para interactuar con una aplicación es *Put-that-there*[5] [26] desarrollado por Richard A. Bolt en 1980 (Figura 2.3).

[4] *Logo* es un lenguaje muy fácil de aprender, permite hacer "gráficos de tortuga", pintando el movimiento del cursor (conocido como tortuga) mediante comandos que indican el desplazamiento y giro a partir de la posición actual. Logo puede usarse para pintar fractales mediante funciones recursivas.

[5] *Put-that-there* permite que el usuario agregue figuras a la pantalla, pronunciando comandos de voz como "crea un círculo verde" o "mueve eso", mientras apunta a la pantalla con el brazo y pronuncia "ahí".

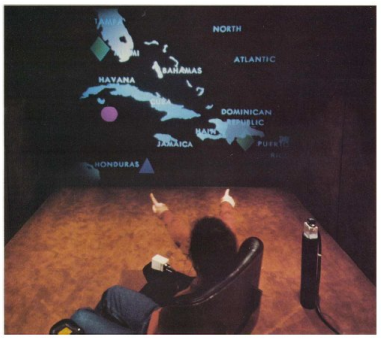


Figura 2.3: Sistema put-that-there de Richard A. Bolt

Figura 2.4: EyeTap de Steve Mann

En 2006 se estableció una comunidad con el objetivo de expandir la investigación y desarrollo de tecnologías relacionadas a las INU [27]. Microsoft también ha estado investigando en la mejora de las interfaces, entre sus desarrollos se encuentra Microsoft Surface[6], mostrada en la Figura 2.5, con el que Microsoft declaró que la INU será la siguiente etapa de las interfaces de usuario y el sensor Kinect, que desde el 2010 ha posicionado a la consola Xbox 360 como la primer INU comercial (Figura 2.6).

Figura 2.5: Microsoft Surface

Figura 2.6: Interfaz Natural con el Sensor Kinect

Entre las investigaciones de la INU, se ha tratado de volverla portable, entre estas ideas se encuentra la llamada "tecnología de sexto sentido"[7] de Pranav Mistry [28], con el reemplazo del monitor por un picoproyector, con el que es posible proyectar la interfaz en cualquier lugar, e interactuar con dicha proyección mediante el reconocimiento de gestos con dedales de color en los dedos a través de una cámara (Figura 2.7).

[6] *Microsoft Surface* es una mesa formada por una pantalla multi-táctil que permite interacción entre varios usuarios y objetos al mismo tiempo, también tiene la capacidad de reconocer objetos y otros dispositivos para acceder a su información. La la imagen 2.5 fue tomada de `http://en.wikipedia.org/wiki/Microsoft_Surface`

[7] *Video demostrativo de la tecnología sexto sentido* `http://www.ted.com/talks/pranav_mistry_the_thrilling_potential_of_sixthsense_technology.html`

Sistema | Interactuando con un mapa

Figura 2.7: Tecnología de Sexto Sentido de Pranav Mistry

2.3. Kinect

Kinect fue construido con el objetivo de revolucionar la experiencia de juego mediante un sistema de interacción completamente nuevo: interacción natural con el videojuego utilizando únicamente movimientos del cuerpo y comandos de voz.

Además de la tecnología empleada, la base de este enfoque es el entendimiento del movimiento del cuerpo, es decir, la computadora debe interpretar y entender el movimiento del usuario antes de que el videojuego pueda responderle. Este hecho implica que el procesamiento debe ser en tiempo real para que el usuario no perciba una demora durante la interacción, lo cual ha sido muy complejo de lograr en trabajos previos, dado que, además de que el ambiente debe ser controlado (iluminación, marcadores de color, tonos de piel o uso de dispositivos sensores como guantes o trajes de movimiento), el tiempo de procesamiento es muy alto.

Kinect es una tecnología de control para la consola Xbox 360 de Microsoft que permite interactuar con un videojuego sin la necesidad de usar un control; utiliza la tecnología de cámara de profundidad desarrollada por la compañía israelí *PrimeSense* [11] la cual permite a Kinect "ver" la *escena* (al usuario y su entorno en tres dimensiones) en tiempo real. Las *imágenes de profundidad*[8] se obtienen con Kinect y posteriormente se procesan por software en la consola para interpretar la escena, detectar personas y rastrear sus movimientos (*rastreo del esqueleto*). Kinect provee captura tridimensional del cuerpo en un esqueleto virtual formado por un conjunto de puntos 3D (articulaciones) relacionados a las partes del cuerpo. Esta abstracción permite detectar gestos del cuerpo al comparar los valores de las articulaciones virtuales contra los valores propuestos de un gesto específico.

El impacto de Kinect fué más allá de los videojuegos, incluso se rompió el record *Guinness*[9] por sus ventas. Su bajo costo, amplia disponibilidad y el desarrollo de bibliotecas para accederlo han permitido que se experimente con él en el desarrollo de aplicaciones creativas para buscar nuevas formas de interactuar con la computadora.

[8] A diferencia de una imagen convencional donde cada pixel representa un color, los pixeles de una imagen de profundidad representan la distancia a la que se encuentra el objeto formado por dichos pixeles.

[9] *Record Mundial Guinness* del "dispositivo de consumo electrónico más rápidamente vendido" por más de 10 millones de unidades en sus primeros 3 meses.

En la sección 1.1 (en la pág. 1) se dió una breve reseña sobre el uso de Kinect en las computadoras convencionales.

2.3.1. Arquitectura de Kinect

La estructura del sensor Kinect es similar a una cámara web. Incluye una *cámara de profundidad*, una cámara RGB y una matriz de 4 micrófonos, que aíslan las voces del ruido ambiental permitiendo utilizar al Kinect como un dispositivo para charlas y comandos de voz en la consola Xbox. Se encuentra sobre una base motorizada, controlada por un acelerómetro de 3 ejes, que le permite rotar horizontalmente para ajustar el campo de vista de las cámaras para ver el cuerpo completo del usuario.

La Figura 2.8 muestra un diagrama de los componentes de hardware del procesador de imagen de Kinect. Es operado por el sistema en chip PS1080 [29, 30], desarrollado por *PrimeSense*, que se encarga de la generación y sincronización de las *imágenes de profundidad* e imágenes de color. El chip ejecuta todos los algoritmos de adquisición de imágenes de profundidad de la escena a partir del sistema de proyección de un patrón de puntos infrarrojos llamado *LightCoding*.

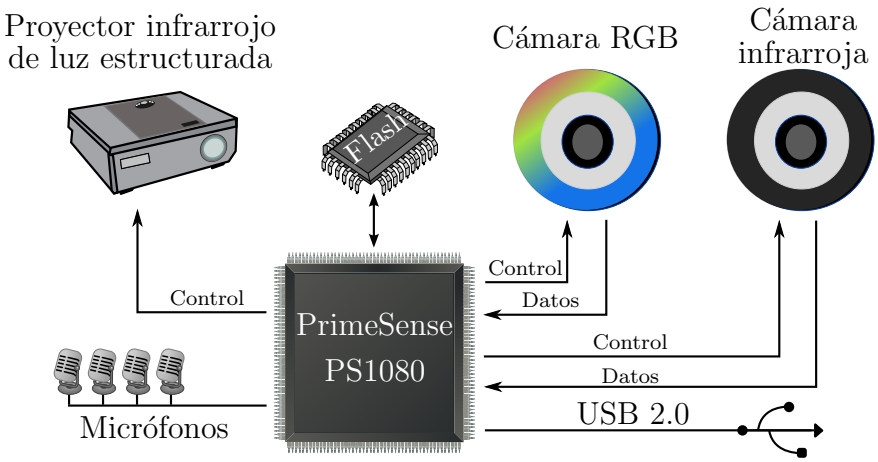


Figura 2.8: Diagrama del hardware de Kinect [30]

Kinect funciona bajo un esquema maestro-esclavo, donde el maestro es la computadora y el esclavo es el Kinect. El chip genera y mantiene en memoria los *cuadros*[10] de imagen de profundidad y color a una velocidad de 30 cps (cuadros por segundo). El acceso a estos datos se realiza a través de un puerto USB 2.0 especial[11]. El procesamiento del audio y el control de USB es realizado por un microprocesador Marvell Technology que funciona independientemente al procesamiento de imágenes.

[10] Un *cuadro* denota la obtención de nuevos datos de imagen. En Kinect, cuadro se refiere a la obtención de una imagen de color, una imagen de profundidad o ambas.

[11] Kinect utiliza un puerto USB 2.0, pero su conector es más grueso para diferenciarlo del puerto normal debido a que requiere un voltaje de 12 volts, mismo que es suministrado por un adaptador de corriente.

La tabla 2.1 presenta las especificaciones de Kinect. Una descripción detallada de cada componente de Kinect se encuentra en [30].

Elemento del sensor	**Rango de especificación**
Captura de imágenes de color y profundidad	1.2 a 3.5 metros
Rastreo del esqueleto	1.2 a 3.5 metros
Campo de vista	43° vertical, 57° horizontal
Rotación de la base	±28°
Cámara de profundidad[12]	11 bits, SXGA (1280 × 1024) a 10 cps, VGA (640 × 480) a 30 cps, QVGA (320 × 240) a 60 cps, sin autoenfoque
Cámara de color	8 bits, 1.3MP(1280 × 960) a 10 cps, VGA (640 × 480) a 30 cps, QVGA (320 × 240) a 60 cps, sin autoenfoque
Memoria	512 MB DDR2 SDRAM
Formato de audio	16 kHz, 16 bits mono PCM
Entrada de audio	Arreglo de 4 micrófonos con ADC de 24 bits y procesamiento de huesped-Kinect, cancelación de eco acústico y supresión de ruido.
Conectividad	Puerto USB 2.0 (propietario modelo S de la consola) para proveer alimentación al motor, se adapta a USB 2.0 convencional con el adaptador eléctrico de 12V.

Tabla 2.1: Especificaciones de Kinect.

2.3.2. Cámara de profundidad

La cámara de profundidad se compone por la cámara infrarroja y el proyector infrarrojo de luz estructurada, como se muestra en la Figura 2.9.

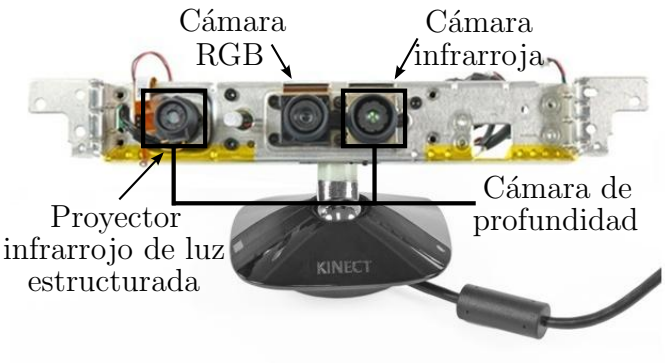
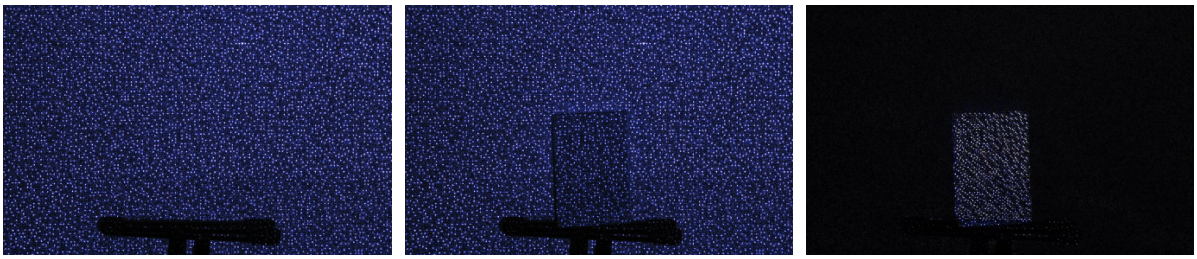


Figura 2.9: Cámaras de Kinect

[12]Las resoluciones de color y profundidad mostradas en la tabla corresponden a los modos de trabajo de la cámara, sin embargo en *OpenNI* fue operable únicamente en VGA a 30 cps.

La cámara de profundidad utiliza una tecnología de codificación por luz llamada *Light-Coding* desarrollada por *PrimeSense* que actúa como un escáner 3D para realizar una reconstrucción tridimensional de la escena. *LightCoding* es similar a los escáners de *luz estructurada*[13], pero en lugar de desplazar una línea de luz, se proyecta un patrón de puntos infrarrojos en la escena. El patrón de puntos se localiza en un difusor (rejilla) frente al proyector infrarrojo. Al emitir la luz infrarroja, el patrón se dispersa por la escena, proyectándose sobre las personas y objetos presentes en ésta (como se puede observar en la Figura 2.10), mientras que la imagen de dicha proyección se obtiene mediante la cámara infrarroja.



(a) (b) (c)

Figura 2.10: Fotografías de la proyección del patrón de puntos infrarrojos (a) sobre un libro cerrado, (b) un libro abierto. En (c) se muestra la diferencia[14].

Como se describe en la patente del sistema de *PrimeSense* [31], el patrón se mantiene constante a través del eje Z ("constante" se refiere a que la forma del patrón no cambia si se proyecta en una superficie perpendicular al eje focal de la cámara dentro del rango de especificación de 1.2 a 3.5 m), sin embargo, al existir objetos en la escena el patrón se proyecta o adapta a la forma de éstos, con lo que se obtiene un desplazamiento del patrón en el plano XY de la imagen infrarroja en relación al patrón, como se observa entre las Figuras 2.10 (a) y (b). Este desplazamiento es el que permite realizar una triangulación entre los puntos del patrón y los puntos del patrón proyectados en la imagen, para realizar la reconstrucción tridimensional de la escena y obtener la imagen de profundidad.

Se tomará el siguiente ejemplo para explicar la generación de las imágenes de profundidad: imaginemos que se rocía horizontalmente la escena con una pintura en aerosol para aplicar una capa uniforme de pintura de forma instantánea. Los objetos que se encuentran más cerca al dispersor serán los primeros en ser pintados y el fondo de la escena (y los objetos más lejanos al dispersor) tendrán zonas sin pintura en donde existan objetos al frente. La proyección del patrón actúa como esta capa de pintura al dispersarse por la escena. Si pudiéramos despegar esta capa de pintura notaremos que sería similar a un "molde de la escena". Este molde sería la imagen de profundidad de la escena.

[13]Un escáner de luz estructurada proyecta y desplaza una línea de luz infrarroja a sobre el objeto (de forma similar a un escáner de documentos), la deformación de la línea captada por una cámara permite estimar la forma del objeto y reconstruirlo tridimensionalmente. Sin embargo es un procedimiento costoso en tiempo y procesamiento que no permite su uso en aplicaciones de tiempo real.

[14]Fotografías obtenidas de `http://www.futurepicture.org/?p=116`, `http://mirror2image.wordpress.com/2010/11/30/how-kinect-works-stereo-triangulation/`

2.3.3. Reconstrucción de objetos

Kinect reconstruye la escena visualizada en 3D para obtener las imágenes de profundidad. La reconstrucción de objetos [32] es una área especial de *visión por computadora*[15] [33]. Se encarga del desarrollo e investigación de técnicas para reconstruir objetos tridimensionales y el cálculo de la distancia entre el sensor y los objetos de la escena. Es utilizada para identificar y conocer la distancia hacia objetos en la escena y detección de obstáculos (robótica y sistemas de control vehicular), inspección de superficies en sistemas de control de calidad y navegación de vehículos autónomos. También es utilizada en la estimación de objetos tridimensionales (sistemas de ensamblado automático). La reconstrucción de objetos involucra otras áreas como el procesamiento de imágenes (filtrado, restauración y mejora de imágenes, entre otras) y reconocimiento de patrones (segmentación, detección de bordes y extracción de características, entre otras).

Una de las primeras técnicas usadas para estimar la forma de un objeto o mapa 3D se basa en la triangulación [34], que utiliza dos cámaras viendo el mismo objeto. El cambio de posición del objeto en ambas imágenes se relaciona con la distancia a dicho objeto, siendo similar al sistema de visión humano. La desventaja que presenta es la baja resolución de la reconstrucción 3D, que depende de la definición y orientación relativa de las cámaras (ángulo y distancia), además el procesamiento de alto nivel que requiere no hace factible su uso en tiempo real.

Existen técnicas que permiten obtener el mapa 3D a partir de las sombras de los contornos del objeto en la imagen [35], sin embargo requieren procesamiento de alto nivel y no es apropiado dado que son áreas ruidosas de la imagen. Otras técnicas se basan en las diferencias existentes de color y textura para distinguir objetos o personas del fondo de la imagen, sin embargo, tienen limitaciones en cuanto a las condiciones de luz y/o colores presentes en la imagen, lo cual provoca que la segmentación de la persona u objeto sea compleja e ineficiente. Otra tecnología utilizada son las *cámaras de tiempo de vuelo (Time-Of-Flight)*, las cuales son similares a un sonar que emite rayos de luz infrarroja a la escena, al conocer el tiempo que los rayos tardan en regresar siendo reflejados por los objetos presentes, se puede conocer la distancia a la que se encuentran y generar una imagen de profundidad de la escena, esta tecnología es poco utilizada debido a su alto costo.

La tecnología de *PrimeSense* evita los problemas de ambigüedad de colores e iluminación al convertir la segmentación de la persona en un problema de clasificación de imágenes de profundidad obtenidas en tiempo real mediante un dispositivo de bajo costo.

2.3.4. Rastreo del esqueleto

El rastreo del esqueleto consta de procesar las imágenes de profundidad obtenidas con Kinect para detectar formas humanas e identificar las partes del cuerpo del usuario presente en la imagen. Cada parte del cuerpo es abstraída como una coordenada 3D o *ar-*

[15] *Visión por computadora* se encarga del estudio de la recuperación de la información tridimensional de una escena a partir de imágenes bidimensionales de la misma.

ticulación. Un conjunto de articulaciones forman un *esqueleto virtual* para cada imagen de profundidad de Kinect, es decir, se obtienen 30 esqueletos por segundo. Las articulaciones generadas varían de acuerdo a la biblioteca de Kinect que se utilice. En *OpenNI/NITE*, cada esqueleto (Figura 2.11) está formado por 15 articulaciones $a_i = \{x_i, y_i, z_i\}$ con $z_i > 0$ cuyas coordenadas se encuentran expresadas en milímetros con respecto a la posición de Kinect en la escena (Figura 2.13). En el SDK de Microsoft y en la consola Xbox se añaden 5 articulaciones (los tobillos, las muñecas y el centro de la cadera).

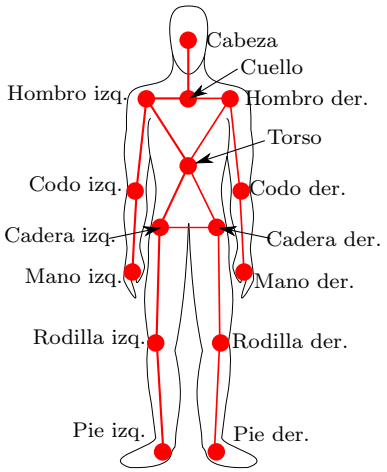


Figura 2.11: Esqueleto virtual de *OpenNI/NITE*.

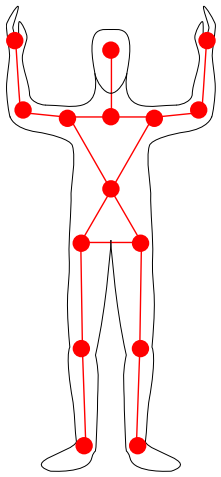


Figura 2.12: Pose "psi" de inicio de rastreo del esqueleto *OpenNI/NITE*

El rastreo del esqueleto de *OpenNI/NITE* se inicia al realizar la pose de calibración "psi"[16] levantando los brazos durante dos segundos como se muestra en la Figura 2.12.

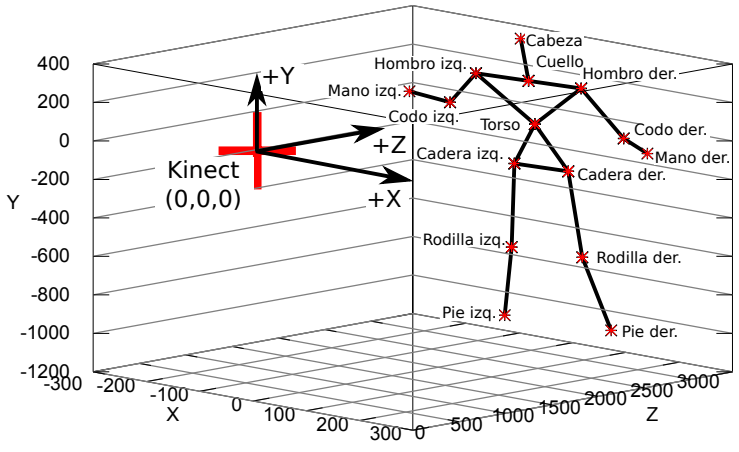


Figura 2.13: Configuración de la escena del esqueleto en *OpenNI/NITE*. Las articulaciones se encuentran en milímetros con respecto al origen del sistema de coordenadas (posición del Kinect) y en espejo (el eje X está invertido para que el lado izquierdo del esqueleto corresponda al lado derecho del cuerpo y viceversa). Kinect mira hacia el lado positivo del eje Z.

[16]La pose se conoce por este nombre, dado el parecido con la letra griega ψ. A partir de la versión 1.5 de *NITE* la pose ya no es necesaria para iniciar el rastreo del esqueleto.

La técnica de rastreo del esqueleto usada en la consola Xbox 360 se explica en [36, 37]. En la Figura 2.14 se muestran las etapas de la técnica de rastreo del esqueleto, misma que toma un enfoque de reconocimiento de objetos con la representación intermedia de las partes del cuerpo que se relacionan a cada articulación del esqueleto virtual. La representación en partes del cuerpo permite tratar el problema de estimación de pose (posición del cuerpo) como un problema de clasificación por pixel. Algunas de las partes permiten encontrar las articulaciones de interés, mientras que la combinación de varias partes permite predecir la posición de otras articulaciones.

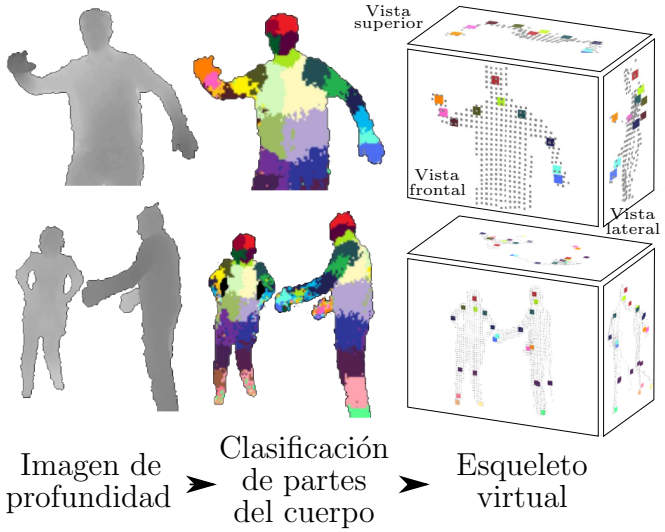


Figura 2.14: Rastreo del esqueleto en la consola Xbox 360 [36]

Para segmentar cada parte del cuerpo los autores construyeron un clasificador de árboles de decisión aleatorios o *bosque aleatorizado de árboles*[17] y un conjunto de más de 500,000 imágenes de profundidad de secuencias de humanos (reales y generadas por computadora) de diversas complexiones y estaturas realizando acciones como correr, saltar, nadar y apuntar, entre otras. El clasificador se entrenó con un subconjunto de 100,000 imágenes de profundidad de la base de datos en donde las poses difieren en más de 5 cm. El clasificador no usa información temporal, es decir, busca poses estáticas en cada imagen de entrada y no movimientos entre imágenes de entrada consecutivas.

El clasificador de bosque es un conjunto de T árboles de decisión, donde cada nodo hoja almacena la distribución c_i aprendida, que etiqueta a la parte c_i del cuerpo y los nodos rama contienen en una característica f y un umbral τ.

La técnica de rastreo del esqueleto inicia tomando una sola imagen de profundidad de Kinect (de 640×480 pixeles). Para clasificar un pixel $\boldsymbol{x}$ de la imagen de entrada, se comienza en la raíz de un árbol del clasificador y se evalúa repetitivamente la Ec. 2.1 (que

[17]Los autores de [36] denominaron el clasificador usado como *randomized tree forest* debido a que generan un árbol compuesto por un conjunto de subárboles.

permite saber si un pixel $\boldsymbol{x}$ está cerca o lejos de Kinect en la imagen I) para seguir la rama izquierda o derecha de acuerdo al umbral τ en el nodo. Si la ruta conduce a un nodo hoja, se sabe que el pixel $\boldsymbol{x}$ pertenece a la parte c_i del cuerpo.

Para cada pixel $\boldsymbol{x}$ de la imagen de entrada, la característica f se calcula con

$$f(I, \boldsymbol{x}) = d_I\left(\boldsymbol{x} + \frac{\boldsymbol{u}}{d_I(\boldsymbol{x})}\right) - d_I\left(\boldsymbol{x} + \frac{\boldsymbol{v}}{d_I(\boldsymbol{x})}\right) \tag{2.1}$$

donde $d_I(\boldsymbol{x})$ es la profundidad del pixel $\boldsymbol{x}$ y los parámetros $(\boldsymbol{u}, \boldsymbol{v})$ son desplazamientos en coordenadas globales que al normalizarse por $\frac{1}{d_I(\boldsymbol{x})}$ permiten que la característica sea invariante a la profundidad y a la traslación en el espacio tridimensional.

Finalmente, cada articulación del esqueleto se encuentra a partir del cálculo de la media de cambio (mean- shift) de los pixeles que forman la parte del cuerpo c_i que se relaciona a la articulación de interés.

Los autores de [36] mencionan que una versión optimizada del algoritmo se ejecuta a 200 cps en paralelo en el GPU[18] de la consola.

2.4. Objetos virtuales

Los objetos virtuales, la manipulación de éstos y de la escena donde se visualizan son parte del área de *Graficación por Computadora*[19] [38].

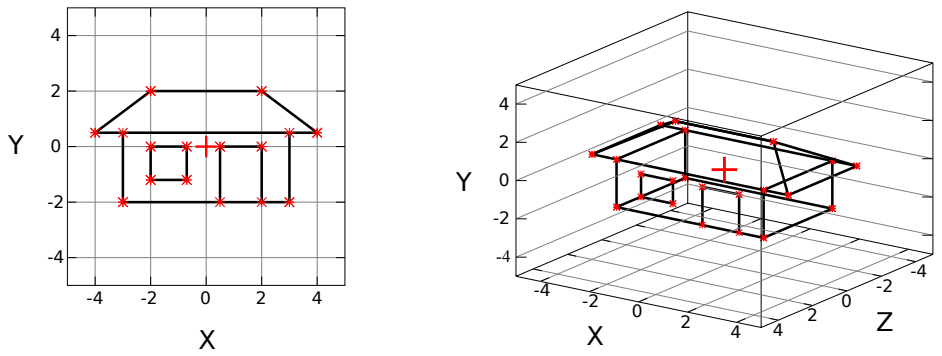
Podemos definir un objeto virtual (bi o tridimensional) como un conjunto de puntos o *vértices* en $\Re^2$ o en $\Re^3$, un conjunto de *aristas* (líneas entre pares de vértices) y un conjunto de *caras*.

Una cara se forma cuando tres o más vértices del exterior del objeto son coplanares y las aristas entre éstos forman un polígono. Una cara puede tener un color o una *textura* (cuando un objeto virtual esá texturizado, una o varias imágenes "envuelven" al objeto, por lo que cada cara contiene un fragmento de la imagen que la cubre).

Un ejemplo de un objeto simple se muestra en la Figura 2.15, el objeto está formado por 22 vértices, 26 aristas y 13 caras.

[18] ATI Xenos 512 MB de RAM GDDR3 a 500 MHz.

[19] *Graficación* es la construcción de escenas bi o tridimensionales a partir de modelos matemáticos de las formas dentro de la escena.



(a) (b)

Figura 2.15: Objeto virtual simple. En (a) se muestra una casa en $\Re^2$, en (b) se muestra en $\Re^3$. El símbolo $*$ indica los vértices y $+$ el origen de coordenadas.

El dibujado de objetos puede realizarse en OpenGL. OpenGL dibuja los objetos en base a caras triángulares o cuadradas, por lo que los polígonos que tienen más de 4 lados y que constituyen una cara del objeto son divididos en triángulos o cuadrados automáticamente al dibujarse.

Además del dibujado, es posible alterar la forma y posición del objeto a través de la aplicación de *transformaciones geométricas* a cada vértice del objeto.

2.4.1. Transformaciones geométricas

Las transformaciones geométricas básicas son traslación, escalamiento y rotación.

Con la finalidad de representar las transformaciones como un producto de matrices de la forma

$$\mathbf{p}' = M \cdot \mathbf{p}$$

las coordenadas se homogeneízan al agregar una coordenada $w = 1$ a cada punto $\mathbf{p}_i$ del objeto, de tal forma que $\mathbf{p}_{2D_i}\{x, y, w\} \in \Re^2$ y $\mathbf{p}_{3D_i}\{x, y, z, w\} \in \Re^3$.

A un punto $\mathbf{p} \in \Re^3$ (y de forma similar a un punto $\mathbf{p} \in \Re^2$) se le puede aplicar:

- Traslación. El punto $\mathbf{p}$ cambia de posición al agregarle un desplazamiento $\mathbf{t} \in \Re^3$ (positivo o negativo).

  $$x' = x + t_x, \qquad y' = y + t_y, \qquad z' = z + t_z$$

  En forma matricial:

  $$\mathbf{p}' = \mathbf{p} + \mathbf{t}$$

  $$\mathbf{p}' = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}, \qquad \mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \qquad \mathbf{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

  En forma matricial con coordenadas homogéneas:

  $$\mathbf{p}' = T(t_x, t_y, t_z) \cdot \mathbf{p}$$

$$T(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Escalamiento. El punto $\mathbf{p}$ puede estirarse o comprimirse en cualquier eje con respecto al origen en $\mathfrak{R}^3$ al multiplicarse por un factor de escala $s \in \mathfrak{R}^3$.

$$x' = s_x \cdot x, \qquad y' = s_y \cdot y, \qquad z' = s_z \cdot z$$

En forma matricial:

$$\mathbf{p}' = S(s_x, s_y, s_z) \cdot \mathbf{p}$$

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Rotación. El punto $\mathbf{p}$ puede rotarse un ángulo θ alrededor de un eje de $\mathfrak{R}^2$ o $\mathfrak{R}^3$ y con respecto al origen del sistema de coordenadas. La rotación alrededor un eje en $\mathfrak{R}^3$ involucra operaciones sobre los dos ejes restantes, así la rotación en $\mathfrak{R}^2$ es similar a la rotación alrededor el eje Z en $\mathfrak{R}^3$.

  La rotación sigue el sentido de la regla de la mano derecha, *i.e.*, se extiende el pulgar de la mano derecha en la dirección positiva del eje y el sentido de la rotación esta dado en la dirección de los demás dedos.

  Rotación en $\mathfrak{R}^2$.

$$x' = x \cdot \cos\theta - y \cdot \operatorname{sen}\theta, \qquad y' = x \cdot \operatorname{sen}\theta + y \cdot \cos\theta$$

En forma matricial:

$$\mathbf{p}' = R \cdot \mathbf{p}$$

$$R = \begin{bmatrix} \cos\theta & -\operatorname{sen}\theta & 0 \\ \operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotación alrededor del eje X en $\mathfrak{R}^3$.

$$x' = x, \qquad y' = y \cdot \cos\theta - z \cdot \operatorname{sen}\theta, \qquad z' = y \cdot \operatorname{sen}\theta + z \cdot \cos\theta$$

$$\mathbf{p}' = R_x(\theta) \cdot \mathbf{p}$$

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\operatorname{sen}\theta & 0 \\ 0 & \operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotación alrededor del eje Y en $\mathfrak{R}^3$.

$$x' = x \cdot \cos\theta - z \cdot \operatorname{sen}\theta, \qquad y' = y, \qquad z' = x \cdot \operatorname{sen}\theta + z \cdot \cos\theta$$

$$\mathbf{p}' = R_y(\theta) \cdot \mathbf{p}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \operatorname{sen}\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\operatorname{sen}\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotación alrededor del eje Z en $\Re^3$.

$$x' = x \cdot \cos\theta - y \cdot \operatorname{sen}\theta, \qquad y' = x \cdot \operatorname{sen}\theta + y \cdot \cos\theta, \qquad z' = z$$

$$\mathbf{p}' = R_z(\theta) \cdot \mathbf{p}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\operatorname{sen}\theta & 0 & 0 \\ \operatorname{sen}\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

En la Figura 2.16 se presenta un ejemplo de traslación, escalado y rotación aplicadas al objeto de la Figura 2.15.

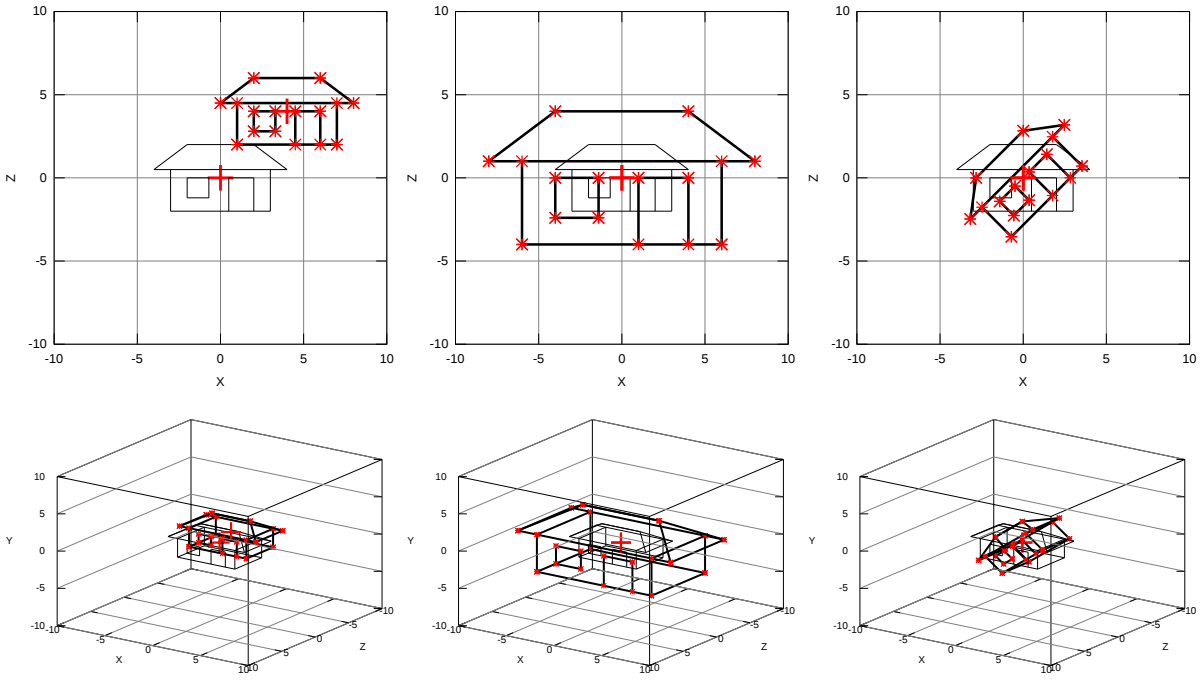


Figura 2.16: Ejemplo de transformaciones geométricas. En la columna (a) se presenta la traslación $T(4,4,4)$, en (b) el escalado $S(2,2,2)$ y en (c) la rotación $R_z(45°)$ del objeto de la Figura 2.15. La primer fila de gráficas corresponde al objeto en 2D, la segunda en 3D.

2.4.2. Transformaciones geométricas compuestas

Una transformación geométrica compuesta se refiere a la aplicación sucesiva de transformaciones geométricas, resultando en un producto de matrices de la forma

$$\mathbf{p}' = M_n \cdots M_2 \cdot M_1 \cdot M_0 \cdot \mathbf{p}$$

Esta es la razón de utilizar coordenadas homogéneas. La transformación se aplica de derecha a izquierda, siendo el extremo derecho cada punto $\mathbf{p}$ del objeto.

En las transformaciones compuestas se debe tomar en cuenta que la rotación y el escalamiento se realizan con respecto al origen, por lo tanto:

- Para rotar el objeto sobre su centro y alrededor de cualquier eje, hay que trasladar el objeto a su centro, aplicar la rotación y regresarlo adonde estaba, es decir: $\mathbf{p}' = T(c_x, c_y, c_z) \cdot R_{eje}(\theta) \cdot T(-c_x, -c_y, -c_z) \cdot \mathbf{p}$.
- Para escalar el objeto sobre cualquier eje el procedimiento es similar a la rotación: $\mathbf{p}' = T(c_x, c_y, c_z) \cdot S(s_x, s_y, s_z) \cdot T(-c_x, -c_y, -c_z) \cdot \mathbf{p}$.

En la Figura 2.17 se presenta la transformación compuesta $T(-1.5, -2, 1) \cdot R_y(300°) \cdot S(1.5, 1.5, 1.5)$ al objeto de la Figura 2.15.

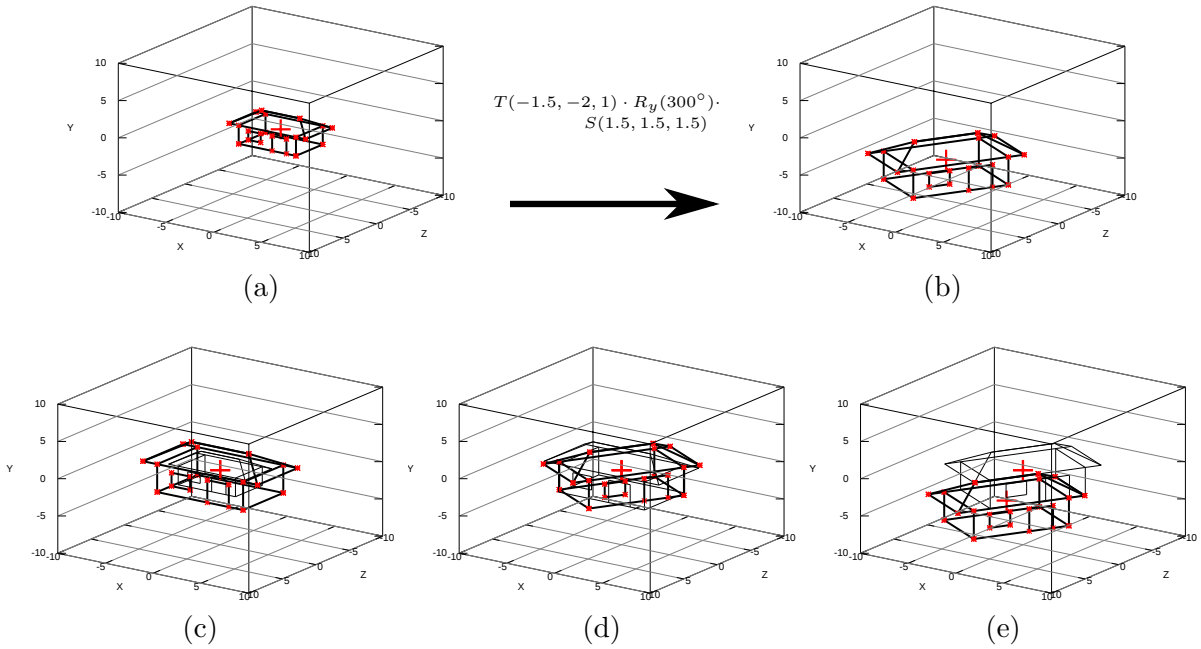


Figura 2.17: Ejemplo de transformación geométrica compuesta. Las subfiguras (c), (d) y (e) muestran la secuencia de transformaciones realizadas para pasar de (a) a (b). La transformación del objeto inicial (a) a (c) es $S(1.5, 1.5, 1.5)$, de (c) a (d) es $R_y(300°)$ y de (d) a (e) es $T(c_x, c_y, c_z)$.

Capítulo 3
Sistema de desarrollo

3.1. Extracción de datos de Kinect

En esta sección se explica la generación y extracción de datos de Kinect mediante *OpenNI*.

La Figura 3.1 muestra la arquitectura de una aplicación que hace uso de Kinect a través de *OpenNI/NITE*. A nivel de hardware se cuenta con el Kinect conectado por el puerto USB a la computadora. La biblioteca *SensorKinect* [39] permite que *OpenNI* acceda al Kinect a través del puerto USB. A nivel de usuario del sistema operativo se encuentra *OpenNI*, que realiza la conexión con Kinect y permite extraer las imágenes de profundidad, color e infrarrojas. Sin embargo para realizar el rastreo del esqueleto *OpenNI* requiere *NITE*.

La Figura 3.2 presenta el flujo de datos de *OpenNI*. Se utilizará el término *generación* para indicar la generación de datos a partir de las cámaras de Kinect y el esqueleto obtenido mediante *OpenNI* y el término *extracción* a los procedimientos implementados en el sistema desarrollado para acceder y utilizar dichos datos generados.

OpenNI se basa en el concepto de *nodos de producción*, que son un grupo de clases que permiten la generación y extracción de los datos obtenidos con Kinect. El control de flujo de generación de datos de Kinect se realiza por una instancia de la clase *Context*, a la que se le registra una única instancia de cada tipo de nodo de producción a utilizar. Adicionalmente, cada tipo de nodo se especifica en el archivo XML de configuración de *OpenNI*. Los nodos utilizados en el sistema de interfaz natural son los siguientes:

- **DepthGenerator**. Generador de imágenes de profundidad.
- **UserGenerator.** Generador de usuarios. Requiere el nodo **DepthGenerator** para aplicar el rastreo del esqueleto a la secuencia de imágenes de profundidad y obtener las coordenadas 2D y 3D del mismo.
- **ImageGenerator.** Generador de imágenes de color.

- **IRGenerator.** Generador de imágenes del patrón de puntos infrarrojos proyectados en la escena.

Es importante señalar que sólo se deben especificar en el archivo XML los nodos que se van a utilizar, de lo contrario se desperdiciará tiempo de ejecución, incluso si los datos no son visualizados o utilizados. Los nodos mínimos utilizados en el sistema desarrollado son **UserGenerator** y **DepthGenerator**.

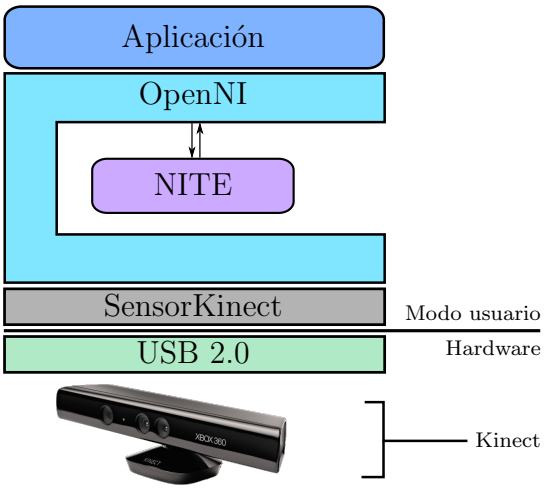


Figura 3.1: Arquitectura de una aplicación basada en *OpenNI/NITE*

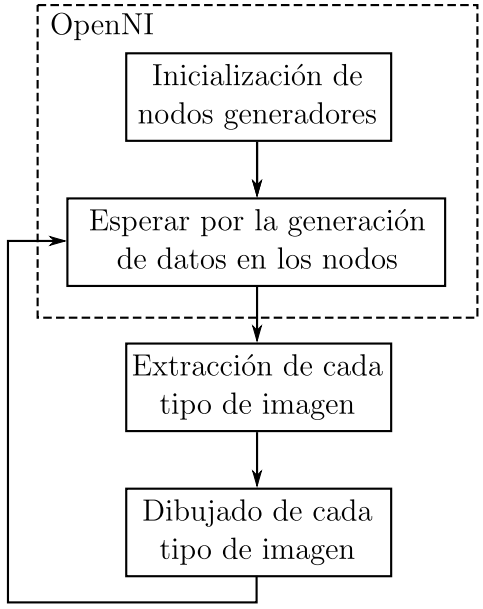


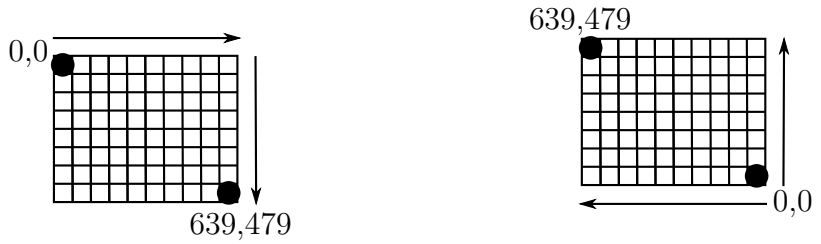
Figura 3.2: Flujo de datos de *OpenNI*

En el flujo de datos de *OpenNI* (ver Figura 3.2), la etapa *"esperar por la generación de datos de los nodos"* se refiere a un mecanismo de *barrera* en la espera por la generación de nuevos datos de cada nodo registrado al contexto de *OpenNI*. Se utilizó el método **WaitAndUpdateAll()**, el cual espera a que todos los nodos tengan datos nuevos para continuar el flujo de datos de *OpenNI*.

3.1.1. Extracción y dibujado de imágenes de profundidad, color e infrarrojas

Kinect genera las imágenes de profundidad, color e infrarrojas a 30 cps (cuadros por segundo) a una resolución de 640×480 pixeles. Estos pixeles son almacenados línea por línea, como se muestra en la Figura 3.3(a), en el buffer del nodo de producción relacionado a cada tipo de imagen (**DepthGenerator**, **ImageGenerator** o **IRGenerator**).
OpenNI solo permite la generación de un par de tipos de imágenes a la vez: profundidad y color, o profundidad e infrarrojas. Es importante señalar que la generación de imágenes de profundidad es necesaria para el rastreo del esqueleto, sin embargo, su dibujado y la generación de las imágenes de color e infrarrojas son opcionales.

El procedimiento para la extracción y dibujado de los 3 tipos de imagen es el mismo,



(a) Buffer de la imagen extraída (b) Buffer de dibujado de imagen

Figura 3.3: Estructura del buffer de imagen

sin embargo las características de cada tipo de imagen varían en cuanto a los tipos de datos y formas de dibujarse. Más adelante se explicarán las consideraciones de cada una.

Para utilizar las imágenes generadas por los nodos, éstas se extraen del buffer de cada nodo y se almacenan mediante un mecanismo de *doble buffer* protegido, el cual fue implementado en la aplicación desarrollada en este trabajo de tesis. Para cada tipo de imagen se cuenta con dos buffers adicionales; mientras se almacena una imagen en uno de los buffers, se lee una imagen del otro buffer para ser dibujada y viceversa. Esto evita que en el mismo buffer se guarde una imagen antes de que se termine de guardar la anterior y que durante el dibujado se muestren imágenes mezcladas.

El algoritmo 1 presenta el almacenamiento de las imágenes en los buffers protegidos.

Algoritmo 1 Algoritmo para el almacenamiento de imágenes en los buffers protegidos

Entrada: Imagen de uno de los generadores (*imagen*)
Salida: Imagen almacenada en uno de los buffers ($buffer[0]$ o $buffer[1]$)

1: Inicialización de la variable protegida $b\_used \rightarrow 0$
2: **si** $b\_used == 1$ **entonces**
3: Bloqueo de variable protegida
4: $buffer[0] \rightarrow imagen$
5: $b\_used \rightarrow 0$
6: Desbloqueo de variable protegida
7: **si no**
8: Bloqueo de variable protegida
9: $buffer[1] \rightarrow imagen$
10: $b\_used \rightarrow 1$
11: Desbloqueo de variable protegida
12: **fin si**

El objetivo del índice *b_used* y de protegerlo en una de las secciones críticas es que cada vez que se termine de almacenar una imagen en uno de los buffers, *b_used* apunte al otro buffer, así la siguiente imagen a almacenar no sobreescribirá a la primera. Este procedimiento se realiza cada vez que todos los nodos de producción hayan generado nuevos datos, *i.e.* cuando la barrera **WaitAndUpdateAll()** se supera (como se observa en la Figura 3.2) lo cual es cada $\frac{1}{30}$ segundos. El código 1 del apéndice A (en la pág. 87) muestra

el almacenamiento de la imagen.

El dibujado de la imagen se realizó con OpenGL utilizando la función **glDrawPixels()**, la cual dibuja el arreglo de pixeles indicado. Sin embargo, dado que las matrices en OpenGL se trabajan de forma transpuesta, el arreglo de pixeles es un recorrido transpuesto del buffer de la imagen como se muestra en la Figura 3.3(b).

En el algoritmo 2 se puede observar que el dibujado de la imagen es similar al almacenamiento en los buffers y tiene la finalidad de dibujar la imagen del buffer que ya se terminó de guardar

Algoritmo 2 Algoritmo para el dibujado de la imagen

Entrada: Número de buffer de imagen a dibujar *b_used*, buffer a dibujar $buffDibujar$
Salida: Imagen almacenada en uno de los buffers ($buffer[0]$ o $buffer[1]$)

```
1: Dibujado de imágenes cada 1/30 segundos
2: si b_used == 1 entonces
3:    Bloqueo de variable protegida
4:    Copiar buffDibujar → buffer[1]
5:    Desbloqueo de variable protegida
6: si no
7:    Bloqueo de variable protegida
8:    Copiar buffDibujar → buffer[0]
9:    Desbloqueo de variable protegida
10: fin si
11: Dibuja buffDibujar
```

Cuando el índice *b_used* es 1, la imagen generada se está almacenando en *buffer*[0], en tanto que en *buffer*[1] se encuentra la última imagen completa almacenada (y viceversa cuando *b_used* es 0), la cual se copia a un tercer buffer, que se manda a dibujar con **glDrawPixels()**. El código 2 del apéndice A (en la pág. 87) muestra el almacenamiento de la imagen.

A continuación se detalla la extracción y las características de las imágenes de profundidad, color e infrarrojas.

Para mostrar un ejemplo de los tipos de imagen generados por Kinect, se muestra el área de trabajo del autor en una imagen de profundidad, color e infrarroja en las Figuras 3.4, 3.5 y 3.6 (respectivamente).

Extracción de imágenes de profundidad

Las imágenes de profundidad son generadas de la cámara de profundidad por el nodo **DepthGenerator** y guardadas en el buffer de éste, de donde se toman mediante el mecanismo de doble buffer para posteriormente dibujarlas.

A diferencia de las imágenes de color, donde cada pixel se refiere a un color, en las

imágenes de profundidad cada pixel es un valor entero entre 0 y 10000, que es la distancia en milímetros entre el Kinect y un objeto o persona en la escena que contiene dicho pixel. En el código 3 del apéndice A (en la pág. 88) se muestra la extracción de la imagen de profundidad.

El dibujado de la imagen de profundidad requiere representar los pixeles de profundidad como pixeles de color, por lo que el valor de cada pixel de profundidad se normalizó a 255 (para una imagen de 255 tonos de gris). La Figura 3.4 presenta un ejemplo de una imagen de profundidad.

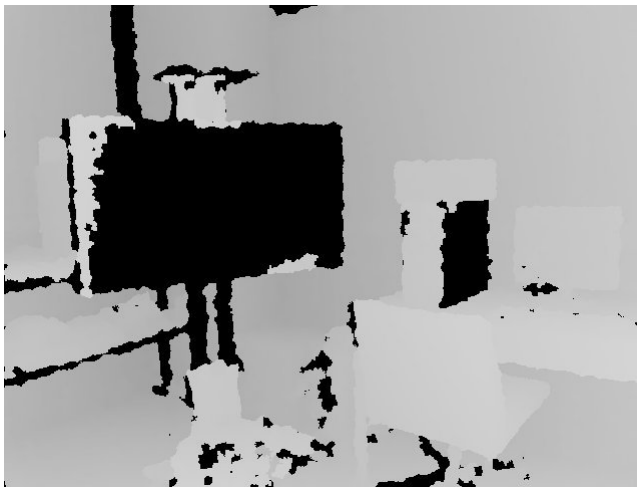


Figura 3.4: Imagen de profundidad del área de trabajo del autor capturada con Kinect

Extracción de imágenes de color

Las imágenes de color son generadas por el nodo **ImageGenerator**. El formato de color de la cámara es RGB24, utilizando un valor entero entre 0 y 255 por cada canal (rojo, verde y azul) del pixel. Dado que las imágenes de color se obtienen distorsionadas (o pixeladas), se tuvo que reemplazar una parte del código de adquisición de imágenes de *SensorKinect* y recompilarlo para eliminar la deformación causada por el filtro de Bayes[1].

En la Figura 3.5 se presenta un ejemplo de una imagen de color capturada con Kinect. En el código 4 del apéndice A (en la pág. 88) se muestra la extracción de la imagen de color.

Extracción de imágenes infrarrojas

Las imágenes infrarrojas son generados por el nodo **IRGenerator** con un formato de color de escala de grises, *i.e.* un entero entre 0 y 255 por pixel.

[1] *Modificación de SensorKinect:* `https://groups.google.com/forum/?fromgroups#!topic/openni-dev/2_HFu47v0NU` y `https://groups.google.com/forum/?fromgroups#!topic/openni-dev/0rpxArKjl3k`

Figura 3.5: Imagen de color del área de trabajo del autor capturada con Kinect

En la Figura 3.6 se presenta un ejemplo de una imagen infrarroja capturada con Kinect. En el código 5 del apéndice A (en la pág. 89) se muestra la extracción de la imagen infrarroja.

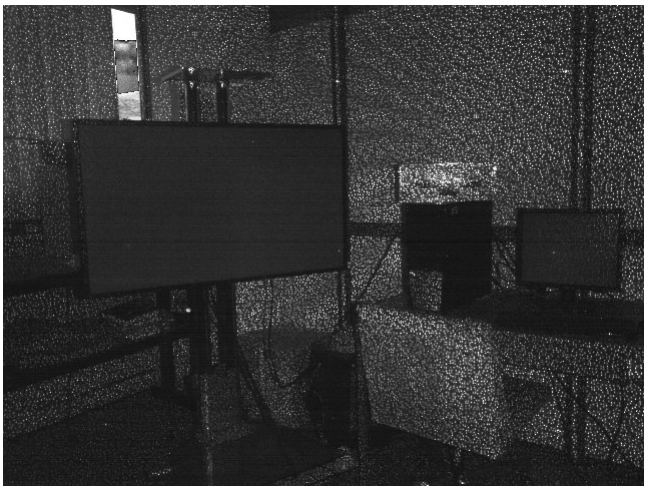


Figura 3.6: Imagen del patrón de puntos infrarrojos del área de trabajo del autor capturada con Kinect

3.1.2. Extracción del esqueleto 3D y 2D

El nodo **UserGenerator** rastrea a las personas que se mueven en la escena y mantiene una lista de los esqueletos 3D estimados a partir de la imagen de profundidad actual. El acceso al esqueleto de cada usuario rastreado se realiza mediante un identificador entero mayor o igual a 0. De forma similar, cada articulación del esqueleto se identifica por una constante entera definida en *OpenNI*. Por ejemplo, el fragmento de código 3.2 es usado para obtener la posición 3D de la cabeza del primer usuario (cuyo id es 0).

Código 3.1: Extracción de articulaciones 3D

```
m_UserGenerator->GetSkeletonCap().GetSkeletonJointPosition(0, XN_SKEL_HEAD, &cabeza3D);
```

El esqueleto 2D se obtiene a partir del esqueleto 3D y la proyección de las articulaciones de éste sobre la imagen de profundidad, para lo que se requiere el nodo **DepthGenerator**. Por ejemplo, para obtener la posición 2D de la cabeza del primer usuario se utiliza la posición 3D de la cabeza, como se muestra en el fragmento de código 3.2.

Código 3.2: Extracción de articulaciones 2D

```
XnPoint3D proj;
m_DepthGenerator->ConvertRealWorldToProjective(1, &cabeza3D, &proj);
cabeza2D = Point2D(proj.X, proj.Y);
```

3.1.3. Dibujado del esqueleto 2D

La forma más simple de dibujar el esqueleto es a base de líneas entre las articulaciones. Para dibujar las líneas se utilizó OpenGL. El dibujado del esqueleto 2D puede realizarse sobre la imagen de profundidad o directamente en un widget o ventana de Qt. El código 6 del apéndice A (en la pág. 89) muestra el método que, al llamarse al final del código 2 (también del apéndice A), dibuja el esqueleto sobre la imagen de profundidad.

La Figura 3.7 muestra un ejemplo de un esqueleto 2D dibujado sobre la imagen de profundidad del usuario levantando la mano izquierda.

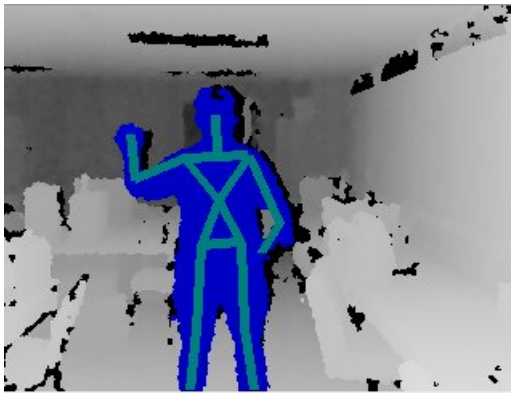


Figura 3.7: Esqueleto 2D sobre una imagen de profundidad

3.1.4. Dibujado del esqueleto 3D

El dibujado del esqueleto en 3D con base en líneas es similar al dibujado del esqueleto en 2D, con la diferencia de que tanto las líneas que unen a las articulaciones como la cámara o punto de vista de OpenGL se encuentran en el espacio tridimensional.

Para ver la escena tridimensional como si se observara desde el Kinect, hay que posicionar una cámara virtual en OpenGL. Por este motivo, se realizó la calibración de la

cámara de profundidad para obtener los parámetros intrínsecos y extrínsecos [40] (utilizando la herramienta RGBDemo[2] [41]) para configurar la matriz M de proyección de OpenGL[3]:

$$M = \begin{bmatrix} \frac{-2 \cdot f_x}{anchoimagen} & 0 & \frac{2 \cdot u_0}{anchoimagen} - 1 & 0 \\ 0 & \frac{2 \cdot f_y}{altoimagen} & \frac{2 \cdot v_0}{altoimagen} - 1 & 0 \\ 0 & 0 & -\frac{z_{Far}+z_{Near}}{z_{Far}-z_{Near}} & \frac{-2 \cdot z_{Far} \cdot z_{Near}}{z_{Far}-z_{Near}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

La Figura 3.8 muestra una fotografía del sistema dibujando el esqueleto 3D del autor levantando la mano izquierda, también se muestra la malla de acciones en 2D y 3D (que se explicara en la sección 3.2.1, en la pág. 40).

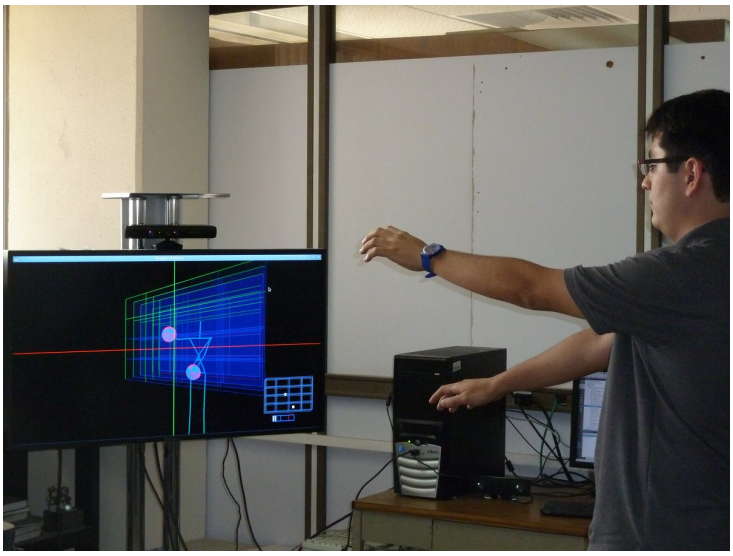


Figura 3.8: Esqueleto 3D y malla de acciones

La malla de acciones 3D se forma por el conjunto de líneas horizontales y verticales frente al esqueleto. La malla en 2D es el rectángulo subdividido de la derecha. Los cuadrados negros pintados dentro de la malla 2D indican la posición de las manos dentro de la malla. El rectángulo dividido con líneas verticales que se encuentra en la parte inferior derecha de la imagen (cada línea corresponde a un plano Z_n, donde Z_0 la primer línea de la derecha) muestra la profundidad de las manos (indicada por las líneas gruesas) dentro de la malla.

[2] *RGBDemo* es una herramienta de uso libre basada en *libfreenect* y *OpenNI*, que entre sus funcionalidades cuenta con la calibración de la cámara de profundidad de Kinect.

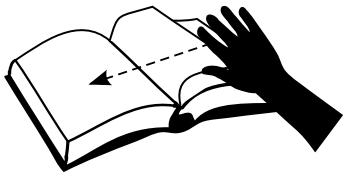
[3] La matriz de proyección se obtuvo de la función *cvPOSIT* de OpenCV. Artículo en `http://opencv.willowgarage.com/wiki/Posit`.

3.2. Traducción de movimientos del cuerpo a comandos de interacción

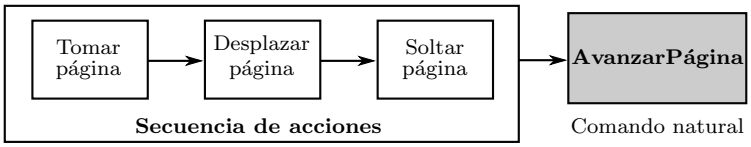
La interacción con la computadora se realiza dando órdenes o *comandos* en la interfaz. En la interfaz de línea de comandos, estasórdenes son palabras claves con opciones y parámetros, en tanto que en la interfaz gráfica las órdenes se presentan como clicks de ratón sobre elementos de la interfaz y atajos del teclado. En los dispositivos móviles las órdenes se realizan al tocar la pantalla, cambiar la orientación del dispositivo o mediante el reconocimiento de comandos de voz.

Cuando hablamos de interacción con objetos físicos, *e.g.* al cambiar a la siguiente página de un libro resulta obvio que se debe tomar la página de la derecha y llevarla a la izquierda como en la Figura 3.9(a). Sin embargo, para llegar al mismo objetivo en un visor de documentos el usuario debe saber, con anterioridad, que requiere presionar un botón o utilizar el ratón en la barra de desplazamiento del visor de documentos, lo cual no resulta *tan natural* como pasar a la siguiente página con la mano.

De forma similar a la que una secuencia de movimientos nos permite manipular objetos en la vida diaria, en la interfaz de usuario desarrollada una secuencia de movimientos específicos de las manos, que denominamos **secuencia de acciones**, lleva a la identificación y ejecución de un **comando de interacción**.



(a) Avanzar a la siguiente página de un libro



(b) Avanzar a la siguiente página de un documento en la interfaz natural de usuario

Figura 3.9: Secuencia de acciones para definir un comando de interacción

En la Figura 3.9(b) se presenta el comando **AvanzarPágina**, mismo que se define por la secuencia de acciones *Tomar página*, *Desplazar página* y *Soltar página*. Para realizar este comando se debe mover la mano derecha comenzando al frente del cuerpo y desplazarla a la izquierda (como se muestra en el lado derecho de la Figura 3.10).

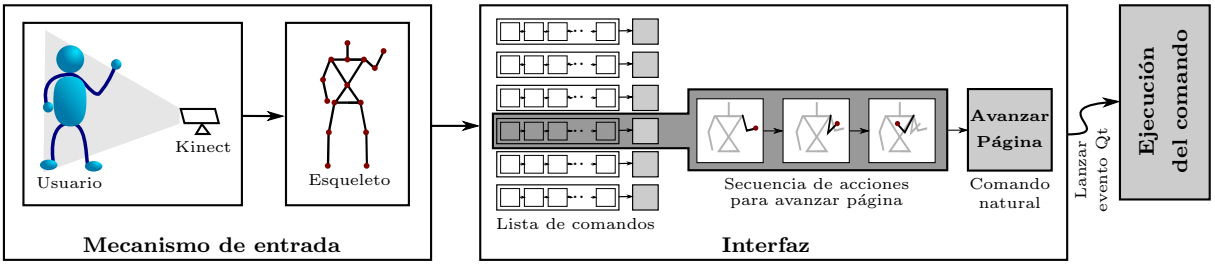


Figura 3.10: Diagrama de detección del comando **AvanzarPágina**[4]

La interfaz natural desarrollada funciona detectando **acciones** (movimientos de las manos) e identificando si dichas acciones permiten realizar una transición en alguna de las secuencias de acciones que definen a un comando.

La Figura 3.10 presenta un diagrama general para la detección del comando **AvanzarPágina** (que es uno de los comandos en la lista de comandos a detectar) y muestra las partes de la interfaz natural involucradas en dicho fin. Para detectar las acciones se utiliza el rastreo del esqueleto del usuario y la **malla de acciones**. Cuando el comando es detectado se lanza un evento de Qt notificando la ocurrencia del mismo y posteriormente se ejecuta la implementación asignada a éste.

Para explicar el funcionamiento de la **detección de comandos** se presenta el diagrama de la Figura 3.11. La lista de comandos a detectar se especifica en el archivo de lista de comandos; posteriormente se inicializa la interfaz natural y se ingresa a un ciclo de procesamiento en el que se obtiene el esqueleto de n; a partir de éste se detectan las acciones a_{d_n} y a_{i_n} (que corresponden a la acción detectada con la mano derecha e izquiera para el esqueleto de la iteración n del ciclo); estas acciones pueden permitir iniciar, continuar o finalizar alguna de las secuencias que definen a un comando. En caso de detectar un comando, *i.e.* que a_{d_n} y a_{i_n} permitan una transición al final de una secuencia de acciones, se lanza un evento de Qt con el identificador de dicho comando y cuando el evento es atendido se ejecuta la función asignada al comando. Finalmente se puede reproducir un sonido como retroalimentación para notificar al usuario que se ha ejecutado un comando.

En las siguientes secciones se explica a detalle cada etapa del flujo de la interfaz natural (de la Figura 3.11). Se optó por explicar primero la malla de acciones, dado que es la base del funcionamiento de las demás etapas.

3.2.1. Malla de acciones

La malla de acciones establece un entorno virtual de interacción que permite sensar e interpretar los movimientos que el usuario realice con sus manos, *i.e.* actúa como un panel de control con instrumentos, como palancas o manijas, que siempre se encuentra frente al usuario y donde el comportamiento de cada instrumento (o comando) se define mediante una secuencia de acciones. En la Figura 3.8 (en la pág. 38) se muestra la malla de acciones en el esqueleto del usuario.

La malla está formada por un conjunto de planos sobre los ejes del sistema de coordenadas S_m de la malla (5 en X_m, 5 en Y_m y 5 en Z_m), donde el origen del sistema O_m se encuentra en la intersección de los planos H_2, V_2 y Z_1 (ver las Figuras 3.12 y 3.13).

Se tienen dos sistemas de coordenadas. El primero es el global $X_gY_gZ_g$, cuyo origen está dado por el Kinect viendo hacia ^+Z_g y el segundo sistema $X_mY_mZ_m$ está dado por malla de acciones. El eje Z_m de la malla de acciones se encuentra sobre el vector normal $\vec{n}$ al plano del pecho del esqueleto, mismo que se forma por las articulaciones *hombro_derecho*, *torso*

[4]El esquema del esqueleto se muestra en espejo.

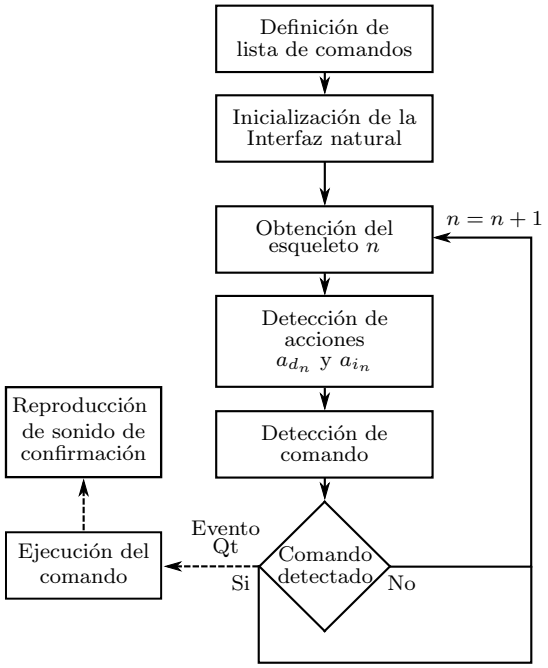


Figura 3.11: Flujo de control de la detección de comandos

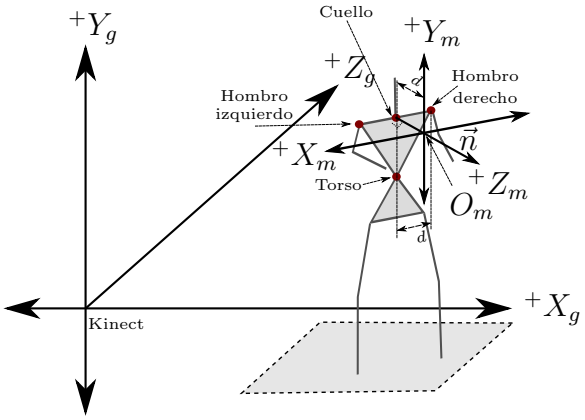


Figura 3.12: Sistema de coordenadas de la malla de acciones

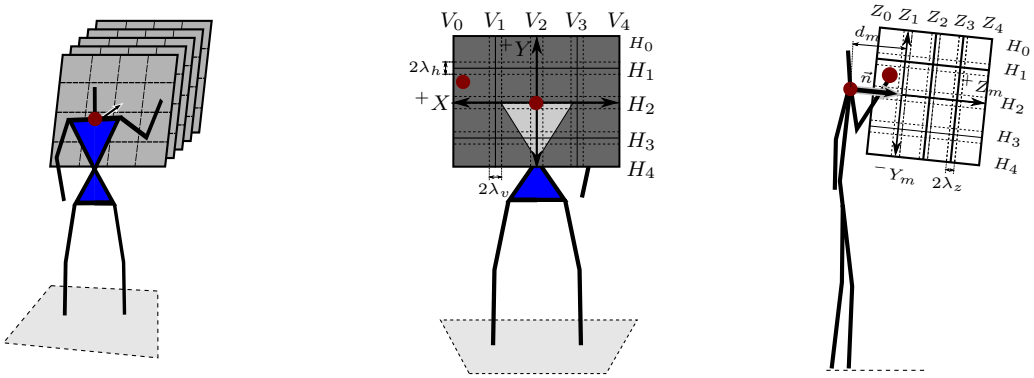
y *hombro_izquierdo*. El origen de la malla (O_m) se localiza a una distancia d sobre $\vec{n}$ a partir de la articulación *cuello*.

La malla y el pecho del esqueleto siempre son paralelos, lo que permite que *la detección de acciones se adapte a la posición en la que se encuentre el usuario*, es decir, el usuario puede caminar, inclinarse o girar dentro del campo de vista de Kinect y siempre tendrá la malla frente a él. Como se observa en la Figura 3.12, las acciones son detectadas cuando el desplazamiento de las manos es al frente del usuario y no necesariamente apuntando a Kinect.

Las distancias entre los planos y la longitud de los umbrales de la malla se presentan en la tabla 3.1. Estas distancias fueron tomadas con mediciones experimentales de tal forma que resultara cómodo (*i.e.* sin estiramientos bruscos o exagerados de los brazos)

e intuitivo al usuario desplazar sus manos entre las celdas. Para hacer esto posible, se experimentó y seleccionó la distancia euclidiana[5] $d = dist(h_d, c)$ entre el hombro derecho y el cuello del esqueleto para adaptar la malla al cuerpo de cada usuario normalizando por d las dimensiones de ésta, lo que significa los movimientos se "sienten" igual para cualquier usuario.

Con el objetivo de evitar ambigüedades cuando las manos se encuentren entre dos o más celdas adyacentes, se agregó un umbral de separación de $2\lambda_h$, $2\lambda_v$ y $2\lambda_z$. De esta forma, las celdas se encuentran delimitadas por los umbrales y no por los planos (como se muestra en la Figura 3.13).



(a) Vista trasera (b) Vista frontal (c) Vista lateral

Figura 3.13: Malla de acciones

Puntos de referencia	**Distancia**
$dist(Hombro\_derecho, Cuello)$	d
$dist(Cuello, Z_1)$	d
$dist(H_0, H_1) = dist(H_3, H_4)$	$1.5d$
$dist(H_1, H_2) = dist(H_2, H_3)$	$1.5d$
$dist(V_0, V_1) = dist(V_3, V_4)$	$2.1d$
$dist(V_1, V_2) = dist(V_2, V_3)$	$2.5d$
$dist(Z_0, Z_1)$	d
$dist(Z_1, Z_2)$	$1.6d$
$dist(Z_2, Z_3)$	$0.8d$
$dist(Z_3, Z_4)$	$0.8d$
λ_h	$3cm$
λ_v	$3cm$
λ_z	$3cm$
Origen $O_m = (0, 0, 0)$	$O_m = (H_2, V_2, Z_1)$

Tabla 3.1: Tabla de dimensiones de la malla de acciones

[5] *Distancia euclidiana en* $\Re^3$ $dist(p_1, p_2) = \sqrt{(p_{1_x} - p_{2_x})^2 + (p_{1_y} - p_{2_y})^2 + (p_{1_z} - p_{2_z})^2}$

Las intersecciones entre los planos forman un conjunto de $4^3 = 64$ celdas en cuatro niveles de profundidad (entre Z_0 y Z_1, entre Z_1 y Z_2, entre Z_2 y Z_3 y entre Z_3 y Z_4). Cada celda de la malla se identifica mediante una etiqueta única en tanto que la unión de celdas se identifica por una etiqueta calculada a partir de las etiquetas de las celdas involucradas en la unión.

Cada celda fue etiquetada con un número binario a de 12 bits usando parte del algoritmo de recortado de líneas de Cohen-Sutherland [38], como se muestra en la Figura 3.14. Los 12 bits se dividieron en 3 secciones $a = \{a_0, a_1, a_2\}$ de 4 bits, donde a_0 identifica a la celda en Z_m, a_1 en el eje X_m y a_2 en el eje Y_m. En cada sección de a los 4 bits valen 0 excepto el que indica la profundidad, columna o fila de la celda en cuestión, de tal forma que solo un bit por sección vale 1 y a solo contiene 3 bits con valor 1. a en decimal es el entero que etiqueta a la celda.

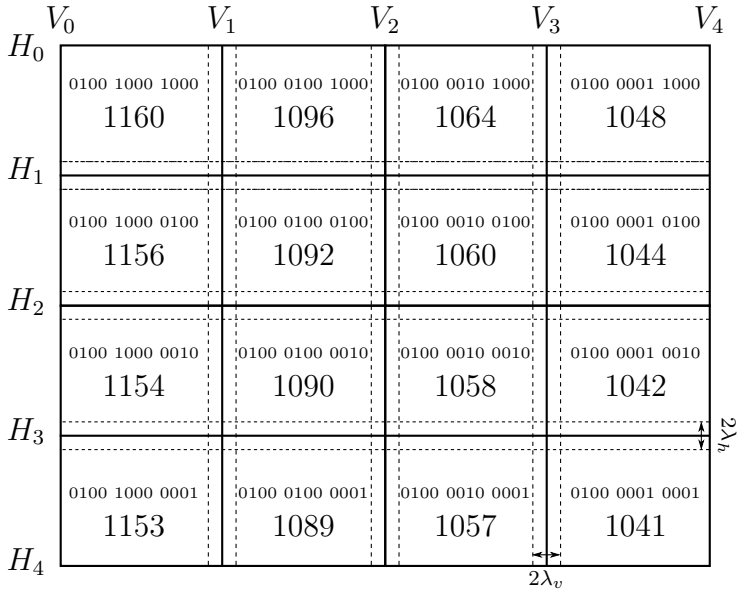
La Figura 3.15 muestra los 64 códigos de las celdas de la malla de acciones.

a de 12 bits:

1	2	3	$\cdots$	10	11	12

Bit	El bit vale 1 si la celda
1	Entre Z_0 y $Z_1 - \lambda_z$
2	Entre $Z_1 + \lambda_z$ y $Z_2 - \lambda_z$
3	Entre $Z_2 + \lambda_z$ y $Z3$
4	Entre $Z_3 + \lambda_z$ y $Z4$
5	Entre V_0 y $V_1 - \lambda_v$
6	Entre $V_1 + \lambda_v$ y $V_2 - \lambda_v$
7	Entre $V_2 + \lambda_v$ y $V_3 - \lambda_v$
8	Entre $V_3 + \lambda_v$ y $V_4 - \lambda_v$
9	Entre H_0 y $H_1 - \lambda_h$
10	Entre $H_1 + \lambda_h$ y $H_2 - \lambda_h$
11	Entre $H_2 + \lambda_h$ y $H_3 - \lambda_h$
12	Entre $H_3 + \lambda_h$ y $H_4 - \lambda_h$

(a) Tabla de codificación de celdas



(b) Codificación de celdas correspondientes al segundo nivel de profundidad (entre $Z_1 + \lambda_z$ y $Z_2 - \lambda_z$).

Figura 3.14: Codificación de la malla de acciones

Cálculo de las coordenadas de la manos con respecto a la malla de acciones

Las coordenadas $\mathbf{m}_s(m_x, m_y, m_z)$ de las manos del esqueleto se encuentran en el sistema de coordenadas global S_g, para obtener las coordenadas $\mathbf{m}_m$ de las manos con respecto al sistema S_m de malla de acciones, mismas que se utilizan en la detección de acciones. La transformacón geométrica de las manos en S_g a S_m es:

$$\mathbf{m}_{g_n} = R_x(\phi_n) \cdot R_y(\theta_n) \cdot T_{xyz}(-\vec{n}_{x_n}, -\vec{n}_{x_n}, -\vec{n}_{x_n}) \cdot \mathbf{m}_{s_n}$$

2184	2120	2088	2072
2180	2116	2084	2068
2178	2114	2082	2066
2177	2113	2081	2065

(a) Celdas entre Z_0 y $Z_1 - \lambda_z$

1160	1096	1064	1048
1156	1092	1060	1044
1154	1090	1058	1042
1153	1089	1057	1041

(b) Celdas entre $Z_1 + \lambda_z$ y $Z_2 - \lambda_z$

648	584	552	536
644	580	548	532
642	578	546	530
641	577	545	529

(c) Celdas entre $Z_2 + \lambda_z$ y $Z_3 - \lambda_z$

392	328	296	280
388	324	292	276
386	322	290	274
385	321	289	273

(d) Celdas entre $Z_3 + \lambda_z$ y Z_4

Figura 3.15: Códigos de las 64 celdas de la malla de acciones (en base 10) divididas en 4 niveles de profundidad

donde $\vec{n}_n$ es la normal al plano del pecho del esqueleto n, ϕ_n y θ_n los ángulos de rotación de $\vec{n}_n$ con respecto a los ejes X_g y Y_g (respectivamente).

Los ángulos ϕ y θ se obtienen a partir del producto punto de $\vec{n}$ y un vector unitario $\vec{v}$ y mediante :

$$\vec{v'} = \vec{n}_n \cdot \vec{v}$$

$$\beta = \arctan 2(\vec{v'}_z, \vec{v'}_x) \cdot \frac{180}{\pi}$$

$$\phi = \beta - 90$$

$$\theta = \arctan 2(\vec{v'}_y, \sqrt{\vec{v'}_x^2 + \vec{v'}_z^2}) \cdot \frac{180}{\pi}$$

3.2.2. Definición de lista de comandos

Se nombró como *definición de lista de comandos* a la etapa en la que se establecen las secuencias de acciones para cada comando de interacción que se desee detectar con la interfaz natural.

En la Figura 3.16 se presentan las fases involucradas en esta etapa, las cuales se describen a continuación.

Definición de cada comando en la lista de comandos

La lista de comandos es un archivo de texto plano donde cada línea $c_0, c_1, \cdots, c_n$ define un comando c_i. Cada comando c_i se define por una secuencia o lista de m acciones $a_0, a_1, \cdots, a_m$ separadas por "," de la acciones de la malla (ver Figura 3.15). El comando c_i queda definido por:

$$c_i = a_0, a_1, \cdots, a_m$$

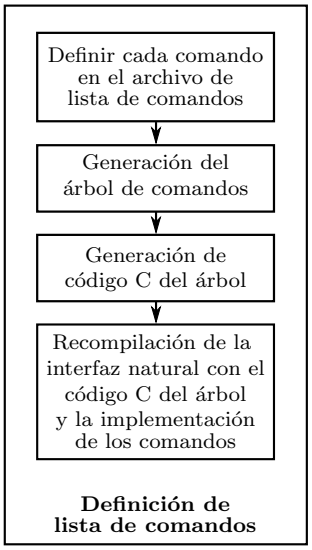


Figura 3.16: Diagrama de control de flujo de la definición de lista de comandos

donde cada $a_j, j = 1 \cdots m$ es el código de una acción (o la unión de p acciones), seguido por el identificador de la mano (0 para la derecha y 1 para la izquierda) con la que se desea detectar la acción, de tal forma que cada acción a_j se define por:

$$a_j = a_{j_0}||a_{j_1}|| \cdots ||a_{j_p}@id\_mano \tag{3.1}$$

De acuerdo a la notación expuesta se diseñaron tres tipos de acciones, mismos que son la base de la definición y detección de comandos:

1. **Acción simple**. Involucra una sola celda para una mano, *i.e.* $p = 1$.
2. **Acción compuesta**. Involucra una unión de celdas para una mano, *i.e.* $p > 1$.
3. **Acción doble**. Involucra una acción simple o compuesta para cada mano.
   Para detectar un movimiento con ambas manos se utiliza el indicador && entre una acción a_d para la mano derecha y una acción a_i para la mano izquierda. Éstas pueden ser simples o compuestas, dando la siguiente estructura a una acción doble a_j:
   $$a_j = a_d \&\& a_i$$

Para ejemplificar la notación de los comandos, se muestra la definición del comando *doble click izquierdo* de la aplicación experimental ratón virtual que se realiza al mover la mano izquierda hacia el frente (ver la tabla 4.1, en la pág. 60 y la tabla 3.15, en la pág. 44). El comando se define por la siguiente secuencia de dos acciones $c = a_0, a_1$:

1160||1096||1156||1092@1, 648||584||644||580@1

donde a_0 = 1160||1096||1156||1092@1 es la primer acción compuesta por la unión de las celdas 1160, 1096, 1156, y 1092 para la mano izquierda (que se indica por @1); a_1 = 648||584||644||580@1 es la segunda acción y está compuesta por la unión de las celdas 648, 584, 644 y 580 también para la mano izquierda. El comando se realiza al mover la mano izquierda hacia el frente (a la altura de la cara).

Generación del árbol de comandos

A partir de la lista de comandos se genera un árbol en el que cada recorrido desde la raíz a una hoja i describe la secuencia de acciones para el comando c_i, *i.e.* los nodos interiores del árbol guardan los códigos de las acciones $a_0, a_1, \cdots, a_m$ y los nodos hoja almacenan el identificador i del comando c_i.

Si a_i es una acción simple, el código de verificación $v = a$. En caso de que a_i sea una acción compuesta, a_i se reemplaza por la máscara de la acción y se calcula el código de verificación v. Si a_i es una acción doble entonces contiene un par de máscaras y un par de códigos de verificación (un par para la mano izquierda y el otro para la derecha). En la sección 3.2.2 se explica la obtención de la máscara y el código de validación de la acción compuesta.

La estructura de los nodos (Figura 3.17) fue diseñada para facilitar la generación de la máquina de estados del árbol de comandos en código C (que se explica en la sección 3.2.2, en la pág. 50). El nodo contiene listas de componentes (para poder almacenar acciones simples (o compuestas) y dobles en un mismo tipo de nodo). Los elementos del nodo son:

1. Tipo de acción.
   Vale 0 si la acción que almacena el nodo es para la mano derecha, 1 si es para la izquierda y 2 si es una acción doble.

2. Una lista de máscaras a.
   Si la acción es simple o compuesta, la lista solo contiene una máscara. Si la acción es doble, la lista contiene dos máscaras (una para la mano derecha y otra para la izquierda).

3. Una lista de códigos de verificación v.
   Al igual que la lista de máscaras el número de códigos de verificación en la lista está en función del tipo de acción que almacena el nodo.

4. Nodos hijos.
   Los nodos hijos se almacenan en tres listas de acuerdo al tipo de acción que almacenan
   Los nodos hijos simples y compuestos están ordenados de acuerdo a la mano que representan y para cada mano, están ordenados en forma creciente por el valor de la máscara.

   - Lista de nodos de acciones simples.
     Contiene p nodos de acciones simples de mano derecha y q nodos de acciones simples de mano izquierda.
   - Lista de nodos de acciones compuestas.
     Contiene r nodos de acciones compuestas de mano derecha y s nodos de acciones compuestas de mano izquierda.
   - Lista de nodos de acciones dobles.
     Contiene t nodos de acciones dobles.

5. Identificador de comando.
   Si el nodo es hoja el identificador es un entero mayor que cero, de lo contrario vale -1.

6. Nodo padre.
   El padre de la raíz del árbol es nulo.

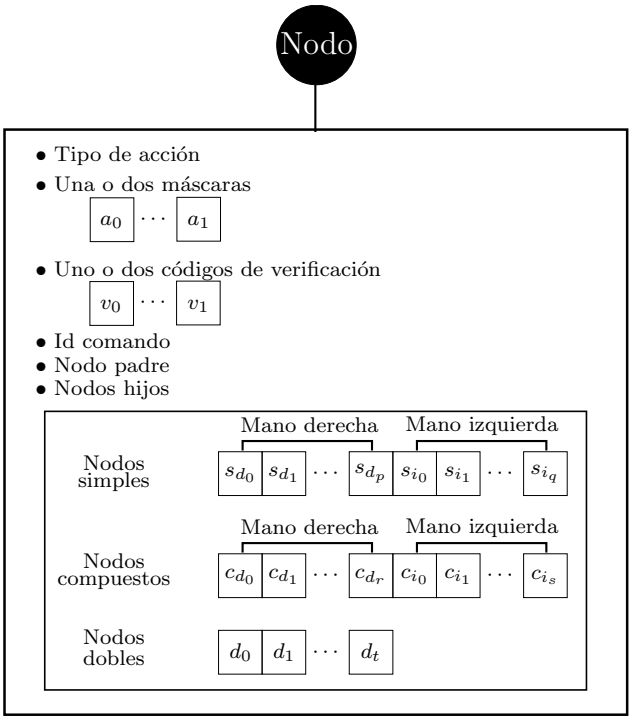


Figura 3.17: Estructura del nodo del árbol de comandos

Los algoritmos 3 y 4 presentan la generación del árbol de comandos.

Algoritmo 3 Construcción y generación del árbol de comandos

Entrada: Archivo de lista de comandos.
Salida: Construcción del árbol de comandos en memoria.

```
1: nraíz es la raíz del árbol
2: Encolar las líneas del archivo en ColaLineas
3: mientras ColaLineas.numLineas > 0 hacer
4:     cadComando ← ColaLineas.desencolar()
5:     Guardar en ColaAcciones las subcadenas separadas por comas de cadComando
6:     insertarComando(nraíz, ColaAcciones) //algoritmo 4.
7: fin mientras
```

Cálculo de la máscara y el código de verificación de una acción compuesta

Para identificar la acción $a_j = a_{j_0}||a_{j_1}||\cdots||a_{j_p}@id\_mano$, compuesta por un grupo de p celdas (o subacciones) y almacenarla como un sólo entero a_j en lugar de una lista de

Algoritmo 4 Insertar un comando al árbol de comandos

Entrada: *nraíz* la raíz del árbol y *ColaAcciones* la secuencia de acciones del comando a insertar.
Salida: El comando insertado y ordenado en el árbol

1: **mientras** *ColaAcciones*.numAcciones > 0 **hacer**
2: *acción* $\leftarrow$ *ColaAcciones*.desencolar()
3: *nodo* $\leftarrow$ construirNodo(*acción*) //Genera el nodo de acuerdo al tipo de acción, calcula la(s) máscara(s) y el (los) códigos de verificación. Establece el tipo del nodo y el identificador de comando en -1
4: **si** *ColaAcciones*.numAcciones $== 0$ **entonces**
5: *nraiz.idComando* $\leftarrow i$ //El padre de *nodo* (el cual es *nraiz*) es la últma acción del comando c_i.
6: **regresar**
7: **fin si**
//Insertar *nodo* en *nraíz* se acuerdo al tipo de acción y de la mano que la define.
8: **si** el nodo almacena una acción doble **entonces**
9: Insertar *nodo* al final de la lista de nodos dobles.
10: **regresar**
11: **fin si**
12: Búsqueda binaria del primer nodo cuyo valor de máscara sea mayor o igual a *nodo.mascaras[0]*. La búsqueda se realiza en la sección de la misma mano de *nodo* de la lista de los hijos de *nraíz* que son del mismo tipo que *nodo*.
13: **si** se encontró un nodo *nod* con la misma máscara **entonces**
14: *nod.idComando* $\leftarrow i$
15: **regresar**
16: **si no**, **si** se encontró un nodo *nod* con valor de máscara mayor **entonces**
17: Insertar *nodo* a la izquierda de *nod*
18: **si no**
19: Insertar *nodo* al final de la lista de nodos del mismo tipo de *nodo* de *nraíz*
20: **fin si**
21: El padre de *nodo* es *nraíz*
22: *nraíz* $\leftarrow$ *nodo*
23: **fin mientras**

enteros, se calcula: (1) una máscara (que reemplazará al código de una acción simple en el árbol de comandos) y (2) un código de verificación, el cual permite saber si una celda pertenece a la unión de celdas.

Para generar la **máscara** a_j, se obtiene el binario de 12 bits de las p celdas que forman la unión y se colocan en una tabla, si todos los bits de la columna i tienen el mismo valor, se asigna el valor 1 al bit i de la máscara, de lo contrario se asigna 0. El resultado después de recorrer todas las columnas es la máscara de la acción compuesta.

La máscara identifica a la unión de celdas, sin embargo ésta no identifica únicamente a la unión con la que fue calculada, *i.e.* la obtención de la máscara a para cualquier otra

unión de celdas en el mismo nivel de profundidad de la malla de acciones dará como resultado la misma máscara a, por lo que se requiere una segunda validación para asegurar si una celda pertenece o no a la unión.

El **código de verificación** v_j se calcula realizando la operación lógica AND entre la máscara y cualquiera de los p códigos de las celdas de la unión. El resultado de la operación $v = a_j \ AND \ a_{j_i}$ siempre es igual a v_j para $i = 0, 1, \cdots, p-1$, *i.e.* para cualquier otra celda fuera de la unión sucede que $v \neq v_j$, lo cual finalmente permite saber si una celda se encuentra o no en la unión de celdas.

La Figura 3.18 presenta un ejemplo de la obtención de la máscara a y el código de verificación v para la acción 1160||1096||1156||1092@1 y el código de validación v para la misma. Lo que se almacena en el nodo del árbol de comandos que representa a esta acción es la máscara $a = 3891$ (que tomará el lugar de una acción simple) y el código de verificación $v = 1024$.

1160||1096||1156||1092@1

(a) Acción formada por una unión de celdas

1160	010010001000
1096	010001001000
1156	010010000100
1092	010001000100
Máscara	$111100110011_2 = 3891_{10}$
Cod. verificación	$010000000000_2 = 1024_{10}$

(b) Obtención de la máscara y el código de verificación para una acción compuesta

Figura 3.18: Ejemplo de obtención de máscara y código de verificación

Representación del árbol de comandos en texto

La generación del árbol de comandos también entrega una representación del recorrido por niveles de éste en una línea de texto $t = n_0, n_1, \cdots, n_p$. Cada subcadena n_i, con $i = 1, \cdots p$ representa un nodo del árbol y tiene la siguiente estructura:

$$n_i = m_d \# v_d @1 : num\_nodos\_hijos \& \& m_i \# v_i @0 : num\_nodos\_hijos, \ < id\_comando >$$

donde el subíndice $_d$ y el identificador @1 se refieren a la mano derecha, el subíndice $_i$ y el identificador @0 a la mano izquierda. Los elementos de n_i son:

- m es la máscara de la unión de celdas del nodo (o un solo código si es una celda).
- v es el código de verificación para la máscara m.
- *num_nodos_hijos* es el número de nodos hijos del nodo n_i, *i.e.* indica el número de subcadenas n separadas por comas de t, que representan a los nodos hijos de n_i.
- && es usado si la acción es de tipo doble.

- , $< id\_comando >$ indica si el nodo es hoja, *i.e.* $num\_nodos\_hijos : 0$, en tal caso *id_comando* es el número del comando de la lista de comandos. Si el nodo no es hoja, la sección de código "$, < id\_comando >$" no existe.

n_0 representa la raíz del árbol con el texto $RAÍZ{:}num\_nodos\_hijos$.

Un ejemplo de la interpretación del texto para un árbol se presenta en la Figura 4.3 (en la pág. 61).

Generación de código fuente C del árbol de comandos

El árbol de comandos se procesa para convertirlo en una máquina de estados en código C para detectar si las acciones a_{d_n} y a_{i_n} que se obtienen durante la **detección de acciones** permiten iniciar, continuar o finalizar alguna de las secuencias de acciones que definen a un comando[6].

La máquina de estados está basada en una estructura *switch* que identifica al estado (o acción) actual q entre los estados de la máquina y estructuras *if* que evaluán la transición al segundo estado q' (el cual se indica mediante las acciones a_{d_n} y a_{i_n}), de acuerdo a las posibles rutas entre los estados de la máquina (*i.e.* las rutas entre nodos del árbol de comandos).

Las transiciones toman en cuenta los valores de a_{d_n} y a_{i_n}, dado que si una mano se encuentra fuera de la malla de acciones, el valor de la acción detectada con dicha mano es 0, (o -1 si la mano se encuentra en los umbrales de la malla). Los tipos de transiciones se explican a continuación:

1. **Transición simple**.
   Cuando una mano se encuentra fuera de la malla, para verificar la transición sólo se considera la acción generada por la mano que está en la malla. Mientras ambas manos estén fuera de la malla la máquina de estados no avanza.

2. **Transición ignorada**.
   Si el valor de las acciones detectadas es el mismo que se detectó en la iteración anterior, la máquina no cambia de estado. Si la acción indica que una o ambas manos están en los umbrales de la malla de acciones la máquina espera en el último estado válido.

3. **Transición doble**.
   Cuando ambas manos están en la malla, la transición toma en cuenta a_{d_n} y a_{i_n} para detectar acciones dobles.

La generación de la máquina de estados del árbol de comandos permite saber con pocas comparaciones lógicas si las acciones a_{d_n} y a_{i_n} permiten a q realizar una transición de estados, *i.e.* el mismo comportamiento que se tendrá al buscar las acciones entre los nodos hijos del nodo en el que se encuentre q, pero eliminando la necesidad de aplicar un algoritmo de

[6]En esta sección se considera de que cada uno de los p nodos rama n_i con $i = 0, 1, \cdots, p-1$ del árbol de comandos, es un *estado* q_i en la máquina de estados generada.

búsqueda en el árbol en memoria, lo que repercute en un menor tiempo de procesamiento.

La generación de código se explica en el algoritmo 5 (en la pág. 57). Se inicia identificando los p nodos rama del árbol de comandos (incluyendo la raíz) al mapear el árbol a una lista $A[p]$ con p apuntadores a nodo. Para esto, el árbol de comandos se recorre por niveles, iniciando en la raíz con el índice $i = 0$ y finalizando con $i = p - 1$, para que cada $A[i]$ apunte al nodo rama n_i.

El resultado del mapeo es la lista A de los p estados q_i de la máquina de estados, éstos se encuentran ordenados por cada nivel del árbol (y ordenados de izquierda a derecha con respecto a dicho nivel).

Para construir la estructura *switch* con los p casos que ordenan a los p estados q_i, se utilizó una lista $S[p]$ con p listas de cadenas de caracteres donde el bloque de código del caso i, que evalúa al estado q_i, se encuentra en $S[i]$. Para generalizar un ejemplo, el bloque con l líneas de código para el caso i es:

$$S[i][0] = \text{“}case\ q_i : \text{''}$$

$$\vdots$$

$$S[i][l-1] = \text{“}break; \text{''}$$

Entonces se procede a codificar las transiciones entre nodos del árbol, mismas que son las transiciones de la máquina de estados. Las transiciones se obtienen al mapear los estados q_i (*i.e.* los nodos hoja n_i) a k estructuras *if*, donde k es el número total de hijos del nodo q_i. **Cada bloque *if* permite realizar la transición entre el estado q_i y el estado relacionado a uno de sus k hijos** al comparar la máscara y el código de verificación entre éste y las acciones a_{d_n} y a_{i_n} (que son los parámetros de la máquina de estados). Los nodos hijos del estado q_i se reagrupan en: (1) acciones para la mano derecha, (2) acciones para la mano izquierda y (3) acciones para ambas manos. Formando tres sub-bloques de estructuras *if* y *else-if* en el caso i del *switch* para evaluar las posibles transiciones para cada mano.

$S[i][0] = \text{“}case\ q_i : \text{''}$
$if(a_{d_n} != 0\ \&\&\ a_{i_n} == 0)\ \{$
    $//Bloque\ de\ transiciones\ de\ la\ mano\ derecha$
$\}\ else$
$if(a_{d_n} == 0\ \&\&\ a_{i_n} != 0)\ \{$
    $//Bloque\ de\ transiciones\ de\ la\ mano\ izquierda$
$\}\ else$
$if(a_{d_n} != 0\ \&\&\ a_{i_n} != 0)\ \{$
    $//Bloque\ de\ transiciones\ de\ ambas\ manos$
$\}\ //Fin\ del\ caso\ i$
$S[i][l-1] = \text{“}break; \text{''}$

Para ejemplificar las sentencias *if* generadas, se retoma la acción 1160||1096||1156||1092@1 para la mano izquierda, cuya máscara es 3891 y código de validación es 1024 de la Figura 3.18 (en la pág. 49) considerando que la transición lleva a q_1. El *if* generado se encuentra en el sub-bloque *Bloque de transiciones de la mano izquierda* y en código C es:

```
switch(estado) {
    ...
    //Bloque de transiciones de la mano izquierda
    if( (acciones[0].act_code&3891)==1024 ) {
        estado = q1;
    }
    ...
}
```

Retomando el ejemplo anterior, si la acción fuese 1160||1096||1156||1092@0 && 160||1096||1156||1092@1 para ambas manos el *if* generado sería:

```
switch(estado) {
    ...
    //Bloque de transiciones de ambas manos
    if( (acciones[0].act_code&3891)==1024 && (acciones[1].act_code&3891)==1024) {
        estado = q1;
    }
    ...
}
```

Finalmente cada línea de código de la máquina de estados se encuentra en S, y al imprimir todas las cadenas de caracteres se obtiene el código completo.

En los códigos 7, 8 y 9 del apéndice A (en las págs. 90, 91 y 92) se muestra la máquina de estados de las aplicaciones de prueba desarrolladas en este trabajo de tesis.

Recompilación del código fuente de la interfaz natural

El código C de la máquina de estados del árbol se agrega al código fuente de la interfaz natural para ser recompilado.

3.2.3. Detección de acciones

Detectar acciones se refiere a notificar las celdas de la malla de acciones en las que se encuentran manos para el esqueleto n, *i.e.* se regresa el código (de la lista de la Figura 3.15) de la celda en la que se encuentre la mano derecha y la mano izquierda. Si alguna de las manos se encuentra fuera de la malla la acción vale 0, pero si la mano está dentro de la malla, pero se encuentra entre los umbrales de separación el valor de la acción es -1.

El algoritmo 6 (en la pág. 58) presenta la detección de acciones.

3.3. Filtrado de las manos en la malla de acciones

El rastreo del esqueleto provee las articulaciones (coordenadas en $\Re^3$) del esqueleto virtual del usuario en cada imagen de profundidad generada con Kinect, sin embargo existe ruido en este proceso al presentar variaciones significativas en los valores de las

articulaciones entre imagen e imagen resultando una continua vibración del esqueleto, incluso si el usuario se mantiene quieto. Esto se debe al hecho de que Kinect no es un sensor 3D, si no que las posiciones de las partes del cuerpo asociadas al esqueleto se estiman a partir de cada imagen de profundidad.

Debido a la presencia de ruido se implementó un filtro de Kalman [42] para suavizar las coordenadas de las manos con respecto a la malla de acciones.

El filtro de Kalman es un algoritmo que utiliza una serie de mediciones (ruidosas) de un proceso en tiempo discreto para producir una salida más precisa que la que se obtiene únicamente con la medición del proceso. El filtro de Kalman opera de forma *recursiva*[7] sobre los datos ruidosos de la lectura del proceso para producir una estimación óptima del estado del sistema, *i.e.*, refina la medición del sistema mediante un modelo de predicción-corrección.

La implementación del filtro se presenta en la Figura 3.19. La medición z_k se refiere a una de las coordenadas de cada mano del esqueleto virtual, *i.e.* el filtro se aplica por cada coordenada de cada mano con respecto a la malla de acciones resultando en la coordenada suavizada $\hat{x}^-$. Q y R son las varianzas del error y de la medición, con valores 0.001 y 0.351 (respectivamente). El filtro fue inicializado con la estimación del estado *a priori* $\hat{x}_k^- = 0$ y la varianza del error estimado *a priori* $P_{k-1} = 1.0$.

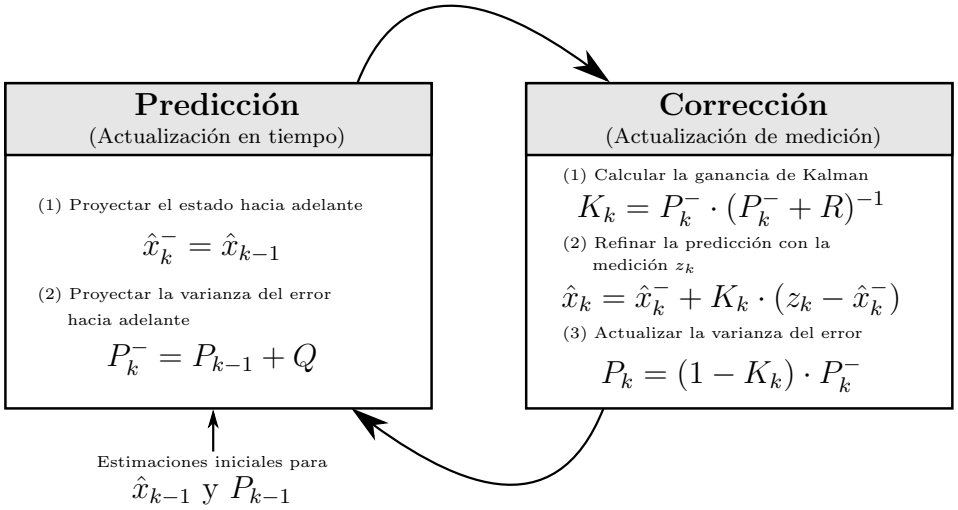


Figura 3.19: Ciclo del filtro de Kalman implementado para suavizar las posiciones de las manos

En las Figuras 3.20, 3.21 y 3.22 se muestra el resultado de la aplicación del filtro de Kalman mientras se mantiene fija la mano derecha dentro de la malla de acciones durante 30 cuadros del esqueleto. La Figura 3.20 muestra la comparativa entre el resultado de las coordenadas en X con filtro y sin la aplicación del filtro. De forma similar en las Figuras 3.21 y 3.22 se muestra la comparativa para los ejes Y y Z.

[7]Filtrado recursivo se refiere a que la estimación del estado actual en t, se basa en la estimación del estado anterior en $t-1$, remontándose así hasta la estimación del primer estado t_0 basada en el resultado de pruebas sobre la varianza del proceso.

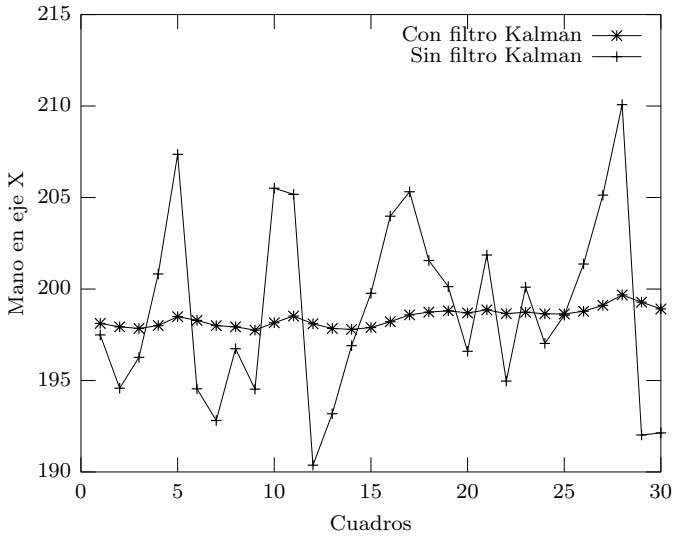


Figura 3.20: Gráfica de la coordenada X de la mano en la malla de aciones durante 30 cuadros del esqueleto

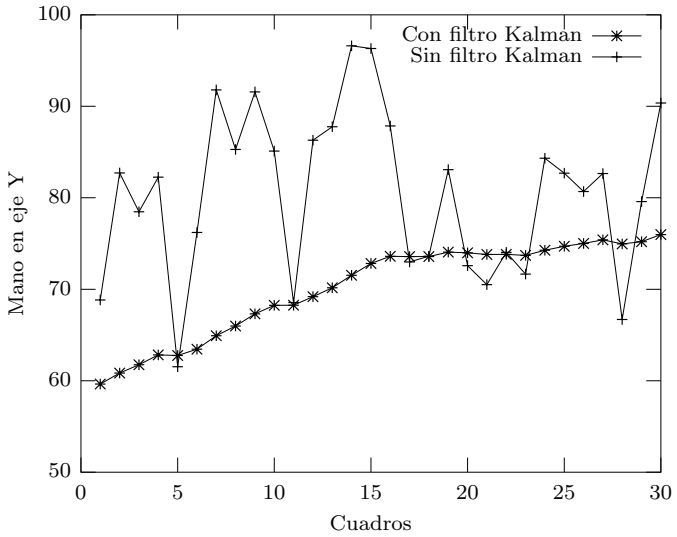


Figura 3.21: Gráfica de la coordenada Y de la mano en la malla de aciones durante 30 cuadros del esqueleto

El resultado esperado es que al mantener la mano fija, las gráficas deberían ser prácticamente lineales (debido a la vibración repentina del cuerpo), sin embargo el ruido que conlleva el esqueleto durante el rastreo genera una vibración considerable de las coordenadas de la mano. En la tabla 3.2 se muestra la comparativa entre varianza de las coordenadas de la mano con y sin filtrado para las coordenadas X, Y y Z (de las Figuras 3.20, 3.21 y 3.22).

Mediante la aplicación del filtro de Kalman se logró reducir a un 0.9217 % el ruido en el eje X, a 33.0647 % en Y y 1.1848 % en Z (con respecto a las varianzas obtenidas sin el filtro). Esta disminución del ruido deja utilizables las coordenadas de las manos, o

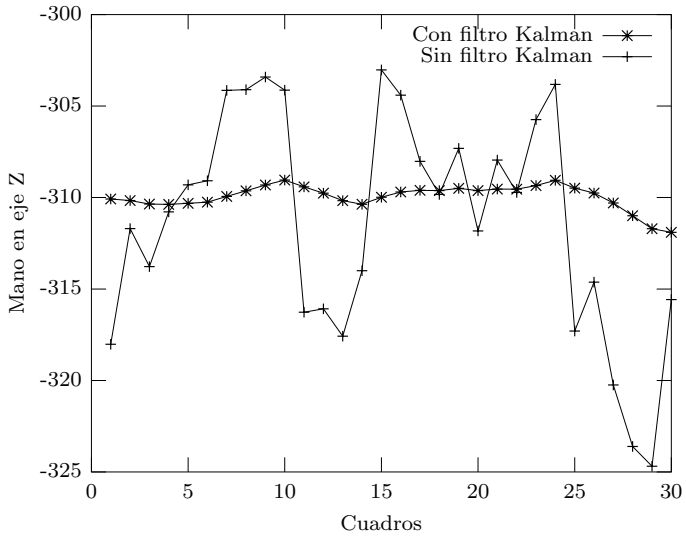


Figura 3.22: Gráfica de la coordenada Z de la mano en la malla de aciones durante 30 cuadros del esqueleto

Eje	**Varianza sin filtro**	**Varianza con filtro**
X	25.8320	0.2381
Y	79.9980	26.4511
Z	37.7000	0.4467

Tabla 3.2: Varianzas para la mano fija durante 30 cuadros del rastreo del esqueleto

cualquier otra articulación del esqueleto, dentro de la malla de acciones para controlar aplicaciones directamente con el mapeo del movimiento de las manos de las manos, como en el caso de dos de las aplicaciones experimentales de este trabajo de tesis: el ratón virtual (que se presenta en la sección 4.1, pág. 59) y en el manipulador de objetos virtuales (que se presenta en la sección 4.3, pág. 67).

3.4. Reproducción de sonido

Se incluyó la reproducción de sonidos para que el usuario identifique la navegación en la barra de menús sin necesidad de ver la pantalla.

Qt incluye un API multimedia; entre sus clases se dispone de **QSound** que permite reproducir archivos de audio, sin embargo, esta API no se encuentra totalmente desarrollada para sistemas Linux, por lo que la alternativa a usar es el framework **Phonon** [43].

Phonon es el framework multimedia estándar de **KDE 4**, incluido en Qt desde la versión 4.4, para la reproducción de archivos de audio y video codificados en formatos estándar.

Phonon se basa en tres conceptos principales que se explicarán a continuación. Para describir su arquitectura se optó por la Figura 3.23 al reproducir archivos de video.

1. **Objeto de medios**: es un objeto de tipo **MediaObject** que permite iniciar, detener

y pausar la reproducción de una cola de archivos multimedia.
MediaObject maneja los archivos a reproducir en una cola, liberando los recursos de los archivos que se van reproduciendo. Cada archivo a reproducir se carga con un objeto **MediaSource**, mismo que se encola al **MediaObject**.

2. **Puente**: es un mecanismo interno de **Phonon** que se encarga de *renderizar* y reproducir los archivos multimedia al enviar cada flujo de éste al dispositivo de salida adecuado. Por ejemplo, un archivo de video contine dos flujos multimedia: video y audio. Como se muestra en la Figura 3.23, el **puente** se encarga de mandar el flujo de audio a la tarjeta de sonido y el flujo de video a la tarjeta de video.
Además, el **puente** abstrae el dispositivo de salida a **Phonon**, lo cual le permite, por ejemplo, controlar el volumen del flujo de audio.

3. **Ruta**: conecta objetos **MediaObject** a los **puentes**, *i.e.*, accede a los dispositivos de salida adecuados al tipo de flujo a reproducir.

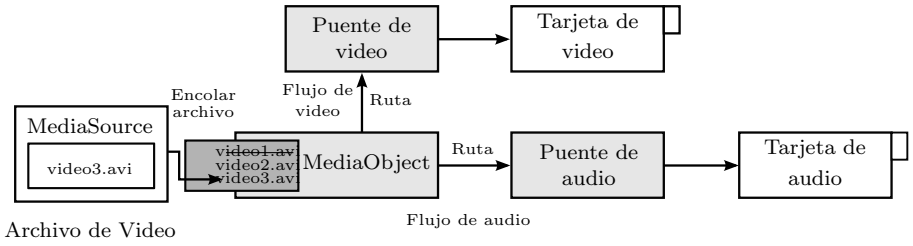


Figura 3.23: Diagrama de la arquitectura de Phonon

Algoritmo 5 Generación de código C de la máquina de estados del árbol de comandos

Entrada: *nraíz* la raíz del árbol de comandos
Salida: Generación de código C del árbol de comandos

1: Inicializar $A[p]$ y $S[p]$
2: Obtener A al recorrer el árbol por niveles
3: **para** $i = 0$ hasta p en A **hacer**
4: Iniciar el caso i del *switch* $S[i][0] \leftarrow$ "*case* q_i : "
5: Reagrupar los nodos hijos de $A[i]$ para la mano derecha, izquierda y ambas
6: **si** Hay nodos hijos con mano derecha **entonces**
7: Agregar a $S[i]$ el inicio del bloque de transiciones de mano derecha.
$if(a_{d_n}! = 0$ && $a_{i_n} == 0)$ {
8: **para** cada nodo n de la mano derecha **hacer**
9: Calcular máscara y código de validación de n
10: Generar el código del *if* y agregarlo al final de $S[i]$
11: **fin para**
12: Finalizar *if* agregando } a $S[i]$.
13: **fin si**
14: **si** Hay nodos hijos con mano izquierda **entonces**
15: **si** Hubo nodos hijos con mano derecha **entonces**
16: Agregar *else* a $S[i]$.
17: **fin si**
18: Agregar a $S[i]$ el inicio del bloque de transiciones de mano izquierda.
$if(a_{d_n} == 0$ && $a_{i_n}! = 0)$ {
19: **para** cada nodo n de la mano izquierda **hacer**
20: Calcular máscara y código de validación de n
21: Generar el código del *if* y agregarlo al final de $S[i]$
22: **fin para**
23: Finalizar *if* agregando } a $S[i]$.
24: **fin si**

Entrada: *nraíz* la raíz del árbol de comandos
Salida: Generación de código C del árbol de comandos

25: **si** Hay nodos hijos con ambas manos **entonces**
26: **si** Hubo nodos hijos con mano izquierda **entonces**
27: Agregar *else* a $S[i]$.
28: **fin si**
29: Agregar a $S[i]$ el inicio del bloque de transiciones de ambas manos.
$if(a_{d_n}! = 0$ && $a_{i_n}! = 0)$ {
30: **para** cada nodo n de de ambas manos **hacer**
31: Calcular máscara y código de validación de para las dos manos del nodo n
32: Generar el código del *if* y agregarlo al final de $S[i]$
33: **fin para**
34: Finalizar *if* agregando } a $S[i]$.
35: **fin si**
36: Finalizar caso i agregando *break;* a $S[i]$.
37: **fin para**
38: Agregar código inicial y final del método
39: Recorrer S imprimiendo cada línea en un archivo

Algoritmo 6 Detección de acciones

Entrada: Coordenadas 3D $\mathbf{m}(m_x, m_y, m_z)$ de la mano en S_m.

Salida: Acción a detectada (código de la celda que contiene a la mano o 0 si ninguna celda la contiene).

$a = 0$, $a_z = 0$, $a_v = 0$, $a_h = 0$
si la mano está dentro de la malla de acciones **entonces**
  si $m_z > Z_0 + \lambda_z$ **y** $m_z < Z_1 - \lambda_z$ **entonces**
    $a_z = 2048$
  si no, **si** $m_z > Z_1 + \lambda_z$ **y** $m_z < Z_2 - \lambda_z$ **entonces**
    $a_z = 1024$
  si no, **si** $m_z > Z_2 + \lambda_z$ **y** $m_z < Z_3 - \lambda_z$ **entonces**
    $a_z = 512$
  si no, **si** $m_z > Z_3 + \lambda_z$ **y** $m_z < Z_4 - \lambda_z$ **entonces**
    $a_z = 256$
  fin si
  si $m_x > V_0 + \lambda_v$ **y** $m_x < V_1 - \lambda_v$ **entonces**
    $a_v = 128$
  si no, **si** $m_x > V_1 + \lambda_v$ **y** $m_x < V_2 - \lambda_v$ **entonces**
    $a_v = 64$
  si no, **si** $m_x > V_2 + \lambda_v$ **y** $m_x < V_3 - \lambda_v$ **entonces**
    $a_v = 32$
  si no, **si** $m_x > V_3 + \lambda_v$ **y** $m_x < V_4 - \lambda_v$ **entonces**
    $a_v = 16$
  fin si
  si $m_y < H_0 + \lambda_h$ **y** $m_y > H_1 - \lambda_h$ **entonces**
    $a_h = 8$
  si no, **si** $m_y < H_1 + \lambda_h$ **y** $m_y > H_2 - \lambda_h$ **entonces**
    $a_h = 4$
  si no, **si** $m_y < H_2 + \lambda_h$ **y** $m_y > H_3 - \lambda_h$ **entonces**
    $a_h = 2$
  si no, **si** $m_y < H_3 + \lambda_h$ **y** $m_y > H_4 - \lambda_h$ **entonces**
    $a_h = 1$
  fin si
  si $a_z == 0$ **o** $a_v == 0$ **o** $a_h == 0$ **entonces**
    $a = -1$
  si no
    $a = a_z || a_v || a_h$
  fin si
fin si
regresar a

Capítulo 4
Aplicaciones

En este capítulo se presentan las tres aplicaciones de prueba que se desarrollaron sobre el sistema de interfaz natural. En la sección 4.1 se presenta el ratón virtual que permite mover el cursor del ratón sobre el entorno de escritorio con la mano derecha, mientras que con la mano izquierda se realizan los cliks. En la sección 4.2 se presenta la navegación de menús de Qt que permite desplazarse por los elementos de la barra de menús y mandar ejecutar alguna de sus operaciones. Posteriormente en la sección 4.3 se explica el manipulador de objetos virtuales que es una aplicación que permite visualizar y aplicar transformaciones geométricas a un objeto virtual mediante movimientos de las manos.

En cada sección se explica el funcionamiento, la secuencia de acciones de cada comando, el árbol de comandos generado a partir cada lista de comandos y el código C generado a partir de este último que permite detectar los comandos de cada aplicación.

4.1. Ratón virtual

La aplicación ratón virtual permite simular la entrada de un ratón convencional desplazando la mano derecha en la malla de acciones mientras se realizan los clicks de ratón con la mano izquierda.

La Figura 4.1 muestra un esquema de la aplicación. El desplazamiento de la mano derecha sobre el plano de mapeo P, el plano color gris dentro de la malla de acciones en la Figura 4.1, que se encuentra en el plano Z_2 y está limitado por los planos H_1 y V_1 es usado para mover el cursor del ratón en la pantalla. Los clicks de ratón se realizan mediante comandos de interacción de la mano izquierda.

Los comandos implementados para esta aplicación se presentan en la tabla 4.1, el árbol de comandos generado se presenta en la Figura 4.3 y el código C generado a partir del árbol se presenta en el Código 7 del apéndice A (en la pág. 90).

En la Figura 4.3(b) se muestra la representación en texto del árbol del comandos de la Figura 4.3(a) que el sistema de interfaz natural genera durante la creación del árbol

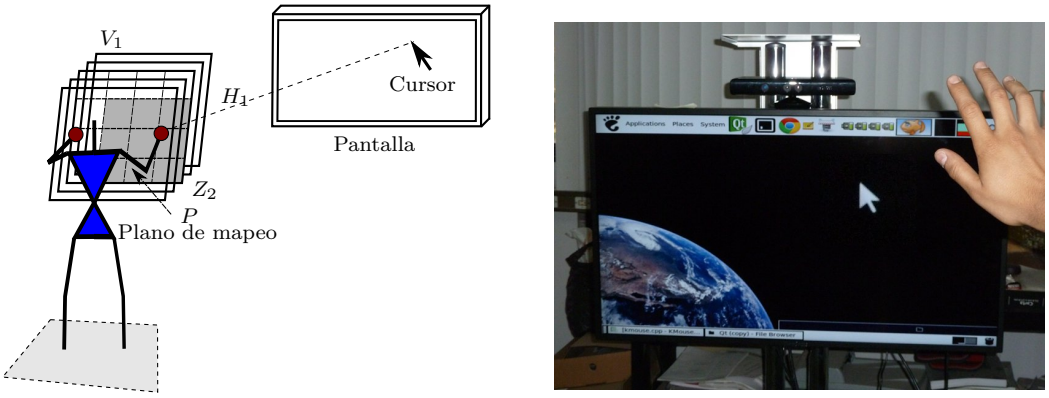


Figura 4.1: Ratón virtual y plano P de rastreo de la mano derecha

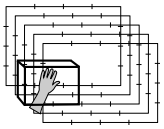
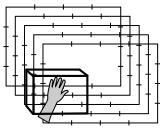
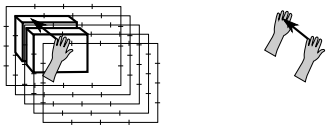
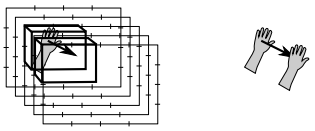
id	Comando	Secuencia de acciones	Figura
0	*Click izquierdo*	642||578||641||577@1	
1	*Mantener click izquierdo*	1154||1090||1153||1089@1	
2	*Doble click izquierdo*	1160||1096||1156||1092@1, 648||584||644||580@1	
3	*Click derecho*	1160||1096||1156||1092@1, 2184||2120||2180||2116@1	

Tabla 4.1: Lista de comandos del ratón virtual

de comandos. Esta representación se explica en la sección "Representación del árbol de comandos en texto" en la pág. 49.

El **id** de cada comando en la tabla 4.1 se refiere al <**id**> de comando en la Figura 4.3 y al identificador del evento Qt que le fue asignado al comando.

Los comandos *click izquierdo* y *mantener click izquierdo* se encuentran relacionados entre sí, dado que ambos involucran un mismo movimiento de la mano izquierda: desplazarla hacia adelante. El comando *mantener click izquierdo* simula la entrada de ratón dejando el botón izquierdo presionado al mantener la mano izquierda cerca del cuerpo (a la altura del torso). Al realizar el comando *click izquierdo i.e.* estirar el brazo, se suelta el botón izquierdo del ratón generando el evento de click izquierdo en el sistema gráfico.

El comando *doble click izquierdo* se realiza empujando la mano izquierda hacia adelante (pero a la altura de la cara para diferenciarlo de un solo click izquierdo). El comando *click derecho* es similar al doble click, pero desplazando la mano izquierda hacia atrás (cerca de la oreja izquierda).

En la Figura 4.2 se muestra un ejemplo del comando *mantener click izquierdo* en el que se arrastra la ventana del esqueleto 3D. El click se sostiene con la mano izquierda (por lo que el cursor es un ícono de una mano cerrada) mientras que el cursor se desplaza con la mano derecha.

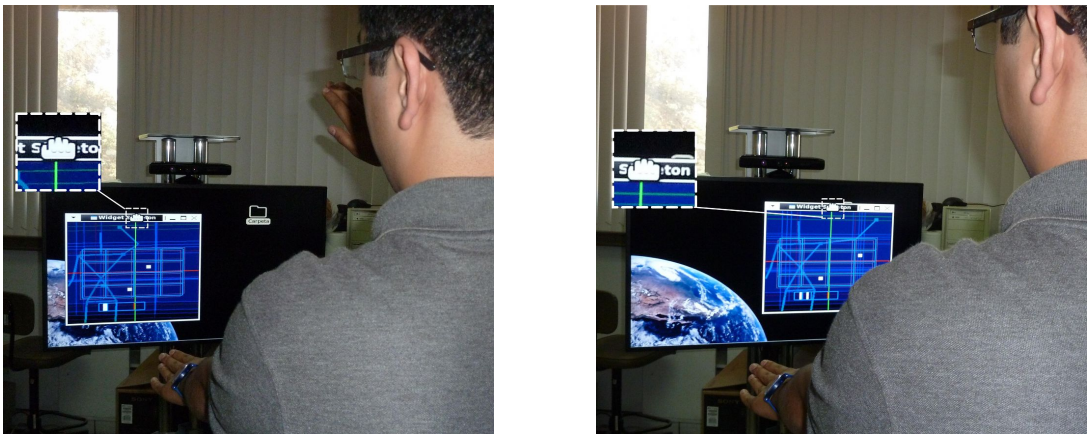
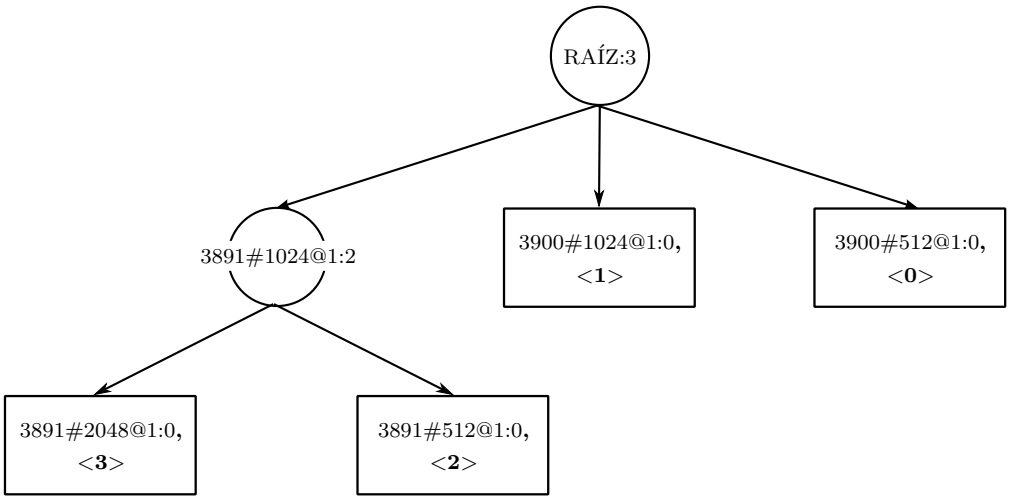


Figura 4.2: Arrastrando una ventana con el ratón virtual



(a) Diagrama del árbol de comandos

RAÍZ:3, 3891#1024@1:2, 3900#1024@1:0, <**1**>, 3900#512@1:0, <**0**>, 3891#2048@1:0, <**3**>, 3891#512@1:0, <**2**>

(b) Representación del árbol de comandos en texto

Figura 4.3: Árbol de comandos del ratón virtual

4.1.1. Plano virtual

El cursor del ratón es un ícono que representa una flecha que apunta hacia un punto o pixel $\mathbf{s}(s_x, s_x)$ en la pantalla, de acuerdo a su resolución en *ancho_escritorio* × *alto_escritorio* pixeles.

Para mapear la posición $\boldsymbol{m}_n(m_{x_n}, m_{y_n})$ de la mano derecha a un punto $\mathbf{s_n}$ de la pantalla primero se consideró una región donde el desplazamiento de la mano no implicara estiramientos forzosos del brazo, por lo que el desplazamiento que se acotó al plano de mapeo P (que se muestra en la Figura 4.1, en la pág. 60), mismo que estableció con un tamaño de 640×480 unidades o pixeles. Los valores del ancho d_w y alto d_h de cada pixel en unidades de la malla de acciones (*i.e.* en milímetros) son:

$$d_w = \frac{d(V_1, V_4)}{640}, \qquad d_h = \frac{d(H_1, H4)}{480}$$

donde $d(\cdot)$ es una función que mide la distancia euclidiana entre dos puntos de entrada.

Las coordenadas $\boldsymbol{m}_n(m_{x_n}, m_{y_n})$ de la mano en el plano Z_2 de la malla de acciones se mapean a un pixel $\mathbf{p}_n(p_{x_n}, p_{y_n})$ en el plano P con:

$$\mathbf{p}_n = (\frac{m_{x_n}}{d_w}, \frac{m_{y_n}}{d_h})$$

Posteriormente el pixel $\mathbf{p}_n$ se mapea al pixel $\mathbf{s}_n(s_{x_n}, s_{y_n})$ de la pantalla con:

$$s_x = p_{x_n} \cdot \frac{ancho\_escritorio}{640}, \qquad s_y = (480 - p_{y_n}) \cdot \frac{alto\_escritorio}{480}$$

Finalmente se aplica el filtro kalman implementado en esta tesis a $\mathbf{s}_n$ y el cursor del ratón se posiciona en punto filtrado.

El resultado es que el usuario puede desplazar el cursor del ratón por la pantalla mediante los movimientos de la mano derecha dentro de P.

4.1.2. Botones del ratón

Qt permite crear y enviar eventos de click de ratón, pero con la limitante de que funciona únicamente en interfaces desarrolladas en Qt, *i.e.* los clicks de ratón no funcionan fuera de la ventana de la aplicación Qt.

Para evitar esta problemática y permitir que los clicks de ratón funcionen en todo el entorno de escritorio se utilizó la biblioteca **XTest** para generar eventos de entrada $X11$ con las funciones **XTestFakeButtonEvent** y **XSync**, que serán descritas brevemente:

- **XTestFakeButtonEvent** (*display*, *botón*, *presionado*, *CurrentTime*)
  donde
  - *display*. Apuntador a la pantalla activa, *i.e.* la pantalla en la que se encuentra el cursor.

- *botón*. Entero que identifica los botones del ratón (el izquierdo es 1 y el derecho 3).
- *presionado*. Booleano que indica si el botón está o no está presionado.
- *CurrentTime*. Indica el tiempo en el evento debe generarse inmediatamente después de la llamada a la función.

- **XSync**(*display*, 0)
  Sincroniza el evento generado con la pantalla *display*. Debe llamarse después de **XTestFakeButtonEvent** para que el evento generado se envíe a la cola de eventos de *X*11.

En los códigos 4.1, 4.2, 4.3 y 4.4 se muestran los fragmentos de código que permiten realizar los comandos *click izquierdo*, *mantener click izquierdo*, *doble click izquierdo* y *click derecho*.

Código 4.1: Click izquierdo

```
1 XTestFakeButtonEvent (dpy, 1, True, CurrentTime);
2 XSync(dpy, 0);
3 XTestFakeButtonEvent (dpy, 1, False, CurrentTime);
4 XSync(dpy, 0);
```

Código 4.2: Mantener click izquierdo

```
1 if(mantener) {
2     XTestFakeButtonEvent (dpy, 1, True, CurrentTime);
3     XSync(dpy, 0);
4 } else {
5     XTestFakeButtonEvent (dpy, 1, False, CurrentTime);
6     XSync(dpy, 0);
7 }
```

Código 4.3: Doble click izquierdo

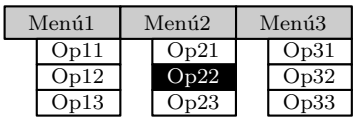
```
1 XTestFakeButtonEvent (dpy, 1, True, CurrentTime);
2 XSync(dpy, 0);
3 XTestFakeButtonEvent (dpy, 1, False, CurrentTime);
4 XSync(dpy, 0);
5 XTestFakeButtonEvent (dpy, 1, True, CurrentTime);
6 XSync(dpy, 0);
7 XTestFakeButtonEvent (dpy, 1, False, CurrentTime);
8 XSync(dpy, 0);
```

Código 4.4: Click derecho

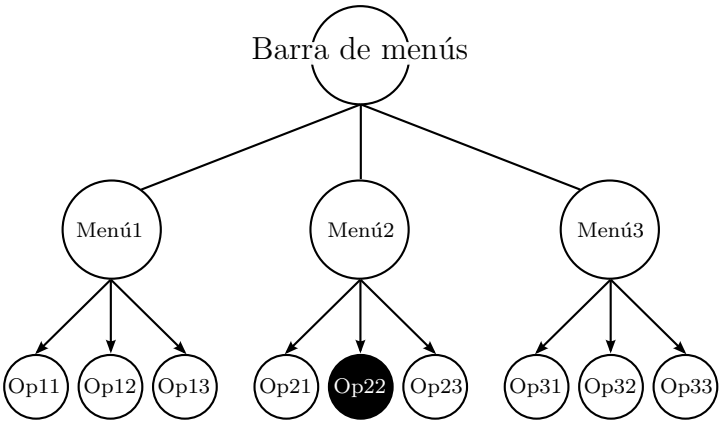
```
1 XTestFakeButtonEvent (dpy, 3, True, CurrentTime);
2 XSync(dpy, 0);
3 XTestFakeButtonEvent (dpy, 3, False, CurrentTime);
4 XSync(dpy, 0);
```

4.2. Navegación de menús de Qt con Kinect

La barra de menús, como la mostrada en la Figura 4.4(a), es uno de los elementos característicos de las interfaces gráficas. Mantiene una jerarquía de *elementos*, que son las *operaciones*[1] y los *menús* o submenús que agrupan operaciones similares de la aplicación.



(a) Barra de menús simple



(b) Árbol de la barra de menús de la izquierda

Figura 4.4: Barra de menús de una aplicación (el elemento *Op22* está seleccionado)

Como se observa en la Figura 4.4, los elementos *Menú1*, *Menú2* y *Menú3* son los menús que agrupan las operaciones *Op11*, *Op12*, *Op13*, $\cdots$ *Op33* de la aplicación.

Se construyó una aplicación para navegar en menús y que permite recorrer los elementos de la barra de menús de la ventana para seleccionar y ejecutar alguna de las operaciones disponibles. Los comandos de la interfaz natural implementados para esta aplicación se presentan en la tabla 4.2, el árbol de comandos generado se presenta en la Figura 4.5 y el código C generado a partir del mismo se presenta en el Código 8 del apéndice A (en la pág. 91). La interpretación de la Figura 4.5(b) se explica en la sección "Representación del árbol de comandos en texto" en la pág. 49. El **id** de cada comando en la tabla 4.2 se refiere al <**id**> de comando en la Figura 4.5 y al identificador del evento Qt que le fue asignado al comando.

En la Figura 4.6 se muestra un ejemplo de la aplicación para navegar los menús. La barra de menús se encuentra en la parte inferior de la ventana de la aplicación.

A continuación se describen los comandos implementados para esta aplicación.

- *Abrir barra/submenú.*
  Despliega el primer menú de la barra de menús (o en caso de que un submenú esté seleccionado, se despliega su primer elemento) al levantar ambas manos, simulando que se levanta una cortina.

- *Ejecutar operación.*

[1] Con *operaciones* nos referimos a las acciones ejecutables de un menú, *e.g.* copiar, pegar, abrir, etc.

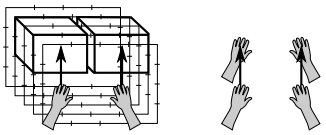
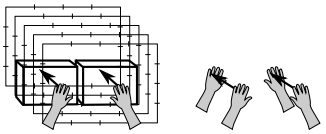
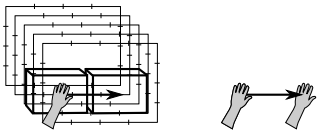
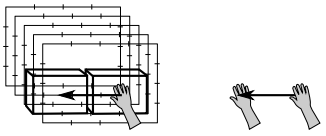
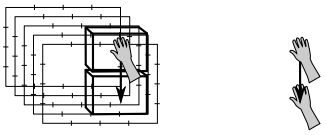
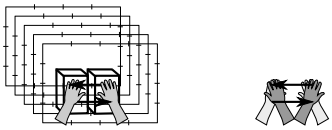
id	Comando	Secuencia de acciones	Figura
0	*Abrir barra/-submenú*	1064||1048||1060||1044|| 552||536||548||532@0&& 1160||1096||1156||1092|| 648||584||644||580@1	
1	*Ejecutar operación*	546||530||545||529@0&& 642||578||641||577@1	
2	*Siguiente elemento*	1154||1090||1153||1089@1, 1058||1042||1057||1041@1	
3	*Elemento anterior*	1058||1042||1057||1041@0, 1154||1090||1153||1089@0	
4	*Cerrar barra*	1064||1048||1060||1044@0, 1058||1042||1057||1041@0	
5	*Cerrar aplicación*	1090||1089@0&&1058||1057@1	

Tabla 4.2: Lista de comandos de la navegación de menús de Qt

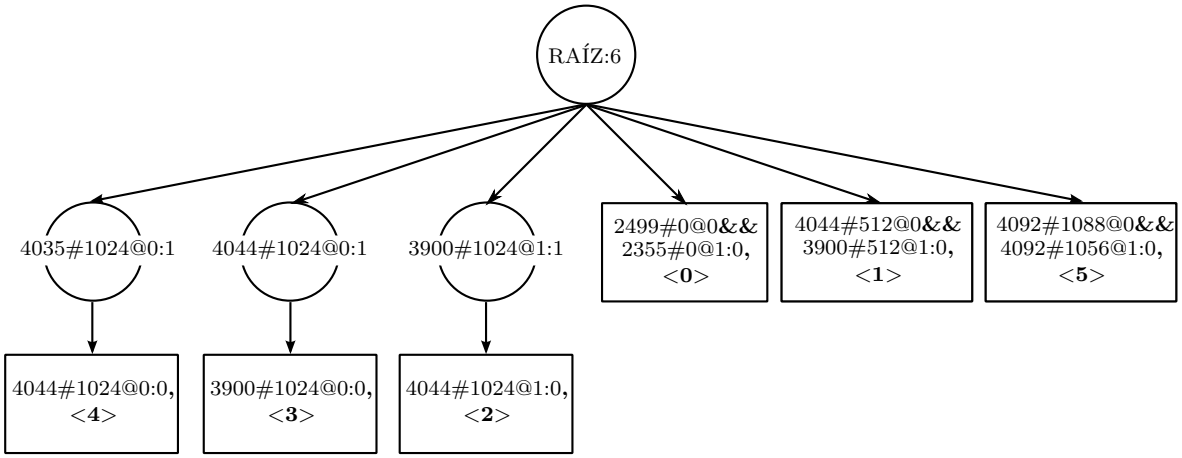
Si el elemento seleccionado es una operación de la barra de menús, ésta puede mandarse a ejecutar al desplazar ambas manos hacia el frente.

- *Siguiente elemento.*
  Permite seleccionar el siguiente elemento al actualmente seleccionado al desplazar la mano izquierda hacia la derecha, *i.e.* se recorren los elementos del último menú abierto.

- *Elemento anterior.*
  Permite seleccionar el elemento anterior al desplazar la mano derecha hacia la izquierda.

- *Cerrar barra.*
  Cierra la barra al desplazar la mano derecha de arriba hacia abajo. La barra se cierra automaticamente tras 5 segundos de inactividad

- *Cerrar aplicación.*
  La aplicación se termina al cruzar las manos, *i.e.* desplazar la mano izquierda a la derecha y la mano derecha a la izquierda



(a) Diagrama del árbol de comandos

RAÍZ:6, 4035#1024@0:1, 4044#1024@0:1, 3900#1024@1:1, 2499#0@0&&2355#0@1:0, <**0**>, 4044#512@0&&3900#512@1:0, <**1**>, 4092#1088@0&&4092#1056@1:0, <**5**>, 4044#1024@0:0, <**4**>, 3900#1024@0:0, <**3**>, 4044#1024@1:0, <**2**>

(b) Representación del árbol de comandos en texto

Figura 4.5: Árbol de comandos del navegador de menús

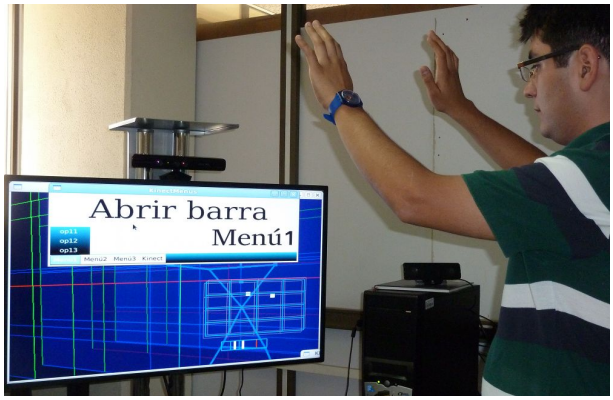


Figura 4.6: Navegador de menús de Qt

4.2.1. Implementación de la navegación de menús

En **Qt**, la barra de menús es un widget de tipo **QMenuBar**, que contiene las operaciones (objetos **QAction**) y submenús (objetos **QMenu**) de la aplicación.

La navegación de menús se basa en la construcción de un *árbol* (ver algoritmo 7) a partir de los elementos de la barra de menús y un mecanismo de *máquina de estados* para recorrerlo de acuerdo a los comandos detectados.

La barra de menús (y los submenús) mantienen sus elementos en un arreglo de objetos **QAction**. El método **QMenu* QAction::menu()** permite discernir si un determinado elemento es un submenú o si es una operación. Este método se utiliza durante un recorrido

Algoritmo 7 Algoritmo recursivo para la generación del árbol jerárquico de una barra de menús

Entrada: Menú m, nodo r del árbol de la barra.
(En la primer llamada al algoritmo, m es la barra de menús y r es el nodo raíz del árbol).

Salida: Generación del árbol jerárquico de la barra de menús.

1: **para** cada elemento e_i de la lista de elementos de m **hacer**
2: Construir el nodo n copiando el elemento e_i y un apuntador al padre r.
3: Agregar e_i a los hijos de r.
4: **si** e_i es un submenú **entonces**
5: Llamada recursiva a este algoritmo con el menú e_i y el nodo del árbol $r.ultimoHijo()$
6: **fin si**
7: **fin para**

recursivo de los elementos de la barra de menús para construir un árbol (como el de la Figura 4.4(b) para la barra de menús de la Figura 4.4(a)), en el que la raíz es la barra de menús, los nodos interiores o ramas son los submenús (o menús cuando el nodo padre es la barra) y los nodos hoja son las operaciones.

Al realizar el comando *abrir barra/submenú* el estado e de la máquina de estados se posiciona en r, la raíz del árbol (*i.e.* la barra de menús) y el apuntador a nodo h se posiciona sobre el primer hijo del nodo apuntado por e, *i.e.* $e = r$ y $h = e_i, i = 0$.

Al detectar los comandos *siguiente elemento* o *elemento anterior* se realiza un recorrido al nodo $h = e_i$ siguiente (con $i++$) o anterior (con $i--$), el resultado se visualiza en la pantalla como el cambio de resaltado entre elementos de la barra. Si se detecta el comando *ejecutar operación* y h es un nodo hoja se ejecuta la implementación asignada a la operación, de forma similar, si se detecta el comando *abrir barra/submenú* y h es un nodo rama, entonces e apuntará a h, *i.e.* $e = h$ y $h = e_0$. De lo contrario el comando es ignorado.

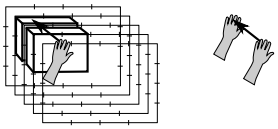
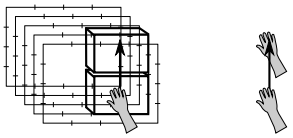
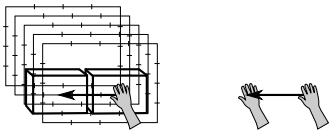
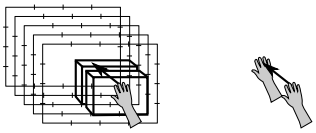
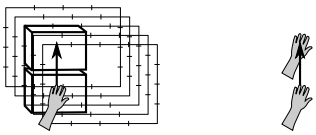
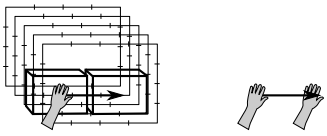
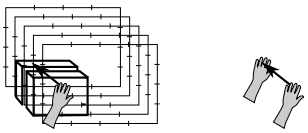
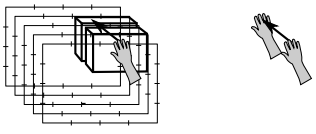
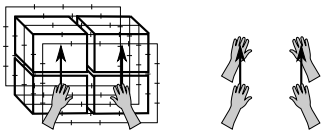
4.3. Interfaz de manipulación de objetos virtuales

La interfaz de manipulación de objetos virtuales fue desarrollada teniendo en mente el poder manipular un objeto virtual con las manos como si éste se encontrara al frente del usuario (como se observa en la Figura 4.9(a)) y cuyo uso resulte *natural* después de pocas repeticiones de los comandos propuestos.

La interfaz permite manipular el objeto al aplicar las transformaciones geométricas *rotación*, *escalamiento* y *traslación* a partir de la interpretación los movimientos de las manos con respecto a la malla de acciones y la detección de los comandos definidos en la Tabla 4.3. La Figura 4.7 muestra el árbol de comandos generado a partir de la lista de comandos, y el Código 9 del apéndice A (en la pág. 92). La interpretación de la Figura 4.7(b) se explica en la sección "Representación del árbol de comandos en texto" en la pág. 49.

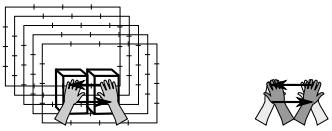
El **id** de cada comando en la tabla 4.3 se refiere al <**id**> de comando en la Figura 4.7 y al identificador del evento Qt que le fue asignado al comando.

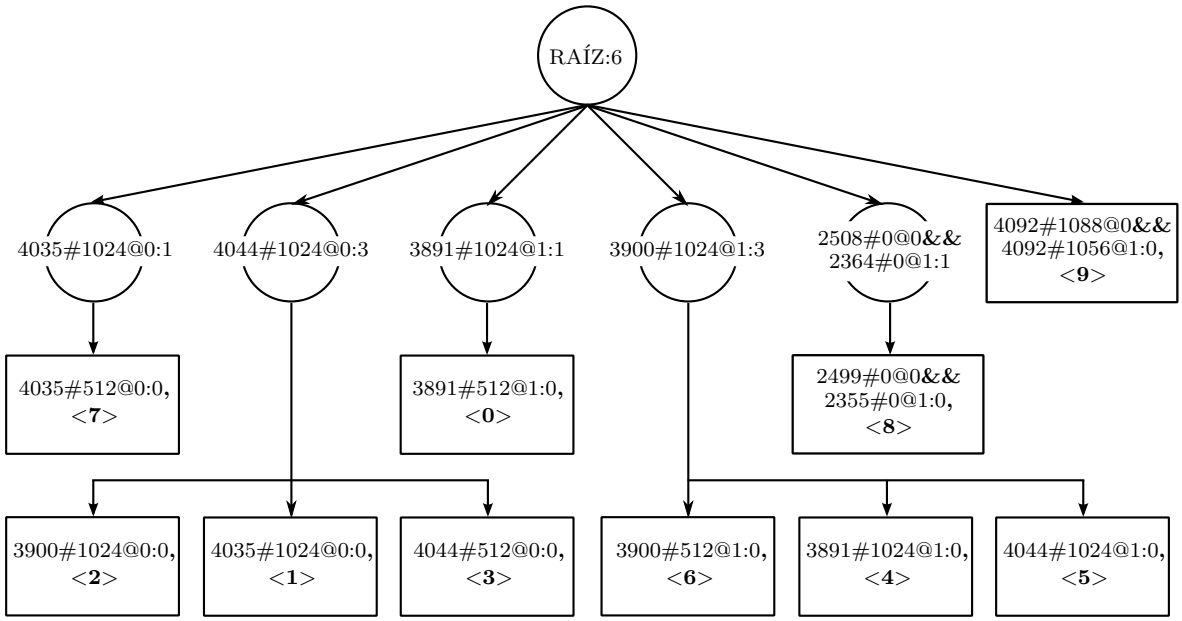
Tabla 4.3: Lista de comandos de la interfaz de manipulación de objetos virtuales

id	Comando	Secuencia de acciones	Figura
0	*Cambiar modo*	1160||1096||1156||1092@1, 648||584||644||580@1	
1	*Iniciar rotación alrededor de X*	1058||1042||1057||1041@0, 1064||1048||1060||1044@0	
2	*Iniciar rotación alrededor de Z*	1058||1042||1057||1041@0, 1154||1090||1153||1089@0	
3	*Iniciar rotación alrededor de Y*	1058||1042||1057||1041@0, 546||530||545||529@0	
4	*Detener rotación alrededor de X*	1154||1090||1153||1089@1, 1160||1096||1156||1092@1	
5	*Detener rotación alrededor de Z*	1154||1090||1153||1089@1, 1058||1042||1057||1041@1	
6	*Detener rotación alrededor de Y*	1154||1090||1153||1089@1, 642||578||641||577@1	
7	*Reiniciar objeto*	1064||1048||1060||1044@0, 552||536||548||532@0	
8	*Mostrar ventana completa/normal*	1058||1042||1057||1041||546||530||545||529@0 **&&** 1154||1090||1153||1089||642||578||641||577@1, 1064||1048||1060||1044||552||536||548||532@0 **&&** 1160||1096||1156||1092||648||584||644||580@1	

Continúa en la siguiente pág.

Tabla 4.3 – *Continuación de la pág. anterior.*

id	Comando	Secuencia de acciones	Figura
9	*Cerrar aplicación*	1090||1089@0&&1058||1057@1	



(a) Diagrama del árbol de comandos

RAÍZ:6, 4035#1024@0:1, 4044#1024@0:3, 3891#1024@1:1,
3900#1024@1:3, 2508#0@0&&2364#0@1:1,
4092#1088@0&&4092#1056@1:0, <**9**>, 4035#512@0:0, <**7**>,
3900#1024@0:0, <**2**>, 4035#1024@0:0, <**1**>, 4044#512@0:0, <**3**>,
3891#512@1:0, <**0**>, 3900#512@1:0, <**6**>, 3891#1024@1:0, <**4**>,
4044#1024@1:0, <**5**>, 2499#0@0&&2355#0@1:0, <**8**>

(b) Representación del árbol de comandos en texto

Figura 4.7: Árbol de comandos de la interfaz de manipulación de objetos virtuales

El funcionamiento de la interfaz se basa en dos modos de interacción, mismos que se intercambian al realizar el comando *cambiar modo*. Los modos son:

1. Manipulación directa del objeto.
2. Rotación del objeto mediante comandos de interacción.

Los comandos *reiniciar objeto*, *mostrar ventana completa/normal* y *cerrar aplicación* se pueden realizar en cualquier momento, éstos se describen a continuación.

- *Reiniciar objeto.*
  Vuelve a colocar el objeto en su posición inicial cuando se mueve la mano derecha hacia adelante (como si se pulsara un botón a la altura de la frente).

- *Mostrar ventana completa/normal.*
  Cambia el tamaño de la ventana de normal a pantalla completa y viceversa al levantar ambas manos (como si se levantara una cortina).

- *Cerrar aplicación.*
  El manipulador se cierra al cruzar las manos al mismo tiempo dejando los brazos cruzados en forma de X, *i.e.* con la mano derecha a la izquierda y la mano izquierda a la derecha.

Manipulación directa del objeto

En este modo se elige entre rotación, escalamiento y traslación del objeto al realizar el comando *cambiar modo*. Las tres transformaciones se realizan tomando las coordenadas de las manos con respecto a la malla de acciones.

La rotación y traslación se realizan al mover la mano derecha dentro de la malla de acciones. El escalamiento se realiza con ambas manos simulando que el objeto es estirado o comprimido al cambiar la distancia entre las manos.

La transformación indicada se inicia cuando la mano (o ambas manos en el caso del escalamiento) sobrepasan el plano Z_2 de la malla de acciones. La transformación se detiene cuando la(s) mano(s) regresa(n) junto al cuerpo, *i.e.* cuando la mano se encuentre entre Z_1 y Z_2.

Las transformaciones geométricas del objeto se realizan con la llegada de cada esqueleto n, por lo que el objeto es transformado y dibujado con OpenGL aproximadamente 30 veces por segundo.

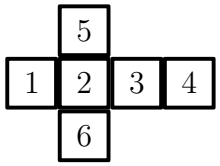
En las secciones 4.3.4, 4.3.5 y 4.3.6 (en las págs. 73, 74 y 75) se explica la implementación del modo directo de la rotación, escalamiento y traslación del objeto virtual.

Rotación del objeto mediante comandos de interacción

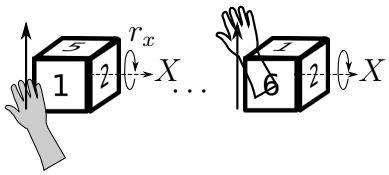
Este modo permite iniciar (o detener) una *rotación continua* alrededor del eje X, Y o Z. Cuando este modo está activo, las rotaciones se inician mediante comandos con la mano derecha y se detienen mediante comandos similares para la mano izquierda. El funcionamiento de dichos comandos (definidos en la tabla 4.3, en la pág. 68) se describe a continuación utilizando la Figura 4.8.

En la Figura 4.8 se explica la rotación de un objeto (el cubo de la Figura 4.8(a)) alrededor de los ejes X, Y y Z. La rotación alrededor de un eje en particular puede entenderse como si dicho eje fuese una varilla que atraviesa al objeto por su centro, entonces el movimiento del objeto queda limitado a la varilla, *i.e.* el objeto se rota alrededor de la varilla (los números de las caras no se muestran rotados para facilitar su lectura).

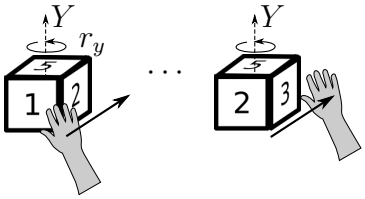
- *Iniciar rotación alrededor de X.*
  La rotación se inicia al subir la mano derecha, el objeto se mantiene rotando en



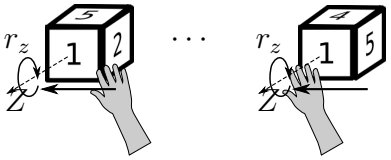
(a) Figura de un cubo y valores de sus caras



(b) Rotación alrededor del eje X



(c) Rotación alrededor del eje Y



(d) Rotación alrededor del eje Z

Figura 4.8: Rotación de un objeto en el modo de comandos

sentido contrario a las manecillas del reloj alrededor del eje X. Como se muestra en la Figura 4.8(b) después de un tiempo de la animación de rotación, la cara 6 del cubo pasará de ser la cara inferior a ser la cara frontal del cubo.

- *Detener rotación alrededor de X.*
  La rotación se detiene al subir la mano izquierda simulando que el objeto forzado a detenerse.

- *Iniciar rotación alrededor de Y.*
  La rotación se inicia al mover la mano derecha hacia adelante y el objeto se mantiene rotando alrededor del eje Y en sentido contrario a las manecillas del reloj. Como se muestra en la Figura 4.8(c) después de un tiempo de la animación de rotación, la cara 2 pasará de la derecha a ser la cara frontal del cubo.

- *Detener rotación alrededor de Y.*
  La rotación se detiene al mover la mano izquierda hacia adelante.

- *Iniciar rotación alrededor de Z.*
  La rotación se inicia al al mover la mano derecha hacia la izquierda y el objeto se mantiene rotando en sentido de las manecillas del reloj alrededor del eje Z. Como se muestra en la Figura 4.8(d) la cara 5 pasará de ser la cara superior a ser la cara derecha y la cara 1 se mantiene al frente.

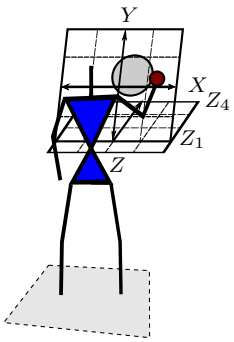
- *Detener rotación alrededor de Z.*
  La rotación se detiene al mover la mano izquierda hacia la derecha.

La rotación continua se refiere a una animación en la que el objeto permanece rotando en el eje o ejes indicados por el usuario.

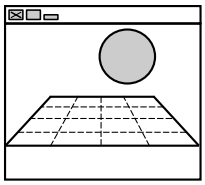
El la sección 4.3.3 se explica la implementación de la rotación del objeto mediante comandos.

4.3.1. Configuración de la interfaz y el objeto virtual

El sistema de coordenadas XYZ del manipulador de objetos se presenta en la Figura 4.9.



(a) Sistema de coordenadas del manipulador de objetos



(b) Ventana del manipulador de objetos

Figura 4.9: Configuración de la interfaz de manipulación de objetos

El objeto virtual (que en la figura es la esfera gris) se encuentra al frente del usuario, el origen del sistema de coordenadas del manipulador corresponde al origen de la malla de acciones. El eje X positivo se encuentra hacia la derecha, el eje Y positivo apunta hacia arriba. La zona de interacción para el modo de manipulación directa se encuentra de Z_1 a Z_4.

La interfaz de manipulación consta en una ventana (widget Qt) en donde se dibuja el objeto y el espacio de interacción (Figura 4.9).

4.3.2. Dibujado del objeto virtual

En la sección 2.4 (en la pág. 25) se trató la composición de los objetos virtuales.

Con la finalidad de dibujar objetos virtuales complejos (como el que se muestra en la Figura 4.10 y la Figura 4.11) se utilizó la biblioteca **libg3d** [44].

Libg3d permite cargar objetos virtuales almacenados en archivos (entre ellos el formato *wavefront*[3]) y en mantener una representación de éstos como objetos en memoria. Para dibujar el objeto en un widget de Qt con OpenGL, se adaptó el código fuente de **libg3d-viewer** [45], el visor de objetos libg3d, que dibuja el objeto con OpenGL sobre una ventana GTK.

[2]Modelo 3D de un conejo reconstruido en 3D en Stanford, disponible en `http://graphics.stanford.edu/data/3Dscanrep/#bunny`

[3] *Wavefront* es un formato de archivo de texto plano para almacenar objetos virtuales. Se basa en la definición de la lista de vértices del objeto y una lista de caras (formadas por un grupo de vértices de la lista y el color de ésta).

Figura 4.10: Reconstrucción 3D de una figura de un conejo realizada en Stanford[2]

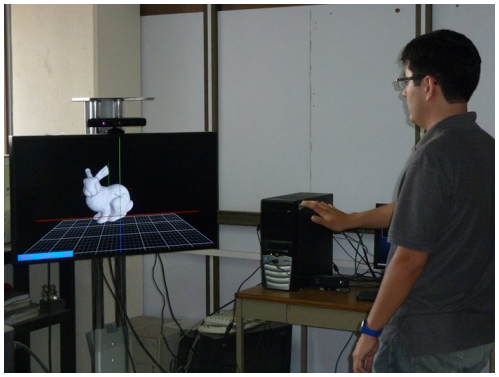


Figura 4.11: Conejo de Stanford en la interfaz de manipulación de objetos virtuales

Configuración de la cámara de OpenGL

La cámara se configuró con perspectiva, lo que permite notar la traslación sobre el eje Z, *i.e.* los objetos se ven más pequeños al alejarse de la cámara y más grandes al acercarse a ésta (como se observa en la Figura 4.16(b), en la pág. 76).

4.3.3. Rotación del objeto mediante comandos de interacción

El resultado de la rotación del objeto mediante comandos se visualiza mediante una animación a 25 cps en la que el objeto permanece rotando alrededor del eje o ejes indicados por los comandos realizados por el usuario (como se indicó en la pág. 70 de la sección 4.3).

El algoritmo 8 presenta la rotación del objeto mediante comandos. Los ejes de rotación activos se identifican por tres banderas globales b_x, b_y y b_z. Cuando el usuario indica, el inicio o fin de la rotación sobre un eje i, el valor de la bandera b_i se niega. Esto permite que se realice o no la rotación del objeto alrededor del eje i.

El control del tiempo de la animación se realiza con un temporizador periódico que envía una señal para invocar a los métodos de rotación y dibujado del objeto cada $\frac{1}{25}$ segundos. Cada $\frac{1}{25}$ segundos los ángulos de rotación r_x, r_y y r_z para los ejes activos se incrementan en 2° y se repinta el objeto aplicando la rotación con los ángulos resultantes.

4.3.4. Rotación directa del objeto virtual

La rotación en el modo directo del manipulador se realiza mapeando las coordenadas $\mathbf{m_n}(m_x, m_y)$ de la mano derecha sobre el plano X_mY_m de la malla de acciones a ángulos de rotación $\mathbf{r_n}(\phi, \theta)$ con $\phi \in [-90°, 90°]$ (alrededor de X) y $\theta \in [-180°, 180°]$ (alrededor de Y).

En la Figura 4.12 se muestra la configuración de la rotación. La rotación se percibe como si $\vec{v}$ fuese una varilla fija al centro del objeto y éste se rotara al mover la varilla con la mano (*e.g.* si la mano se desplazara a la izquierda veríamos la cajuela del auto, si la mano se desplazara hacia arriba se vería la suspensión). La rotación aplicada al objeto es

Algoritmo 8 Dibujado de cuadro de rotación del objeto

Entrada: Señal del temporizador cada $\frac{1}{25}$.
Salida: Aplicación de rotación alrededor de los ejes indicados y dibujado del objeto.

si b_x == true **entonces**
  r_x += 2°
  $r_x = r_x \ mod \ 360°$
  Rotar el objeto r_x° alrededor del eje X
fin si
si b_y == true **entonces**
  r_y += 2°
  $r_y = r_y \ mod \ 360°$
  Rotar el objeto r_y° alrededor del eje Y
fin si
si b_z == true **entonces**
  r_z += 2°
  $r_z = r_z \ mod \ 360°$
  Rotar el objeto r_z° alrededor del eje Z
fin si
Dibujar el objeto

entonces

$$R_n = R_y(\theta)R_x(\phi)$$

para los ángulos $\mathbf{r_n}(\phi, \theta)$ obtenidos de la mano derecha del esqueleto n.

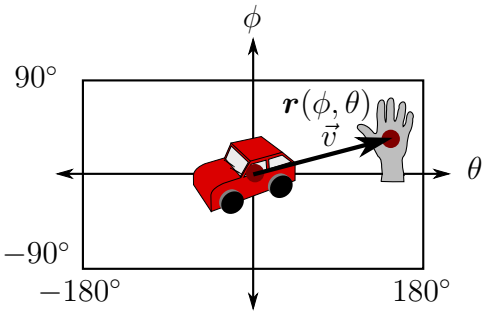


Figura 4.12: Rotación directa del objeto

En la Figura 4.13 se presenta un ejemplo de la aplicación de rotación al conejo.

4.3.5. Escalamiento directo del objeto virtual

El objeto se escala por el mismo factor e_n en los tres ejes, *i.e.* para el esqueleto n la matriz E_n de escalamiento es:

$$E_n(e_n, e_n, e_n) = \begin{bmatrix} e_n & 0 & 0 & 0 \\ 0 & e_n & 0 & 0 \\ 0 & 0 & e_n & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

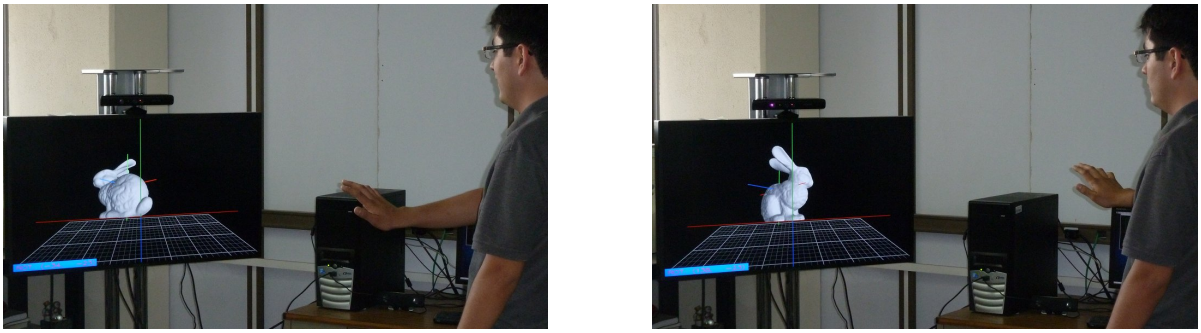


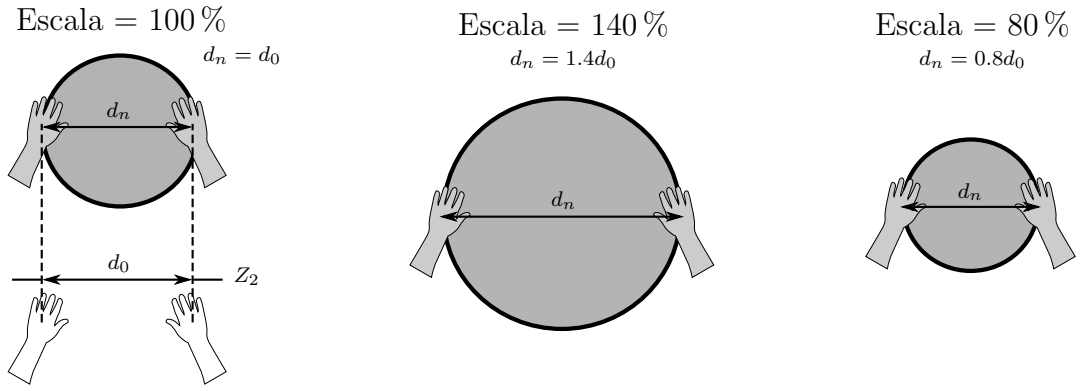
Figura 4.13: Rotación del conejo

Al iniciar el escalamiento se toma la distancia euclidiana d_0 entre las manos como el 100 % del tamaño actual del objeto (ver Figura 4.14(a)). Para cada esqueleto n se toma la distancia d_n (también entre las manos), el factor de escalado e_n se calcula como la razón entre éstas

$$e_n = \frac{d_n}{d_0}$$

El escalamiento inicia con $e_0 = 1$, donde $d_{n=0} = d_0$.

El resultado es que mientras se separen las manos el objeto aumentará de tamaño (como se muestra en la Figura 4.14(b) y en la Figura 4.15), por el contrario, mientras las manos estén más juntas el tamaño de éste disminuirá (Figura 4.14(c)). El procedimiento se repite al acercar las manos al cuerpo y volver a superar de Z_2, *i.e.* se puede continuar agrandando o disminuyendo el tamaño del objeto en caso de que las manos estén demasiado separadas y se salgan de los límites de la malla de acciones (o demasiado juntas) para seguir escalando el objeto.



(a) Inicio del escalamiento (b) Agrandado del objeto (c) Disminución del objeto

Figura 4.14: Escalamiento directo del objeto

4.3.6. Traslación directa del objeto virtual

La traslación se realiza mapeando las coordenadas $\mathbf{m_n}(m_x, m_y, m_z)$ de la mano derecha en la malla de acciones al sistema de coordenadas del mundo del objeto. Para el esqueleto

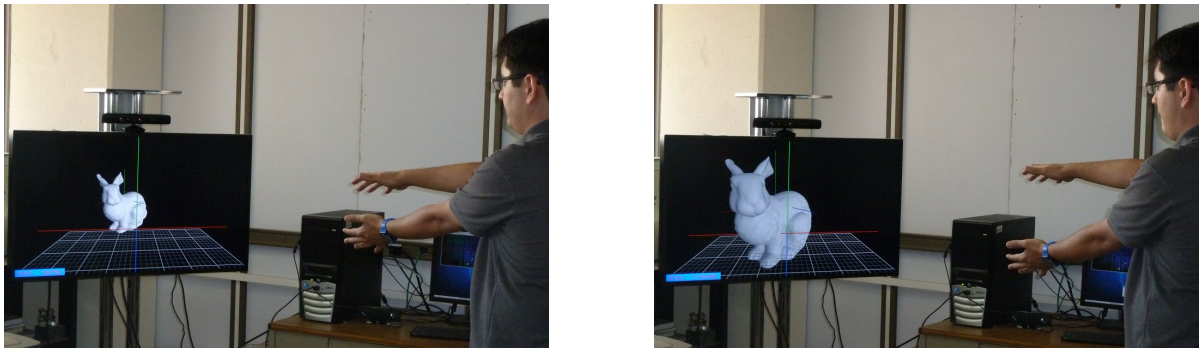


Figura 4.15: Escalamiento del conejo

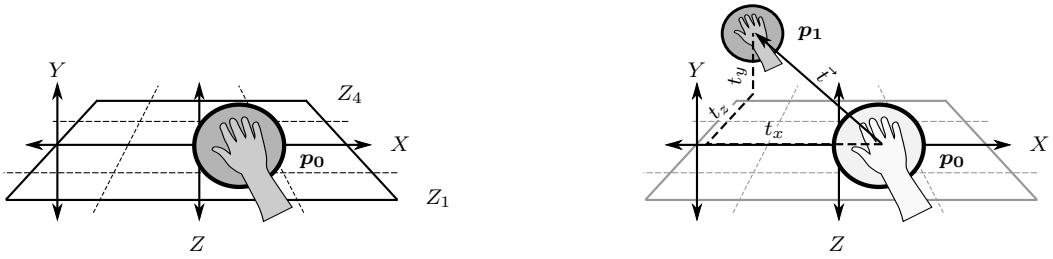
n se tiene la matriz de traslación T_n

$$T_n(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

donde $t_x = 10 \cdot m_x$, $t_y = 10 \cdot m_y$ y $t_z = 10 \cdot m_z$.

El resultado de la traslación es que el objeto sigue el movimiento de la mano derecha.

La Figura 4.16(a) muestra el inicio de la traslación (cuando la mano derecha atraviesa el plano Z_1 de la malla), la mano derecha se encuentra en $\mathbf{m_0}$ y el centro del objeto en $\mathbf{p_0} = 10 \cdot \mathbf{m_0}$. Después de realizar un desplazamiento $\vec{t}(t_x, t_y t_z)$ (como se muestra en la Figura 4.16(b)) el objeto se encuentra en $\mathbf{p_1}$. En este ejemplo la traslación para colocar el objeto en $\mathbf{p_1}$ es $T(\mathbf{p_{1_x}}, \mathbf{p_{1_y}}, \mathbf{p_{1_z}})$.



(a) Inicio de la traslación

(b) Traslación de $\mathbf{p_0}$ a $\mathbf{p_1}$

Figura 4.16: Traslación directa del objeto

En la Figura 4.17 se presenta un ejemplo de traslación del conejo. El conejo es trasladado hacia atrás al empujar la mano derecha al frente del cuerpo.

La traslación se pausa mientras la mano izquierda se encuentre adelante de Z_1 y se detiene si al estar pausada se regresa primero la mano derecha y luego la izquierda junto al cuerpo (antes de Z_1).

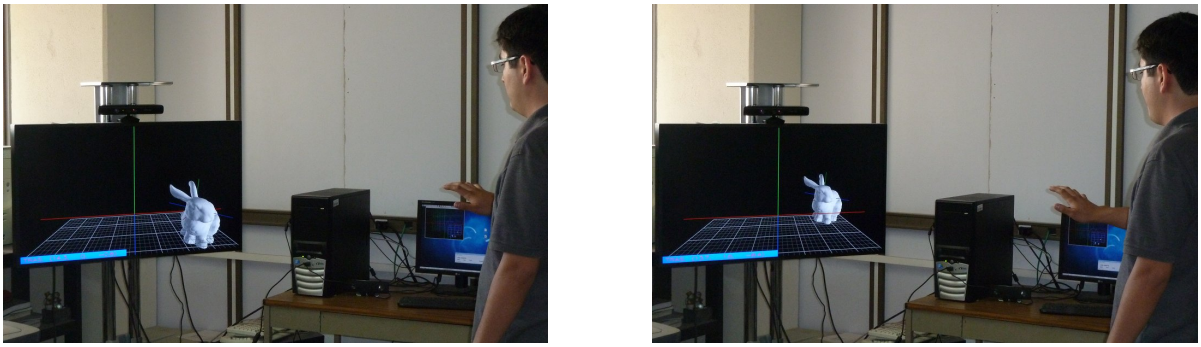


Figura 4.17: Traslación del conejo

Capítulo 5
Conclusiones

Se implementó un sistema que permite la construcción de una interfaz natural de usuario. El sistema traduce los movimientos del cuerpo a comandos de interacción utilizando el sensor Kinect. La implementación fue realizada en el lenguaje C/C++ utilizando las bibliotecas Qt, *OpenNI/NITE*, OpenGL, XTest, libg3d y Phonon en un entorno GNU/-Linux.

Como parte de este trabajo de tesis se realizó un estudio sobre las interfaces naturales de usuario y la interacción natural para controlar aplicaciones en la computadora.

Se estudió el funcionamiento de Kinect y las capacidades del rastreo del esqueleto de *OpenNI*, lo que nos permitió explorar y experimentar con los movimientos del cuerpo que podríamos considerar *naturales* y que fuesen fáciles de aprender en pocas repeticiones, es por ello que se optó la detección secuencias de movimientos de las manos definidos por una secuencia de reglas o acciones. La combinación de acciones para definir comandos brinda flexibilidad para detectar movimientos de una o ambas manos, permitiendo utilizar el sistema para controlar una variedad de aplicaciones bajo un paradigma de interacción natural.

El mecanismo desarrollado para detectar las acciones se adapta al tamaño del usuario y se adapta a la posición de éste en la escena (mientras se mantenga en el rango de vista de Kinect), debido a esto la interfaz natural funciona diferente a la visual, sin que el usuario tenga la necesidad de ver la pantalla. Para brindar una retroalimentación se agregó la reproducción de sonido.

Inicialmente la detección de comandos se diseñó en tres etapas: (1) la generación fuera de línea del árbol de comandos y su almacenamiento un archivo, (2) la carga y reconstrucción en memoria del árbol durante la inicialización del sistema y (3) la detección de comandos basada en la transición de estados en el árbol causada por las acciones detectadas en cada esqueleto. Sin embargo esta última etapa causaba una reducción significativa de la velocidad del sistema (de 30 a 24 cps) repercutiendo en la efectividad de la detección de comandos por la necesidad de buscar transiciones en el árbol ocasionadas por las acciones

detectadas.

Dado que no es necesario modificar las definiciones de los comandos en tiempo de ejecución, *i.e.* el árbol de comandos se mantiene constante, se optó por diseñar e implementar la generación de código C de una máquina de estados con las mismas reglas del árbol de comandos, reemplazando la estructura de datos del árbol en memoria y la búsqueda inherente en éste por una serie de comparaciones lógicas simples entre la acción detectada y los nodos del árbol.

Se incorporó un filtro de Kalman para reducir el ruido de las articulaciones del esqueleto para las aplicaciones que requieran trabajar con las coordenadas del esqueleto, tal caso se presentó en dos de las aplicaciones de prueba: el ratón virtual y el manipulador de objetos virtuales.

Finalmente este trabajo de tesis produjo los siguientes resultados:

- Una herramienta desarrollada con software libre, extendiendo el uso de Kinect a una computadora convencional, mismo que establece una base sólida para construir (o adaptar) aplicaciones para interactuar bajo un paradigma de interacción natural basada en un mecanismo de detección e interpretación de acciones o desplazamientos de las manos y secuencias de acciones. Tres aplicaciones de prueba construidas sobre el sistema de interfaz natural para explorar las posibilidades de uso del mismo. El presente documento de tesis de maestría.

5.1. Trabajo a futuro

El sistema desarrollado presenta una forma simple de controlar interfaces gráficas mediante movimientos del cuerpo utilizando el sensor Kinect, por lo que la mayoría de las sugerencias a realizar como trabajos futuros se refieren a extender las capacidades del sistema, asímismo se presentan algunas propuestas de aplicaciones que podrían hacer uso del sistema.

- Construir una malla de acciones sobre cada mano para detectar acciones con movimientos de los dedos, lo cual requeriría aplicar una técnica de clasificación sobre las imágenes de profundidad para obtener las coordenadas 3D de los dedos. Esto permitirá desarrollar aplicaciones con las que se pueda interactuar directamente con los dedos (*e.g.* un teclado virtual).
- Adaptar el sistema para que funcione con un segundo Kinect, mismo que podría usarse para seguir detectando comandos cuando el usuario de la espalda al primer Kinect.
- Agregar un módulo de detección de gestos con las manos (*e.g.* abrir o cerrar mano).
- Experimentación detectando comandos de dos o más usuarios para interactuar en una aplicación cooperativa.
- Extender la detección de comandos para interactuar con aplicaciones remotas.

- Utilizar el sistema en una interfaz para modelar o deformar objetos virtuales.
- Experimentar con la interacción en un ambiente de realidad aumentada.

Bibliografía

[1] Stuart K. Card, Allen Newell, and Thomas P. Moran. *The Psychology of Human-Computer Interaction*. L. Erlbaum Associates Inc. ISBN: 0898592437, Hillsdale, NJ, USA, 1983.

[2] Jean Vanderdonckt, Joëlle Coutaz, Gaëlle Calvary, and Adrian Stanciulescu. *Multimodality for Plastic User Interfaces: Models, Methods, and Principles*, chapter 4, páginas 61–84. Springer, 2008. D. Tzovaras (ed.), Lecture Notes in Electrical Engineering, Springer-Verlag, Berlin, 2007.

[3] Jhilmil Jain, Arnold Lund, and Dennis Wixon. The future of natural user interfaces. In *Proceedings of the 2011 annual conference extended abstracts on human factors in computing systems*, CHI EA '11, páginas 211–214, Vancouver, BC, Canada, 2011. ACM.

[4] Alessandro Valli. The design of natural interaction. *Multimedia Tools and Applications*, 38:295–305, 2008.

[5] Alessandro Valli. Notes on natural interaction, 2005. Disponible en `http://www.idemployee.id.tue.nl/g.w.m.rauterberg/Movies/NotesOnNaturalInteraction.pdf`. Accesado el 26/08/2012.

[6] Matthias Rauterberg and Patrick Steiger. Pattern recognition as a key technology for the next generation of user interfaces. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 4 of *SMC '96*, páginas 2805–2810. Piscataway: IEEE, 1996.

[7] Steve Mann. *Intelligent image processing*. IEEE John Wiley and Sons. ISBN: 0-471-40637-6, New York, NY, USA, noviembre 2001.

[8] OpenNI. Open Natural Interaction: organización y biblioteca de desarrollo de aplicaciones de interacción natural. Sitio Web. Disponible en `http://www.openni.org/`. Accesado el 19/08/2012.

[9] Rick Rogers. Kinect with Linux. *Linux Journal*, 207:50–53, julio 2011.

[10] Microsoft Research. *Kinect for Windows SDK Programming Guide*. Microsoft. Disponible en `http://msdn.microsoft.com/en-us/library/hh855348.aspx`. Accesado el 26/08/2012.

[11] PrimeSense. PrimeSense Natural Interaction. Sitio Web. Disponible en `http://www.primesense.com/`. Accesado el 19/08/2012.

[12] NITE. Implementación de rastreo del esqueleto de PrimeSense para OpenNI. Sitio Web. Disponible en `http://www.primesense.com/Nite/`. Accesado el 19/08/2012.

[13] Evan Suma, Belinda Lange, Albert Rizzo, David M. Krum, and Mark Bolas. FAAST: the flexible action and articulated skeleton toolkit. In *Proceedings of the IEEE Virtual Reality Conference*, IEEE VR '11, páginas 245–246, Singapur, marzo 2011.

[14] KinEmote. Kinemote: Controla el ratón con la mano. Sitio Web. Disponible en `http://www.kinemote.net/`. Accesado el 26/08/2012.

[15] Maged N. Kamel Boulos, Bryan J. Blanchard, Cory Walker, Julio Montero, Aalap Tripathy, and Ricardo Gutierrez-Osuna. Web GIS in practice X: a Microsoft Kinect natural user interface for Google Earth navigation. *International Journal of Health Geographics*, 10:1–14, 2011.

[16] E. S. Santos, E. A. Lamounier, and A. Cardoso. Interaction in augmented reality environments using Kinect. In *Proceedings of the XIII Symposium on Virtual Reality*, CVPR '91, páginas 112–121, mayo 2011.

[17] Samuel A. Lacolina, Alessandro Soro, and Riccardo Scateni. Natural exploration of 3D models. In *Proceedings of the 9th ACM SIGCHI Italian Chapter International Conference on Computer-Human Interaction: Facing Complexity*, CHItaly '11, páginas 118–121, Alghero, Italy, 2011. ACM.

[18] L. Gallo, A. P. Placitelli, and M. Ciampi. Controller-free exploration of medical image data: Experiencing the Kinect. In *Proceedings of the 24th International Symposium on Computer-Based Medical Systems*, CBMS '11, páginas 1–6, junio 2011.

[19] Ming-Chun Huang, Ethan Chen, Wenyao Xu, and Majid Sarrafzadeh. Gaming for upper extremities rehabilitation. In *Proceedings of the 2nd Conference on Wireless Health*, WH '11, páginas 27:1–27:2, San Diego, California, 2011. ACM.

[20] G. Odowichuk, S. Trail, P. Driessen, W. Nie, and W. Page. Sensor fusion: Towards a fully expressive 3D music control interface. In *2011 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, PacRim '11, páginas 836–841, agosto 2011.

[21] Johnny Chung Lee. In search of a natural gesture. *The ACM Magazine for Students XRDS: Crossroads*, 16:9–12, junio 2010.

[22] Jasmin Blanchette and Mark Summerfield. *C++ gui programming with Qt 4*. Prentice Hall Press. ISBN: 9780137143979, Upper Saddle River, NJ, USA, segunda edición, 2008.

[23] Douglas C. Engelbart and William K. English. A research center for augmenting human intellect. In *Proceedings of the joint computer conference, parte I*, AFIPS '68, páginas 395–410, New York, NY, USA, Diciembre 9-11, 1968. ACM.

[24] Kin Yeung Wong. Cell phones as mobile computing devices. *IT Professional*, 12(3):40–45, mayo 2010.

[25] Steve Mann. Continuous lifelong capture of personal experience with eyetap. In *Proceedings of the the 1st ACM workshop on Continuous archival and retrieval of personal experiences*, CARPE '04, páginas 1–21, New York, NY, USA, 2004. ACM.

[26] Richard A. Bolt. Put-that-there: Voice and gesture at the graphics interface. In *Proceedings of the 7th annual conference on Computer Graphics and interactive techniques*, SIGGRAPH '80, páginas 262–270, Seattle, Washington, USA, 1980. ACM.

[27] UI Group Community. Sitio web. Disponible en `http://nuigroup.com/log/comments/forums_launched/`. Accesado: 19/06/2012.

[28] Pranav Mistry and Pattie Maes. Sixthsense: a wearable gestural interface. In *Proceedings of the ACM SIGGRAPH ASIA 2009 Sketches*, SIGGRAPH ASIA '09, páginas 11:1–11:1, Yokohama, Japón, 2009. ACM.

[29] PrimeSense. Sistema en chip PS1080. Sitio Web, Octubre 2011. Disponible en `http://www.primesense.com/en/company-profile/114-the-ps1080`. Accesado el 26/08/2012.

[30] The teardown: the Kinect for Xbox 360. *The Engineering and Technology Journal*, 6(3):94–95, abril 2011.

[31] Zalevsky Zeev and Shpunt Alex. Method and system for object reconstruction, marzo 2006. Patente disponible en `http://www.cs.cmu.edu/afs/cs/academic/class/15869-f11/www/readings/primesense07_patent.pdf`. Accesado el 26/08/2012.

[32] Reinhard Klette. *Computer vision: three-dimensional data from images*. Springer-Verlag, ISBN: 9813083719, Singapur, 2001.

[33] Yi Ma, Stefano Soatto, Jana Kosecka, and Shankar S. Sastry. *An Invitation to 3-D Vision: From Images to Geometric Models*. Interdisciplinary applied mathematics: Imaging, vision, and graphics. Springer. ISBN: 9780387008936, 2004.

[34] A.Shashua S. Avidan. Trajectory triangulation: 3D reconstruction of moving points from a monocular image sequence. *Pattern Analysis and Machine Intelligence (PAMI)*, 22:248–357, 2000.

[35] Y. G. Leclerc and A. F. Bobick. The direct computation of height from shading. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR '91, páginas 552–558, junio 1991.

[36] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, páginas 1297–1304, Washington, DC, USA, 2011. IEEE Computer Society.

[37] Christopher Bishop. Embracing uncertainity: Applied machine learning comes of age. In *Procedings of the International Conference on Machine Learning*, ICML '11, Bellevue, Washington, USA, 2011. Video de la conferencia disponible en `http://techtalks.tv/talks/54443/`. Accesado el 25/08/2012.

[38] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: principles and practice*. Addison-Wesley Longman Publishing Co., Inc. ISBN: 0-201-12110-7, Boston, MA, USA, segunda edición, 1990.

[39] PrimeSense. Sensorkinect: Módulo de acceso a kinect para openni. Repositorio, 2012. Disponible en `https://github.com/avin2/SensorKinect`. Accesado el 28/08/2012.

[40] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, segunda edición, 2004.

[41] Nicolas Burrus. RGBDemo: Software de código abierto para visualizar, calibrar y procesar la salida de las cámaras de Kinect. Sitio Web. Disponible en `http://labs.manctl.com/rgbdemo/`. Accesado el 26/08/2012.

[42] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[43] Phonon. Framework multimedia estándar de KDE 4 en Qt. Sitio Web, Marzo 2012. Disponible en `http://qt-project.org/doc/qt-4.8/phonon-overview.html`. Accesado el 26/08/2012.

[44] Markus Dahms. libg3d: GNU Biblioteca para cargar modelos virtuales de diversos formatos. `http://gna.org/projects/libg3d/`, 2006–2009.

[45] Markus Dahms. G3DViewer: GNU Visualizador de objetos virtuales basado en libg3d. `http://gna.org/projects/g3dviewer`, 2006–2009.

A. Código C

Código 1: Almacenamiento de la imagen

```
/**
Almacenamiento o actualizaci\'on de la nueva imagen
El m\'etodo "extractFrame" es el que cambia de acuerdo al tipo de imagen a extraer y act
    \'ua sobre el buffer "ibuf" que se encuentre libre en el momento de la actualizaci\'
    on del nodo generador de imagen
  \see (C\'odigo 3 del ap\'endice B)  DepthImageHandler::extractFrame(bool)
  \see (C\'odigo 4 del ap\'endice B)  RGBImageHandler::extractFrame(bool)
  \see (C\'odigo 5 del ap\'endice B)  IRImageHandler::extractFrame(bool)
 **/
void IImageHandler::update() {
    //Si el manejador de imagen est\'a deshabilitado, ignorar la extracci\'on y
        almacenamiento
    if(!m_enabled) return;
    if(m_CurrBuff==1) { //Almacenar en el buffer 1 y habilitar el 0
        m_mWriteBuff1.lock();
            extractFrame(m_CurrBuff);
            m_CurrBuff=0;
        m_mWriteBuff1.unlock();
    } else { //Almacenar en el buffer 0 y habilitar el 1
        m_mWriteBuff0.lock();
            extractFrame(m_CurrBuff);
            m_CurrBuff=1;
        m_mWriteBuff0.unlock();
    }
    //Indicar que se ha terminado de almacenar la nueva imagen
    emit frameProcessed();
}
```

Código 2: Dibujado de la imagen

```
/**
  \brief Dibujado de la imagen
  La imagen de profundidad, color e infrarroja se dubuja con el este mismo codigo
      mediante glDrawPixels()
 **/
void KGLImageWidget::paintGL() {
    // Limpiar el buffer de dibujado
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    //glViewport(0, 0,  m_widgetWidth, m_widgetHeight);

    // Configurar la matriz de proyecci\'on en OpenGL
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0, m_widgetWidth, m_widgetHeight, 0, -1.0, 1.0);

```

```
15      // Configurar la matriz de vista del modelo
16      glMatrixMode(GL_MODELVIEW);
17      glLoadIdentity();
18
19      glDepthMask(0); //Solo para dibujar el esqueleto en 2D sobre la imagen
20          //Escalar la imagen conforme al escalado de la ventana (tama\~no por default de
                640 por 480 pixeles)
21          glPixelZoom ((float)m_widgetWidth / 640.f, (float)m_widgetHeight / 480.f);
22      //Dibujado de la imagen almacenada en "m_frame" con glDrawPixels
23      glDrawPixels(640,480,GL_RGB,GL_UNSIGNED_BYTE,m_frame);
24  }
```

Código 3: Extracción de imagen de profundidad

```
1   /**
2   Extracci\'on de imagen de profundidad
3   La imagen de profundidad se extrae del generador de profundidad "g_DepthGenerator" y se
        almacena en el buffer "ibuf" de las im\'agenes de profundidad.
4    **/
5   void DepthImageHandler::extractFrame(bool ibuf) {
6       xn::SceneMetaData smd;
7       xn::DepthMetaData dmd;
8       //Obtener el arreglo de pixeles de la imagen de profundidad, y de los pixeles que
            forman a los usuarios encontrados en la imagen
9       g_DepthGenerator->GetMetaData(dmd);
10      g_UserGenerator->GetUserPixels(0, smd);
11
12      const XnDepthPixel* pDepth = dmd.Data();
13      const XnLabel* pLabels = smd.Data();
14      int y=0, x=0;
15      unsigned char *buffptr = m_ImageBuffer[ibuf];
16      //Recorrer las filas de pixeles de la imagen
17      for(int y=m_ImageHeight-1; y>-1; y--) {
18          //Tomar el primer pixel de la fila "y"
19          unsigned char *pixelptr = &buffptr[y*m_ImageWidth*m_ImageChannels];
20          //Recorrer las columnas de pixeles de la imagen
21          for(x=0; x<m_ImageWidth; x++) {
22              unsigned depth = *pDepth;
23              unsigned label = *pLabels;
24              unsigned maxdist=10000;
25              //Normalizar los pixeles de profundidad
26                  //Si depth==0 es inv\'alido.
27                  //Si depth==255 es el pixel m\'as cercano.
28                  //Si depth==1 es el pixel m\'as lejano
29
30              if(depth>maxdist) depth=maxdist;
31              if(depth) depth = (maxdist-depth)*255/maxdist+1;
32
33              //Convirtiendo la imagen a RGB (3 bytes por pixel)
34              if(label) { //Si el pixel etiqueta a un usuario, darle el color del usuario
35                  pixelptr[0] = SkeletonColors[label][0]*2*depth/255;
36                  pixelptr[1] = SkeletonColors[label][1]*2*depth/255;
37                  pixelptr[2] = SkeletonColors[label][2]*2*depth/255;
38              } else {    //El pixel es no etiqueta a un usuario, darle un color RGB
39                  pixelptr[0] = depth*127/255;
40                  pixelptr[1] = depth*63/255;
41                  pixelptr[2] = depth*0/255;
42              }
43              pDepth++;
44              pixelptr+=m_ImageChannels;
45              pLabels++;
46          }
47      }
48  }
```

Código 4: Extracción de imagen de color

```
/**
Extracci\'on de imagen de color
La imagen de color se extrae del "g_ImageGenerator" y se almacena en el buffer "ibuf" de las im\'agenes de color.
 **/
void RGBImageHandler::extractFrame(bool ibuf) {
    //Obtener el arreglo de pixeles de la imagen de color.
    xn::ImageMetaData imd;
    g_ImageGenerator->GetMetaData(imd);

    const XnUInt8 *pImg = imd.Data();

    unsigned char *buffptr = m_ImageBuffer[ibuf];
    //Recorrer las filas de pixeles de la imagen
    for(int y=m_ImageHeight-1; y>-1; y--) {
        //Posicionarse en el primer pixel de la fila "y"
        unsigned char *pixelptr = &buffptr[y*m_ImageWidth*m_ImageChannels];
        //Recorrer las columnas de pixeles de la imagen
        for(int x=0; x<m_ImageWidth; x++) {
            //Copiar los valores para rojo, verde y azul
            pixelptr[0] = pImg[0];
            pixelptr[1] = pImg[1];
            pixelptr[2] = pImg[2];

            pImg+=3;
            pixelptr+=3;
        }
    }
}
```

Código 5: Extracción de imagen infrarroja

```
/**
Extracci\'on de im\'agen infrarroja
La imagen infrarroja se extrae del generador "g_IRGenerator" y se almacena en el buffer "ibuf" de las im\'agenes infrarrojas.
 **/
void IRImageHandler::extractFrame(bool ibuf) {
    //Obtener el arreglo de pixeles de la imagen infrarroja.
    xn::IRMetaData imd;
    const XnIRPixel *pIR;

    g_IRGenerator->GetMetaData(imd);
    pIR = imd.Data();
    unsigned char *buffptr = m_ImageBuffer[ibuf];
    //Recorrer las filas de pixeles de la imagen
    for(int y=m_ImageHeight-1; y>-1 ; y--) {
        //Posicionarse en el primer pixel de la fila "y"
        unsigned char *pixelptr = &buffptr[y*m_ImageWidth*m_ImageChannels];
        //Recorrer las columnas de pixeles de la imagen
        for(int x=0; x<m_ImageWidth ; x++) {
            //Convertir el pixel a RGB
            unsigned ir = *pIR;
            pixelptr[0] = ir;
            pixelptr[1] = ir;
            pixelptr[2] = ir;
            pixelptr+=3;
            pIR++;
        }
    }
}
```

Código 6: Dibujado del esqueleto 2D sobre la imagen de profundidad

```
void KGLDepthImageWidget::paintGL(){
```

```
2       //C\'odigo 2 del ap\'endice B
3       KGLImageWidget::paintGL();
4
5       ...
6
7       //Dibujar las l\'ineas entre las articulaciones del esqueleto "skel"
8       glBegin(GL_LINES);
9           drawLimb2D(skel,Head,Neck);
10
11          drawLimb2D(skel,Neck,LeftShoulder);
12          drawLimb2D(skel,LeftShoulder,LeftElbow);
13          drawLimb2D(skel,LeftElbow,LeftHand);
14
15          drawLimb2D(skel,Neck,RightShoulder);
16          drawLimb2D(skel,RightShoulder,RightElbow);
17          drawLimb2D(skel,RightElbow,RightHand);
18
19          drawLimb2D(skel,LeftShoulder,Torso);
20          drawLimb2D(skel,RightShoulder,Torso);
21
22          drawLimb2D(skel,Torso,LeftHip);
23          drawLimb2D(skel,LeftHip,LeftKnee);
24          drawLimb2D(skel,LeftKnee,LeftFoot);
25
26          drawLimb2D(skel,Torso,RightHip);
27          drawLimb2D(skel,RightHip,RightKnee);
28          drawLimb2D(skel,RightKnee,RightFoot);
29
30          drawLimb2D(skel,LeftHip,RightHip);
31      glEnd();
32  }
33
34  ...
35
36  void KGLDepthImageWidget::drawLimb2D(const Skeleton &skel, SkeletonJoints j1,
        SkeletonJoints j2) {
37      glVertex2f(skel.joints2d[j1].x*((float)m_widgetWidth/640.f), skel.joints2d[j1].y*((
            float)m_widgetHeight/480.f));
38      glVertex2f(skel.joints2d[j2].x*((float)m_widgetWidth/640.f), skel.joints2d[j2].y*((
            float)m_widgetHeight/480.f));
39  }
```

Código 7: Código C del árbol de comandos del ratón virtual

```
1   /*********************************************************************
2     C'odigo generado autom'aticamente a partir de la lista de comandos:
3    "/home/speralta/Tesis/Development/Qt/lstComandos"
4     <0> 642||578||641||577@1,
5     <1> 1154||1090||1153||1089@1,
6     <2> 1160||1096||1156||1092@1,648||584||644||580@1,
7     <3> 1160||1096||1156||1092@1,2184||2120||2180||2116@1,
8    'Arbol de comandos:
9    "QKEVENT_TREE_ROOT:3, 3891#1024@1:2, 3900#1024@1:0, <1>, 3900#512@1:0, <0>, 3891#2048@1
         :0, <3>, 3891#512@1:0, <2>, "
10   *********************************************************************/
11  void QKinectEvent2::detectCommand(QVector<ACTION> acciones) {
12      if(acciones.size()==0) return;
13      if(acciones.at(0).act_code == -1 || acciones.at(1).act_code ==-1) return;  /*Si
            alguna mano est'a en la separaci'on entre las celdas de la malla, ignorar*/
14      if(acciones[0].act_code==0 && acciones[1].act_code==0) {                    /*Si las
            manos est'an fuera de la malla*/
15          estado = q0;
16          return;
17      }
18      switch(estado) {
19      case q0:  /*QKEVENT_TREE_ROOT:3*/
20          estado = q0;
21          if(acciones[1].act_code!=0 && acciones[0].act_code==0) {
```

```
            if( (acciones[1].act_code&3891)==1024 ) {
                estado = q1;
            } else
                if( (acciones[1].act_code&3900)==1024 ) {
                    QApplication::postEvent(m_event_receiver, new QEvent(QEvent::Type(
                        eventBaseNumber+1)));
                } else
                    if( (acciones[1].act_code&3900)==512 ) {
                        QApplication::postEvent(m_event_receiver, new QEvent(QEvent::Type
                            (eventBaseNumber+0)));
                    }
        }
        break;
    case q1:  /*3891#1024@1:2*/
        estado = q0;
        if(acciones[1].act_code!=0 && acciones[0].act_code==0) {
            if( (acciones[1].act_code&3891)==1024 ) {
                estado = q1;
            } else
                if( (acciones[1].act_code&3891)==2048 ) {
                    QApplication::postEvent(m_event_receiver, new QEvent(QEvent::Type(
                        eventBaseNumber+3)));
                } else
                    if( (acciones[1].act_code&3891)==512 ) {
                        QApplication::postEvent(m_event_receiver, new QEvent(QEvent::Type
                            (eventBaseNumber+2)));
                    }
        }
        break;
    }
}/*Fin detectCommand*/
```

Código 8: Código C del árbol de comandos del navegador de menús

```
/**************************************************************************
    C'odigo generado autom'aticamente a partir de la lista de comandos:
    "/home/speralta/Tesis/Development/Qt/lstComandos"
        <0> 1064||1048||1060||1044||552||536||548||532@0
            &&1160||1096||1156||1092||648||584||644||580@1,
        <1> 546||530||545||529@0&&642||578||641||577@1,
        <2> 1154||1090||1153||1089@1,1058||1042||1057||1041@1,
        <3> 1058||1042||1057||1041@0,1154||1090||1153||1089@0,
        <4> 1064||1048||1060||1044@0,1058||1042||1057||1041@0,
        <5> 1090||1089@0&&1058||1057@1,
    'Arbol de comandos:
        "QKEVENT_TREE_ROOT:6, 4035#1024@0:1, 4044#1024@0:1, 3900#1024@1:1, 2499#0@0
            &&2355#0@1:0, <0>, 4044#512@0&&3900#512@1:0, <1>, 4092#1088@0&&4092#1056@1:0,
             <5>, 4044#1024@0:0, <4>, 3900#1024@0:0, <3>, 4044#1024@1:0, <2>, "
 **************************************************************************/

typedef enum estados{ q0=0, q1, q2, q3 } estados;   /*Poner en el .h de la clase
    QKinectEvent2*/

void QKinectEvent2::detectCommand(QVector<ACTION> acciones) {
    if(acciones.size()==0) return;
    if(acciones.at(0).act_code == -1 || acciones.at(1).act_code ==-1) return;  /*Si
        alguna mano est'a en la separaci'on entre las celdas de la malla, ignorar*/
    if(acciones[0].act_code==0 && acciones[1].act_code==0) {                   /*Si las
        manos est'an fuera de la malla*/
        estado = q0;
        return;
    }
    switch(estado) {
        case q0:  /*QKEVENT_TREE_ROOT:6*/
            estado = q0;
            if(acciones[0].act_code!=0 && acciones[1].act_code==0) {
                if( (acciones[0].act_code&4035)==1024 ) {
                    estado = q1;
```

```
29                  } else
30                  if( (acciones[0].act_code&4044)==1024 ) {
31                      estado = q2;
32                  }
33              }
34              else
35              if(acciones[1].act_code!=0 && acciones[0].act_code==0) {
36                  if( (acciones[1].act_code&3900)==1024 ) {
37                      estado = q3;
38                  }
39              }
40              else
41              if(acciones[0].act_code!=0 && acciones[1].act_code!=0) {
42                  if( (acciones[0].act_code&2499)==0 && (acciones[1].act_code&2355)==0) {
43                      QApplication::postEvent(m_event_receiver, new QEvent(QEvent::Type(
                            eventBaseNumber+0)));
44                  } else
45                  if( (acciones[0].act_code&4044)==512 && (acciones[1].act_code&3900)==512)
                        {
46                      QApplication::postEvent(m_event_receiver, new QEvent(QEvent::Type(
                            eventBaseNumber+1)));
47                  } else
48                  if( (acciones[0].act_code&4092)==1088 && (acciones[1].act_code&4092)
                        ==1056) {
49                      QApplication::postEvent(m_event_receiver, new QEvent(QEvent::Type(
                            eventBaseNumber+5)));
50                  }
51              }
52              break;
53          case q1:  /*4035#1024@0:1*/
54              estado = q0;
55              if(acciones[0].act_code!=0 && acciones[1].act_code==0) {
56                  if( (acciones[0].act_code&4035)==1024 ) {
57                      estado = q1;
58                  } else
59                  if( (acciones[0].act_code&4044)==1024 ) {
60                      QApplication::postEvent(m_event_receiver, new QEvent(QEvent::Type(
                            eventBaseNumber+4)));
61                  }
62              }
63              break;
64          case q2:  /*4044#1024@0:1*/
65              estado = q0;
66              if(acciones[0].act_code!=0 && acciones[1].act_code==0) {
67                  if( (acciones[0].act_code&4044)==1024 ) {
68                      estado = q2;
69                  } else
70                  if( (acciones[0].act_code&3900)==1024 ) {
71                      QApplication::postEvent(m_event_receiver, new QEvent(QEvent::Type(
                            eventBaseNumber+3)));
72                  }
73              }
74              break;
75          case q3:  /*3900#1024@1:1*/
76              estado = q0;
77              if(acciones[1].act_code!=0 && acciones[0].act_code==0) {
78                  if( (acciones[1].act_code&3900)==1024 ) {
79                      estado = q3;
80                  } else
81                  if( (acciones[1].act_code&4044)==1024 ) {
82                      QApplication::postEvent(m_event_receiver, new QEvent(QEvent::Type(
                            eventBaseNumber+2)));
83                  }
84              }
85              break;
86      }
87  }/*Fin detectCommand*/
```

Código 9: Código C del árbol de comandos de la interfaz de manipulación de objetos virtuales

```
/**********************************************************************
    C'odigo generado autom'aticamente a partir de la lista de comandos:
    "/home/speralta/Tesis/Development/Qt/lstComandos"
        <0> 1160||1096||1156||1092@1,648||584||644||580@1,
        <1> 1058||1042||1057||1041@0,1064||1048||1060||1044@0,
        <2> 1058||1042||1057||1041@0,1154||1090||1153||1089@0,
        <3> 1058||1042||1057||1041@0,546||530||545||529@0,
        <4> 1154||1090||1153||1089@1,1160||1096||1156||1092@1,
        <5> 1154||1090||1153||1089@1,1058||1042||1057||1041@1,
        <6> 1154||1090||1153||1089@1,642||578||641||577@1,
        <7> 1064||1048||1060||1044@0,552||536||548||532@0,
        <8> 1058||1042||1057||1041||546||530||545||529@0
            &&1154||1090||1153||1089||642||578||641||577@1
            ,1064||1048||1060||1044||552||536||548||532@0
            &&1160||1096||1156||1092||648||584||644||580@1,
        <9> 1090||1089@0&&1058||1057@1,
    'Arbol de comandos:
        "QKEVENT_TREE_ROOT:6, 4035#1024@0:1, 4044#1024@0:3, 3891#1024@1:1, 3900#1024@1:3,
             2508#0@0&&2364#0@1:1, 4092#1088@0&&4092#1056@1:0, <9>, 4035#512@0:0, <7>,
            3900#1024@0:0, <2>, 4035#1024@0:0, <1>, 4044#512@0:0, <3>, 3891#512@1:0, <0>,
             3900#512@1:0, <6>, 3891#1024@1:0, <4>, 4044#1024@1:0, <5>, 2499#0@0&&2355#0
            @1:0, <8>, "
 **********************************************************************/

typedef enum estados{ q0=0, q1, q2, q3, q4, q5 } estados;   /*Poner en el .h de la clase
    QKinectEvent2*/

void QKinectEvent2::detectCommand(QVector<ACTION> acciones) {
    if(acciones.size()==0) return;
    if(acciones.at(0).act_code == -1 || acciones.at(1).act_code ==-1) return;  /*Si
        alguna mano est'a en la separaci'on entre las celdas de la malla, ignorar*/
    if(acciones[0].act_code==0 && acciones[1].act_code==0) {                   /*Si las
        manos est'an fuera de la malla*/
        estado = q0;
        return;
    }
    switch(estado) {
        case q0:  /*QKEVENT_TREE_ROOT:6*/
            estado = q0;
            if(acciones[0].act_code!=0 && acciones[1].act_code==0) {
                if( (acciones[0].act_code&4035)==1024 ) {
                    estado = q1;
                } else
                if( (acciones[0].act_code&4044)==1024 ) {
                    estado = q2;
                }
            }
            else
            if(acciones[1].act_code!=0 && acciones[0].act_code==0) {
                if( (acciones[1].act_code&3891)==1024 ) {
                    estado = q3;
                } else
                if( (acciones[1].act_code&3900)==1024 ) {
                    estado = q4;
                }
            }
            else
            if(acciones[0].act_code!=0 && acciones[1].act_code!=0) {
                if( (acciones[0].act_code&2508)==0 && (acciones[1].act_code&2364)==0) {
                    estado = q5;
                } else
                if( (acciones[0].act_code&4092)==1088 && (acciones[1].act_code&4092)
                    ==1056) {
                    QApplication::postEvent(m_event_receiver, new QEvent(QEvent::Type(
                        eventBaseNumber+9)));
                }
            }
            break;
```

```
57          case q1:  /*4035#1024@0:1*/
58              estado = q0;
59              if(acciones[0].act_code!=0 && acciones[1].act_code==0) {
60                  if( (acciones[0].act_code&4035)==1024 ) {
61                      estado = q1;
62                  } else
63                  if( (acciones[0].act_code&4035)==512 ) {
64                      QApplication::postEvent(m_event_receiver, new QEvent(QEvent::Type(
                            eventBaseNumber+7)));
65                  }
66              }
67              break;
68          case q2:  /*4044#1024@0:3*/
69              estado = q0;
70              if(acciones[0].act_code!=0 && acciones[1].act_code==0) {
71                  if( (acciones[0].act_code&4044)==1024 ) {
72                      estado = q2;
73                  } else
74                  if( (acciones[0].act_code&3900)==1024 ) {
75                      QApplication::postEvent(m_event_receiver, new QEvent(QEvent::Type(
                            eventBaseNumber+2)));
76                  } else
77                  if( (acciones[0].act_code&4035)==1024 ) {
78                      QApplication::postEvent(m_event_receiver, new QEvent(QEvent::Type(
                            eventBaseNumber+1)));
79                  } else
80                  if( (acciones[0].act_code&4044)==512 ) {
81                      QApplication::postEvent(m_event_receiver, new QEvent(QEvent::Type(
                            eventBaseNumber+3)));
82                  }
83              }
84              break;
85          case q3:  /*3891#1024@1:1*/
86              estado = q0;
87              if(acciones[1].act_code!=0 && acciones[0].act_code==0) {
88                  if( (acciones[1].act_code&3891)==1024 ) {
89                      estado = q3;
90                  } else
91                  if( (acciones[1].act_code&3891)==512 ) {
92                      QApplication::postEvent(m_event_receiver, new QEvent(QEvent::Type(
                            eventBaseNumber+0)));
93                  }
94              }
95              break;
96          case q4:  /*3900#1024@1:3*/
97              estado = q0;
98              if(acciones[1].act_code!=0 && acciones[0].act_code==0) {
99                  if( (acciones[1].act_code&3900)==1024 ) {
100                     estado = q4;
101                 } else
102                 if( (acciones[1].act_code&3900)==512 ) {
103                     QApplication::postEvent(m_event_receiver, new QEvent(QEvent::Type(
                            eventBaseNumber+6)));
104                 } else
105                 if( (acciones[1].act_code&3891)==1024 ) {
106                     QApplication::postEvent(m_event_receiver, new QEvent(QEvent::Type(
                            eventBaseNumber+4)));
107                 } else
108                 if( (acciones[1].act_code&4044)==1024 ) {
109                     QApplication::postEvent(m_event_receiver, new QEvent(QEvent::Type(
                            eventBaseNumber+5)));
110                 }
111             }
112             break;
113         case q5:  /*2508#0@0&&2364#0@1:1*/
114             estado = q0;
115             if(acciones[0].act_code!=0 && acciones[1].act_code!=0) {
116                 if( (acciones[0].act_code&2508)==0 && (acciones[1].act_code&2364)==0) {
117                     estado = q5;
118                 } else
```

```
119                     if( (acciones[0].act_code&2499)==0 && (acciones[1].act_code&2355)==0) {
120                         QApplication::postEvent(m_event_receiver, new QEvent(QEvent::Type(
                                eventBaseNumber+8)));
121                     }
122                 }
123                 break;
124         }
125     }/*Fin detectCommand*/
```