Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

NEKO: un *tabletop* basada en Kinect para manipulación de objetos virtuales 2D y 3D

Tesis que presenta

Paul Casillas Alcalá

para obtener el Grado de

Maestro en Ciencias

en Computación

Director de tesis:

Jorge Buenabad Chávez

México, Distrito Federal **Noviembre 2012**

Resumen

Desde que el hombre creó las primeras máquinas, las interfaces hombre-máquina han tomado gran importancia, debido a que una interfaz de usuario bien diseñada puede facilitar significativamente el control de sistemas complejos. A pesar del uso habitual y extendido del teclado y el ratón por la mayoría de los usuarios de computadoras, no siempre son las herramientas más adecuadas para interactuar con todas las aplicaciones. Gracias a los recientes avances en la tecnología de sensores, visión por computadora y procesamiento de señales, hoy existen mejores herramientas para la creación de interfaces de usuario donde el control de las aplicaciones se realiza a través de varios sentidos del usuario.

Esta tesis presenta el diseño de NEKO (del inglés *Natural Environment for Kinect-based Object interaction*) un prototipo de *tabletop* que permite el despliegue y manipulación *natural* de un entorno virtual acorde a la percepción del usuario. El primordial objetivo en la construcción de NEKO fue crear un espacio interactivo que permitiera manipular objetos virtuales 2D y 3D a través de las manos de una forma semejante a como se haría sobre una mesa convencional. En particular se buscó evitar las barreras que imponen los dispositivos de entrada y salida convencionales (como el teclado y el ratón). Para cumplir dicha meta se diseñó un prototipo de *tabletop*, con componentes de hardware comercial de bajo costo, que facilitan al usuario interactuar sin la necesidad de dispositivos de control artificial. Asimismo se desarrolló una arquitectura de software capaz de identificar patrones relevantes de las manos y rostro de los usuarios con un tiempo de respuesta corto.

Abstract

Since the time man created the first machines, human-machine interfaces have been very important, as a well-designed user interface may significantly simplify the control of complex systems. Despite the common and widespread use of the keyboard and mouse by most computer users today, these devices are not the most appropriate tools to interact with software applications. Thanks to recent advances on sensor technologies, computer vision algorithms, and signal processing, there are now better tools to create user interfaces whose interaction can be made more natural and through various senses of the users.

This thesis presents the desings of NEKO (*Natural Environment for Kinect-based Object interaction*), a *tabletop* prototype that allows a virtual environment to be displayed and manipulated according to the user's perception. The primary objective in building NEKO was to create an interactive space that allows users to manipulate 2D and 3D virtual objects through their hands, as is similarly done on a conventional table. Particularly, we wanted to avoid the barriers imposed by convetional input/output devices (such as keyboard and mouse). To meet this goal we designed an ad-hoc structure composed of ordinary components and low cost comercial hardware. Furthermore we develop a software architecture able to identify relevant patterns from the face and hands of the user in a short reponse time.

Agradecimientos

Al Centro de Investigación y Estudios Avanzados del I.P.N.
por brindarme las herramientas necesarias en el desarrollo
del presente trabajo de investigación.

Al Consejo Nacional de Ciencia y Tecnología
por los recursos económicos proporcionados
para mi formación académica.

A los profesores del Cinvestav
por sus enseñanzas a lo largo
de la maestría.

A mi asesor de tesis, el Dr. Jorge Buenabad Chávez
por orientarme en el desarrollo de mi tesis.

Sobre todo a Dios y a mis seres queridos
por el apoyo y enseñanzas en estos
dos últimos años.

Índice general

1 Introducción **1**

1.1 Planteamiento del problema 5

1.2 Objetivos del trabajo de investigación 6

1.3 Metodología 6

1.4 Organización de la tesis 7

2 Interacción hombre-computadora **9**

2.1 Interfaces de usuario 10

2.1.1 Interfaz por lotes 10

2.1.2 Interfaz de línea de comandos 12

2.1.3 Interfaz gráfica de usuario 13

2.1.4 Interfaz de usuario natural 16

3 Tecnologías base de NUIs **21**

3.1 Tecnologías ópticas para superficies interactivas 23

3.1.1 Reflexión interna total frustrada 24

3.1.2 Iluminación difusa 25

3.1.3 Plano de luz 26

3.1.4 Iluminación de superficie difusa 27

3.2 Controles de videojuegos 28

3.2.1 Nintendo Wii 29

3.2.2 PlayStation Move 30

3.2.3 Microsoft Kinect 31

3.3 Trabajo relacionado 34

3.3.1 Usando una cámara de profundidad como sensor táctil 34

3.3.2 OmegaDesk 34

3.3.3 Interacciones en el aire 36

3.3.4 Esfera 37

3.3.5 Aumentando mesas interactivas con teclados y ratones 37

4 NEKO **41**

4.1 Aspectos de diseño de NEKO 42
4.2 Prototipo de *tabletop* 44
4.3 Arquitectura de software 50
4.3.1 Inicialización de *tabletop* 52
4.3.2 Captura de eventos 54
4.3.3 Interacción con objetos virtuales 55
4.3.4 Dibujado del entorno virtual 57
4.3.5 Cerrar y guardar configuración 58

5 Modelos de clasificación **61**

5.1 Detección de rostro 63
5.1.1 Procesamiento de marcos de video 63
5.1.2 Preparación de muestras 65
5.1.3 Construcción de la cascada de clasificadores 68
5.1.4 Uso del modelo de clasificación 73
5.2 Detección de manos 75
5.2.1 Procesamiento de marcos de video 77
5.2.2 Preparación de muestras 78
5.2.3 Construcción de la *cascada de clasificadores* 80
5.2.4 Uso del modelo de clasificación 81
5.3 Detección de gestos 82
5.3.1 Detección de pose 84
5.3.2 Construcción del modelo de clasificación 87
5.3.3 Clasificación de movimientos 90

6 Interacción con objetos virtuales **93**

6.1 Interacción 2D 94
6.1.1 Construcción de la superficie táctil 94
6.1.2 Interacción táctil 96
6.2 Interacción 3D 98
6.2.1 Manipulación de objetos virtuales con los dedos 101
6.2.2 Manipulación de objetos virtuales a través de *partículas* 104
6.2.3 Fuerzas físicas 112

7 Resultados **117**
7.1 Interacción 2D 117
7.2 Interacción 3D 120
7.2.1 Manipulación de objetos virtuales con los dedos 120
7.2.2 Manipulación de objetos virtuales a través de *partículas* 121
7.3 Aplicaciones 122

8 Conclusiones **125**
8.1 Trabajo a futuro 127

A Detector de manos basado en OpenNI/NITE **129**

B Detector de pose basado en SVM **133**

Índice de figuras

1.1 Primer prototipo de ratón para computadora desarrollado por Douglas Engelbart. 2
1.2 Evolución de las interfaces de usuario (de izquierda a derecha). 3
1.3 Ejemplos de *tabletops* comerciales. 4
1.4 Prototipo de *tabletop* desarrollado en este trabajo de investigación. 5

2.1 Perforadora de tarjetas IBM 029. 11
2.2 Teletipo ASR-33. 13
2.3 Proyecto de tesis de Sutherland: Sketchpad. 14
2.4 Star, pionera en áreas de trabajo con ventanas e iconos. 15
2.5 Interfaz del sistema operativo LisaOS. 16
2.6 Componentes de una interfaz de usuario natural. 18

3.1 Popularidad de la palabra *multi-touch* en los últimos ocho años. 22
3.2 Elementos que constituyen a FTIR. 24
3.3 Elementos que constituyen a FDI. 25
3.4 Elementos que constituyen a RDI. 26
3.5 Elementos que constituyen a LP. 27
3.6 Elementos que constituyen a DSI. 28
3.7 Wiimote: vista lateral, vista frontal y posterior. 30
3.8 Componentes del sistema *PlayStation Move* 31
3.9 Componentes de Kinect. 32
3.10 Componentes de *OmegaDesk* 35
3.11 Interacción de las dos tabletops propuestas por Hilliges *et al.* 36
3.12 Componentes de hardware que permiten la detección táctil y la proyección de contenido sobre *Esfera* 38

4.1 Arquitectura de hardware de NEKO. 46
4.2 "Huecos" ocasionados por la refracción de los rayos infrarrojos. 47

4.3 Especificaciones del prototipo de mesa interactiva de NEKO 49
4.4 Arquitectura de software de NEKO 52
4.5 Tareas realizadas para la inicialización de la *tabletop*. 53
4.6 Tareas realizadas para la captura de eventos 55
4.7 Tareas realizadas para la interacción con objetos virtuales. 56
4.8 Tareas involucradas para el dibujado del entorno virtual. 58
4.9 Arquitectura de hardware de NEKO. 58
4.10 Tareas involucradas al cerrar la aplicación 59

5.1 Clasificación de un conjunto de atributos $\mathbf{x}$ a una etiqueta de clase y predefinida. 62
5.2 Tareas involucradas en la detección de rostro. 64
5.3 Ajuste en la iluminación de un marco de video. 65
5.4 Ejemplo de rostro extraído. 67
5.5 Ejemplo muestras negativas. 68
5.6 Estructura de una cascada de clasificadores. 68
5.7 Atributos basados en las funciones Haar. 69
5.8 Ejemplo de superposición de el atributo 3a. 70
5.9 Tasa de rechazo alcanzada para cada nodo. 72
5.10 Cálculo de los ángulos α y β. 73
5.11 Cambio de perspectiva de un objeto físico y un objeto virtual. 75
5.12 Esqueleto básico creado con la biblioteca NITE. 76
5.13 Tareas involucradas en la detección de manos. 77
5.14 Ajuste de brillo y contraste a un marco de video capturado por la cámara RGB del Kinect. 78
5.15 Extracción de muestra 79
5.16 Tasa de rechazo alcanzada para cada nodo. 80
5.17 Detección de manos mediante una *cascada de clasificadores*. 81
5.18 Extracción de elementos relevantes de los marcos de video. 82
5.19 Conjunto de movimientos reconocidos como gestos. 83
5.20 Detección de gestos 84
5.21 Poses detectadas mediante un modelo de clasificación. 84
5.22 Detección de dedos. 85
5.23 Puntos y vectores en la trayectoria de un gesto circular. 88
5.24 Separación de dos clases. 88
5.25 Gesto para la calibración de la superficie táctil. 90

6.1 Tareas involucradas en la interacción 2D. 94

6.2 Construcción de la superficie táctil. 95
6.3 Detección de tacto sobre el cristal claro. 96
6.4 Coordenadas Kinect a coordenadas OpenGL. 98
6.5 Tareas involucradas en la interacción 3D. 101
6.6 Cubo contenedor de la malla de un objeto. 102
6.7 Partículas de polvo sobre el aire. 105
6.8 Interacción objeto virtual-partícula. 106
6.9 Cubo contenedor para objetos con dimensiones desiguales. 107
6.10 Construcción de partículas. 108

7.1 Menú construido para la interacción 2D. 118
7.2 Objeto virtual constituido de dos triángulos. 119
7.3 Manipulación de objetos 2D. 119
7.4 Manipulación de un objeto virtual mediante los dedos. 120
7.5 Fuerzas físicas entre objetos virtuales. 121
7.6 Manipulación de un objeto virtual mediante partículas. 123

A.1 Detección de manos y dedos sobre los marcos del Kinect 130

B.1 Muestras ejemplo de posturas de mano. 133
B.2 Conjunto de direcciones que conforman a un polígono de una mano. 134
B.3 Detección de posturas de manos mediante SVM. 135

Índice de tablas

3.1 Comparativo de trabajos relacionados. 39

Capítulo 1

Introducción

"Ciencia es lo que entendemos suficientemente bien como para explicárselo a una computadora. Arte es todo lo demás."

Donald Ervin Knuth

Desde que el hombre creó las primeras máquinas, las interfaces hombre-máquina han tomado gran importancia, debido a que una interfaz de usuario bien diseñada puede facilitar significativamente el control de sistemas complejos. Por ejemplo, el conductor de un automóvil puede controlar la velocidad del vehículo pisando los pedales acelerador y freno aun sin conocer la cantidad precisa de oxigeno y combustible necesarias para mover el conjunto de pistones dentro de su motor o la fricción requerida para detener las ruedas. Es gracias a una interfaz relativamente sencilla que muchas personas pueden conducir, a pesar de su poco o nulo conocimiento sobre mecánica.

La interacción hombre-computadora (HCI, por sus siglas en inglés) es un área de investigación dedicada al diseño de interfaces para sistemas de cómputo utilizados en diversas ramas de la ciencia como: medicina, acelerando la rehabilitación de pacientes [Gama et al., 2012]; psicología, ayudando al tratamiento de fobias [Gregg and Tarrier, 2007]; en robótica, facilitando el control de los robots [Henry et al., 2010]; y en muchas otras, reduciendo las barreras del mundo digital mediante interfaces intuitivas que mejoran la experiencia del usuario.

El diseño de una buena interfaz de usuario es una parte esencial del desarrollo de sistemas de software y/o hardware. Una interfaz de usuario debe ser fácil de usar, intuitiva y enfocada en las necesidades de los usuarios. Aun así, las interfaces para computadoras de escritorio han usado los mismos principios básicos de diseño durante la última década, debido princi-

palmente a la mínima evolución en el paradigma de interacción que rige actualmente sobre los dispositivos de entrada.

Pese a que el ratón y teclado son dispositivos de entrada muy comunes entre la mayoría de los usuarios de computadoras, no siempre son las herramientas más adecuadas para interactuar con todas las aplicaciones. El ratón [Barnes, 1997] fue presentado al público en el centro de convenciones de San Francisco el 9 de diciembre de 1968. Y aún 43 años después, es uno de los dispositivos de entrada predominantes para el uso diario en las computadoras. No obstante, el artilugio mostrado al final de la década de los 60s difiere de los modelos de hoy en día principalmente por aspectos ergonómicos y tecnológicos. En la figura 1.1 se muestra el primer prototipo de ratón, constituido por una caja de madera, un botón, una rueda debajo de la caja y un cable con conector.

Figura 1.1: Primer prototipo de ratón para computadora desarrollado por Douglas Engelbart.

Gracias a los recientes avances en la tecnología de sensores, visión por computadora y procesamiento de señales, actualmente existen mejores herramientas para la creación de nuevos conceptos en interfaces y dispositivos de entrada más allá de lo convencional. Un ejemplo de ello son las *interfaces naturales de usuario*.

El término interfaz de usuario natural (NUI) es una disciplina de interacción con la computadora, enfocada en el uso de habilidades humanas para el control de sistemas o aplicaciones. Es considerado el siguiente paradigma en HCI y está conformado de arquetipos de interacción humana, como el uso del tacto, visión y el habla [Liu, 2010]. Por ello, el significado de *natural* hace énfasis en el uso de habilidades que se adquieren al vivir en el mundo real. NUI tiene como objetivo replicar entornos del mundo real mediante el uso de sensores y habilidades tecnológicas emergentes que permitan una interacción más precisa y óptima entre objetos físicos y digitales.

La figura 1.2 muestra la evolución de la interfaz de usuario en los sistemas de cómputo [Liu, 2010]. El primer prototipo fue la *interfaz por lotes*, donde los dispositivos de entrada eran tarjetas perforadas; para interpretar la salida se usaba una impresora. Posteriormente, las bases de HCI surgieron con la *interfaz de línea de comandos* (CLI), remplazando el uso de tarjetas perforadas por comandos introducidos mediante un teclado. A principio de la decada de los 80s el ratón y el desarrollo de sistemas operativos gráficos encaminaron a una revolución en la creación de interfaces de sistemas informáticos, surgiendo así la *interfaz de usuario gráfica* (GUI), la cual utiliza la metáfora de un escritorio (una interacción semejante a la realizada sobre un escritorio físico).

Figura 1.2: Evolución de las interfaces de usuario (de izquierda a derecha).

Por otra parte, pocas tecnologías existentes proveen una interacción tan rica y fluida como cuando se trabaja sobre una mesa con medios tradicionales (tales como el papel y bolígrafos) sobre todo en espacios colaborativos. Tomando como base los beneficios de estos sitios de trabajo, muchos investigadores han explorado y desarrollado sistemas denominados *tabletops* [Buxton et al., 1985], los cuales permiten la manipulación de contenidos digitales en diversas actividades, principalmente, colaborativas.

Las *tabletops* son mesas interactivas, que permiten manipular contenido digital sobre su superficie, generalmente a través de una interfaz de usuario multitáctil. Debajo o por encima de la mesa se encuentra un proyector que despliega el contenido sobre la superficie del tablero (constituido habitualmente de un marco con acrílico reflejante); se utiliza una cámara infrarroja para localizar objetos físicos que tocan al tablero. Los usuarios interactúan con la mesa al tocar o arrastrar sus manos sobre la superficie, además pueden colocar objetos con etiquetas especiales que la cámara sea capaz de reconocer.

La gran mayoría de *tabletops* son sistemas que permiten a los usuarios realizar tareas muy intuitivas imitando comportamientos muy similares a la vida diaria. Por ejemplo, organizar fotos es bastante natural en una *tabletop* debido a la semejanza de organizar papel sobre una mesa en el mundo real. En la figura 1.3 se muestra dos ejemplos de *tabletops* multitáctiles comerciales: ReacTable y Microsoft Surface. La primera es una *tabletop* multitáctil desarrollada por el Music Technology Group de la Universidad Pompeu Fabra de España y fue diseñada para la creación de música sintética. En cambio, Microsoft Surface es un producto profesional introducido al mercado por Microsoft en abril 17 de 2008 y está fabricada para su uso en el hogar o negocios.

Figura 1.3: Ejemplos de *tabletops* comerciales.

Sin embargo, a pesar de que las *tabletops* ofrecen una manipulación directa mediante el tacto, su área de interacción está restringida a una superficie, con lo cual es complicado realizar técnicas para el despliegue y manipulación de objetos virtuales en tres dimensiones.

La motivación de este trabajo de investigación surge de la necesidad de explorar una forma para mezclar la manipulación de objetos 2D y 3D sobre una *tabletop* a través de las manos sin la necesidad de instrumentar al usuario. En la figura 1.4 se muestra el prototipo de *tabletop* producto de este trabajo de investigación. El propósito fue diseñar una *tabletop* que permitiera al usuario visualizar y manipular objetos virtuales en 2D y 3D de forma semejante a la vida real. La idea básica es que el usuario sea una parte integral de la interfaz y se sienta influenciado de la misma forma que en el mundo real.

Figura 1.4: Prototipo de *tabletop* desarrollado en este trabajo de investigación.

1.1 Planteamiento del problema

Los sistemas de escritorio generalmente están construidos por una interfaz de usuario de dos dimensiones, que requiere el uso de teclado y ratón para realizar la interacción. Estas interfaces deben proporcionar toda la funcionalidad necesaria para que el usuario pueda manipular el sistema, inclusive, cuando se trata de ambientes virtuales u objetos en tres dimensiones. No obstante, algunos aspectos de usabilidad presentes en esas interfaces de software puede ocasionar que algunos usuarios experimenten un cierto grado de frustración hasta para efectuar tareas relativamente sencillas.

La interacción en 3D es crucial para diversas tareas, tales como animaciones, diseño asistido por computadora o entrenamiento virtual. Sin embargo, una gran parte del desarrollo de la interacción con la computadora se ha enfocado en interfaces sobre pantallas bidimensionales por lo que la manipulación y visualización no es del todo apropiada para entornos en 3D, para ello se requiere el uso de interfaces y técnicas que permitan al usuario interactuar de forma adecuada.

El presente trabajo de tesis propone un prototipo de *tabletop* para manipular, a través

de las manos y sus gestos, con objetos virtuales en 2D y 3D. El propósito esencial es disolver las barreras sobre tres dimensiones, proporcionadas por los dispositivos de entrada y salida convencionales, mediante la construcción de una interfaz que permita manipular contenido digital semejante a la vida cotidiana.

De acuerdo a lo planteado anteriormente, se desea dar respuesta con el proyecto de investigación a la siguiente pregunta:

¿Se puede diseñar y construir un prototipo de *tabletop* con componentes de hardware comercial que permita la visualización y manipulación de objetos virtuales 2D/3D de forma semejante a la vida real?

1.2 Objetivos del trabajo de investigación

Objetivo general

Investigar, diseñar y construir un prototipo de *tabletop* con el fin de manipular objetos virtuales 2D y 3D de forma semejante a la vida cotidiana. En particular, evitar las barreras en la manipulación de entornos virtuales que imponen los dispositivos de entrada y salida convencionales (como el teclado y el ratón) mediante una interfaz de usuario natural.

Objetivos específicos

- Diseñar y construir un prototipo de *tabletop* con componentes de hardware comercial con la finalidad de facilitar la reproducción del mismo.
- Diseñar y desarrollar un mecanismo para la manipulación de objetos virtuales 2D/3D sobre la *tabletop*.
- Diseñar y desarrollar un mecanismo para la visualización de objetos virtuales 2D/3D sobre la *tabletop*.

1.3 Metodología

La metodología que se siguió para el desarrollo del proyecto se llevó a cabo conforme a los siguientes pasos:

1. Recopilación y análisis de información relacionada con el desarrollo de *tabletops* y superficies interactivas.

2. Estudio y análisis de las tecnologías de hardware comerciales convenientes para el desarrollo de *tabletops*.

3. Diseño y construcción de un prototipo de *tabletop* mediante tecnologías de hardware comerciales.

4. Estudio y análisis de las tecnologías de software libre que faciliten el desarrollo de los componentes lógicos de la *tabletop*.

5. Diseño y desarrollo del software para la manipulación de objetos virtuales 2D/3D.

6. Realización de pruebas para la manipulación de objetos virtuales 2D/3D.

Cabe mencionar que para el desarrollo del software para la manipulación de objetos virtuales 2D/3D se utilizó un modelo evolutivo, específicamente el modelo incremental [Bennett and Rajlich, 2000]. Mediante este modelo se desarrolló una primera versión del sistema para satisfacer un subconjunto de requisitos esenciales y en versiones posteriores el sistema se incrementó con nuevas funcionalidades.

1.4 Organización de la tesis

Tras explicar la motivación y los objetivos del presente trabajo de investigación en este capítulo a continuación se describe el resto de la organización de esta tesis:

- **Capítulo 2.** Describe la evolución de las interfaces más representativas de la interacción hombre-computadora. Comenzando con las interfaces por lotes hasta las interfaces de usuario natural.

- **Capítulo 3.** Detalla las tecnologías empleadas para la construcción de *tabletops* y superficies interactivas. Asimismo también se analizan los trabajos relacionados con este tema de esta tesis.

- **Capítulo 4.** Se describen los componentes de hardware que instrumentan al prototipo de *tabletop* y los elementos de diseño considerados en su construcción. Además se especifica la arquitectura de software desarrollada para la interacción con el prototipo de *tabletop*.

- **Capítulo 5.** Este capítulo entra en detalle con la arquitectura de software. En especial con los componentes de software para la construcción y uso de modelos de clasificación en la detección de manos, gestos y rostros.
- **Capítulo 6.** De forma similar al capítulo anterior, en este capítulo describen los componentes de software para la manipulación de objetos virtuales en 2D y 3D .
- **Capítulo 7.** En este capítulo se describen los resultados obtenidos de los componentes de software encargados de la manipulación de objetos virtuales 2D y 3D.
- **Capítulo 8.** Finalmente, las conclusiones de la investigación llevada acabo y las posibles ideas a realizar como trabajo a futuro se expresan en este capítulo.

Capítulo 2

Interacción hombre-computadora

"Una imagen vale más que mil palabras. Una interfaz vale más que mil imágenes."

Ben Shneiderman

En la sociedad actual las computadoras son una herramienta esencial en gran parte de las actividades cotidianas. Ya sea en el hogar, la escuela o trabajo, las personas se encuentran en contacto con dispositivos o sistemas de cómputo. Por el consiguiente, los diseñadores y desarrolladores de esos sistemas deben contemplar varios requisitos para garantizar que el sistema realmente resuelva una necesidad, sea fácil de usar y sobre todo utilizado. La interacción hombre-computadora (HCI) busca que se cumplan estos requisitos para que exista una experiencia agradable para los usuarios y de acuerdo a sus necesidades.

HCI ha sido definida de diferentes formas. Algunas definiciones sugieren que se encarga de analizar como las personas utilizan los sistemas de computo para satisfacer, de la mejor manera, las necesidades de los usuarios. Mientras que otros investigadores definen a HCI como un campo dedicado a la investigación y diseño de sistemas de cómputo para seres humanos [Preece and Benyon, 1993]. Sin embargo, todas estas definiciones se pueden resumir en el siguiente concepto:

"La interacción hombre computadora es una disciplina que estudia el diseño, evaluación e implementación de sistemas de cómputo interactivos para seres humanos y los fenómenos de su entorno." [Hew, 1992]

HCI es un área de investigación que surgió en la década de los 80s, inicialmente como un área en ciencias de la computación. Con el paso del tiempo, se ha ido involucrando en otras disciplinas e incorporando nuevas metodologías para la creación de sistemas de hardware y software.

El objetivo de este capítulo es describir la evolución en las interfaces de los sistemas de cómputo y su paradigma de interacción hombre-computadora. El capítulo comienza con las interfaces por lotes y línea de comandos donde sus únicos usuarios profesionales en computación y en tecnologías de información. Esto cambió radicalmente con la aparición de la interfaz gráfica de usuario y las computadoras personales, puesto que se creó un entorno accesible para un mayor número de personas. Finalmente, la interfaz de usuario natural surge a partir de los ambientes aumentados por computadora con el propósito de crear sistemas de cómputo acorde a las habilidades adquiridas por los seres humanos.

2.1 Interfaces de usuario

Varias interfaces de usuarios han evolucionado desde los inicios de la interacción hombre-computadora. Es gracias al incremento de procesamiento y disminución de tamaño y costo en las computadoras que nuevos dispositivos, plataformas e infraestructuras se han creado. Si bien el incremento en el poder de procesamiento ha sido continuo, las interfaces de computadora han evolucionado a un ritmo más lento.

El trayecto de las interfaces de computadora están definidas por fases, cuatro de ellas son las más importantes y representativas. La primera es la interfaz por lotes, la cual utilizaba tarjetas perforadas como dispositivo de entrada. La segunda fase es la interfaz de línea de comandos (CLI) basada en entrada y salida textual. La interfaz gráfica de usuario (GUI) utilizó la metáfora de escritorio en la tercera fase. Finalmente, la cuarta fase corresponde a la interfaz de usuario natural (NUI), la cual, intenta enriquecer la interacción entre humanos y computadoras más allá del uso del teclado y el ratón. A continuación se describe cada una de ellas.

2.1.1 Interfaz por lotes

En la época de las interfaces por lotes, computadoras con gran poder de procesamiento eran escasas y caras. Los equipos más poderosos de aquella época poseían un menor número

de ciclos lógicos por segundo que un horno de microondas actual, y bastante menos que los teléfonos inteligentes de hoy en día. Por ende, las interfaces de usuario eran bastante rudimentarias, principalmente porque el software era diseñado para aprovechar al máximo el carente poder de procesamiento sobre las tareas y no en la interfaz en sí. Las circunstancias de aquellos tiempos obligaban a los usuarios a adaptarse a las computadoras y no al revés.

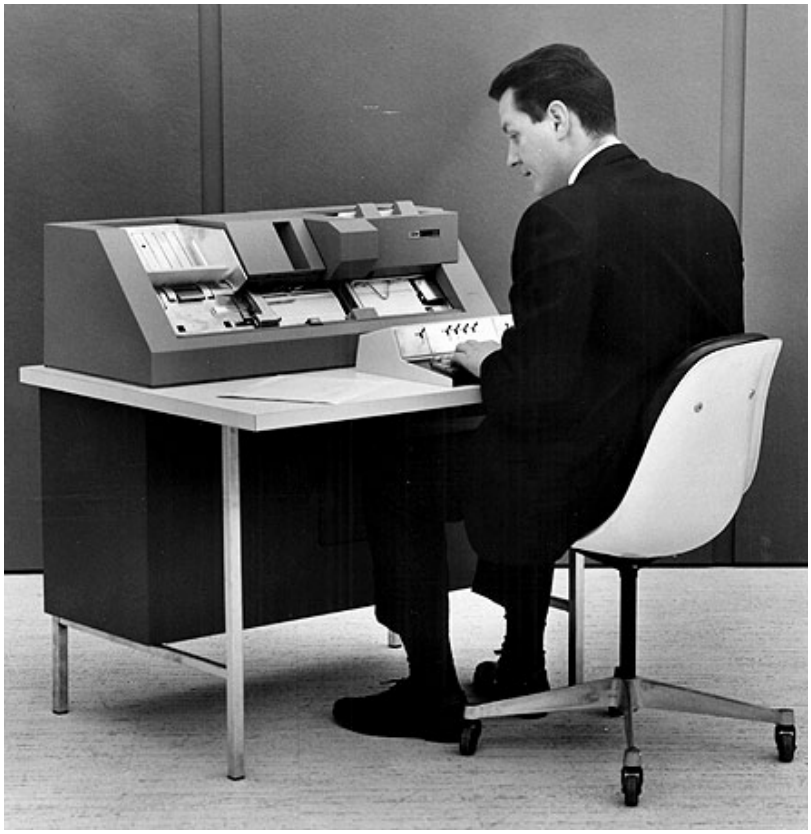


Figura 2.1: Perforadora de tarjetas IBM 029.

En las primeras etapas de desarrollo de computadoras, los usuarios utilizaban como dispositivos de entrada tarjetas perforadas; como salida se imprimían los resultados sobre cintas magnéticas. La realización de un trabajo en la máquina por lotes requería primeramente perforar las tarjetas (las cuales describían el programa y los datos de entrada) en máquinas de escribir especiales, complejas y propensas a fallas mecánicas. Una vez perforadas las tarjetas, los operadores/usuarios las introducían a la computadora donde se iba a ejecutar la tarea. Posteriormente, se imprimían los resultados sobre cintas magnéticas para generar nuevas tarjetas perforadas o para corregir algún error en la ejecución. Estas actividades, en conjunto con el carente poder de procesamiento, prolongaban hasta días (o quizás, con un poco de suerte y destreza, sólo algunas horas) el tiempo requerido para una realizar una tarea. En la figura 2.1 se muestra la IBM 029 card punch, anunciada el 14 de octubre de 1964, utiliza-

da para perforar y verificar las tarjetas. El operador/usuario interactuaba con una interfaz, constituida de un teclado, para describir la tarea y los datos a procesar.

A finales de la década de los 50s, se comenzó a utilizar un *programa monitor*, residente en la computadora, el cual permitía revisar los errores de las tareas en ejecución y controlar sus servicios. Los programas monitores representaron un primer paso en los sistemas operativos y en el diseño de nuevas interfaces de usuario.

2.1.2 Interfaz de línea de comandos

Las interfaces de línea de comandos evolucionaron a partir de los monitores de la interfaz por lotes. CLI utiliza el modelo de interacción petición-respuesta. Las peticiones son expresadas a través de un conjunto de comandos introducidos por el usuario mediante un teclado; posteriormente el sistema procesa la petición y proporciona una respuesta. La ejecución de tareas era mucho más rápida que en los sistemas por lotes, disminuyendo de días a segundos.

Un gran inconveniente de CLI es que el usuario deben conocer los comandos y argumentos necesarios para realizar una operación. Una errata en el comando o en sus argumentos produce un error en la ejecución; el usuario debe encontrar y corregir el error manualmente para ejecutar la tarea. A menudo, es difícil para un usuario novato aprender y recordar la sintaxis (sobre todo si los mensajes de error no son del todo explícitos o fáciles de entender). No obstante, CLI puede llegar a ser bastante flexible y rápido para un usuario experto.

Las primeras CLI utilizaban teletipos[1] (ver figura 2.2). Principalmente porque proporcionaban una interfaz que era familiar para muchos ingenieros y usuarios. Posteriormente, gracias a la amplia adopción de los terminales con desplegado de video (VDT), a mediados de la década de los 70s, se redujo aun más el tiempo de respuesta de los sistemas de línea de comandos. Principalmente porque los caracteres se mostraban más rápido como puntos de fósforo sobre los VDT de lo que un cabezal de impresora los plasmaba en papel. La velocidad de despliegue y la accesibilidad para modificar el texto facilitó a los diseñadores de software desarrollar interfaces visuales, carentes de papel y tinta, en lugar de textuales.

[1]Los teletipos son dispositivos para la transmisión y recepción de telégrafos inventado al inicio del siglo XX.

Figura 2.2: Teletipo ASR-33.

2.1.3 Interfaz gráfica de usuario

En lugar de introducir comandos a través de un teclado, GUI provee una interacción gráfica superior a la proporcionada por CLI. A principio de la década de los 80s, el ratón y el desarrollo de sistemas operativos gráficos encaminaron a una revolución en la creación de interfaces de sistemas informáticos. GUI utiliza la metáfora de un escritorio (una interacción semejante a la realizada sobre un escritorio físico). La mayoría de los usuarios están familiarizados con un escritorio, por ello los desarrolladores de sistemas GUI utilizan este conocimiento preexistente para realizar una interacción fácil de recordar y usar.

Los sistemas y aplicaciones GUI utilizan el paradigma de ventanas, íconos, menús y puntero (del inglés Window, Icon, Menu, Pointer device - WIMP) que denota los elementos para realizar la interacción con la interfaz. En GUI, los usuarios no deben memorizar comandos ni su respectiva sintaxis porque los elementos WIMP representan de forma gráfica a una instrucción o tarea. Esto se debe a que GUI está establecida sobre CLI, es decir a través de una información visual "amigable" con ventanas, se selecciona con el puntero un comando implícito en un ícono o menú. Algunos analistas señalan que reconocer y elegir es más sencillo que recordar y enseguida escribir [van Dam, 2000]. Por ello GUI presenta mayor facilidad de uso a los usuarios que CLI.

Ivan Sutherland fue uno de los pioneros en HCI con el desarrollo de su proyecto de tesis

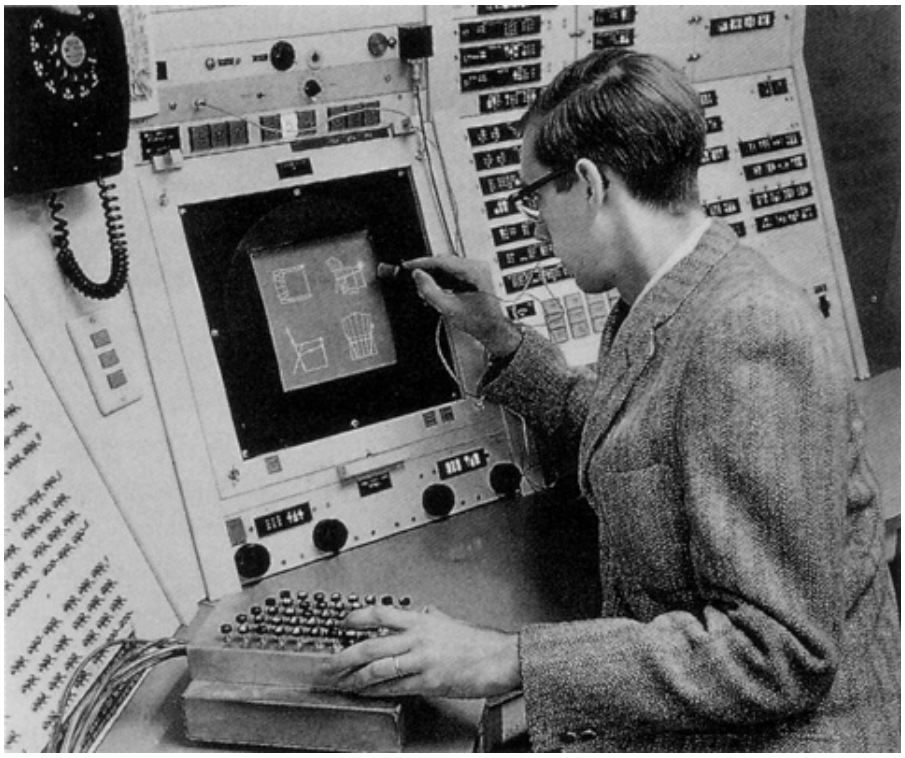


Figura 2.3: Proyecto de tesis de Sutherland: Sketchpad.

Sketchpad: A man-machine graphical communication system [Sutherland, 2003] en el Instituto tecnológico de Massachusetts. Las ideas introducidas en 1963 por Sutherland aun prevalecen en las interfaces de usuario a más de 49 años. Sketchpad es considerado como uno de los sistemas informáticos más reconocido desarrollado por un individuo, por ello fue merecedor del premio Turing en 1988.

La figura 2.3 muestra al autor de Sketchpad, Ivan Sutherland, utilizando un lápiz óptico, dispositivo predecesor del ratón, con el cual podía apuntar e interactuar con objetos desplegados sobre la pantalla; las cuatro perillas bajo la pantalla permiten controlar la posición y escala de la imagen. Bajo la mano izquierda de Sutherland se encuentra un conjunto de botones que permiten controlar funciones de dibujado.

El concepto de Sketchpad influyó sobre David Canfield Smith en su proyecto Pygmalion [Smith, 1993]. Canfield desarrolló un argumento más explicito que Sutherland de los beneficios cognitivos de una interacción directa, estableciéndose así el término de *icono*. En Pygmalion Canfield deduce que las imágenes pueden representar entidades abstractas en un lenguaje de programación.

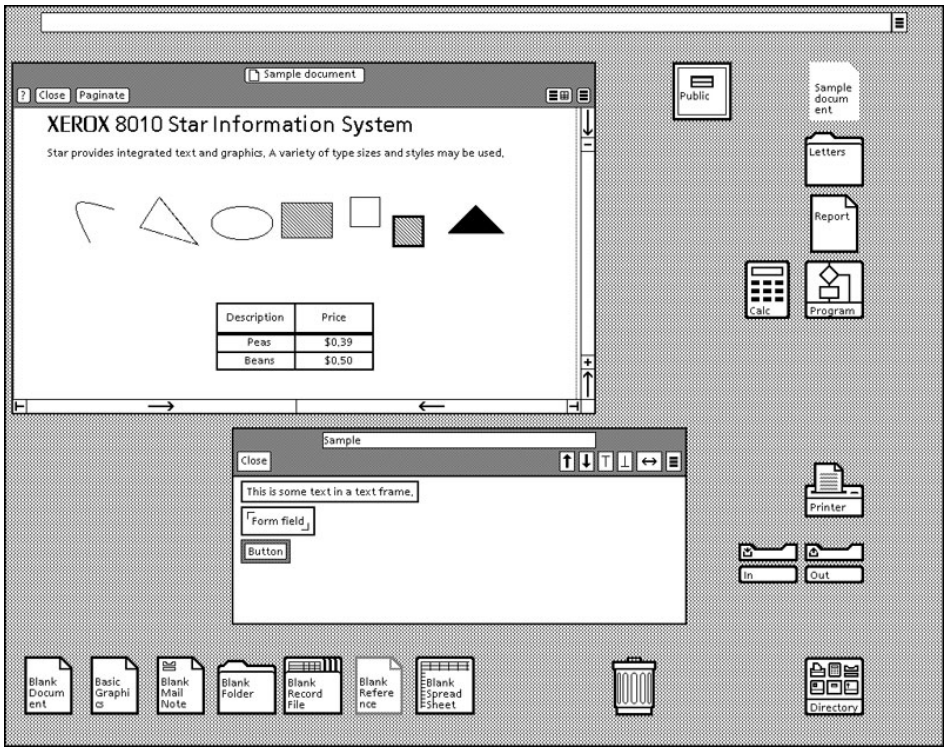


Figura 2.4: Star, pionera en áreas de trabajo con ventanas e iconos.

Una de las primeras GUI comerciales fue *Star* [Johnson et al., 1989] (ver figura 2.4). *Star* fue desarrollada por la corporación Xerox en el centro de investigación de Palo Alto en 1970 y se diseñó para asemejarse a una oficina como tal. La idea de utilizar esta metáfora era crear contrapartes electrónicas de objetos usados en la oficina (documentos, carpetas, botes de basura, etc.) con los cuales los usuarios interactúan frecuentemente. Así, surgió la metáfora de escritorio sobre GUI, asemejando a un escritorio sobre una oficina. Cabe destacar que Canfield formaba parte del equipo de desarrollo de *Star*, por lo cual las ideas de Sketchpad contribuyeron en gran parte del proyecto de Xerox. Sin embargo, fue hasta inicios de la década de los 80s cuando Apple lanzó al mercado *Lisa1*[2] que hizo populares las interfaces gráficas. En la figura 2.5 se muestra la interfaz del sistema operativo de Lisa1, LisaOS, la cual poseía una gran similitud a la interfaz de *Star*.

Desafortunadamente, *Star* no estaba diseñada para utilizarse como un sistema autónomo, sino en conjunto con diversos dispositivos a través de una red. Por ello, los costos de instalación eran demasiado caros (entre los $50, 000 y $100, 000 dólares) lo que ocasionó dificultades en las ventas. Por otra parte, Apple había desarrollado una interfaz que se asemejaba bastante al sistema de Xerox, con la diferencia que su costo era mucho menor. Por esta razón

[2]Apple Lisa1 es considerad como la primera computadora personal, desarrollada en 1983.

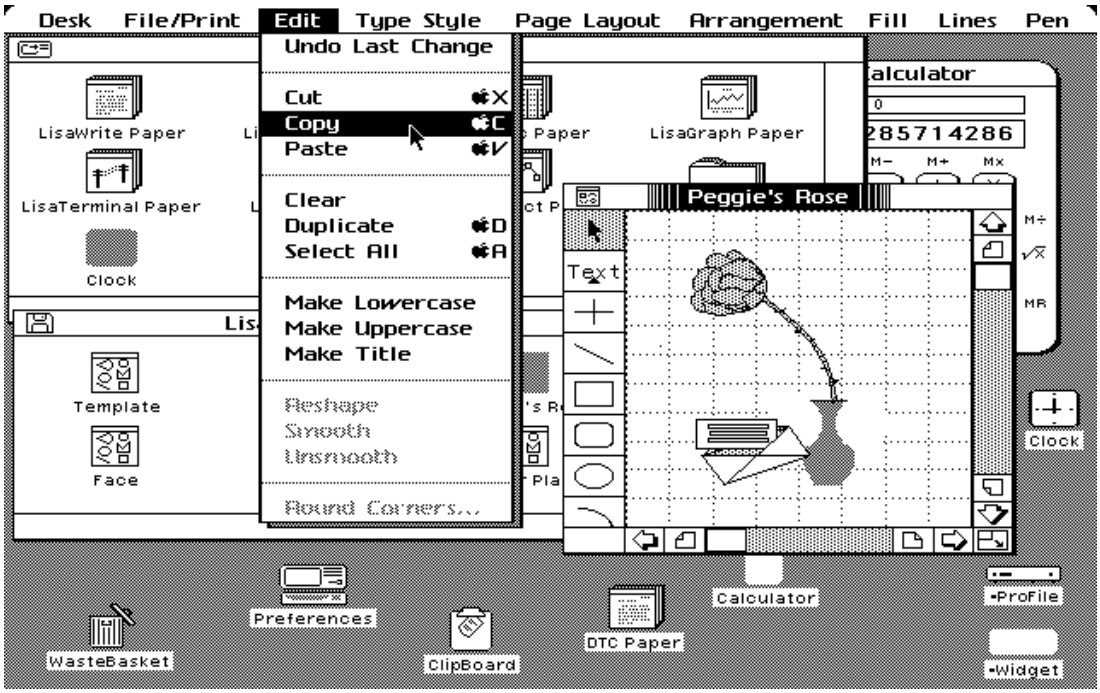


Figura 2.5: Interfaz del sistema operativo LisaOS.

Lisa1 tomó popularidad en el creciente mercado de computadoras personales.

Algunos de los principios desarrollados por la interfaz *Star* incluían manipulación directa, *lo que ves es lo que obtienes* (del inglés What You See Is What You Get), donde los usuarios podían manipular los objetos representados sobre una pantalla de forma análoga a la interacción con objetos reales. Por ejemplo, se mostraba un documento en la pantalla de igual forma como se vería de forma impresa.

2.1.4 Interfaz de usuario natural

NUI se encuentra en una posición similar que GUI a principio de la década de los 80s. Como GUI, NUI propone reducir las barreras en HCI. Gracias a los recientes avances en la tecnología de sensores, visión por computadora y procesamiento de señales, actualmente existen mejores herramientas para la creación de nuevos conceptos en interfaces y dispositivos de entrada más allá de lo convencional.

El término interfaz de usuario natural (NUI) es una metodología de interacción con la computadora enfocada en el uso de habilidades humanas para el control de sistemas o aplicaciones. Es considerado el siguiente paradigma en HCI y está conformado de arquetipos de

interacción humana más tradicionales, como el uso del tacto, visión, el habla y principalmente la creatividad [Liu, 2010]. Por ello, el significado de *natural* hace énfasis en el uso de habilidades que se adquieren de vivir en el mundo real, sin el uso de dispositivos artificiales cuyo funcionamiento se debe aprender.

La cita de Richard Stallman del software libre[3] ilustra una idea similar en la ambigüedad de la palabra *natural*. En NUI, *natural* se refiere al comportamiento del usuario durante la interacción no acerca de la interfaz en sí: [Wigdor and Wixon, 2011]

por ello se lee del inglés como

Natural User Interface

y no como

Natural User Interface

NUI estudia modelos y técnicas computacionales inspirados en la naturaleza para comprender el mundo que nos rodea en términos de procesamiento de información y modelos de interacción. Además NUI procura replicar entornos del mundo real mediante el uso de sensores y habilidades tecnológicas emergentes que permitan una interacción más precisa y óptima entre objetos físicos y digitales.

Los seres humanos estamos acostumbrados al mundo real y sin embargo replicar el mundo real de forma precisa en la computadora es bastante complejo. La realidad aumentada [Wellner et al., 1993] construye sobre el mundo real entornos virtuales con capacidades computacionales. Por consiguiente, la realidad aumentada es una de las principales herramientas en el diseño y desarrollo de NUIs. Principalmente, porque permite construir entornos que proveen información mediante objetos virtuales sobre el mundo real. A continuación, se describen las características principales que distinguen a una NUI:

- Interfaz centrada en el usuario. Los usuarios no requieren aprender a utilizar un dispositivo de control artificial pues la interacción está sustentada sobre habilidades comunes de los seres humanos.
- Multicanal. Al hacer uso de uno o más canales sensoriales y motrices del usuario se puede complementar la interacción sobre la interfaz. Además, se incrementa el ancho de banda en la entrada de información del usuario.

[3]Free like freedom, not like beer.

- Interacción directa. A través de la realidad aumentada es posible introducir objetos virtuales a nuestro entorno, permitiendo su manipulación sin el uso de instrumentos de por medio.

En la figura 2.6 se describen los componentes físicos que integran una NUI [Rauterberg et al., 1997]. El hardware de entrada para identificar la interacción realizada por los usuarios sobre las áreas de trabajo son dispositivos acústicos, sensores visuales y/o táctiles (cámaras, micrófonos, superficies táctiles, etc.). Como hardware de salida se emplean pantallas, dispositivos de proyección/holográficos y bocinas. Si bien el diseño y componentes que integran a una NUI pueden ser distintos entre sistemas, la gran mayoría utiliza una infraestructura similar a la antes mencionada.

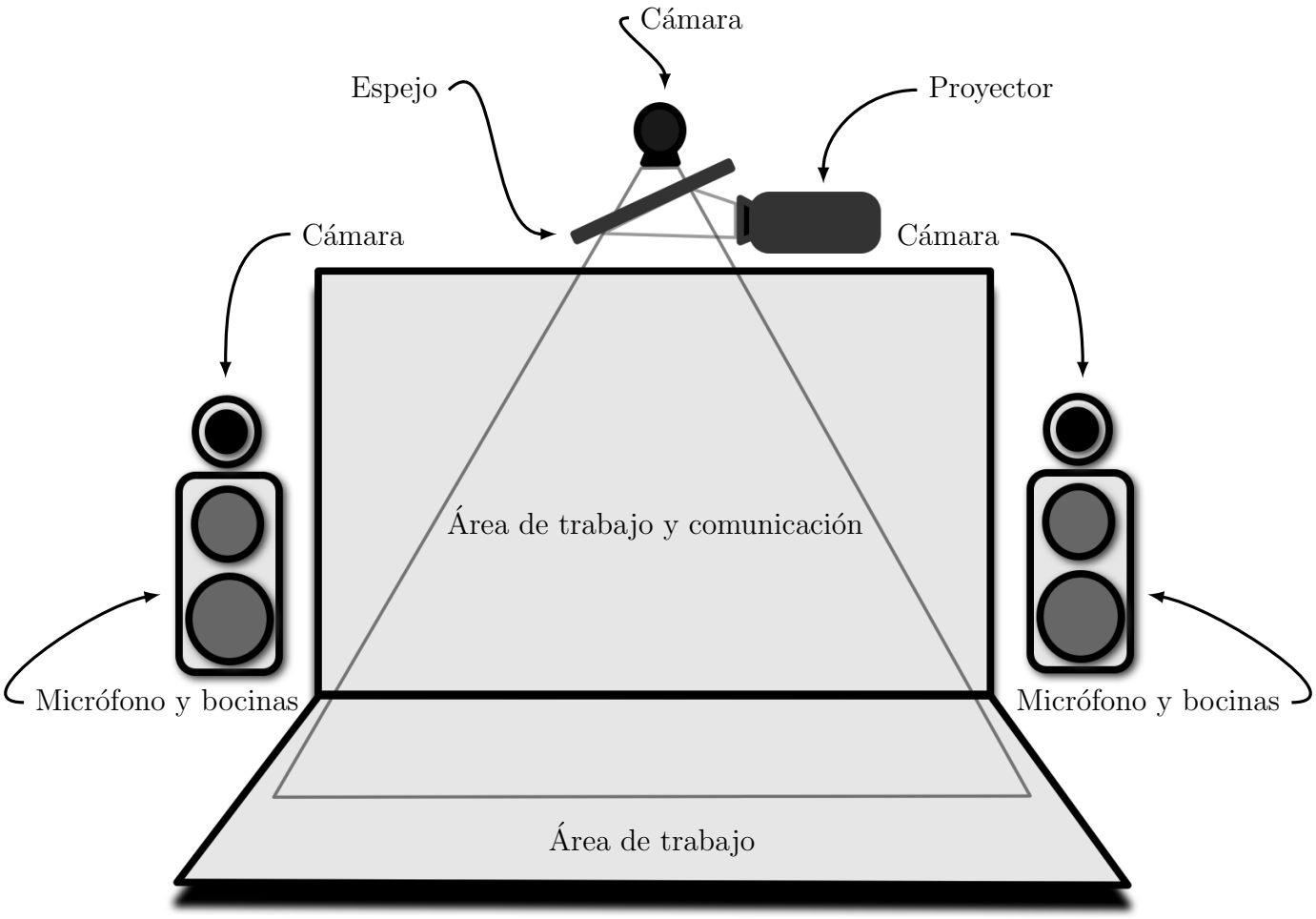


Figura 2.6: Componentes de una interfaz de usuario natural.

Otro elemento esencial para el desarrollo de NUIs son las técnicas para el reconocimiento de patrones, las cuales extraen información de los dispositivos de hardware y permiten

reconocer elementos significativos para el sistema. Para la implementación de estas técnicas se requieren algoritmos robustos de áreas de investigación dedicadas al reconocimiento de objetos, patrones, gestos, voz y procesamiento de imágenes.

Gracias a la entrada/salida paralela que ofrecen en conjunto los diversos componentes de una NUI, la manipulación de contenido digital sobre estas interfaces se ajusta a las habilidades de los usuarios. Por ello las NUI aportan gran ventaja sobre los métodos convencionales de interacción.

Resumen

En este capítulo se presentó la evolución y el paradigma de interacción de las interfaces de los sistemas de cómputo. Si bien el incremento en el poder de procesamiento del hardware ha sido continuo en estos sistemas, sus interfaces han evolucionado a un ritmo más lento. El trayecto de las interfaces de computadora está definido por cuatro fases: interfaz por lotes, interfaz de línea de comandos, interfaz gráfica de usuario e interfaz de usuario natural.

En la época de las interfaces por lotes, computadoras con gran poder de procesamiento eran escasas y caras, por ello las circunstancias de aquellos tiempos obligaban a los usuarios a adaptarse a las computadoras y no al revés. La interfaz de línea de comandos (CLI) evolucionó a partir la interfaz por lotes, en CLI los usuarios realizan peticiones a un sistema a través de un conjunto de comandos introducidos mediante un teclado; posteriormente el sistema procesa la petición y proporciona una respuesta. Con la aparición de la interfaz gráfica de usuario (GUI) y de las computadoras personales se creó un entorno más accesible para los usuarios, en GUI los usuarios no deben memorizar comandos ni su respectiva sintaxis porque los elementos del sistema se representan de forma gráfica. Finalmente, la interfaz de usuario natural (NUI) es una metodología de interacción con la computadora enfocada en el uso de habilidades humanas para el control de sistemas o aplicaciones, NUI está conformado de arquetipos de interacción humana más tradicionales, como el uso del tacto, visión y el habla.

Capítulo 3

Tecnologías base de NUIs

En el capítulo anterior se describió la evolución de las interfaces de los sistemas de cómputo. En este capítulo se detallan las principales tecnologías utilizadas en la construcción de interfaces de usuario natural (NUIs) que emplean tecnologías ópticas y controles de videojuegos.

Una forma de interacción con NUIs bastante común hoy en día es a través de una pantalla táctil. Gracias a la introducción al mercado de teléfonos inteligentes con pantallas táctiles, los desarrolladores de aplicaciones han tomado gran interés por esta clase de interacción con el usuario. Esta metodología de interacción no es precisamente de hogaño. En la década de los 60s ya se habían desarrollado varios mecanismos para la construcción de pantallas táctiles a través de varias capas electrónicas que al hacer contacto entre sí permitían identificar la interacción del usuario. Sin embargo, mediante esta tecnología se limitaba al usuario a realizar solo un contacto a la vez. Con el avance de la tecnología fue posible construir superficies multitáctiles, las cuales, como su nombre lo indica, permiten identificar múltiples contactos en un mismo instante.

Si bien la historia de la interacción multitáctil se remonta a principios de la década de los 80s, esta tecnología no había provocado gran impacto hasta hace algunos años. En la figura 3.1 se muestra una gráfica con la relevancia en la búsqueda de la palabra *multi-touch*[1]. Como se puede observa antes del año 2005 no existía gran tendencia hacia esta tecnología. No obstante después de un mecanismo propuesto por Jeff Han para la construcción de superficies táctiles con tecnologías ópticas el interés incrementó [Han, 2005].

El uso de tecnologías ópticas para la construcción de superficies táctiles propició la con-

[1]http://www.google.com/trends/?q=multi-touch (accedida en septiembre de 2012)

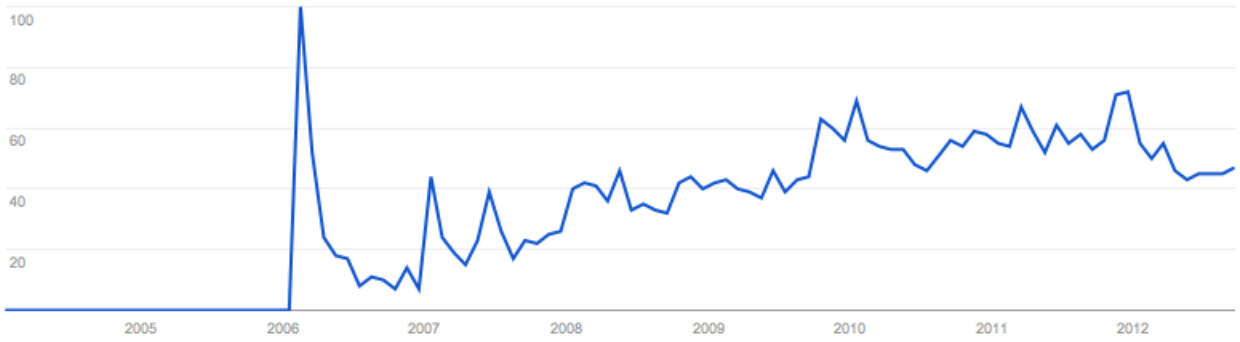


Figura 3.1: Popularidad de la palabra *multi-touch* en los últimos ocho años.

strucción de mesas interactivas escalables y de bajo costo denominadas *tabletops* [Buxton et al., 1985]. Pocas tecnologías existentes proveen una interacción tan rica y fluida como cuando se trabaja sobre una mesa con medios tradicionales (tales como el papel y bolígrafos) sobre todo hablando de espacios colaborativos. Por ello, en busca de aprovechar semejantes cualidades, varios investigadores han explorado la construcción de mesas con superficies interactivas.

Las *tabletops* son mesas multitáctiles construidas habitualmente con tecnologías ópticas que permiten manipular objetos físicos y contenido digital sobre su superficie sin la necesidad de dispositivos de control artificial de por medio. En consecuencia las *tabletops* acometen una interacción natural mediante el uso del tacto; la interacción se suele enriquecer mediante gestos y audio. Debajo o por encima de la *tabletop* se encuentra un proyector que despliega contenido sobre la superficie del tablero (constituido típicamente de un marco de acrílico) y a la par se utiliza una cámara infrarroja para localizar objetos físicos que tocan al tablero. Los usuarios interactúan con la *tabletop* al tocar o arrastrar sus manos sobre la superficie. Además pueden colocar objetos con etiquetas especiales que la cámara sea capaz de reconocer.

Por otra parte, las empresas dedicadas al entretenimiento se han convertido en un sector influyente en el desarrollo de sistemas interactivos. Esencialmente porque propician el desarrollo de ideas novedosas para atraer al mercado con sus productos. Gran parte de los avances en las interfaces de usuario se han favorecido de estas ideas, sobre todo de las aportadas por una rama específica del entretenimiento: los videojuegos.

En su afán de mejorar la interacción de los jugadores, los desarrolladores de videojuegos han creado nuevos periféricos que mejoran la experiencia de usuario. Sin embargo, estos dispositivos no sólo han creado innovación en consolas de videojuegos sino también han

planteado herramientas interesantes para el desarrollo de espacios y superficies interactivas.

Sin mayor exordio el capítulo comienza describiendo las tecnologías ópticas para la construcción de superficies interactivas. Posteriormente se pormenorizan los controles de videojuegos que ofrecen una interacción basada en movimientos de las empresas Nintendo, Sony y Microsoft. Finalmente se analizan los trabajos desarrollados en los últimos años relacionados a este trabajo de investigación.

3.1 Tecnologías ópticas para superficies interactivas

La construcción de superficies interactivas mediante tecnologías ópticas es una técnica donde se emplea sensores de cámaras (de ahí el nombre de tecnologías ópticas) como principal hardware de entrada. El enfoque primordial tras esta técnica es extraer patrones relevantes de las imágenes que capturan los sensores de las cámaras.

Una característica distintiva de esta técnica es el uso de fuentes de luz infrarroja. Esta clase de luz posee una longitud de onda que va desde los 700 y 1000 nanómetros y es una porción del espectro de luz que se encuentra más allá de las capacidades visuales del ojo humano. Sin embargo, los sensores de las cámaras pueden captar sin mayor problema los rayos infrarrojos.

Dado que las superficies interactivas emplean proyectores y/o pantallas como dispositivos de retroalimentación visual, frecuentemente las cámaras capturan las imágenes desplegadas por estos dispositivos. Por ello se emplean fuentes de luz infrarroja para discernir los patrones de interés exclusivamente sobre el espectro infrarrojo.

A la fecha existen cuatro métodos (en constante evolución conforme nuevas tecnologías aparecen) para la construcción de superficies interactivas con tecnologías ópticas:

- *reflexión interna total frustrada* (FTIR) propuesta por Jeff Han [Han, 2005].
- *Iluminación difusa* (DI), como la *tabletop* Microsoft Surface [NUIGroup, 2010].
- *Plano de luz* (LP), desarrollada por Alex Popovich y Nima Motamedi [NUIGroup, 2010].
- *Iluminación de superficie difusa* (DSI) de Tim Roth [Schöning et al., 2008].

Todas estos métodos están diseñados para su implementación como escenarios no intrusivos, donde el usuario no requiere el uso de instrumentos adicionales para realizar la manipulación del sistema. Sin embargo, aunque no es imperativo instrumentar al usuario para interactuar, el espacio de entrada del sistema se puede extender con el uso de guantes, lentes y/o bolígrafos especiales. A continuación se describe con mayor detalle cada una de estos métodos.

3.1.1 Reflexión interna total frustrada

En el 2005, Jefferson Han publicó un método para la creación de una superficie multitáctil utilizando acrílico, una cámara web, luz infrarroja y un proyector [Han, 2005]. En la figura 3.2 se muestra la metodología para la detección de tacto en dicho método, el cual es denominado *reflexión interna total frustrada* (FTIR). La evaluación del tacto en FTIR se realiza a través de la dispersión de luz infrarroja, emitida por leds infrarrojos, hacia una cámara web.

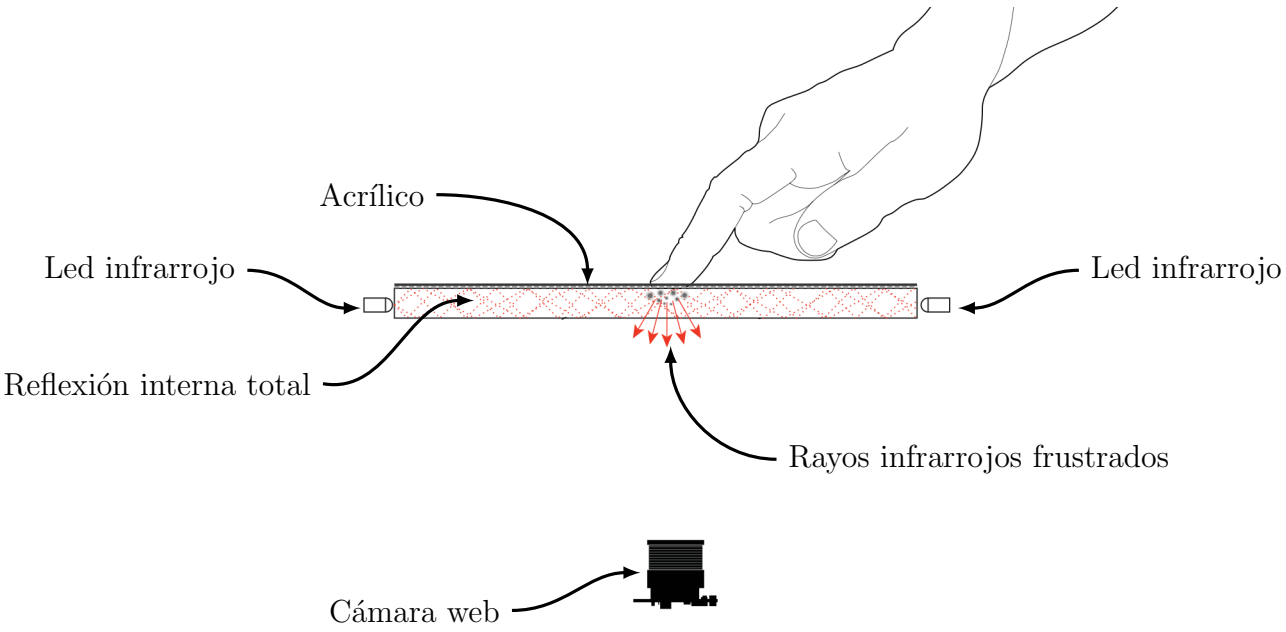


Figura 3.2: Elementos que constituyen a FTIR.

Han aprovechó una propiedad de la luz cuando se encuentra dentro de algunos materiales. Si un rayo de luz impacta a determinado ángulo de incidencia por dentro de un material con alto índice de refracción el rayo rebotará constantemente dentro de la superficie del material.

Empleando esta propiedad Han iluminó una superficie de acrílico con luz infrarroja para

atrapar los rayos. Cuando el usuario hace contacto con el acrílico algunos rayos son "frustrados" de su trayectoria y dispersados hacía abajo en donde una cámara (instalada de un filtro para bloquear la luz visible) los captura. Así, mediante un conjunto de algoritmos de visión por computadora se identifica la posición donde se ha hecho contacto.

FTIR es barato, fácil de construir y permite desarrollar una gran gama de aplicaciones. Además, gracias a la superficie de acrílico transparente, se puede instalar un proyector como dispositivo de despliegue.

3.1.2 Iluminación difusa

Los dispositivos de hardware necesarios para construir una superficie interactiva mediante el método *iluminación difusa* (DI) son similares a los que emplea FTIR. Sin embargo la diferencia entre FTIR y DI radica en la posición de la fuente de luz infrarroja y en el uso de una superficie difusora transparente (de ahí el nombre de iluminación difusa).

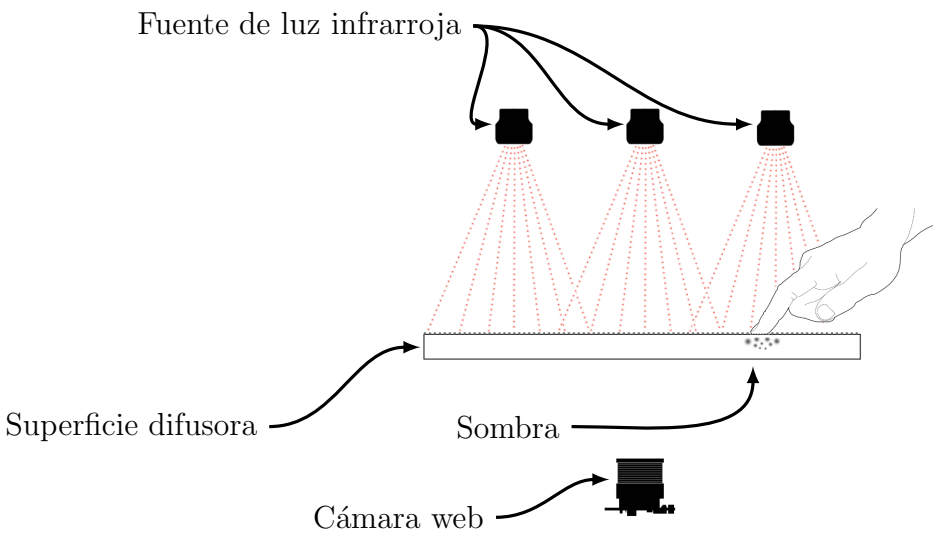


Figura 3.3: Elementos que constituyen a FDI.

DI se puede construir como *iluminación difusa frontal* (FDI) o *iluminación difusa trasera* (RDI). En ambos casos se utiliza el mismo principio: comparar los cambios de iluminación de la superficie difusora.

La figura 3.3 muestra la estructura de FDI. Como se puede observar la fuente de luz infrarroja se encuentra por encima de la superficie difusora. En consecuencia cuando el usuario

acerca lo suficiente algún objeto opaco a la superficie se genera una sombra, la cual es capturada por una cámara ubicada por debajo de la superficie.

En la figura 3.4 se muestra la estructura que constituyen a RDI. A diferencia de FDI, en RDI la fuente de luz infrarroja se encuentra por debajo de la superficie difusora. Por ende se refleja una gran cantidad de rayos infrarrojos cuando algún objeto opaco se aproxima lo suficiente a la superficie; la cámara web, también ubicada por debajo de la superficie, captura los rayos infrarrojos reflejados. Puesto que en RDI la fuente de luz emite en la misma dirección de captura de la cámara web, se puede identificar objetos a través del uso de *fiduciales*[2].

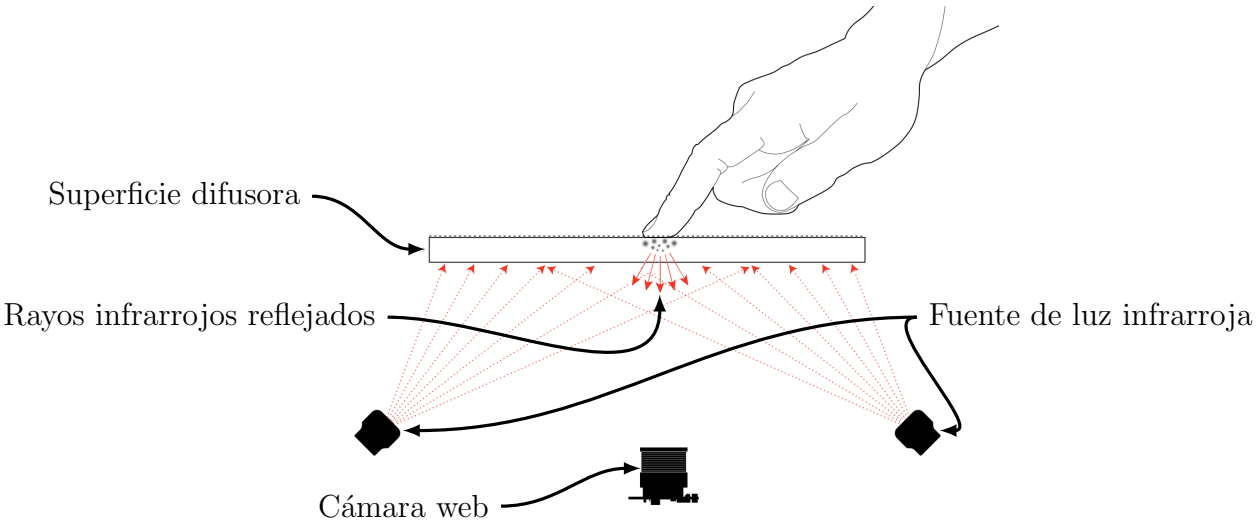


Figura 3.4: Elementos que constituyen a RDI.

3.1.3 Plano de luz

Alex Popovich, en lugar de hacer rebotar rayos de luz dentro de algún material como FTIR, empleó una fuente de luz infrarroja a base de lásers para construir una superficie interactiva. Mediante los lásers Popovich proyectó un plano de luz por encima de una superficie transparente. Así cuando algún objeto opaco, como el dedo de un usuario, toca el plano de luz infrarrojo se desvían algunos rayos hacia una cámara web (situada por debajo de la superficie). Para reducir el número de lásers Popovich utilizó una lente especial que le permitió distribuir hasta 120 grados el haz de luz.

[2]etiquetas con figuras poco convencionales fáciles de identificar

Nima Motamedi se sirvió de la idea de Popovich para construir la superficie interactiva mostrada en la figura 3.5. Motamedi, en contraste con Popovich, empleó una fuente de luz a base leds. La luz que generan los leds, comparada con la luz emitida por los lásers, se dispersa en mayor medida lo cual permite capturar objetos no solo sobre la superficie sino también por encima. Además con el método de Motamedi no existe el riesgo latente de daño ocular por contacto con luz láser. Sin embargo, este método requiere un mayor número de leds para iluminar uniformemente la superficie.

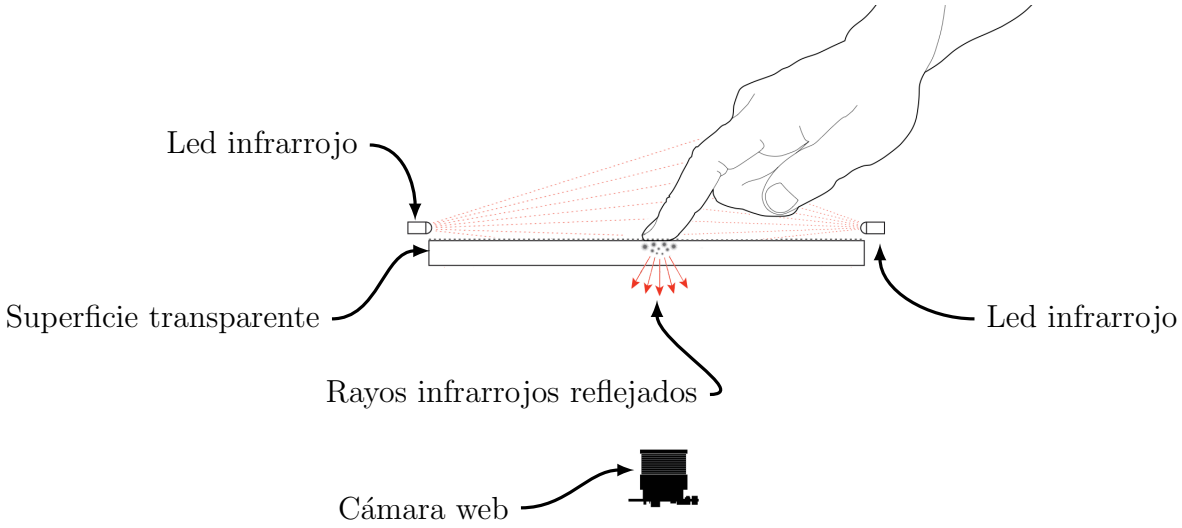


Figura 3.5: Elementos que constituyen a LP.

La construcción de una superficie interactiva mediante el método plano de luz permite utilizar cualquier superficie transparente, no solo acrílico, por lo que su construcción no es compleja ni costosa. Aunque como inconveniente de este método no se puede rastrear objetos con *fiduciales*.

3.1.4 Iluminación de superficie difusa

El método Iluminación de superficie difusa (DSI) surge como mejora al problema de iluminación desigual de los sistemas DI. Tim Roth propuso el uso de *Endlighten*[3], un acrílico especial compuesto de partículas que actúan como espejos.

La figura 3.6 muestra la estructura de DSI. Se puede apreciar que la diferencia entre DSI y FTIR es el uso de *Endlighten*. Mediante este material, la luz infrarroja emitida dentro de

[3]http://www.plexiglas-magic.com/

la superficie es distribuida de manera uniforme.

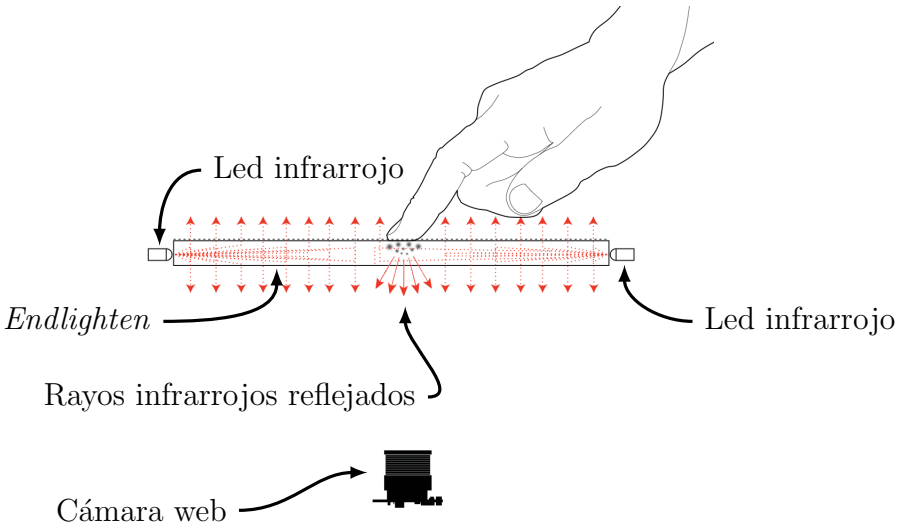


Figura 3.6: Elementos que constituyen a DSI.

Las ventajas y desventajas de utilizar DSI son:

Ventajas

- No existe problema para conseguir una iluminación uniforme.
- Se puede convertir una superficie interactiva FTIR a DSI fácilmente.
- Es posible rastrear objetos con *fiduciales*.

Desventajas

- Menor contraste sobre las imágenes, reduciendo la facilidad de discernir objetos.
- Mayor susceptibilidad a los cambios en la iluminación del entorno.
- Restricción en el tamaño de la superficie interactiva debido a la carencia de firmeza de *Endlighten*.
- Incremento en los costos para la construcción de la superficie interactiva.

3.2 Controles de videojuegos

Durante las últimas décadas el mercado de los videojuegos ha crecido a un ritmo acelerado. La industria de los videojuegos se ha convertido en un nicho atractivo para un gran sector de la población. Por ello, no es de extrañarse que los desarrolladores, con el propósito

de ofrecer una interacción cautivadora para sus usuarios y destacar de la competencia, estén diseñado constantemente nuevos productos e ideas de interacción.

Como es de esperarse entonces, una de las áreas que ha estado recibiendo mucha atención en la industria de los videojuegos es el desarrollo de interfaces de usuario naturales. Y los controles con interacción basada en movimientos es un ejemplo de ello. Sin embargo, el alcance de esta clase de tecnología ha ido más allá de los videojuegos y ha tenido también gran impacto en el desarrollo de sistemas de cómputo.

Increiblemente los controles de videojuegos han evolucionado en mayor velocidad que los dispositivos de entrada de las computadoras. Debido a esto, el uso de controles de videojuegos ha ganado una atención significativa por los investigadores. Actualmente Nintendo, Sony y Microsoft son las únicas empresas que han fabricado sus propios controles de videojuegos con interacción basada en movimientos. La primera fue Nintento con el *control Wii*, le siguió Sony con *PlayStation Move* y, finalmente, Microsoft con el dispositivo Kinect.

Cada uno de estos controles, al igual que las tecnologías ópticas descritas en la sección anterior, utiliza luz infrarroja y sensores para identificar la interacción de los jugadores. Por consiguiente, esta clase de tecnología se puede utilizar como dispositivo de entrada en el desarrollo de espacios interactivos. A continuación se describe cada uno de los controles de videojuegos de Nintendo, Sony y Microsoft.

3.2.1 Nintendo Wii

En noviembre de 2006 la consola de videojuegos *Nintendo Wii* fue lanzada al mercado. La principal novedad ofrecida por la consola *Nintendo Wii* fue el control *Wiimote*, un dispositivo muy semejante a un control remoto de una televisión. Además de botones este control contiene un acelerómetro de 3 ejes, una cámara infrarroja de alta resolución, una bocina y un motor de vibración. En la figura 3.7 se muestran la vista lateral, frontal y posterior del *Wiimote*.

Dependiendo de la interacción implícita en cada videojuego, los usuarios de la consola *Nintendo Wii* pueden desde apuntar el *Wiimote* a la televisión para seleccionar objetos virtuales hasta imitar movimientos característicos de alguna actividad. Por ejemplo simular una batalla medieval y blandir el *Wiimote* de forma semejante a una espada.

Figura 3.7: Wiimote: vista lateral, vista frontal y posterior.

Aunque Nintendo ofrece un kit de desarrollo relativamente económico, el acuerdo legal que establecen para el uso de sus herramientas limita severamente los tipos de aplicaciones potenciales a desarrollar. Sin embargo, con el uso de bibliotecas de software desarrolladas por la comunidad de software libre, es posible realizar las siguientes aplicaciones:

- Rastreo de objetos.
- Pintarrones interactivos.
- Pantallas táctiles.
- Reconocimiento de gestos.
- Interfaces de usuario.

3.2.2 PlayStation Move

PlayStation Move, lanzado en septiembre de 2010, es un sistema que permite interactuar con la consola de videojuegos *PlayStation 3* de Sony. En la figura 3.8 se muestran los tres componentes del sistema *PlayStation Move*: el control *move*, el control de navegación y la cámara *PlayStation Eye*. El control *move* posee un acelerómetro, un giroscopio y un sensor de campo magnético. En cambio el control de navegación cuenta con un conjunto de botones y un joystick analógico.

Figura 3.8: Componentes del sistema *PlayStation Move*

En general, los usuarios de la consola de videojuegos *PlayStation 3* interactuan sujetando el control *move* y el control de navegación con sus manos. A diferencia del *Wiimote*, el control *move* cuenta con una esfera luminosa en la parte superior, la cual en conjunto con la cámara *PlayStation Eye* permiten posicionar el mando en la escena. La consola de videojuegos puede identificar la interacción del usuario combinando la posición del mando con la información del control de navegación. Las posibilidades que la plataforma ofrece para el desarrollo de aplicaciones son:

- Rastreo de objetos.
- Reconocimiento de gestos.
- Multimedia interactiva.
- Realidad aumentada.
- Interfaces de usuario.

3.2.3 Microsoft Kinect

Kinect fue lanzado en noviembre de 2010 originalmente para la consola de videojuegos *Xbox 360* de Microsoft. A diferencia de los controles de Sony y Nintendo, el dispositivo Kinect

permite a los usuarios de la consola de videojuegos *Xbox 360* interactuar a través de su cuerpo o mediante su voz sin la necesidad de un control físico o un dispositivo táctil. Esto lo logra a través de la tecnología de una compañía Israelí llamada PrimeSense[4] que permite obtener en tiempo real la profundidad, color y audio de una escena.

Desde su introducción al mercado es evidente que no sólo ha estado transformando la industria de los videojuegos, sino también muchas otras áreas como robótica y realidad virtual. En la figura 3.9 se muestran los componentes que integran el dispositivo Kinect: una cámara RGB, un sensor de profundidad, un proyector láser de luz infrarroja, múltiples micrófonos y un motor sobre la base que permite modificar la inclinación del dispositivo.

Figura 3.9: Componentes de Kinect.

El sensor de profundidad, un sensor CMOS[5] monocromático, funciona en conjunto con el proyector láser de luz infrarroja. Ambos permiten capturar imágenes incluso en completa obscuridad. A continuación se describen las especificaciones de los componentes de Kinect:

- Visión horizontal de 57 grados.
- Visión vertical de 43 grados.
- ± 31 grados de movimiento del motor (+31 hacia arriba, -31 hacia abajo).

[4]http://www.primesense.com (accedida en junio de 2012)

[5]Complementary metal-oxide-semiconductor o CMOS es una de las familias lógicas empleadas en la fabricación de circuitos integrados.

- Resolución de la cámara de profundidad de 640 x 480 píxeles a 30 fotogramas por segundo.
- Resolución de la cámara RGB de 640 x 480 píxeles a 30 fotogramas por segundo.
- Consumo de energía de 2.25 watts.

Para evaluar la profundidad de los elementos de una escena, Kinect utiliza una técnica propietaria llamada "Codificación de luz", basada en "Luz estructurada" [Scharstein and Szeliski, 2003]. PrimeSense no proporciona información acerca de Codificación de Luz, no obstante, en su página web de preguntas frecuentes[6] escriben lo siguiente:

"La tecnología de PrimeSense para adquirir la imagen de profundidad está basada en Codificación de Luz^{TM}. Codificación de Luz funciona mediante la codificación del volumen en una escena con luz cercana a infrarrojo. La Codificación de Luz infrarroja es invisible para el ojo humano. La solución utiliza un sensor de imagen CMOS para leer la luz codificada en la escena. El SoC[7] de PrimeSense es conectado al sensor de imagen CMOS y ejecuta un algoritmo paralelo para descifrar la luz codificada recibida y producir una imagen de profundidad de la escena. La solución es inmune a luz ambiental y funciona en cualquier entorno interior."

Las posibles aplicaciones que se pueden realizar con el dispositivo Kinect son:

- Rastreo de objetos.
- Reconocimiento de gestos.
- Procesamiento de imágenes
- Multimedia interactiva.
- Realidad aumentada.
- Interfaces de usuario.
- Digitalización 3D.

[6]http://primesense.360.co.il/?p=535 (accedida en junio de 2012)
[7]Sistema sobre chip (System on Chip).

3.3 Trabajo relacionado

A continuación se describen algunos trabajos desarrollados en los últimos años relacionados con este trabajo de investigación. Todos los trabajos se enfocan en la construcción de espacios interactivos para la manipulación de objetos virtuales. Asimismo todos emplean como base alguno de los métodos de tecnologías ópticas o controles de videojuegos descritos con anterioridad.

3.3.1 Usando una cámara de profundidad como sensor táctil

En [Wilson, 2010] se reporta el desarrollo de un método para la construcción de superficies multitáctiles usando el dispositivo Kinect. Dado que las imágenes que captura el sensor de profundidad contienen la distancia de la superficie más cercana en cada pixel, el autor creó un modelo de un tablero de mesa con el cual compara el contacto de las manos del usuario.

El autor primeramente obtiene un valor de profundidad $d_{surface}$ para cada pixel capturado por el sensor de profundidad. Posteriormente, construye un modelo del tablero con los valores en $d_{surface}$ y con un umbral d_{max} seleccionado acorde a la variación de lectura ocasionada por el ruido del dispositivo Kinect. En seguida, el autor elige un valor d_{min} que coincida con el grosor típico de los dedos del usuario. Finalmente evalúa si el usuario hace contacto con el tablero comparando las imágenes obtenidas del sensor de profundidad con el modelo del tablero. Los pixeles correspondientes a las manos del usuario estarán a una distancia d_{min} del modelo del tablero.

3.3.2 OmegaDesk

En el 2011 Febretti *et al.* construyeron un escritorio interactivo para la manipulación de objetos en dos y tres dimensiones denominado *OmegaDesk* [Febretti et al., 2011]. Los principales escenarios que resaltan los autores para el uso de *OmegaDesk* son: visualización científica, interacción con información geoespacial dinámica, análisis de información médica y en general cualquier ambiente donde se desee enriquecer información visual en 3D a través de una espacio adjunto en 2D.

En la figura 3.10 se muestra los componentes que integran a *OmegaDesk*. Como se puede observar el sistema está constituido de varios elementos comerciales. Entre estos elementos se

encuentran dos pantallas estereoscópicas de gran tamaño que delimitan el espacio de interacción. Asimismo se utiliza la superficie multitáctil comercial *Multitouch* G^3 para identificar el contacto por parte del usuario sobre la pantalla en posición horizontal. Además para capturar la interacción que realiza el usuario con sus manos por encima de esta pantalla se utilizan cinco cámaras OptiTrack V100R2; el área de cobertura se complementa con el dispositivo Kinect.

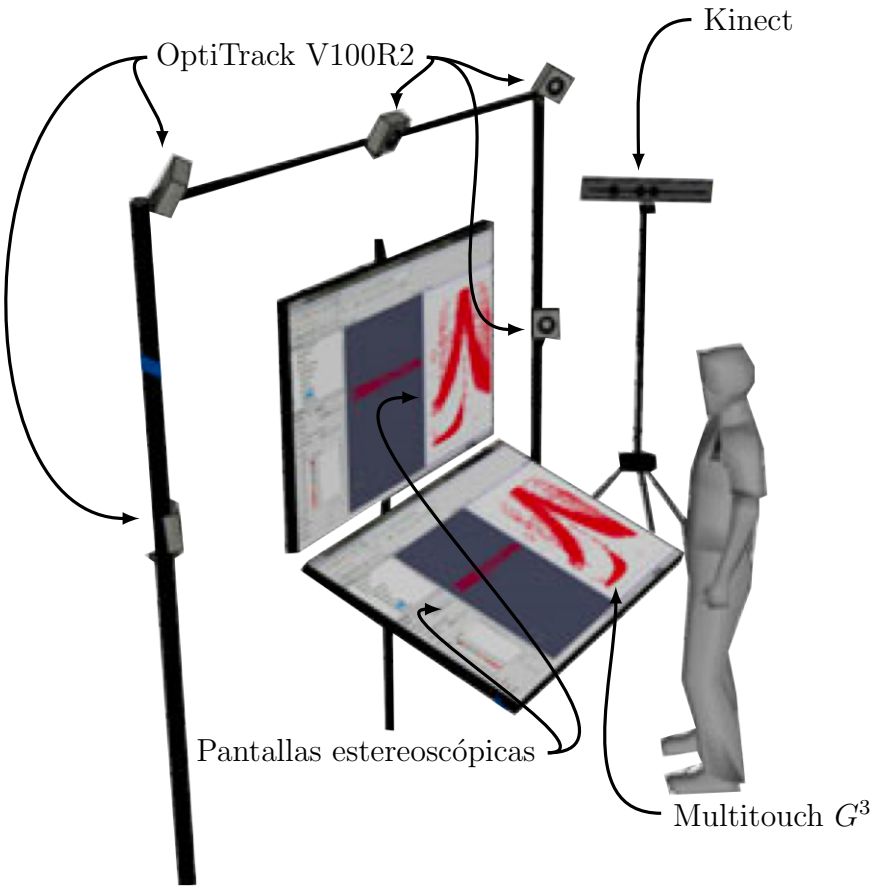


Figura 3.10: Componentes de *OmegaDesk*

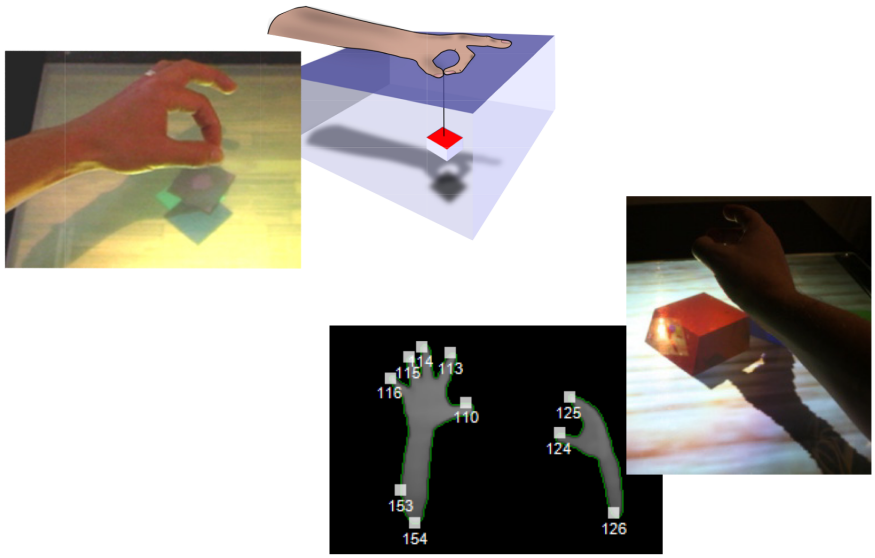
Además del escritorio interactivo, los autores desarrollaron un sistema operativo con dos componentes de software principales que permiten procesar la interacción realizada por el usuario y desplegar la información pertinente.

3.3.3 Interacciones en el aire

Hilliges *et al.* propusieron dos metodologías de interacción para la manipulación de objetos virtuales sobre *tabletops* [Hilliges et al., 2009]. Conceptualmente, su trabajo aporta técnicas que extienden la interacción táctil sobre *tabletops* y acoplan la manipulación de objetos virtuales en 3D.

En la figura 3.11 se muestran las dos metodologías de interacción propuestas por los autores. En (a), utilizan un gesto de "pellizco" para levantar, trasladar y/o rotar objetos virtuales. En cambio sobre (b) las tareas de rotación y traslación se llevan a cabo cuando al menos dos dedos del usuario hacen contacto con la geometría del objeto virtual.

(a) Interacción con "pellizco"



(b) Interacción con dedos

Figura 3.11: Interacción de las dos tabletops propuestas por Hilliges *et al.*

En la primera metodología para reconocer el gesto de "pellizco" se hace uso de un algoritmo de visión por computadora. El algoritmo analiza el centro y orientación del hueco que se genera cuando los dedos pulgar e índice se tocan. Tras la detección, el objeto se levanta y manipula solamente si es interceptado por el hueco. Evidentemente la principal limitante de esta metodología es la escasa libertad de interacción impuesta por un único mecanismo para manipular los objetos.

La segunda metodología utiliza un algoritmo para reconocer los dedos de las manos del usuario y determinar el "contacto" con los objetos virtuales. Las manipulación se lleva a cabo si al menos dos dedos de la mano del usuario tocan la geometría de algún objeto. A pesar de que esta metodología, en comparación con la anterior, propone una mayor libertad de interacción para manipular objetos en tres dimensiones, su primordial limitante es la velocidad de interacción. El tiempo requerido para realizar los cálculos de la detección de los dedos e identificar el contacto sobre el objeto virtual es bastante significativo por lo que se disminuye el desempeño del sistema.

3.3.4 Esfera

En [Benko et al., 2008] reportan la creación de *Esfera*, una pantalla esférica multitácto y multiusuario. Además de la solución de hardware y software, los autores desarrollaron un conjunto de técnicas que facilitan la interacción colaborativa alrededor de *Esfera*.

En la figura 3.12 se muestran los componentes de hardware que integran a *Esfera*. Como se puede observar el hardware de proyección y detección de tácto están acoplados en la base. Con ello, múltiples usuarios pueden interactuar simultáneamente a través de la superficie esférica. Para desplegar el contenido digital se utiliza un proyector, un espejo y una lente gran angular. Por otra parte se emplea una cámara web para capturar e identificar la interacción realizada por el usuario con los elementos virtuales.

Los autores reportan que es complicado proporcionar una iluminación uniforme sobre la superficie esférica. Sobre todo en la base donde es menos brillante que en la parte superior. De igual forma la lente gran angular produce distorsiones significativas sobre la información a proyectar. Lo que involucra invertir procesamiento para transformar y ajustar los objetos a una superficie esférica.

3.3.5 Aumentando mesas interactivas con teclados y ratones

Hartmann *et al.* desarrollaron un conjunto de técnicas para la interacción de dispositivos de entrada convencionales y *tabletops* [Hartmann et al., 2009]. Estas técnicas permiten manipular objetos de forma táctil sobre la *tabletop*, interactuar a través de cursores e introducir texto mediante teclados. Gracias a ello, el usuario puede realizar tareas con las herramientas

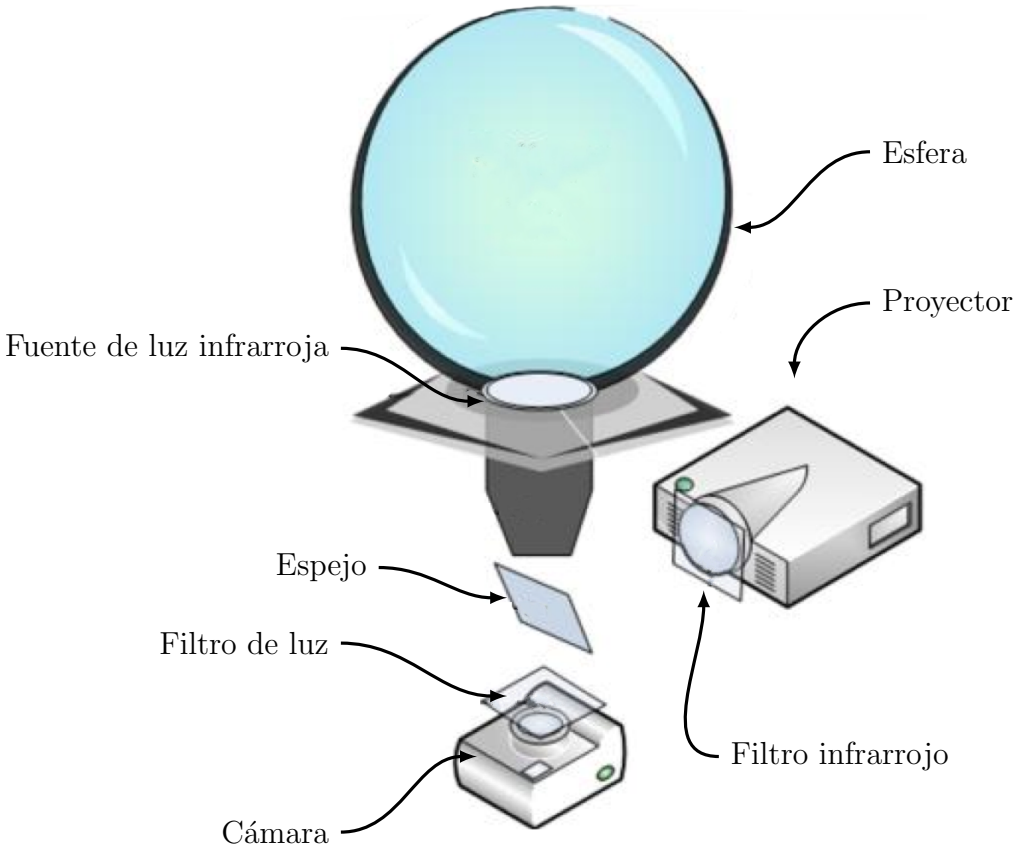


Figura 3.12: Componentes de hardware que permiten la detección táctil y la proyección de contenido sobre *Esfera*

que mejor se adapten a sus necesidades.

Los autores sugieren que los dispositivos de entrada convencionales enriquecen la interacción sobre *tabletops* ya que poseen una alta precisión y desempeño. Además, permiten identificar la posición del usuario y facilitan trabajar con objetos distantes.

El sistema está constituido por una *tabletop*, construida con FTIR, mutiusuario y multitáctil diseñada para facilitar la interacción en grupo. La mesa interactiva está instrumentada por una fuente de luz infrarroja, dos cámaras y dos proyectores. Estos elementos permiten identificar el contacto del usuario con la superficie y reconocer sobre la *tabletop* objetos físicos etiquetados con fiduciales. Cada teclado y ratón situado en el espacio de trabajo posee un fiducial único que en conjunto con un algoritmo de visión por computadora permite obtener la orientación y posición de los dispositivos sobre la *tabletop*.

Las principales limitaciones de este sistema son:

- El usuario debe moverse con los dispositivos de entrada sobre la mesa para que el sistema reconozca las interacciones sucesivas que está realizando.
- Un usuario no puede intercambiar los dispositivos una vez asociado a una herramienta.
- Los dispositivos cubren espacio de trabajo sobre la *tabletop*.

Resumen

En la tabla 3.1 se comparan los trabajos anteriores tomando como puntos de referencia la interacción proporcionada, la superficie para la manipulación de los objetos y la forma de desplegado de información. Los nombres de aquellos trabajos con $\surd$ son de relevancia para este proyecto de investigación.

Trabajo	Interacción	Superficie	Desplegado
$\surd$ Usando una cámara de profundidad como sensor táctil	Táctil	Adaptable	Proyección
OmegaDesk	Táctil y manipulación 3D	Plana	Pantallas estereoscópicas
$\surd$ Interacción en el aire	Táctil y manipulación 3D	Plana	Proyección
Esfera	Táctil	Esférica	Proyección
Mesas interactivas aumentadas con ratones y teclados	Táctil, complementada con teclado y ratón	Plana	Proyección

Tabla 3.1: Comparativo de trabajos relacionados.

Capítulo 4

NEKO

"Simple es difícil. Fácil es más difícil. Invisible es aun más difícil."
Jean-Louis Gassée

Jean-Louis Gassée[1] no podía describir mejor el reto que enfrentan los diseñadores en el desarrollo de interfaces de usuario. A diferencia de la interacción con las interfaces de computadoras tradicionales, donde se utiliza el ratón y teclado para manipular contenido sobre una pantalla, las NUIs involucran el control de sistemas con interfaces 2D-3D y/o multimodales — la vida real no es plana y los seres humanos vivimos e interactuamos sobre un mundo con tres dimensiones. Por lo tanto es razonable suponer que interfaces capaces de facilitar una interacción en tres dimensiones sean naturales para los usuarios, sobre todo si su uso se basa en habilidades que utilizamos todos los días.

En el capítulo anterior se presentaron trabajos desarrollados en los últimos años acerca superficies interactivas. Analizando el uso de estos trabajos se puede apreciar que las técnicas propuestas por los autores para la manipulación de objetos virtuales disuelve las barreras impuestas por los dispositivos de entrada convencionales. Sin embargo los métodos utilizados para desplegar la información están sustentados sobre pantallas en dos dimensiones o en el mejor de los casos, sobre pantallas estereoscópicas/autoestereoscópicas. En ambas situaciones, el usuario difícilmente puede interactuar con objetos virtuales en tres dimensiones como en el mundo real, principalmente porque no puede introducir sus manos a la pantalla/superficie. Por lo tanto, el usuario debe manipular objetos imaginarios sobre sus manos y a la par visualizar la pantalla/superficie donde se está desplegando la información.

[1]Jean-Louis Gassée, ejecutivo de Apple entre 1981 y 1990, es mejor conocido como el fundador de Be Inc.

A sabiendas de esta limitante, la solución desarrollada en trabajos anteriores fue proyectar manos virtuales como un mecanismo simulado de retroalimentación para representar las manos del usuario en la escena. Por ello esta solución carece de naturalidad en la interacción. Adicionalmente, las propuestas antes mencionadas requieren de tecnología costosa, sobre todo al utilizar pantallas estereoscópicas/autoestereoscópicas y múltiples cámaras para la captura del movimiento.

Este capítulo describe un prototipo de mesa interactiva, que propone solucionar de forma alterna los inconvenientes de los trabajos relacionados para la interacción con objetos virtuales en dos y tres dimensiones. NEKO, del inglés *Natural Environment for Kinect-base Object interaction*, es el prototipo de *tabletop* desarrollado con hardware comercial de bajo costo y software libre que permite el despliegue y manipulación de un entorno virtual acorde a la percepción del usuario.

Al igual que la mayoría de *tabletops*, NEKO utiliza cámaras y algoritmos de visión por computadora para identificar la interacción realizada por las manos el usuario. Asimismo se despliega el entorno virtual sobre una superficie a través de un proyector. El capítulo comienza describiendo los aspectos que motivaron el diseño de NEKO así como también los elementos considerados para la elección de los dispositivos de entrada/salida. Posteriormente, se describe de forma general los componentes lógicos de la arquitectura de software y su flujo de datos.

4.1 Aspectos de diseño de NEKO

Los ratones, teclados y pantallas están por doquier funcionando como intérpretes entre el mundo real y el virtual. No obstante, al utilizar éstas y otras herramientas invertimos gran cantidad de tiempo y esfuerzo transfiriendo información entre ambos mundos. Indiscutiblemente esta es una tarea estrictamente necesaria que quizá podría evitarse si en lugar de construir entornos virtuales sobre computadoras de escritorio nos enfocáramos a integrar objetos virtuales en espacios del mundo real familiares para los usuarios.

Para la mayoría de las personas una mesa es un espacio común donde realizan diversas actividades cotidianamente. Desde redactar documentos hasta ingerir alimentos, las personas aprovechan la superficie de una mesa para colocar objetos y realizar tareas específicas. Si las personas relacionan una mesa o un escritorio como un espacio de trabajo ¿Por qué no

aprovechar esta ideología para construir un sistema que mezcle un entorno virtual con el mundo real? Las *tabletops* multitáctiles ya aprovechan esta ideología y hasta cierto punto permiten manipular contenido digital de forma directa como en la vida real. No obstante, su espacio de interacción está restringido a una superficie plana y esto complica el despliegue y manipulación de objetos virtuales en tres dimensiones.

Tomando como base los beneficios de interacción de las *tabletops* y buscando facilitar la manipulación de objetos virtuales 2D y 3D en el mundo real se diseñó NEKO. El objetivo general en el diseño y la construcción de NEKO fue crear un espacio interactivo que permitiera manipular objetos virtuales a través de las manos de una forma semejante a como se haría sobre una mesa convencional. Para cumplir dicha meta se diseñó un prototipo de *tabletop* compuesto de elementos de hardware y componentes de software.

Un aspecto de diseño específico de NEKO fue ofrecer al usuario una interacción sin la necesidad de algún dispositivo de control artificial. Aunque el uso de guantes y controles proporcionan resultados muy precisos de la manipulación que un usuario realiza con sus manos, suelen ser caros y estorbosos. Por el contrario los controles de videojuegos con interacción basada en movimientos, como el dispositivo Kinect, junto con algoritmos para visión por computadora, proveen herramientas para la construcción de interfaces naturales sin la necesidad de dispositivos de control artificial. Si bien los controles *Wiimote* y *PlayStation Move* también ofrecen una interacción basada en movimientos, no fueron convenientes para la construcción del prototipo porque emplean dispositivos de control artificial.

En cambio (en comparación con los controles *Wiimote*, *PlayStation Move* y las tecnologías ópticas para la construcción de superficies interactivas) Kinect ofrece las siguientes características:

- *Permite construir superficies interactivas sin la necesidad de instrumentos.* Con la técnica propuesta en [Wilson, 2010] se puede identificar el contacto con una superficie a través del sensor de profundidad del dispositivo Kinect sin la necesidad de instalar cámaras o fuentes de luz infrarroja.

- *Facilidad en la construcción de NEKO.* Al prescindir de instrumentos o dispositivos de control artificial se simplifica el ensamblaje/desensamblaje del hardware sobre el prototipo de *tabletop*.

- *Flexibilidad en el uso de materiales.* A diferencia de las las tecnologías ópticas mediante el dispositivo Kinect no se requiere acrílico o algún material especial para la construcción

de la superficie interactiva.

- *Posibilita la manipulación de objetos virtuales 3D.* Al contrario de las cámaras convencionales, Kinect captura la distancia de cada elemento en la escena, lo cual favorece la construcción de mecanismos para la manipulación de objetos virtuales 3D.

Otro aspecto de diseño que guió la construcción de NEKO fue mantener una correspondencia en la ubicación de la entrada y salida del prototipo de *tabletop*. A diferencia de los trabajos relacionados, presentados en el capítulo anterior, se deseaba que la *tabletop* de NEKO permitiera visualizar objetos virtuales sobre las manos del usuario y no sobre otro espacio lejos de ellas. Para cumplir con este objetivo se decidió que el prototipo de *tabletop* debería estar compuesto de dos tableros: uno por debajo de las manos del usuario y el otro por encima de ellas. Así al combinar un proyector con técnicas de visión por computadora, como la realidad aumentada [Wellner et al., 1993], sería posible proyectar los objetos virtuales 2D/3D sobre las manos del usuario. Hoy en día existen varias tecnologías que posibilitan un mejor despliegue de objetos en tres dimensiones; no obstante sus costos suelen ser muy elevados. En cambio un proyector es mucho más barato y su portabilidad permite colocarlo en diferentes ubicaciones donde no interfiera con el sensor de profundidad de Kinect.

El último aspecto de diseño tomado en cuenta en la construcción de NEKO fue el desarrollo de una arquitectura de software capaz de identificar la interacción del usuario con un tiempo de respuesta corto. Es bien sabido que los algoritmos de visión por computadora suelen ser costosos computacionalmente hablando. En consecuencia fue necesario diseñar componentes de software que aprovecharan al máximo el hardware y evitarán el derroche de recursos computacionales. NEKO necesariamente debe ejecutarse a una velocidad mínima de 24 marcos por segundo para mantener una interacción adecuada entre el usuario y el entorno virtual. Además, un sistema de software con buen desempeño favorece el desarrollo de nuevas aplicaciones que aprovechen la manipulación de objetos virtuales sobre NEKO. Por ejemplo, aplicaciones educativas que asistan a profesores en sus clases con material didáctico o en sus presentaciones.

4.2 Prototipo de *tabletop*

Como se mencionó en la sección anterior, NEKO está compuesto de un prototipo de *tabletop* y de componentes de software que permiten identificar la interacción del usuario. En esta sección se presenta el diseño de la mesa interactiva y los dispositivos de entrada/salida

utilizados en el prototipo. En general, con el propósito de facilitar la construcción y reproducción del prototipo, la *tabletop* está compuesta de una mesa metálica e instrumentada con dispositivos comerciales de bajo costo.

La figura 4.1 muestra los elementos que integran la arquitectura de hardware del prototipo de *tabletop*. La mesa de fierro, compuesta de dos tableros con cristales, está instrumentada de una cámara web, el dispositivo Kinect, un proyector y una computadora.

El prototipo de la mesa, a diferencia de una mesa convencional, está compuesta de dos tableros, uno posicionado a cierta distancia por encima del otro. Sobre el tablero inferior se encuentra un cristal claro de 90 x 60 centímetros con un grosor de 4 milímetros; el tablero superior posee un cristal obscuro, de 6 milímetros de grosor, con las mismas medidas que el cristal claro. Existen 3 razones por las que se eligió un cristal claro en el tablero inferior:

- la primera es para disminuir los costos en la construcción del prototipo. Un cristal obscuro es más caro que uno claro.
- la segunda se debe a la refracción de los rayos infrarrojos del proyector de Kinect cuando atraviesan el cristal. A mayor grosor existe mayor posibilidad de incrementar el índice de refracción en los rayos infrarrojos, lo que ocasiona "huecos" en la información proporcionada por el dispositivo Kinect. La figura 4.2 muestra una imagen, capturada por el dispositivo Kinect, con huecos debido a la refracción de los rayos infrarrojos.
- y finalmente, la última razón es porque un cristal obscuro disminuye el paso de la luz y por lo tanto la visibilidad de las imágenes que captura la cámara RGB del Kinect.

En cambio, se decidió instalar en el tablero superior un cristal obscuro para facilitar el despliegue del entorno virtual sobre su superficie y a la par permitir al usuario visualizar sus manos. No obstante, el cristal obscuro se puede remplazar por una superficie transparente y una película semi opaca capaz de detener la luz emitida por el proyector. Ambos tableros de la mesa se encuentran inclinados para favorecer el acceso a la superficie de los cristales y también para evitar la refracción de los rayos infrarrojos.

La cámara web (colocada sobre el tablero superior de la mesa) captura el entorno frontal cercano a la *tabletop*. Así, mediante algoritmos de visión por computadora y modelos de clasificación se identifica el rostro del usuario cuando está manipulando los objetos virtuales. Para proporcionar una inmersión de tres dimensiones los objetos virtuales se despliegan continuamente sobre el cristal obscuro respecto a la posición del rostro del usuario.

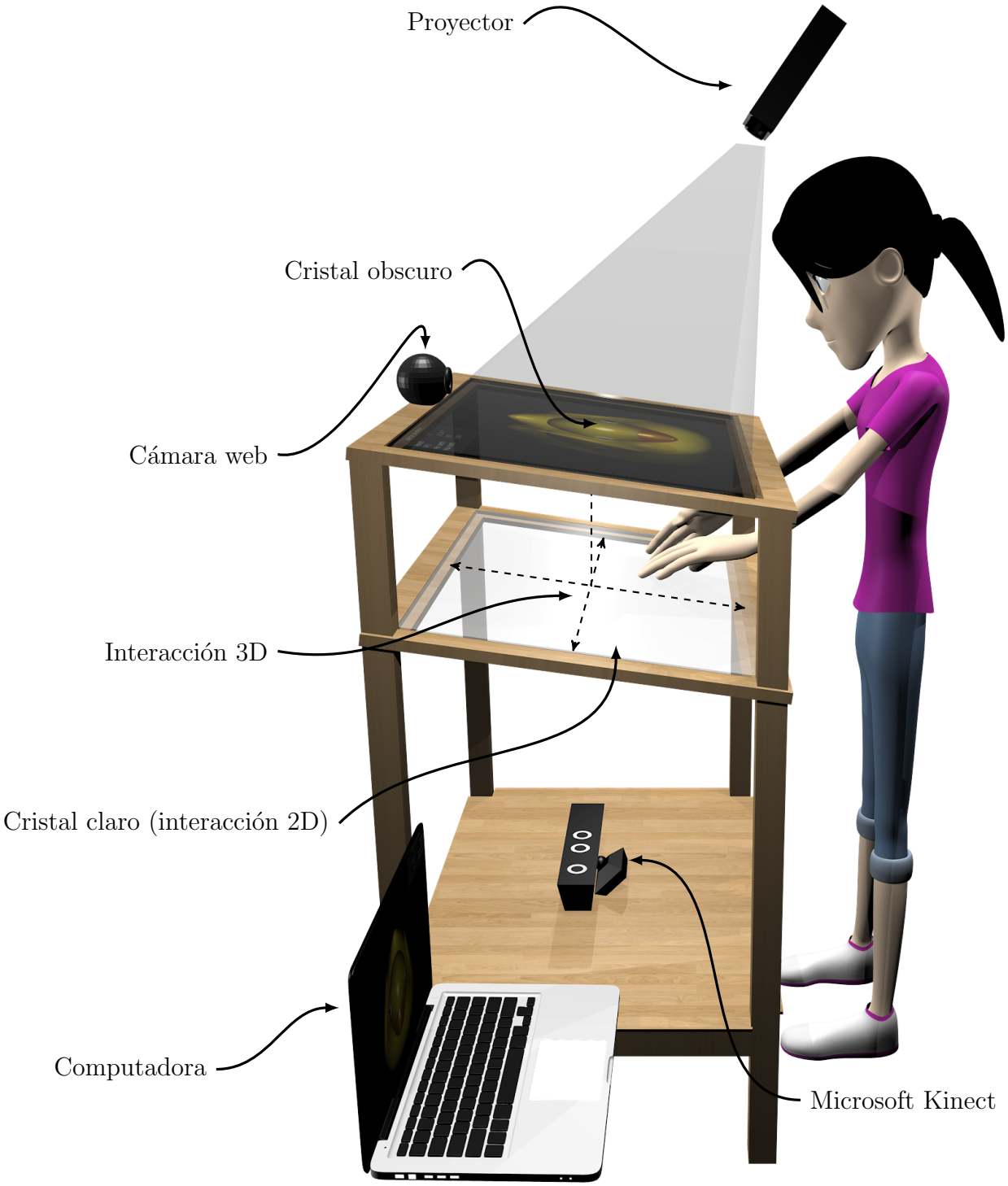


Figura 4.1: Arquitectura de hardware de NEKO.

El dispositivo Kinect, posicionado en la base de la mesa y orientado hacia los tableros, captura imágenes de la interacción realizada por las manos del usuario entre los cristales. Semejante a una interacción sobre un dispositivo multitáctil, la interacción 2D toma lugar

Figura 4.2: "Huecos" ocasionados por la refracción de los rayos infrarrojos.

sobre la superficie del cristal inferior en donde el usuario interactua a través del tacto. La interacción 3D toma lugar en el espacio entre los tableros. Para mantener la correspondencia entrada/salida, los objetos virtuales son proyectados sobre el cristal superior de la mesa, entre las manos y el rostro del usuario. Así el usuario visualiza los objetos como si estuvieran en sus manos. De igual forma que con la cámara web, se utilizan algoritmos de visión por computadora para procesar la información proporcionada por el dispositivo Kinect.

El proyector es el dispositivo de salida que despliega los objetos virtuales sobre la *tabletop*. Un proyector es mucho más barato si se compara con pantallas estereoscópicas o monitores del tamaño de los tableros de la mesa. Además, si en lugar de utilizar un proyector de tamaño común se utiliza un pico proyector las posibilidades de movilidad se incrementan, lo que hace posible colocar el proyector en diferentes ubicaciones. Por ejemplo en la base de la mesa, proyectando por debajo de los tableros, o por encima de la mesa para desplegar los objetos sobre el marco superior.

En la figura 4.3 se muestra con mayor detalle el diseño de la mesa interactiva así como las especificaciones de la base y los tableros que sostienen los cristales. La parte frontal de la mesa tiene una altura de 90 centímetros de las patas hasta el tablero inferior, lo cual es ideal para facilitar el acceso de las manos a personas con una estatura de 162 a 175 centímetros (los mexicanos tenemos una estatura promedio de 164 centímetros). Asimismo se eligió una separación de 30 centímetros entre los tableros por las siguientes razones:

- Evitar que el tablero superior esté demasiado cerca del rostro del usuario. Esto le permite al usuario observar con mayor claridad los objetos virtuales desplegados sobre el cristal del tablero superior.

- Crear un espacio óptimo de interacción. Una separación de 30 centímetros permite al usuario manipular los objetos virtuales de forma cómoda y además evita el punto anterior.

- Mantener la interacción 2D y 3D próximas entre sí. El usuario puede manipular objetos virtuales 3D entre el espacio de los tableros y sin requerir mayor esfuerzo puede mover sus manos al cristal inferior para manipular objetos 2D.

La parte posterior de la mesa posee una altura de 130 centímetros, generando una inclinación en los marcos de aproximadamente 9 °. Como se mencionó anteriormente, esta inclinación evita la refracción de los rayos infrarrojos del proyector infrarrojo del Kinect y además favorece la visualización del entorno virtual sobre el cristal superior. La base de la mesa, donde se coloca el dispositivo Kinect, está situada a 10 centímetros sobre el suelo entre las patas y permite mantener la estructura de la mesa firme.

Cabe mencionar que las medidas de los marcos están diseñadas de manera acorde a las especificaciones de captura del Kinect. La distancia mínima de captura del sensor de profundidad es de 60 centímetros con un ángulo de visión de 70 ° en diagonal, por lo que el dispositivo Kinect (situado en la base de la mesa, a una distancia promedio del marco inferior de 100 centímetros) es capaz de capturar una superficie con una diagonal de 140 centímetros. Sin embargo, si se utilizaran estas medidas, el usuario debería estirar bastante su brazo para alcanzar el extremo posterior del marco. Por ello se optó por utilizar cristales con medidas de 90 x 60 centímetros, proporcionando una superficie con una diagonal de 108 centímetros.

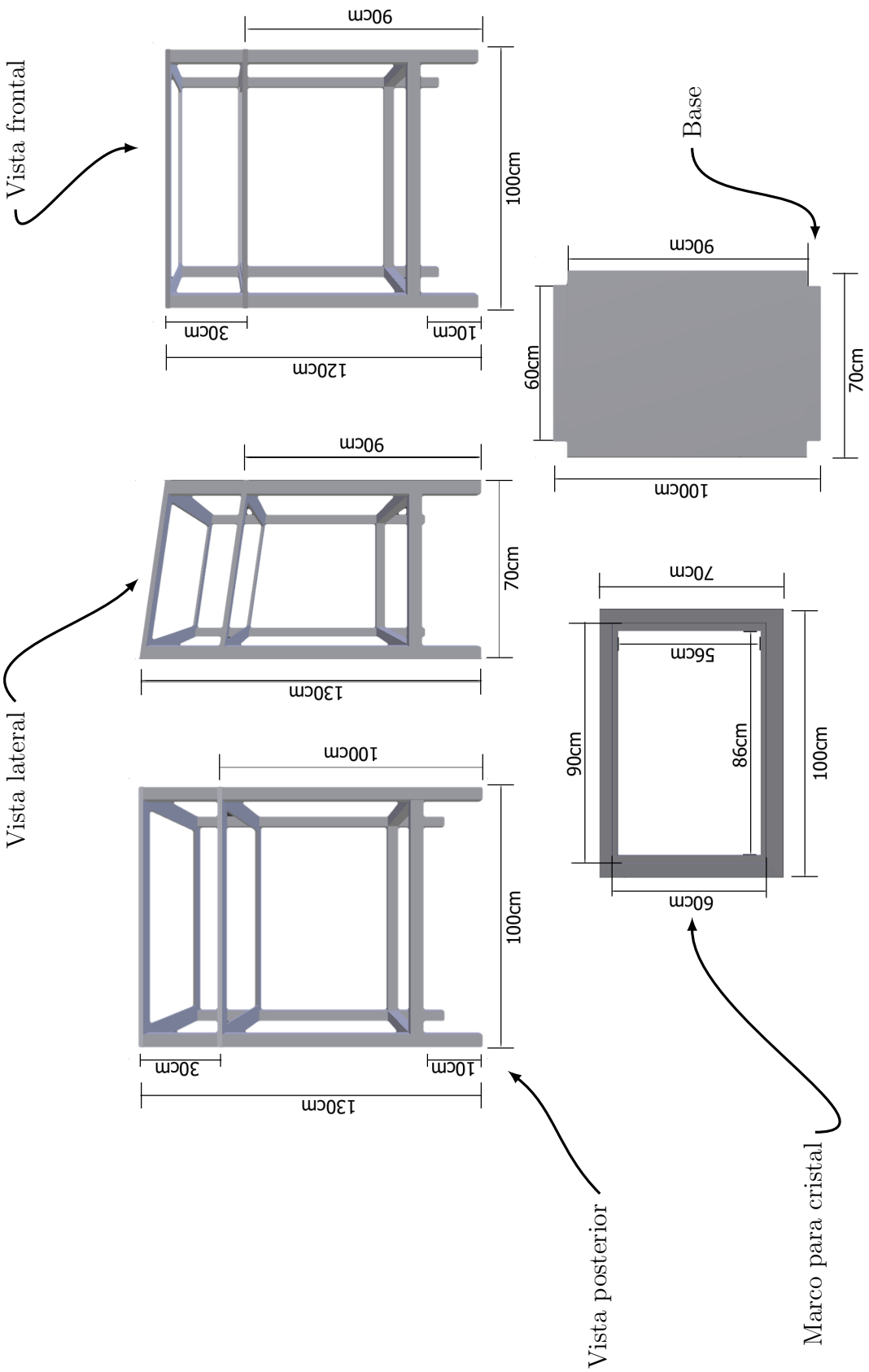


Figura 4.3: Especificaciones del prototipo de mesa interactiva de NEKO

4.3 Arquitectura de software

El software de NEKO permite el despliegue y manipulación, acordes a la percepción del usuario, de un entorno virtual sobre el prototipo de *tabletop*. En general, el software se encarga de capturar imágenes (a través de los dispositivos de hardware) de la interacción realizada por las manos y el rostro del usuario sobre la mesa interactiva. A partir de estas imágenes se extraen patrones relevantes que permiten manipular y visualizar el entorno virtual de forma semejante a la vida real.

Para facilitar la administración y el control en el desarrollo del software de NEKO, se diseñó una arquitectura constituida de cinco módulos. Cada módulo está estructurado a su vez por actividades específicas, las cuales dividen un problema mayor en tareas comunes más simples de resolver. No obstante, como se mencionó en la sección aspectos de diseño, el requerimiento primordial, tomado en cuenta en la construcción de esta arquitectura, fue mantener un flujo entre los módulos sin "cuellos de botella", es decir, un sistema con tiempo de respuesta corto.

La figura 4.4 muestran los cinco módulos principales que componen la arquitectura de software de NEKO. Tales módulos están implementados a partir de las siguientes bibliotecas y herramientas de software libre que permiten la comunicación con los dispositivos de hardware y el desarrollo de algoritmos de visión por computadora:

- *LibUSB*[2]*-1.0.9*. Una biblioteca para C de código abierto que permite la comunicación a través del puerto USB con dispositivos de hardware.

- *OpenNI*[3] *1.5.4*. Del inglés Open Natural Interaction, OpenNI es un framework multilenguaje y multi-plataforma desarrollado por PrimeSense para el desarrollo de aplicaciones utilizando interacción natural. Originalmente, OpenNI solo ofrece controladores para su hardware *PrimeSense 3D Sensor*[4], un dispositivo semejante en apariencia y funcionamiento a Kinect. No obstante, existen controladores que permiten integrar el dispositivo Kinect con este framework uno de ellos es el SensorKinect de *avin2*[5].

- *OpenCV 2.4.1*. Del inglés Open Computer Vision es una biblioteca de uso comercial y académico para el desarrollo de aplicaciones de visión por computadora.

[2]http://www.libusb.org
[3]http://openni.org
[4]http://www.primesense.com/en/solutions/solsensor
[5]https://github.com/avin2/SensorKinect

- OpenGL 3.0. Del inglés Open Graphics Library, OpenGL es el entorno de software libre más popular para el desarrollo de aplicaciones interactivas 2D y 3D. OpenGL permite dibujar escenas complejas a partir de primitivas geométricas simples tales como líneas, puntos y triángulos.

- *Cinder 0.8.4*. Una biblioteca desarrollada por la comunidad que provee herramientas para el desarrollo de aplicaciones con audio, video, procesamiento de imágenes y geometría computacional en C++.

La computadora sobre la cual se emplearon estas bibliotecas de software para el desarrollo de NEKO es una MacBook Pro con las siguientes características:

- Sistema operativo Mac OS X Lion 10.7.4

- Procesador Intel Core 2 Duo T7700 a 2.4 GHz.

- 4 GB DDR2 SDRAM a 667 MHz.

- Disco duro Western Digital de 500 GB a 5400 rpm.

- GPU Nvidia GeForce 8600M GT a 600 MHz con 512 MB GDDR3.

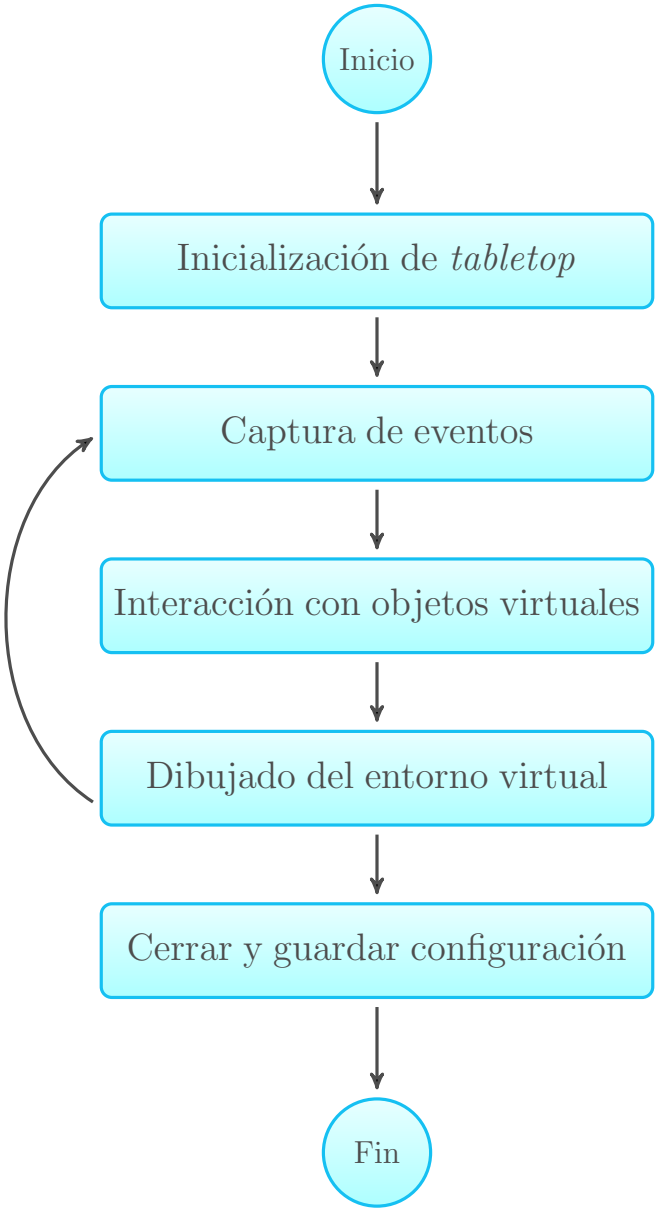


Figura 4.4: Arquitectura de software de NEKO

A continuación se describen las tareas involucradas en cada uno de los módulos de la arquitectura de software en el orden sobre el que fluyen los datos.

4.3.1 Inicialización de *tabletop*

Este módulo se encarga de configurar los componentes de hardware de la *tabletop*, en especial el dispositivo Kinect y la cámara web. Asimismo se carga en memoria la configuración necesaria para el correcto funcionamiento del software de NEKO. La figura 4.5 muestra de manera gráfica el flujo entre las tareas de este módulo. A continuación se describen con mayor

detalle cada una de ellas:

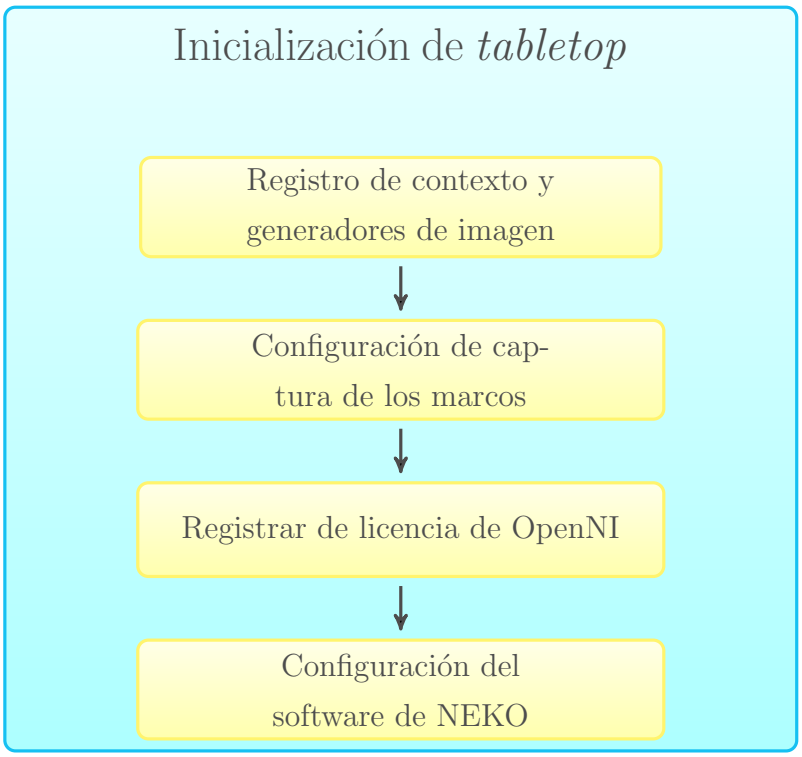


Figura 4.5: Tareas realizadas para la inicialización de la *tabletop*.

- **Registro de contexto y generadores de imagen del dispositivo Kinect**. En esta tarea se extrae una lista de dispositivos conectados a los puertos USB de la computadora mediante la biblioteca LibUSB. Con ello se determina si el dispositivo Kinect y la cámara web están conectados a alguno de los puertos mediante el identificador del fabricante y del producto. De ser el caso, se establece la comunicación con cada uno de estos dispositivos de hardware. Enseguida, a través de la biblioteca OpenNI, se registra un contexto de imagen para el sensor de profundidad y la cámara RGB del Kinect. Este contexto permite configurar los generadores de imagen correspondientes para capturar y extraer los marcos de video[6] del dispositivo Kinect durante la ejecución de NEKO.

- **Configuración de resolución y velocidad de marcos de video**. La resolución del sensor de profundidad y la cámara del Kinect pueden llegar hasta un tamaño de 640 x 480 pixeles a una velocidad de 30 marcos por segundo. No obstante, para realizar los cálculos de forma fluida utilizando esas especificaciones es necesario una o varias

[6]En esta tesis nos referiremos como *marco de video* o *marco* a las imágenes que capturan la cámara RGB del dispositivo Kinect y la cámara web; a las imágenes que captura el sensor de profundidad del dispositivo Kinect nos referiremos como *marco de profundidad*.

computadoras con gran poder de procesamiento. Por ello, para fines prácticos, esta tarea se encarga de configurar una resolución de 160 x 120 pixeles a 30 marcos por segundo al sensor de profundidad y de 120 x 96 pixeles a 30 marcos por segundo a la cámara RGB. Asimismo, para la cámara web, se utilizó la misma configuración que la cámara RGB de Kinect.

- **Registro de licencia OpenNI**. Para el funcionamiento de la biblioteca OpenNI es indispensable introducir una licencia constituida por un proveedor y una llave. A pesar de que solo se utilizará esta biblioteca para extraer los marcos del dispositivo Kinect, el dispositivo no proporciona ninguna información si no se registra esta licencia. Una vez registrada la licencia, se enciende el proyector láser de rayos infrarrojos y se habilita el sensor de profundidad y la cámara RGB del Kinect. Finalmente, se inicia la captura de la cámara web.

- **Configuración del software de NEKO**. El software de NEKO que permite la manipulación de objetos virtuales utiliza varios modelos para la detección de manos, rostro y gestos. Estos modelos se encuentran en archivos los cuales son leídos a memoria cada vez que se inicia la ejecución de NEKO. Las tareas que generan estos modelos se describen a mayor detalle en el capítulo 5.

4.3.2 Captura de eventos

La *captura de eventos* permite, entre otras cosas, identificar los eventos de teclado y ratón sobre la computadora en la cual se ejecuta el software de NEKO. Así mediante estos eventos se efectúan tareas de depuración y configuración de la *tabletop*, tales como delimitar la distancia mínima y máxima de captura del sensor de profundidad del dispositivo Kinect.

Las tareas que constituyen a este módulo (ver figura 4.6) son la *captura de eventos del teclado*, *captura de eventos del ratón* y *captura de eventos de la ventana*. Las primeras dos son utilizadas mientras se ejecuta NEKO, como su nombre lo indica, para capturar los eventos de teclado y ratón respectivamente. De igual forma estas tareas también están vinculadas a labores de depuración durante el desarrollo o modificación de algún módulo. Por otra parte, de forma semejante a cualquier aplicación, NEKO emplea una ventana como método de despliegue del entorno virtual. Por ello se emplea la tarea *captura de eventos de la ventana* para mantener una relación de aspecto coherente del dibujado del entorno virtual por la GPU; esta tarea ajusta la proporción del entorno virtual si la ventana sufre alguna modificación.

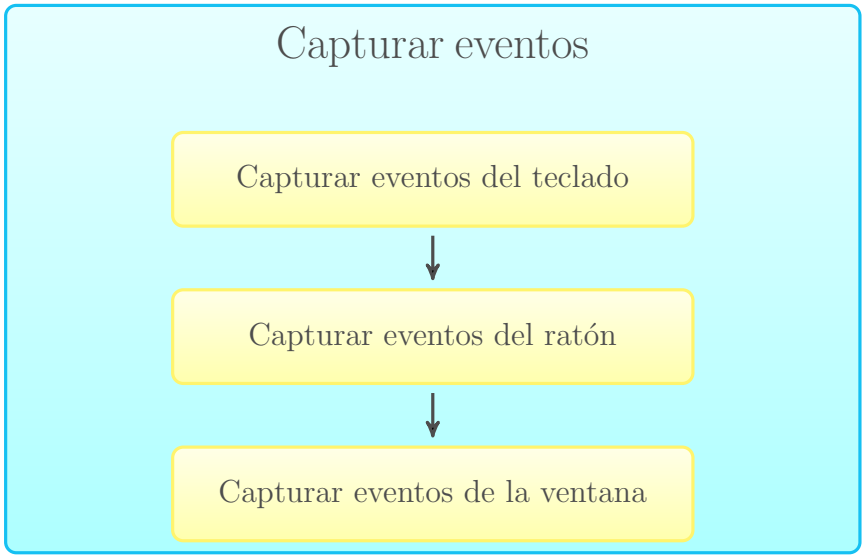


Figura 4.6: Tareas realizadas para la captura de eventos

4.3.3 Interacción con objetos virtuales

La *interacción con objetos virtuales* es el módulo de mayor importancia en la arquitectura de software de NEKO. A grandes rasgos, en este módulo se extraen patrones relevantes de la interacción que realiza el usuario sobre el prototipo de *tabletop* y conforme a estos patrones se manipulan los objetos virtuales.

La figura 4.7 describe las tareas involucradas en la *interacción con objetos virtuales*. Primeramente, con la configuración obtenida de los módulos *inicialización de tabletop* y *captura de eventos*, se extraen los marcos de la cámara web y del dispositivo Kinect. Enseguida se identifican los patrones relevantes que el usuario realiza con sus manos y su rostro mediante modelos de clasificación. Y finalmente se manipulan los objetos virtuales acordemente a los patrones identificados.

Para realizar una interacción semejante a la vida real con los objetos virtuales sobre la *tabletop* es necesario que el usuario observe los objetos desde una perspectiva concorde a la de sus ojos. Por consiguiente, es ineludible el uso de una tarea para la *detección de rostro*. Mediante un modelo de clasificación, previamente cargado a memoria en el modulo *inicialización de la tabletop*, se compara cada marco obtenido de la cámara web para identificar la posición del rostro del usuario. De este modo se calcula una perspectiva para el despliegue del entorno virtual. La *detección de rostro* también implica la creación y almacenamiento de un modelo de clasificación a través múltiples imágenes de muestra.

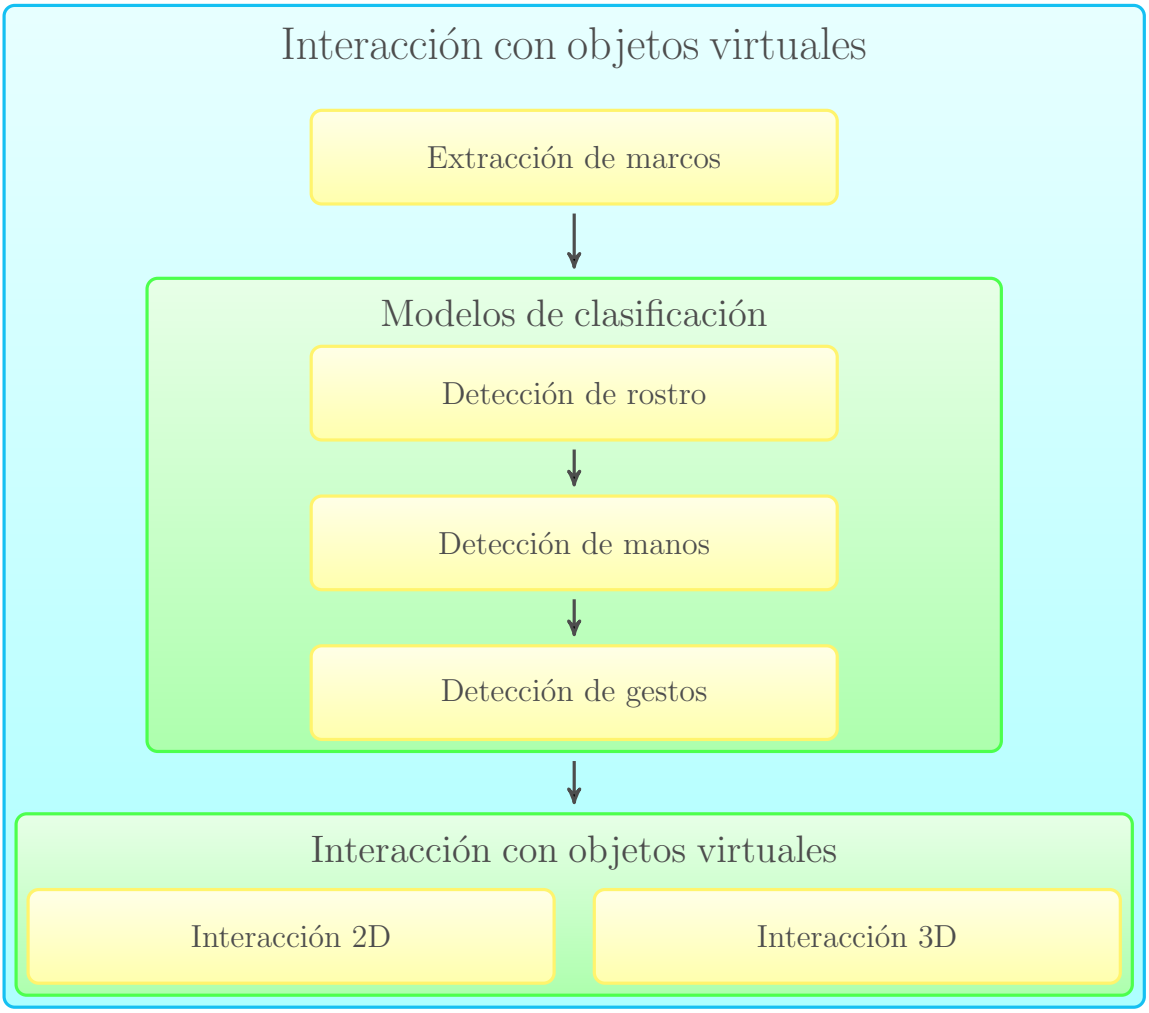


Figura 4.7: Tareas realizadas para la interacción con objetos virtuales.

La *detección de manos* se realiza exclusivamente sobre los marcos que otorga el dispositivo Kinect. De forma similar a la *detección de rostro* o con el uso de técnicas de visión por computadora, en esta tarea se identifican las manos del usuario sobre los tableros de la *tabletop*. La *detección de manos* es la base para realizar las tareas *detección de gestos* e *interacción 2D/3D*.

De forma general, la *detección de gestos* determina los gestos que el usuario realiza con sus manos. Esta tarea evalúa si existe alguna pauta, es decir una silueta o pose, en las manos detectadas por la tarea *detección de manos*. De ser este el caso, se registran los movimientos significativos de la mano y posteriormente se verifica su correspondencia con los movimientos implícitos en algún gesto.

En la tarea *interacción 2D* se identifica el contacto por parte del usuario sobre el cristal

claro ubicado en el tablero inferior de la *tabletop*. En esta tarea se construye un modelo de la superficie del cristal mediante los marcos del sensor de profundidad del dispositivo Kinect. A través de este modelo se determina si algún dedo del usuario hace contacto con la superficie del cristal.

Finalmente, la tarea *interacción 3D* concierne a la manipulación en tres dimensiones de objetos virtuales. A partir de la información proporcionada por el sensor de profundidad del dispositivo Kinect se evalúa la distancia de cualquier cuerpo opaco (como las manos del usuario) introducido entre los tableros de la *tabletop* contra la "distancia" de los objetos virtuales proyectados. Cuando ambas distancias son casi idénticas, el objeto virtual es manipulado acordemente a la interacción inducida por el cuerpo opaco.

4.3.4 Dibujado del entorno virtual

Una vez realizada la *interacción con objetos virtuales*, se dibuja con la GPU todos los elementos del entorno virtual y se despliega tanto en la pantalla de la computadora donde se ejecuta el software de NEKO, como en el cristal obscuro del tablero superior del prototipo de *tabletop*. Cabe mencionar que NEKO maneja múltiples entornos virtuales que permiten al usuario separar los objetos virtuales conforme a sus necesidades. Por ejemplo en un entorno virtual el usuario puede colocar un tablero de ajedrez con sus respectivas piezas, mientras que sobre otro entorno puede colocar un modelo del sistema solar; NEKO permite al usuario seleccionar el entorno que desea desplegar y manipular sobre el cristal obscuro del tablero superior del prototipo de *tabletop*.

Las tareas involucradas en este módulo se pueden observar en la figura 4.8. La tarea *preparación* limpia el buffer de la pantalla y determina la fuente de iluminación del entorno virtual. La tarea *dibujado de espacios* dibuja el espacio de trabajo actual del entorno virtual sobre el que usuario está interactuando ([a] sobre la figura 4.9). El *dibujado de marcos de video* es de uso exclusivo para la depuración, pues permite al desarrollador identificar el comportamiento de los algoritmos implementados sobre los marcos de video proporcionados por la cámara web y el dispositivo Kinect ([b] sobre la figura 4.9). La tarea de *renderizado de objetos y texturas* dibuja las mallas y texturas de los objetos que se encuentran sobre el espacio de trabajo actual ([c] sobre la figura 4.9). Finalmente, el *dibujado de parámetros* dibuja el menu de la aplicación y los parámetros utilizados por los dispositivo Kinect ([d] sobre la figura 4.9).

Figura 4.8: Tareas involucradas para el dibujado del entorno virtual.

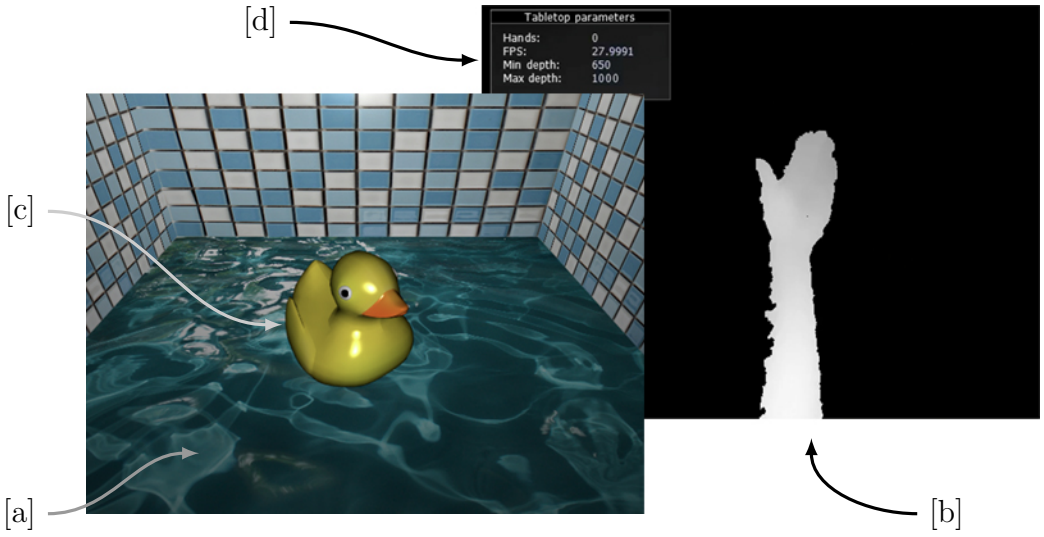


Figura 4.9: Arquitectura de hardware de NEKO.

4.3.5 Cerrar y guardar configuración

El flujo de datos recae sobre este módulo exclusivamente cuando la aplicación se va a cerrar. Las configuraciones realizadas por el usuario o el desarrollador son guardadas sobre

un archivo que será leído nuevamente en la *inicialización de tabletop*. Otra tarea realizada sobre este módulo es detener la comunicación con la cámara web y el dispositivo Kinect.

La primera tarea en ejecutarse sobre este módulo es *detener dispositivos de hardware*, en ella se manda una señal a la cámara web y al dispositivo Kinect para interrumpir la captura de imágenes e indicar al sistema operativo su disponibilidad de uso. Los ajustes realizados a estos dispositivos junto con la ubicación de los modelos de clasificación y superficie táctil se guardan en un archivo *xml* sobre la tarea *guardar configuración* para su uso posterior al iniciar nuevamente el software de NEKO. La figura 4.10 muestra las tareas del módulo *cerrar y guardar configuración*.

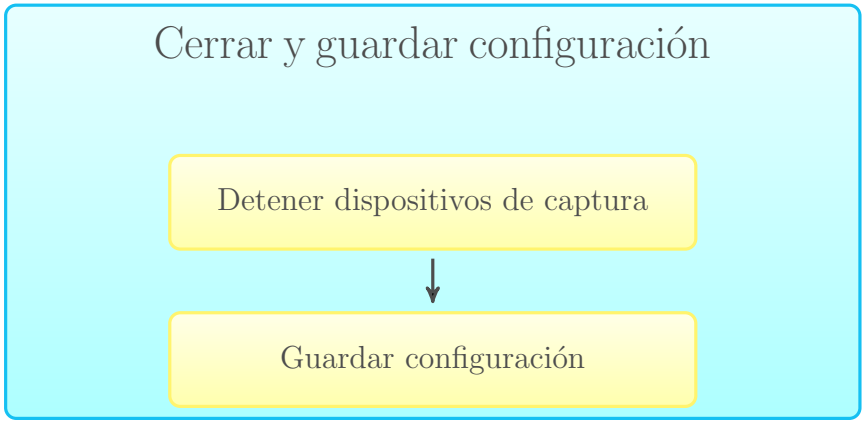


Figura 4.10: Tareas involucradas al cerrar la aplicación

Resumen

Este capítulo presentó NEKO, una nueva *tabletop* para la manipulación de objetos virtuales 2D y 3D mediante las manos. NEKO está compuesto de un prototipo de *tabletop* y de componentes de software que permiten identificar la interacción del usuario. El diseño de NEKO tomó en consideración tres aspectos principales: 1) ofrecer al usuario una interacción sin la necesidad de algún dispositivo de control artificial, 2) mantener una correspondencia en la ubicación de la entrada y salida, y 3) una arquitectura de software capaz de identificar la interacción del usuario con un tiempo de respuesta corto.

El prototipo de *tabletop* a diferencia de una mesa convencional, está compuesta de dos tableros, uno posicionado a cierta distancia por encima del otro. Con el propósito de facilitar

la construcción y reproducción del prototipo, la *tabletop* está compuesta de una mesa elaborada de fierro e instrumentada de una cámara web, un dispositivo Kinect, un proyector y una computadora.

El software de NEKO se encarga de capturar imágenes (a través de los dispositivos de hardware) de la interacción realizada por las manos y el rostro del usuario sobre el prototipo de *tabletop*. A partir de estas imágenes se extraen patrones relevantes que permiten manipular y visualizar el entorno virtual de forma semejante a la vida real. Al igual que la mayoría de *tabletops*, NEKO utiliza algoritmos de visión por computadora y modelos de clasificación. En este capítulo se presentó una visión general de la arquitectura de software de NEKO. En los siguientes dos capítulos se describen con mayor detalle las tareas involucradas para la interacción con objetos virtuales.

Capítulo 5

Modelos de clasificación

"Las ciencias no intentan explicar, tampoco interpretar, simplemente se enfocan en construir modelos."

John von Neumann

En el capítulo anterior se describió la organización de la arquitectura de hardware y, de forma general, los componentes de software que integran a NEKO. Este capítulo presenta una descripción detallada de los componentes de software relacionados con la construcción y uso de modelos de clasificación. En NEKO se construyen y emplean modelos de clasificación para detectar la forma y el movimiento del rostro y las manos del usuario, con el propósito de implementar la manipulación de objetos virtuales 2D/3D y desplegarlos de acuerdo a la perspectiva del usuario.

Más generalmente, la clasificación es la tarea de organizar objetos en una o diversas categorías predefinidas. En otras palabras: "Clasificación es la tarea de aprender una función objetivo f que asigne a un conjunto de atributos[1] $\mathbf{x}$ una etiqueta de clase y predefinida" [Tan et al., 2005]. La función objetivo es también conocida como *modelo de clasificación*. Existen diversas técnicas de clasificación, las cuales en su gran mayoría emplean algoritmos de aprendizaje que permiten construir modelos acordes a la relación existente entre un atributo y su etiqueta de clase. En la figura 5.1 se muestra de forma conceptual el funcionamiento de un modelo de clasificación.

Las situaciones en las cuales se ponen en práctica los modelos de clasificación son real-

[1]Un atributo es una propiedad o característica de un objeto que puede variar, ya sea de un objeto a otro o a través del tiempo.

Figura 5.1: Clasificación de un conjunto de atributos **x** a una etiqueta de clase y predefinida.

mente variadas. Por ejemplo, en 1920 Edwin Hubble fue el primero en estudiar la morfología de las galaxias. Utilizando el telescopio del observatorio Monte Wilson en California, Hubble fotografío una gran cantidad de galaxias y las clasificó con respecto a su forma. En el área de visión por computadora la detección y seguimiento de rostros humanos sobre marcos de videos es una tarea de clasificación bastante popular.

En NEKO la clasificación se emplea como un mecanismo para analizar la interacción del usuario sobre la *tabletop*, permitiendo fundamentalmente, extraer patrones relacionados con el movimiento de rostro y manos. Entre otras cualidades, un rostro provee gran variedad de elementos comunicativos ideales para construir un sinfín de aplicaciones. En nuestro caso la detección de rostros proporciona elementos relevantes con los cuales se acopla el entorno virtual, desplegado sobre la *tabletop*, a la perspectiva del usuario.

Otra de las tareas de clasificación con gran popularidad (y de envergadura considerable en el desarrollo de NEKO) es la detección de manos. Los seres humanos nos expresamos a través de las manos, por lo tanto es elocuente el uso de modelos de clasificación para su detección, en especial para el desarrollo de aplicaciones como el reconocimiento de gestos. Los gestos con las manos son una de las maneras más naturales que los humanos utilizamos para comunicarnos.

Por estas y otras razones, la detección de rostros y manos son de utilidad para el desarrollo de NUIs y, en general, para la selección y manipulación de objetos en espacios de interacción bidimensionales y tridimensionales.

Este capítulo aborda los componentes de software encargados de la *detección de rostro*,

detección de manos y *detección de gestos* en NEKO. A lo largo del capítulo se describen las tareas involucradas en cada uno de estos componentes, así como también las técnicas utilizadas para la construcción de los modelos.

5.1 Detección de rostro

Como se mencionó en los aspectos de diseño en el capítulo 4, para realizar una interacción *natural* con objetos virtuales sobre NEKO, es necesario que los objetos virtuales se desplieguen en conformidad con la perspectiva que atañe a la posición del rostro del usuario. Tal y como sucede en la vida real la perspectiva de los objetos cambia mientras trasladamos nuestra cabeza a una nueva ubicación. Por consiguiente, es ineludible el uso de una técnica para la detección de rostro y ajuste de perspectiva sobre NEKO.

Una técnica sobresaliente para la detección y seguimiento de rostro sobre marcos de videos es la *cascada de clasificadores con características Haar* [Viola and Jones, 2001]. Comúnmente esta técnica es conocida como el *detector Viola-Jones* por sus desarrolladores Paul Viola y Michael Jones. En pocas palabras, esta técnica utiliza una gran cantidad de muestras *positivas* de un objeto de interés en conjunto de muestras arbitrarias consideradas como *negativas* para generar un modelo de clasificación; el modelo se emplea posteriormente sobre alguna imagen donde se desea discernir el objeto de interés dentro de algún entorno en particular.

NEKO emplea las herramientas que ofrece OpenCV para construir e implementar un modelo de clasificación capaz de identificar el rostro del usuario cuando está interactuando sobre la *tabletop*. En la figura 5.2, las actividades a la izquierda (enlazadas mediante una flecha punteada) realizan la construcción del modelo de clasificación. Las actividades a la derecha corresponden al uso del modelo en tiempo de ejecución de NEKO para la detección de rostros y ajuste de perspectiva del entorno virtual. Nótese que las actividades para la construcción del modelo de clasificación se realizan antes de ejecutar el software de NEKO. A continuación se describen con mayor detalle cada una de estas actividades.

5.1.1 Procesamiento de marcos de video

La calidad en la imagen de cualquier cámara digital, entre otras cosas, está restringida a las condiciones de iluminación de su entorno. Analógicamente el desempeño de los sistemas

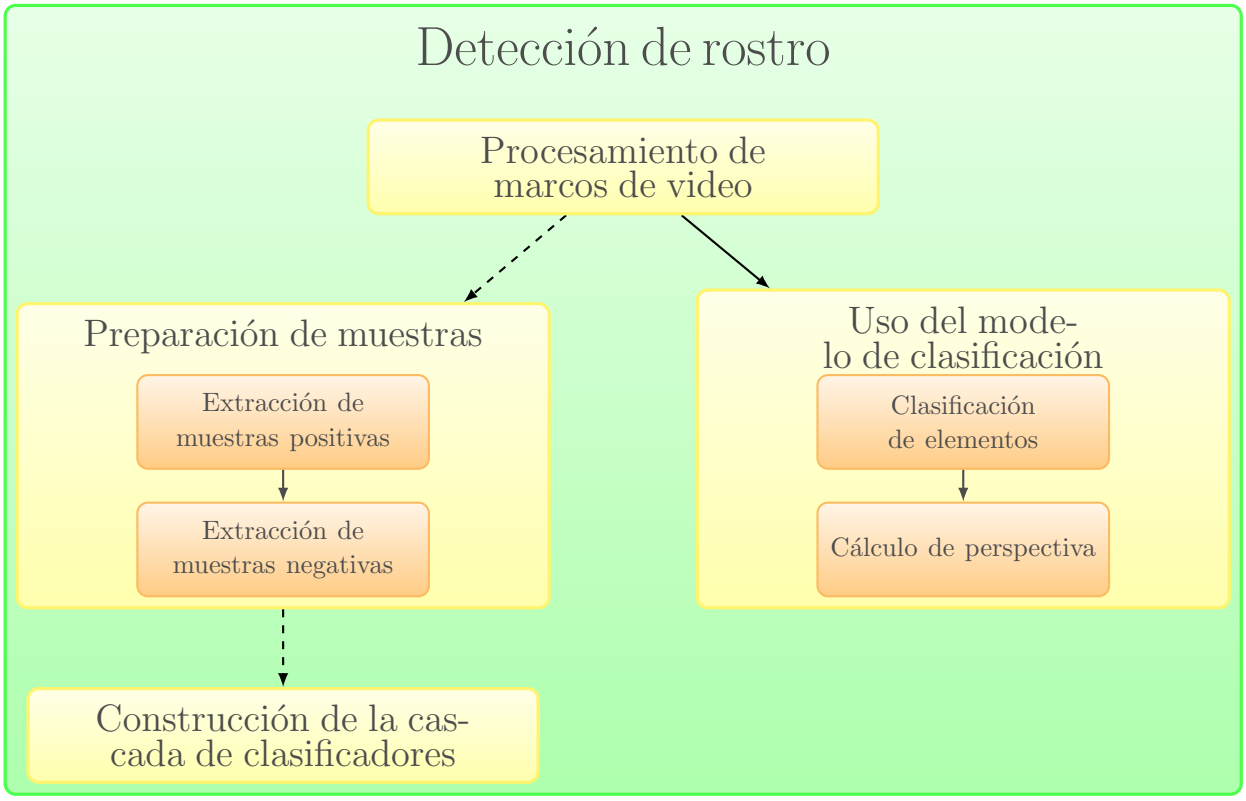


Figura 5.2: Tareas involucradas en la detección de rostro.

de visión por computadora, como NEKO, dependen en gran medida de imágenes con "buena" calidad. Sin embargo, no siempre se puede disponer de una fuente de iluminación adecuada. Por ello, se requiere un mecanismo alterno que otorgue flexibilidad a los cambios de iluminación en el entorno.

Una mecanismo utilizado frecuentemente en la fotografía para rectificar, de forma rápida, características esenciales de una imagen son los histogramas. Un histograma de imagen es una representación abstracta de los pixeles en una imagen. Por ello a menudo se emplean para subsanar, hasta cierto punto, problemas de iluminación.

El *procesamiento de marcos de video* en NEKO es la tarea dedicada, a través de histogramas de imágenes, a enmendar los cambios de iluminación sobre los marcos de videos capturados por la cámara web. Específicamente esta tarea equilibra el brillo y contraste de tales marcos. Cabe mencionar que esta tarea se realiza tanto en la construcción del modelo de clasificación como en el uso del mismo para la detección de rostro.

Para realizar esta tarea, primeramente es necesario extraer el marco de video actual de la cámara web. Después, dicho marco se transforma del espacio de colores RGB a escala de

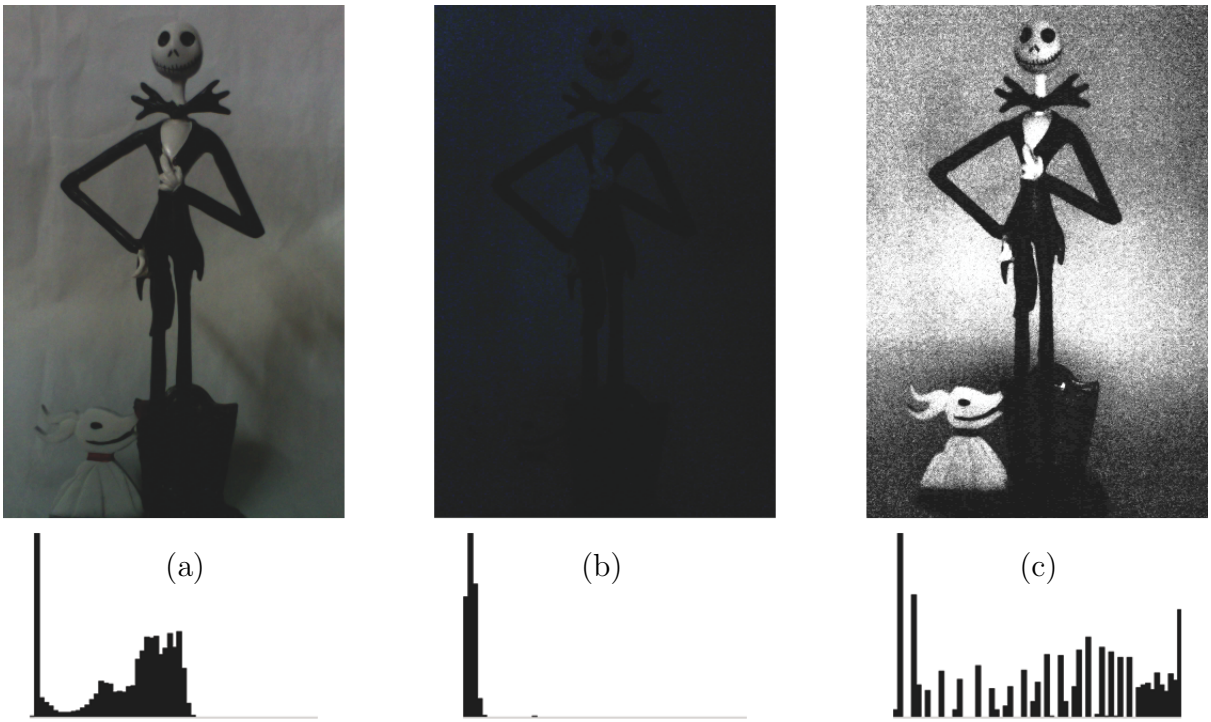


Figura 5.3: Ajuste en la iluminación de un marco de video.

grises con la función de OpenCV *cvtColor()*. Con esta función se obtiene una imagen de un solo canal de color, la cual requiere menos espacio en memoria y facilita la extracción de información en tareas futuras. Como se mencionó en el capítulo anterior, el software debe identificar la interacción del usuario en un tiempo de respuesta corto. Por ello se debe escatimar en lo posible el uso de los recursos. Por último, se ajusta el brillo y contraste de la imagen obtenida a través de la función *equalizeHist()*.

La figura 5.3 muestra tres imágenes con su respectivo histograma. Como se puede apreciar, el marco de video (a) capturado bajo una buena iluminación contiene algunas zonas obscuras que no representan mayor problema para distinguir el muñeco de resina. En cambio en el marco de video (b), capturado con poca luz, vagamente se distingue su silueta. Sin embargo, si al marco de video (b) se le ajusta el brillo y contraste el muñeco se aprecia con mayor claridad (c).

5.1.2 Preparación de muestras

En general, las técnicas de clasificación requieren de un conjunto de muestras ejemplo para construir un modelo. En nuestro caso dado que usamos *detector Viola-Jones* se requieren dos tipos de muestras: *positivas* y *negativas*. Las muestras positivas contienen exclusivamente

imágenes del objeto de interés, es decir rostros. Por el contrario, las muestras negativas son imágenes del entorno cercano a la *tabletop*. En esta tarea, *preparación de muestras*, se reúnen las muestras positivas y negativas que serán utilizadas en la construcción del modelo de clasificación.

Extracción de muestras positivas

La extracción de muestras positivas es el proceso donde se agrupan miles de imágenes que contengan únicamente el objeto de interés, en nuestro caso rostros de personas. Si bien en Internet existen múltiples repositorios que contienen gran cantidad de imágenes con rostros, no es posible emplear estas imágenes para la construcción del modelo porque la mayoría contiene objetos que no son de interés (como hombros o brazos). Por lo cual es necesario especificar la ubicación del rostro sobre cada imagen. Hacer este trabajo de forma manual para miles de imágenes, sin mencionar lo ineficiente de esta metodología, puede tomar días.

Una alternativa a esta problemática es generar nuestras propias imágenes a partir de un video. Con la ayuda de una cámara web, colocada bastante cerca a la faz de una persona, se graba un video donde únicamente se capture el rostro de una persona. Así mediante este video se pueden extraer miles de imágenes con rostros en cuestión de minutos.

El algoritmo 1 describe el proceso utilizado para la extracción de imágenes a partir de un video. Primeramente se crea en disco duro un archivo de texto plano que será de utilidad posteriormente para la generación de las muestras positivas (línea 1). Enseguida para cada marco del video grabado por la cámara web se construye un recuadro con dimensiones acordes al tamaño del rostro de la persona (líneas 2-4). Finalmente se almacena en disco duro el marco de video y se agrega en el archivo de texto plano el tamaño del recuadro (líneas 5-6). Cabe mencionar que las dimensiones del rostro se especifican de forma manual dependiendo del tamaño del rostro sobre el marco de video.

A través de la utilidad de OpenCV *opencv_createsamples* y del archivo de texto plano generado por el algoritmo anterior, se extraen las muestra positivas, es decir el rostro de la persona, de cada marco de video almacenado. Asimismo, mediante la utilidad *opencv_createsamples*, se reducen las dimensiones de cada muestra positiva a un tamaño de 20 x 20 pixeles y se transforma del espacio de colores RGB a escala de grises con el propósito de agilizar la construcción del modelo de clasificación.

Algorithm 1 Extracción de muestras positivas de un video

Require: *videoCamaraWeb*, *dimensionRostro*

Ensure: Marcos de video.

1: Crea el archivo *posicionRostro* en disco duro
2: **for all** $marco \in videoCamaraWeb$ **do**
3: $centroMarco \leftarrow$ extraer el punto medio de *marco*
4: $recuadroRostro \leftarrow centroMarco + dimensionRostro$
5: Almacena *marco* en disco duro.
6: Agregar al archivo *posicionRostro* el *recuadroRostro*
7: **end for**

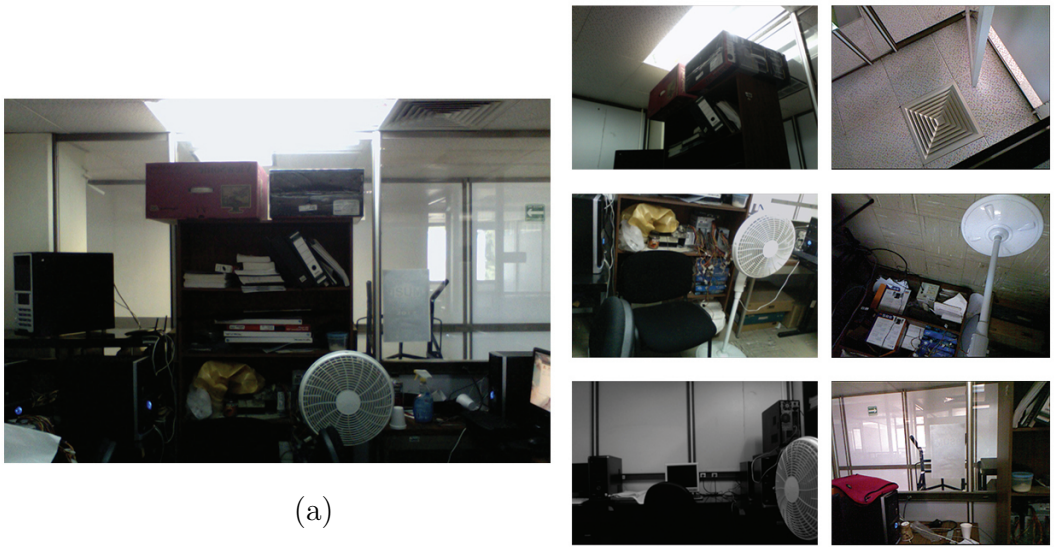
La figura 5.4 contiene un ejemplo de muestra positiva extraída mediante la utilidad *opencv_createsamples* de un marco de video capturado por la cámara web.

Figura 5.4: Ejemplo de rostro extraído.

Extracción de muestras negativas

Por otra parte, la extracción de muestras negativas es un proceso más sencillo de realizar. Simplemente, con una cámara digital se capturan imágenes del entorno cercano a la *tabletop*. Sin embargo, con la finalidad de construir un modelo de clasificación robusto, se decidió capturar mayor número de imágenes a los objetos que la cámara web, ubicada en el tablero superior de la *tabletop*, captura de fondo mientras el usuario interactúa. Las muestras negativas capturadas, a diferencia de las muestras positivas, no requieren algún tratamiento especial.

La figura 5.5 muestra una imagen con los elementos de fondo que captura la cámara web (a) y sus respectivas muestras negativas capturadas (b).



(a) (b)

Figura 5.5: Ejemplo muestras negativas.

5.1.3 Construcción de la cascada de clasificadores

La *construcción de la cascada de clasificadores* es la tarea en la cual, a partir de las muestras positivas y negativas extraídas en la tarea *preparación de muestras*, se construye un modelo de clasificación con la técnica *cascada de clasificadores con características Haar* propuesta por Paul Viola y Michael Jones. En sí, el modelo de clasificación que se construye con esta técnica es un árbol de decisión binario [Breiman et al., 1984] (al cual Viola y Jones le nombraron *cascada de clasificadores*) capaz de identificar sobre una imagen objetos relevantes.

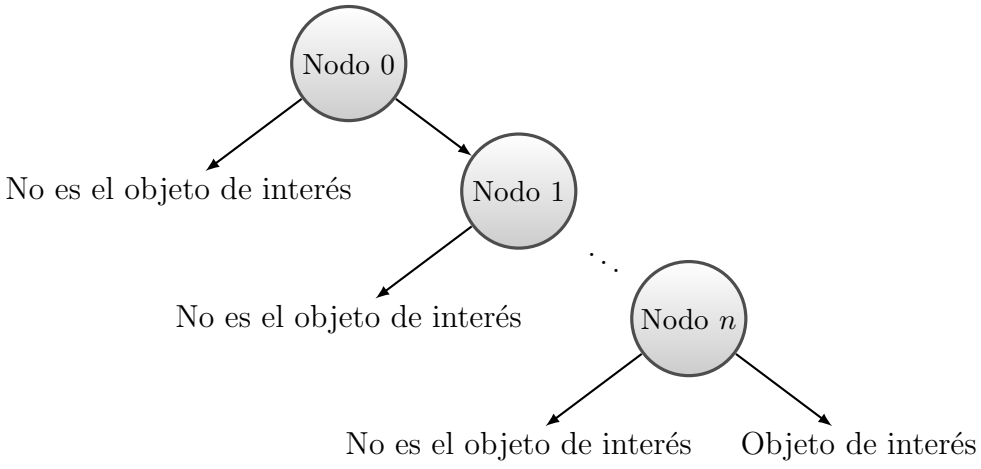


Figura 5.6: Estructura de una cascada de clasificadores.

La figura 5.6 muestra la estructura general de una *cascada de clasificadores*. Cada nodo del árbol de decisión binario es otro árbol de decisión más simple (de ahí el nombre de cascada de clasificadores) comúnmente llamado *clasificador débil*[2]. Para cada nodo i se obtiene una etiqueta de clase e, comparando si un valor v para un atributo a en particular es menor o mayor a un umbral u:

$$e = \begin{cases} +1 \textbf{ si } v_i \geq u_i \\ -1 \textbf{ si } v_i < u_i \end{cases} \tag{5.1}$$

Donde una etiqueta de clase +1 indica que existe un objeto de interés; por el contrario -1 determina que no se han encontrado características del objeto de interés. Los nodos se acomodan de menor a mayor capacidad de reconocimiento permitiendo en primera instancia etiquetar una gran cantidad de objetos como relevantes sobre la imagen. Sin embargo en los nodos con mayor capacidad de reconocimiento se eliminan falsos positivos aceptados por nodos anteriores.

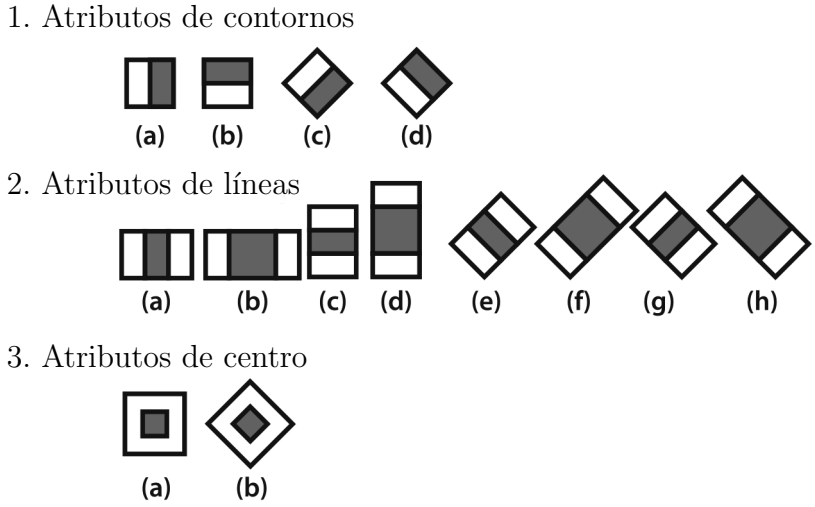


Figura 5.7: Atributos basados en las funciones Haar.

Con la idea de facilitar la construcción de una *cascada de clasificadores* Viola y Jones utilizan unos atributos especiales, basados en las funciones Haar [Papageorgiou et al., 1998], para representar a los elementos de una imagen (ver figura 5.7). A diferencia del uso de pixeles, estos atributos pueden englobar características que facilitan la construcción de una *cascada de clasificadores*. Como se puede apreciar en la figura 5.7, cada atributo contiene

[2]Un *clasificador débil* por sí solo es ineficaz para una tarea de clasificación pero cuando varios se utilizan en conjunto se puede alcanzar un alta grado de precisión utilizando tasas muy bajas de rechazo.

una combinación de rectángulos blancos y negros. Así cuando un atributo se superpone sobre una imagen, se evalúa la diferencia de intensidad que existe en los pixeles debajo de la región blanca con los pixeles de la región obscura. Si la diferencia de intensidad es superior a un umbral entonces el atributo está presente. Por ejemplo, en la figura 5.8 se muestra la superposición del atributo 3a sobre un rostro, puesto que la región de los labios es más obscura que la zona cercana a dicha región es muy probable que el atributo sea considerado como representativo para simbolizar los labios. Así sucesivamente los elementos del rostro se van remplazando mediante estos atributos.

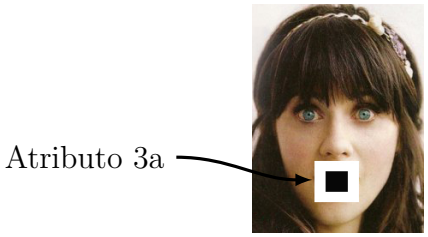


Figura 5.8: Ejemplo de superposición de el atributo 3a.

En general, para la creación de una *cascada de clasificadores* se determinan los atributos basados en las funciones Haar implícitos en las muestras positivas a través de una técnica de *machine learning* denominada *boosting* [Freund and Schapire, 1996]. OpenCV ofrece la aplicación *opencv_traincascade* para la construcción de una *cascada de clasificadores* utilizando algoritmos *boosting*.

Mediante las muestras positivas y negativas, obtenidas de la tarea *preparación de muestras*, se construyó una *cascada de clasificadores* para la detección de rostros sobre los marcos de video de la cámara web, utilizando la aplicación *opencv_traincascade*. Los parámetros que se especificaron a *opencv_traincascade* fueron:

- **Parámetros de las muestras.**

  - Cantidad de muestras positivas. A pesar de que se obtuvieron más de 4,000 muestras positivas, en la tarea *preparación de muestras* se utilizaron únicamente 2,000 de ellas debido a la similaridad que existía entre las imágenes (se utilizó el rostro de una sola persona para la extracción de muestras). Además otra razón de utilizar esta cantidad de muestras fue para agilizar el tiempo en la construcción de la *cascada de clasificadores*.

- Dimensiones de las muestras positivas. La elección del tamaño para cada muestra positiva generada, 20x20 pixeles, se debió por los resultados obtenidos en [Lienhart et al., 2003]. También se hicieron pruebas con dimensiones hasta de 36x36 pixeles. Sin embargo, se observó que durante la construcción del modelo el espacio en memoria de la computadora se agotaba con dimensiones grandes. Por ello se optó en utilizar la misma configuración que en [Lienhart et al., 2003].

- Cantidad de atributos basados en las funciones Haar. Con la finalidad de simplificar la construcción del modelo de clasificación, se decidió emplear únicamente los atributos verticales y horizontales, es decir los atributos 1a, 2a, 2b, 3a, 1b, 2c y 2d mostrados en la figura 5.7. Si bien los atributos rotados ayudan a mejorar la detección de rostros se observó que éstos incrementan el tiempo en la construcción de la *cascada de clasificadores* y en la búsqueda de rostros sobre una imagen.

- Cantidad de muestras negativas. El número de muestras negativas capturadas del entorno cercano al prototipo de *tabletop* de NEKO fue de 700. No obstante, se utilizaron solamente 500 de ellas para evitar mermar los recursos durante la construcción del modelo; cada muestra negativa ostentaba un tamaño de 640x480 pixeles.

- **Parámetros de la *cascada de clasificadores***

  - Número de nodos para la *cascada de clasificadores*. La configuración por omisión que sugiere la aplicación *opencv_traincascade* permite construir una *cascada de clasificadores* de 20 nodos. Se decidió utilizar este número de nodos primordialmente por el tiempo considerable que se requiere en construir un árbol binario de más de 20 nodos con los parámetros antes mencionados para las muestras positivas y negativas.

  - Algoritmo *boosting*. Existen tres algoritmos *boosting* que se pueden utilizar para la construcción de una *cascada de clasificadores*: *Adaboost gentil*, *Adaboost discreto* y *Adaboost real*. Todos ellos son idénticos en cuanto a su complejidad computacional, no obstante su diferencia radica en el algoritmo que cada uno utiliza. En [Lienhart et al., 2003] demuestran que bajo las mismas condiciones *Adaboost gentil* obtiene un mejor desempeño que los otros dos. Bajo esta premisa se decidió utilizar este

algoritmo para la construcción de la *cascada de clasificadores*.

- Tasa de detección por nodo. Este parámetro indica la tasa de detección que se desea obtener para cada árbol de decisión implícito en cada nodo de la *cascada de clasificadores*. Típicamente cada nodo se construye para alcanzar una tasa alta de detección a costa de varios falsos positivos. Con este parámetro se puede estimar la precisión general que alcanzará toda la *cascada de clasificadores* mediante d^n, donde d es la tasa de detección por nodo y n es el número de nodos de la cascada. En nuestro caso se decidió asignar a cada nodo un valor de 0.9999 (es decir 99.99 %), con lo cual se obtuvo una precisión general de $0.9999^{20} = 99.8\,\%$.

- Tasa de rechazo por nodo. La tasa de rechazo indica el porcentaje de falsos positivos que se eliminarán en cada nodo y por lo general este parámetro suele tener valores muy bajos. Un nodo por sí solo con una tasa de rechazo baja es ineficaz para una tarea de clasificación pero cuando varios se utilizan en conjunto se puede alcanzar un alta grado de precisión. La tasa de rechazo general se calcula mediante r^n, donde r es la tasa de rechazo y n es el número de nodos de la cascada. En nuestro caso se decidió un valor de 0.5 (50 %) concediendo una tasa de rechazo general de $0.5^{20} = 9.6{*}10^{-7}$

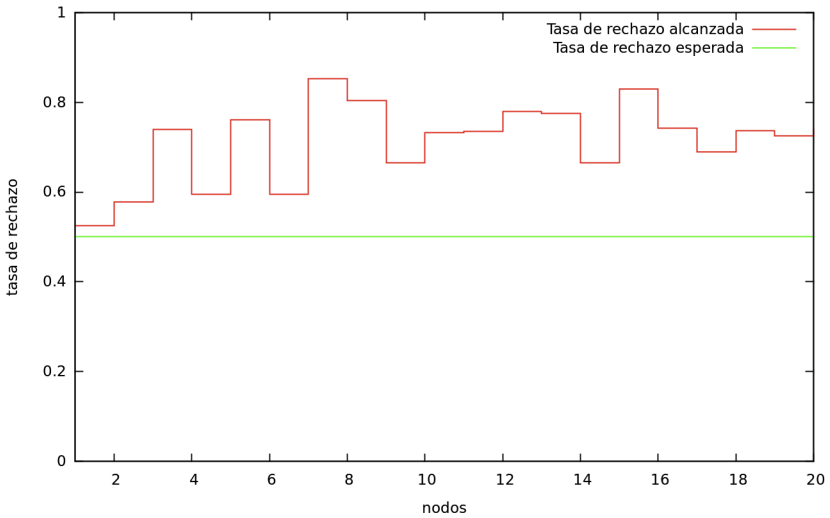


Figura 5.9: Tasa de rechazo alcanzada para cada nodo.

Como se puede observar el tiempo necesario para construir una *cascada de clasificadores* es relativo a la configuración de cada uno de estos parámetros y del número de muestras positivas y negativas. El proceso para la construcción de la *cascada de clasificadores* con los

parámetros antes mencionados tomó un día. La figura 5.16 muestra una gráfica con la tasa de rechazo obtenida para cada nodo. Como se puede observar la tasa de rechazo obtenida por nodo es mayor a la esperada, lo que significa que cada nodo eliminará más del 50 % de los falsos positivos.

5.1.4 Uso del modelo de clasificación

Las tareas anteriores se emplean para construir un modelo de clasificación, es decir una *cascada de clasificadores*, para la detección de rostros. En la tarea, *uso del modelo de clasificación*, descrita a continuación, se emplea dicho modelo de clasificación sobre los marcos de video de la cámara web, capturados durante el uso de NEKO, para desplegar el entorno virtual conforme a la perspectiva del usuario. En general, la *detección multiescala* consiste en recorrer pixel por pixel cada marco de video de la cámara web buscando determinar si existe algún rostro. Este proceso se realiza mediante la función *detectMultiScale()* de la biblioteca OpenCV.

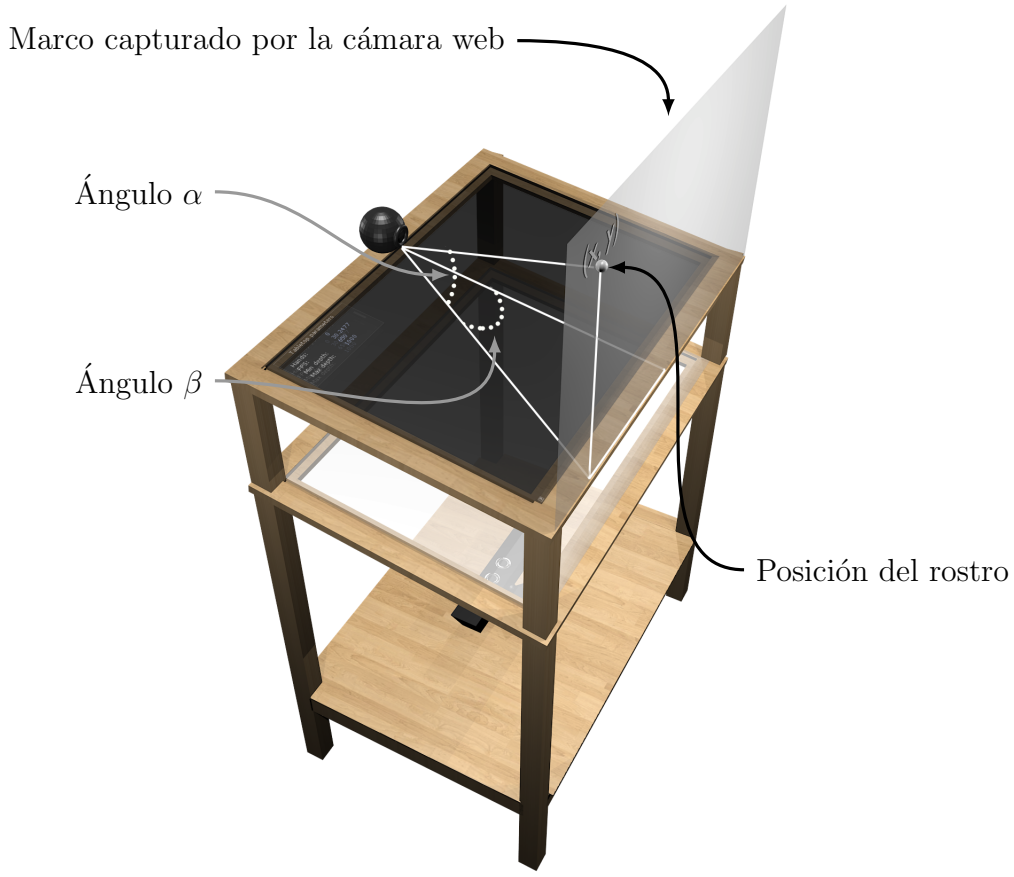


Figura 5.10: Cálculo de los ángulos α y β.

Los ángulos necesarios para calcular la perspectiva del entorno virtual se muestran en la figura 5.10. En razón a la posición del rostro del usuario obtenida con la función de OpenCV *detectMultiScale()* se calcula un vector hasta la posición de la cámara web. Suponiendo que el rostro del usuario siempre se encuentra a una distancia similar al ancho del tablero superior de la *tabletop*, se evalúan los ángulos que forma el vector respecto al tablero superior y al centro del marco capturado por la cámara web. Así se rota la escena respecto al valor de los ángulos calculados.

El algoritmo 2 describe el proceso para el cálculo de perspectiva del entorno virtual. Primeramente, se compara el tamaño de los rostros previamente identificados mediante la función *detectMultiScale()* de la biblioteca OpenCV(líneas 1-6). Suponiendo que en el marco de video exista más de un usuario, el rostro de mayor tamaño sería el del usuario más próximo a la *tabletop*. Posteriormente se calcula el vector del rostro del usuario hacia la posición de la cámara web (línea 8). Finalmente se evalúan los ángulos que forma el vector y se rota el entorno virtual con respecto a estos ángulos (líneas 9-11).

Algorithm 2 Cálculo de perspectiva del entorno virtual

Require: $rostrosMarcoVideo$, $posicionCamaraWeb$

Ensure: Perspectiva del entorno virtual.

1: $rostroMasCercano \leftarrow$ vacío
2: **for all** $rostro \in rostrosMarcoVideo$ **do**
3: **if** $rostroMasCercano$ es menor que $rostro$ **then**
4: $rostroMasCercano \leftarrow rostro$
5: **end if**
6: **end for**
7: **if** $rostroMasCercano$ es diferente de vacío **then**
8: $vectorRostro \leftarrow posicionCamaraWeb - rostroMasCercano$
9: $\alpha \leftarrow atan(vectorRostro.z/vectorRostro.y)$
10: $\beta \leftarrow atan(vectorRostro.z/vectorRostro.x)$
11: Rota entorno virtual con α y β
12: **end if**

La figura 5.11 varias imágenes de un objeto físico “arriba” de un objeto virtual que permiten apreciar el resultado del proceso para el cálculo de perspectiva. El objeto físico está posicionado sobre el cristal obscuro del tablero superior del prototipo de *tabletop* de NEKO; el objeto virtual se está desplegando por debajo del cristal. Las imágenes fueron cap-

turadas mediante una cámara fotográfica desde tres diferentes perspectivas: izquierda, central y derecha. un objeto físico y un objeto virtual capturado desde tres diferentes perspectivas. En las imágenes de la fila superior el objeto virtual no se ajusta a la perspectiva del usuario, por lo que se observa de forma similar en las tres imágenes. En cambio en la fila inferior el objeto virtual asemeja un comportamiento similar al objeto físico.

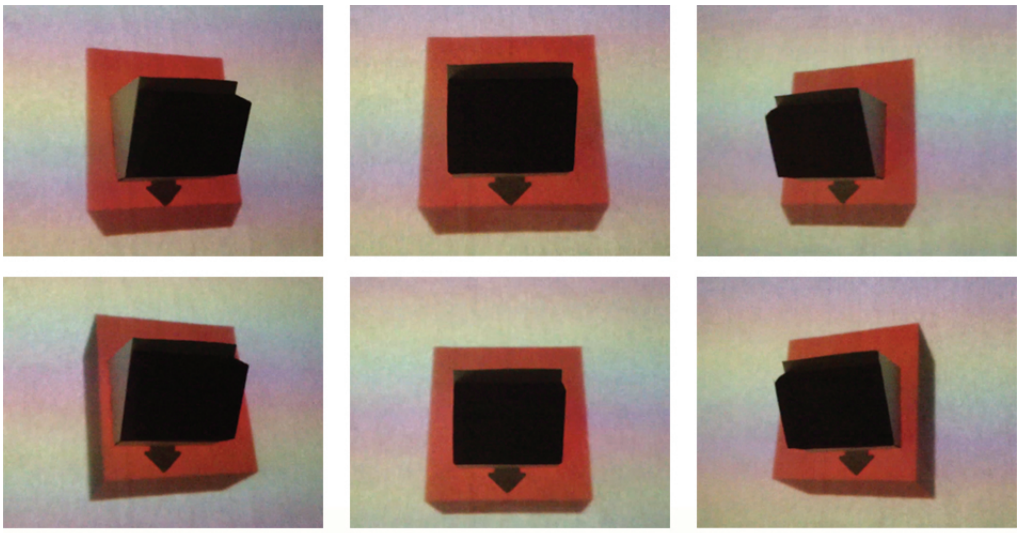


Figura 5.11: Cambio de perspectiva de un objeto físico y un objeto virtual.

5.2 Detección de manos

En los últimos años, las técnicas para el reconocimiento de manos han recibido bastante atención en el área de visión por computadora. La posibilidad de identificar una mano y de seguir los movimientos que ésta realiza proporciona nuevas oportunidades en HCI, sobre todo en el desarrollo de interfaces de usuario.

La detección de las manos es una tarea imprescindible en NEKO, en especial porque las manos son el principal método de interacción sobre el prototipo de *tabletop*. Entre otras cosas la detección de manos facilita la implementación de una tarea para el reconocimiento de gestos y la manipulación de objetos virtuales. En NEKO, se desarrollaron dos mecanismos para realizar el reconocimiento de manos. El primero se realizó mediante la biblioteca OpenNI. El segundo, basado en la biblioteca OpenCV, surgió a partir de las limitaciones encontradas en el primero.

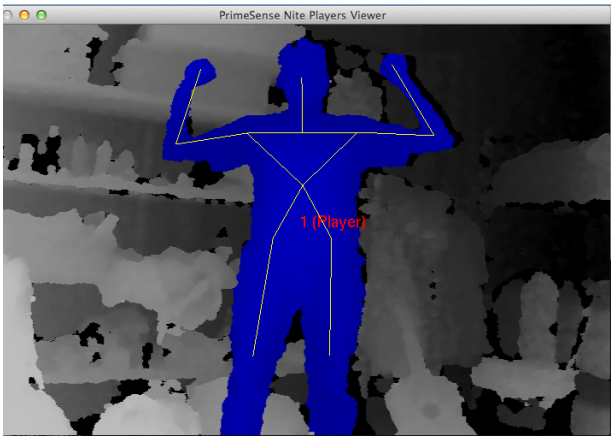


Figura 5.12: Esqueleto básico creado con la biblioteca NITE.

La biblioteca OpenNI en conjunto del *middleware* NITE[3] (ambos desarrollados por PrimeSense) ofrecen herramientas para rastrear movimientos realizados con el cuerpo. A mayor detalle, cuando un usuario está interactuando, NITE permite procesar la información del sensor de profundidad del Kinect para crear estructuras del cuerpo humano y analizar sus movimientos. Entre estas estructuras se encuentra un esqueleto básico con articulaciones, mostrado en la figura 5.12, el cual permite calcular la orientación del usuario y su centro de masa. Otras de las actividades que permite realizar NITE es la detección de "manos" y el reconocimiento de gestos. Estas dos últimas actividades fueron usadas como base para el desarrollo de NEKO (ver Apéndice A:*Detector de manos basado en OpenNI/NITE*).

Desafortunadamente NITE es un *middleware* con algoritmos propietarios y, semejante a una caja negra, no se puede indagar con facilidad la metodología que utilizan para identificar los gestos ni las manos. Además, NITE no está diseñado para funcionar con marcos de video (capturados por el sensor de profundidad del dispositivo Kinect) que contengan únicamente manos. Si a esto le añadimos que el rastreo de las manos presenta problemas cuando algún elemento en la escena está más cerca que las propias manos (como el tablero inferior del prototipo de *tabletop* de NEKO), NITE deja de ser una opción para la detección de manos.

Por otro lado, al igual que con la detección de rostro, si se utiliza una *cascada de clasificadores* es posible identificar las manos del usuario sobre los marcos de profundidad del dispositivo Kinect, sin la necesidad de efectuar un movimiento inicial como en NITE, incluso aun si el tablero inferior está a menor distancia del dispositivo Kinect que las manos.

La figura 5.13 describe el proceso realizado para la creación de una *cascada de clasifi-*

[3]http://openni.org/Downloads/OpenNIModules.aspx

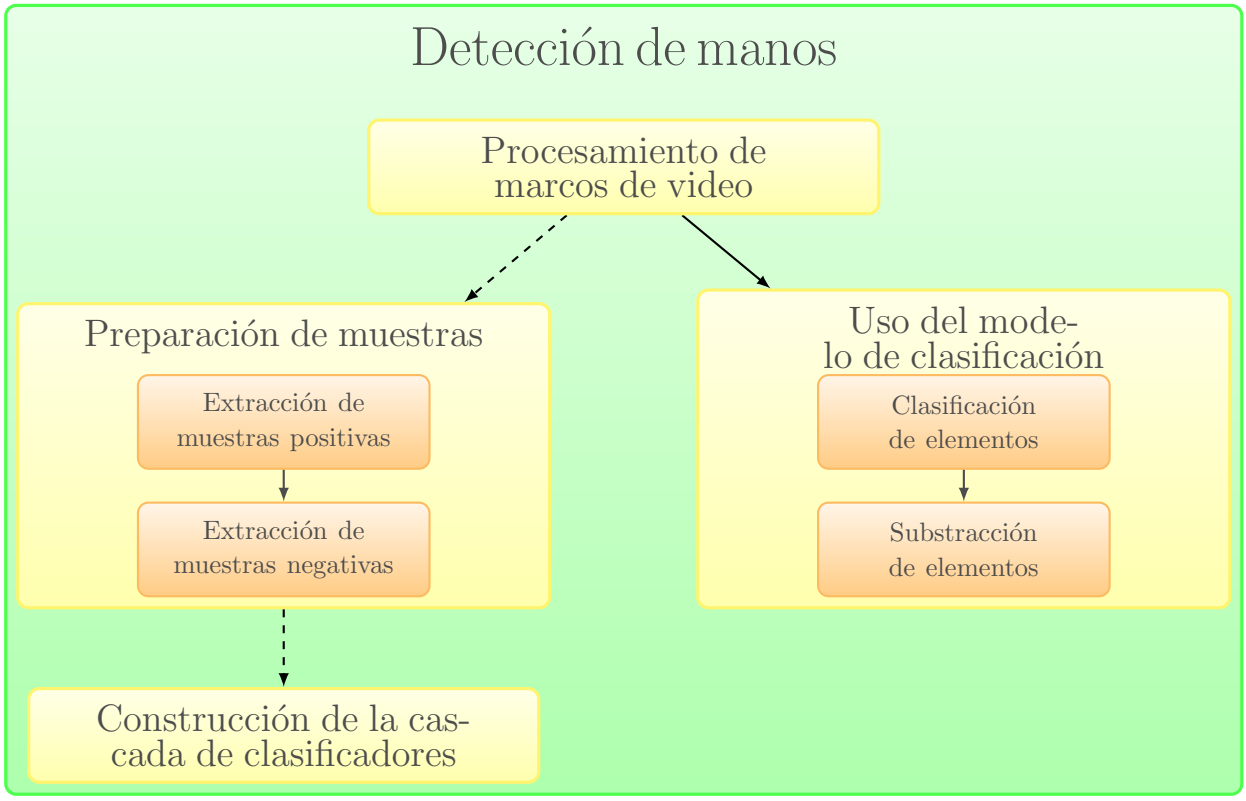


Figura 5.13: Tareas involucradas en la detección de manos.

cadores que permitió la detección de manos sobre NEKO. De forma similar a la detección de rostro, el proceso de la figura 5.13 está conformado de tres actividades para la construcción del modelo (las cuales son externas al flujo de datos de NEKO) y dos actividades para la detección de manos durante el uso de NEKO. A continuación se describe cada una de estas actividades

5.2.1 Procesamiento de marcos de video

A diferencia de la cámara web, el dispositivo Kinect se encuentra ubicado en la base del prototipo de *tabletop* de NEKO. Por ello es de esperarse que la iluminación en los marcos de video de la cámara RGB del Kinect (se recordará que el dispositivo Kinect cuenta con una cámara RGB y un sensor de profundidad) no sea del todo favorable. Sobre todo porque la luz del entorno se obtiene por una fuente de iluminación ubicada en el techo, la cual está emitiendo en dirección opuesta a la orientación de la cámara RGB del Kinect. Para solucionar este problema es necesario, al igual que en la detección de rostro, usar una tarea para ajustar el brillo y contraste de los marcos de video.

Primeramente los marcos de video se transforman del espacio de colores RGB a escala de grises con la función de OpenCV *cvtColor()*. Posteriormente se ajusta el brillo y contraste de la imagen a escala de grises a través de la función *equalizeHist()*. En la figura 5.15 se aprecian los resultados de ajustar el brillo y contraste a un marco de la cámara RGB del Kinect.

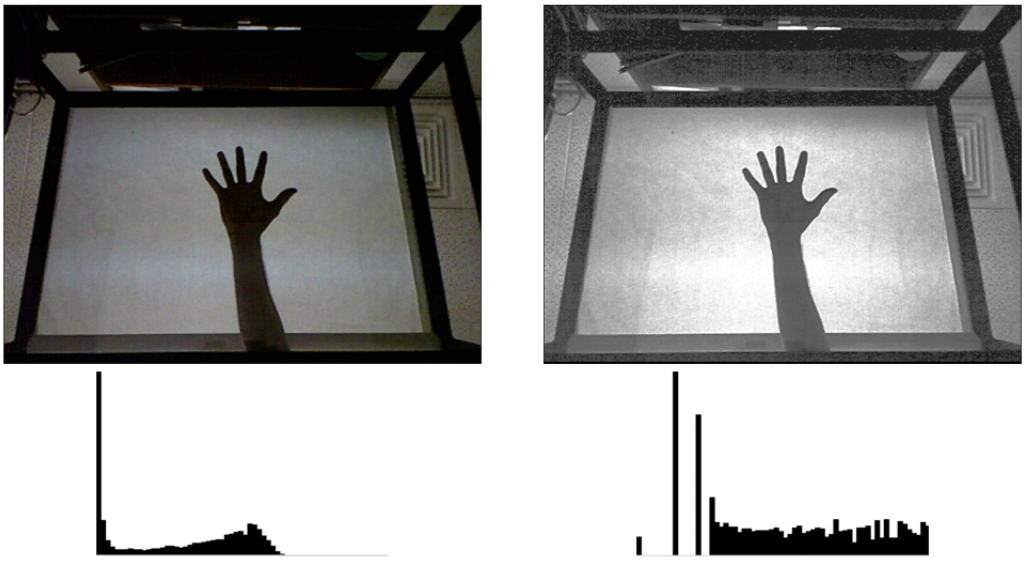


Figura 5.14: Ajuste de brillo y contraste a un marco de video capturado por la cámara RGB del Kinect.

5.2.2 Preparación de muestras

Como se mencionó anteriormente, la técnica propuesta por Paul Viola y Michael Jones requiere de un conjunto de muestras positivas y negativas para construir un modelo de clasificación. Si bien las muestras positivas se pueden generar a partir de un video como en la detección de rostro, en lugar de ello se decidió aprovechar el detector de manos basado en OpenNI/NITE.

El algoritmo 3 describe el proceso para la extracción de muestras positivas. Reutilizando el desarrollo del detector de manos basado en OpenNI/NITE, se identifica la posición de la mano sobre los marcos de video del sensor de profundidad del dispositivo Kinect (líneas 1-7). Dado que el sensor de profundidad y la cámara RGB del Kinect capturan la misma escena de forma simultanea, se emplea la posición de la mano sobre el marco de video del sensor de profundidad para segmentar un recuadro de imagen del marco de video de la cámara RGB (líneas 8-9). Finalmente, se guarda el recuadro de imagen en disco duro (línea 10).

Algorithm 3 Creación de muestras positivas para la detección de manos

Require: *contextoOpenNI*, *marcoVideoRGB*

Ensure: Muestras positivas

1: Inicializa $generadorManos$ con la información de $contextoOpenNI$
2: Inicializa $generadorGestos$ con la información de $contextoOpenNI$
3: Define espacio de búsqueda para $generadorGestos$
4: Inicia captura de $generadorManos$ y $generadorGestos$
5: **for all** $movimiento \in generadorGestos$ **do**
6: **if** *movimiento* es algún gesto **then**
7: $posicionMano \leftarrow$ obtén posición de $generadorManos$
8: Convierte $posicionMano$ a coordenadas proyectivas
9: Extrae $recuadroMano$ de $marcoVideoRGB$ a partir de $posicionMano$
10: Almacena $recuadroMano$ en disco duro
11: **end if**
12: **end for**

Posteriormente con la utilidad *opencv_createsamples* se reduce el tamaño de cada recuadro de imagen almacenado en disco duro a una dimensión de 20x20 pixeles, asimismo cada recuadro se transforma del espacio de colores RGB a escala de grises. La figura 5.15 contiene una muestra positiva, extraída de un marco de video de la cámara RGB del dispositivo Kinect, después de emplear el algoritmo 3 y la utilidad *opencv_createsamples*.

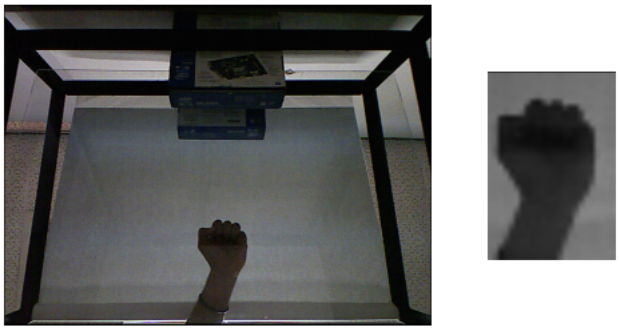


Figura 5.15: Extracción de muestra

Respecto a las muestras negativas, se usaron las mismas imágenes capturadas para la construcción del modelo de clasificación de rostro y se agregaron imágenes de los tableros de

la mesa y del techo.

5.2.3 Construcción de la *cascada de clasificadores*

El proceso para generar la *cascada de clasificadores* para la detección de manos es idéntico al descrito anteriormente en la detección de rostro. No obstante, debido a los constantes cambios de iluminación que sufren los marcos de video de la cámara RGB del Kinect, se decidió proporcionar una mayor tasa de detección en cada nodo de la *cascada de clasificadores*.

La tasa de detección otorgada a cada uno de los 20 nodos fue de un 99.9999 % con la cual se obtuvo una precisión general de $0.999999^20 = 99.99\,\%$,igualmente a cada nodo se le concedió una tasa de rechazo de 50 %. Debido a los resultados obtenidos en [Lienhart et al., 2003] y en la *cascada de clasificadores* para la detección de rostros se utilizó el algoritmo *boosting Adaboost gentil*.

A diferencia de la detección de rostros, el proceso para la construcción de la *cascada de clasificadores* con estos parámetros tomó un poco más de dos días. La figura 5.16 muestra la tasa de rechazo obtenida para cada nodo.

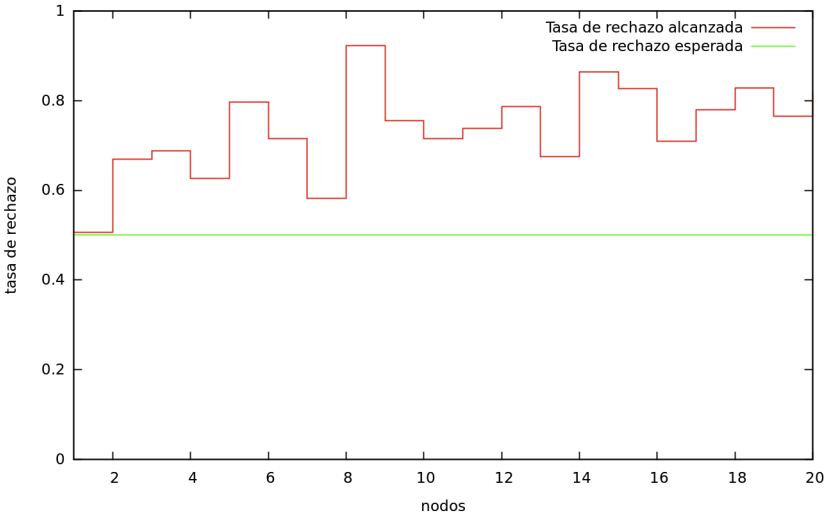


Figura 5.16: Tasa de rechazo alcanzada para cada nodo.

5.2.4 Uso del modelo de clasificación

Una vez construida la *cascada de clasificadores* con las tareas anteriores se utiliza la función *detectMultiScale()* de la biblioteca OpenCV para emplear la cascada e identificar las manos del usuario sobre los marcos de video de la cámara RGB del Kinect. La figura 5.17 muestra una sucesión de marcos de video donde se identifica una mano mediante la *cascada de clasificadores*.

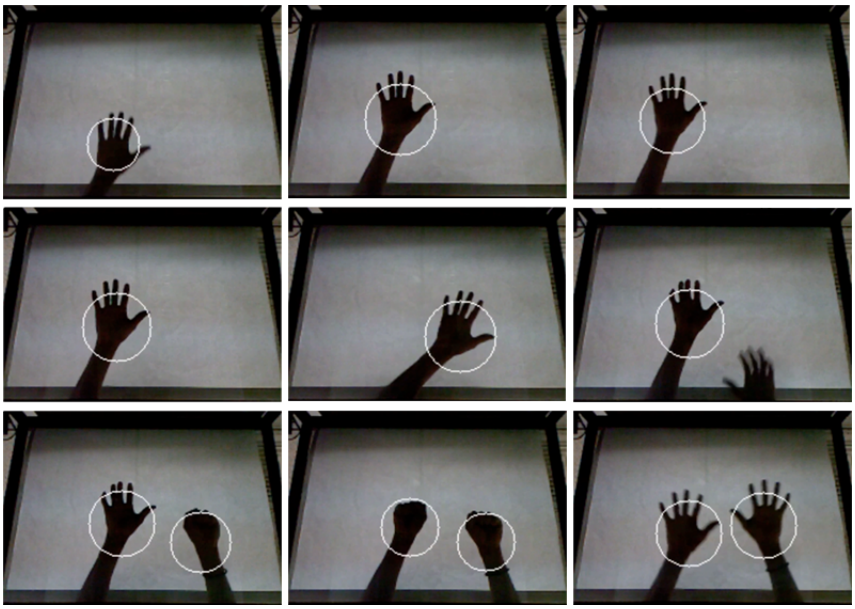


Figura 5.17: Detección de manos mediante una *cascada de clasificadores*.

Cabe mencionar que los tableros de la mesa no representan mayor problema en la detección de las manos como en el detector basado en OpenNI/NITE. Sin embargo, para evitar problemas en la manipulación de objetos virtuales, se decidió eliminar los tableros de los marcos de video del sensor de profundidad. Gracias a la carencia de movilidad del dispositivo Kinect, situado justo en la base de la mesa, es posible distinguir elementos estáticos (como los tableros de la *tabletop*) de las manos del usuario mientras está interactuando.

Debido a que los marcos de profundidad del dispositivo Kinect contienen bastante ruido, se decidió construir un modelo inicial y actualizarlo constantemente con la información de cada marco de profundidad. Para ello se utilizó el *método gaussiano de mezcla* el cual se realiza mediante las siguientes fórmulas:

$$\mu_t = (1 - \alpha)\mu_{t-1} + \alpha p_t \tag{5.2}$$

$$\sigma^2 = \big(1-\alpha\big)\sigma_{t-1}^2 + \alpha(p_t - \mu_t)^2 \tag{5.3}$$

Donde μ es el valor promedio de ejecución en un tiempo t de un pixel p con una tasa de aprendizaje α y una varianza de ejecución σ^2. Es decir el método gaussiano de mezcla obtiene un modelo base a partir del primer marco de video del sensor de profundidad. Posteriormente, con la obtención de nuevos marcos de videos, identifica los pixeles que no pertenecen al modelo inicial y crea otro modelo con estos pixeles. Si un pixel aparece muy frecuentemente sobre los marcos entonces se considerada parte de un elemento irrelevante en la escena. En cambio si su presencia es esporádica se considera como un elemento relevante y se elimina de los modelos.

OpenCV ofrece una implementación del método gaussiano de mezcla a través de la clase *BackgroundSubtractorMOG*. Esta clase únicamente requiere como parámetros una tasa de aprendizaje α y, obviamente, una marco de video. Así mediante las ecuaciones (5.2) y (5.3) se calculan los modelos y se eliminan los elementos no relevantes del marco de video del sensor de profundidad. La figura 5.18 muestra un marco de video al cual se le eliminaron los tableros de la mesa mediante la clase *BackgroundSubtractorMOG*.

Figura 5.18: Extracción de elementos relevantes de los marcos de video.

5.3 Detección de gestos

Laurel utiliza la siguiente definición de gesto [Laurel and Mountford, 1990]: “Un gesto es un movimiento del cuerpo que contiene información. Decir adiós es un gesto. Presionar una tecla no es un gesto, porque el movimiento de un dedo en su camino a pulsar una tecla no es

ni observado ni significativo". Acorde a la definición anterior, la tarea de detección de gestos incluye observar y registrar los movimientos significativos de las manos cuando el usuario está interactuando sobre el prototipo de *tabletop* de NEKO.

En NEKO, se manejan varios gestos que facilitan la interacción con objetos virtuales y el manejo del prototipo de *tabletop*. La figura 5.19 muestra los gestos utilizados para estas funcionalidades, los gestos (a) y (b) permiten al usuario moverse entre los entornos virtuales (como se recordará NEKO emplea entornos virtuales que permiten separar los objetos virtuales). Asimismo, el gesto (c) permite iniciar la calibración de la superficie táctil durante la ejecución de NEKO.

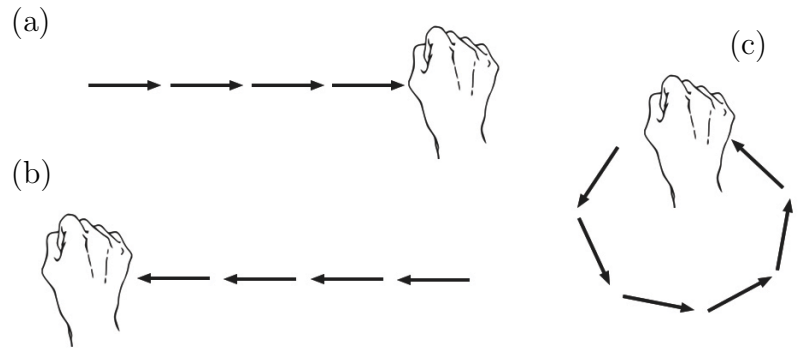


Figura 5.19: Conjunto de movimientos reconocidos como gestos.

La *detección de gestos* utiliza una técnica de clasificación que ha recibido atención considerable: las máquinas de vectores de apoyo (Support Vector Machine, SVM). Esta técnica tiene sus raíces en la teoría del aprendizaje estadístico y ha mostrado resultados prometedores en gran variedad de aplicaciones prácticas. Para nuestro propósito se ha utilizado esta técnica para construir un modelo de clasificación capaz de discernir los movimientos de las manos del usuario entre los tableros de la mesa.

De forma general, la detección de gestos involucra tres tareas (ver figura 5.20): *detección de pose*, *construcción del modelo* y *aplicación del modelo*. A continuación se describe cada una de estas tareas con mayor detalle.

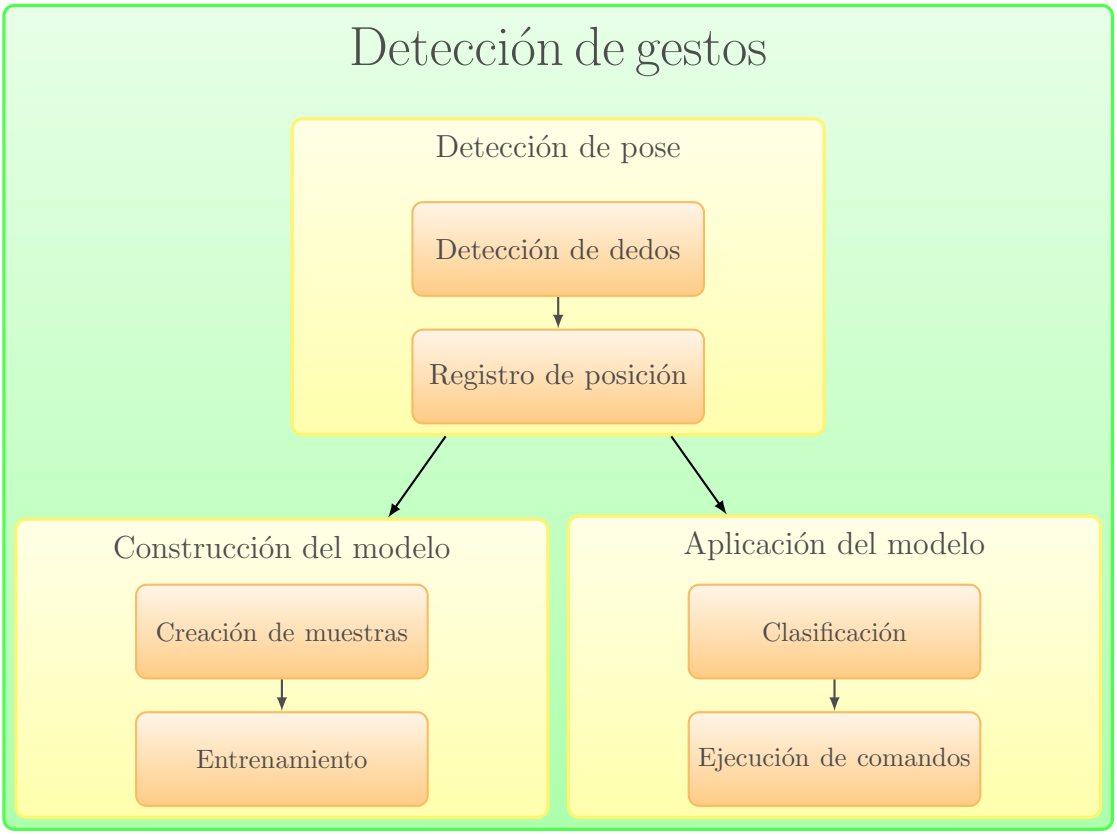


Figura 5.20: Detección de gestos

5.3.1 Detección de pose

Con la idea de facilitar el reconocimiento de gestos en NEKO, se decidió desarrollar una tarea para discernir los movimientos implícitos en un gesto de la interacción con objetos virtuales. Dicha tarea es la *detección de pose* la cual mediante el uso de posturas de mano permite identificar si los movimientos que el usuario realiza con sus manos forman parte de un gesto o son resultado de la manipulación de objetos virtuales.

Figura 5.21: Poses detectadas mediante un modelo de clasificación.

Se desarrollaron dos mecanismos para la detección de posturas de mano. En el primero

se aprovechó el potencial de la técnica SVM para construir un modelo de clasificación, capaz de identificar las posturas de mano mostradas en la figura 5.21 (ver Apéndice B:*detector de pose basado en SVM*). De hecho mediante este mecanismo se concluyó que una postura de mano debía cumplir con los siguientes requisitos:

- Fácil y rápida de realizar.
- No debe interferir con la manipulación de objetos virtuales.
- Sencilla de detectar para el sistema.

Analizando estos requisitos se dedujo que un puño es la postura ideal para el reconocimiento de gestos. Sin embargo también se determinó que el uso de un modelo de clasificación para identificar una postura de puño es "matar moscas con cañones". Por ello se optó en desarrollar y utilizar otro mecanismo para reconocer un puño mediante la cantidad de dedos extendidos de una mano, lo cual es más simple y a la par no involucrara demasiados recursos computacionales. La idea detrás de este mecanismo es que si el usuario retrae todos los dedos de una mano hacia la palma de la misma entonces se identifique una postura de puño.

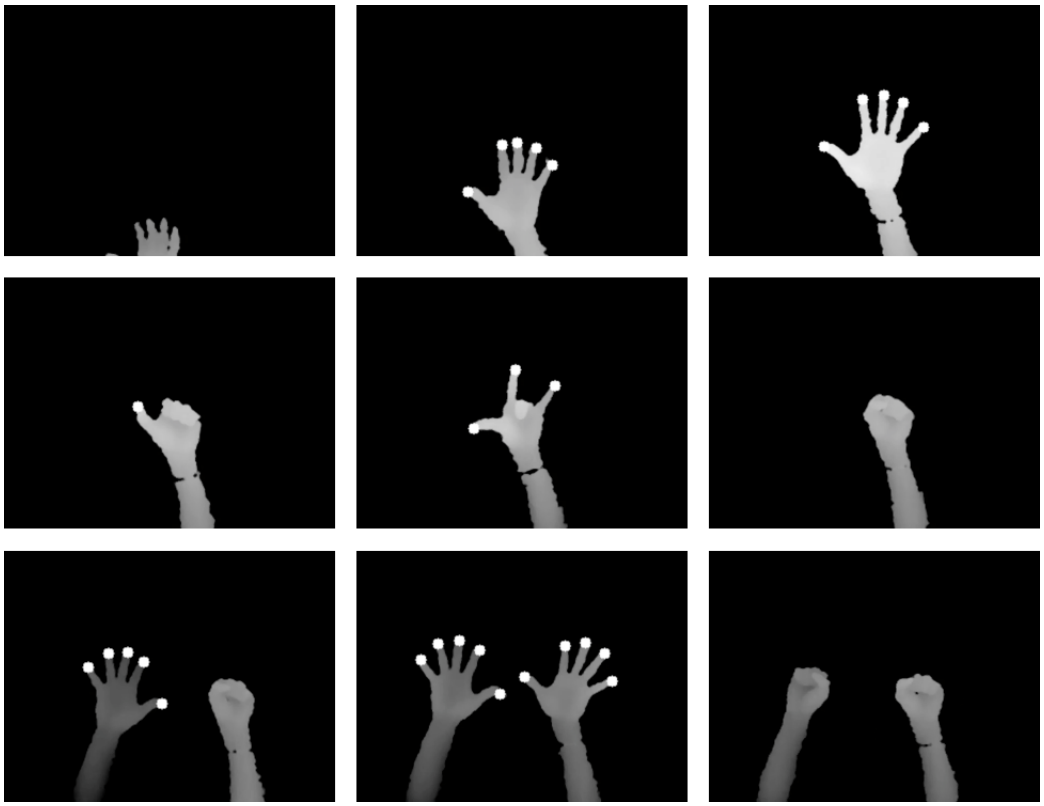


Figura 5.22: Detección de dedos.

El proceso para identificar los dedos de una mano se realiza en el algoritmo 4 descrito a continuación. Primeramente se extrae un recuadro de imagen del marco de video del sensor de profundidad con la posición de la mano (línea 1). Enseguida se obtiene el contorno de la mano con la función de OpenCV *findContours()* y se ajusta una elipse al contorno mediante la función *fitEllipse()* (líneas 2-3). Inmediatamente el contorno se aproxima a un polígono con la función *approxPolyDP()* y se evalúa si alguno de los puntos del polígono está afuera de la elipse (líneas 4-7). De ser este el caso entonces se considera el punto como un dedo (línea 8). Finalmente se compara el número de dedos extendidos en una mano y si no existe alguno entonces se reconoce una postura de puño (líneas 11-16).

Algorithm 4 Detección de dedos

Require: *marcoSensorProfundidad, posicionMano*

Ensure: Dedos

1: Extrae $recuadroMano$ de $marcoSensorProfundidad$ a partir de $posicionMano$
2: Extrae $contornoMano$ de $recuadroMano$
3: Ajusta $elipsePalma$ con $contornoMano$
4: Aproxima a un polígono $contornoMano$
5: $dedos \leftarrow$ contenedor vacío
6: **for all** $punto \in contornoMano$ **do**
7: **if** *punto* está afuera de $elipsePalma$ **then**
8: Agrega *punto* a *dedos*
9: **end if**
10: **end for**
11: **if** *dedos* diferente de vacío **then**
12: Agrega base de $elipsePalma$ a *dedos*
13: retorna *verdadero* para postura de puño
14: **else**
15: retorna *falso* para postura de puño
16: **end if**

La figura 5.22 muestra una secuencia de marcos donde se identifican los dedos de una mano mediante el algoritmo 4. Cuando el usuario retrae los dedos de una mano se identifican una postura de puño.

Cuando se identifica una postura puño sobre una mano del usuario, se registran todos los movimientos que ésta realiza mediante la tarea *registro de posición*; mientras el usuario man-

tenga la postura de puño se almacena cada posición de la mano. Con el afán de disminuir los recursos necesarios para almacenar los movimientos, se decidió guardar la posición de cada mano con una postura puño cada tres marcos de video transcurridos. Es decir, si el sistema se ejecuta a una velocidad de 30 marcos por segundo, únicamente se almacenarán 10 posiciones durante un segundo (en lugar 30 posiciones).

5.3.2 Construcción del modelo de clasificación

De forma semejante a la técnica de Viola-Jones, el proceso para construir un modelo de clasificación con la técnica SVM consta de una tarea para la *extracción de muestras* y otra tarea para el *aprendizaje del modelo*. A continuación se describe cada una de estas tareas.

Extracción de muestras

En NEKO se decidió representar a un gesto como un conjunto de trayectorias, es decir a través de las posiciones registradas del movimiento de la mano con la tarea *detección de pose* se determinó construir vectores que indiquen la trayectoria del movimiento. La figura 5.23 muestra en (a) las posiciones registradas en el movimiento de un gesto circular y en (b) los vectores que indican la trayectoria del movimiento. Como es de esperarse los vectores aportan gran detalle del movimiento de las manos.

Para construir el modelo de clasificación, se capturaron 16 muestras de trayectorias representativas a cada uno de los gestos mostrados en la figura 5.19. También se evaluó la posibilidad de identificar un gesto como un conjunto de puntos, semejantes a pequeñas "migajas" dejadas en la trayectoria de la mano. Sin embargo debido la ambigüedad que podría existir con movimientos similares pero con trayectorias diferentes (por ejemplo un gesto circular realizado de izquierda a derecha poseería los mismos puntos que un gesto circular realizado de derecha a izquierda) se descartó esta posibilidad.

Aprendizaje del modelo

El *aprendizaje del modelo* es la tarea en la cual se construye un modelo de clasificación para la detección de gestos mediante SVMs. La construcción del modelo mediante esta técnica consiste en encontrar un *hiperplano* (margen o línea de separación) óptimo para un conjunto de muestras dado, en nuestro caso para las trayectorias que representan a los gestos. La

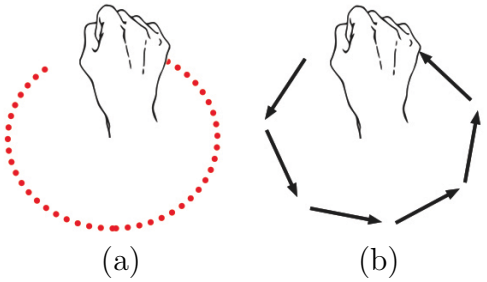


Figura 5.23: Puntos y vectores en la trayectoria de un gesto circular.

figura 5.24 muestra dos de los múltiples *hiperplanos* que pueden separar correctamente a un conjunto de muestras formado por cuadros y círculos. El *hiperplano* B_2 posee mayor margen de separación entre los dos tipos de muestras que el *hiperplano* B_1.

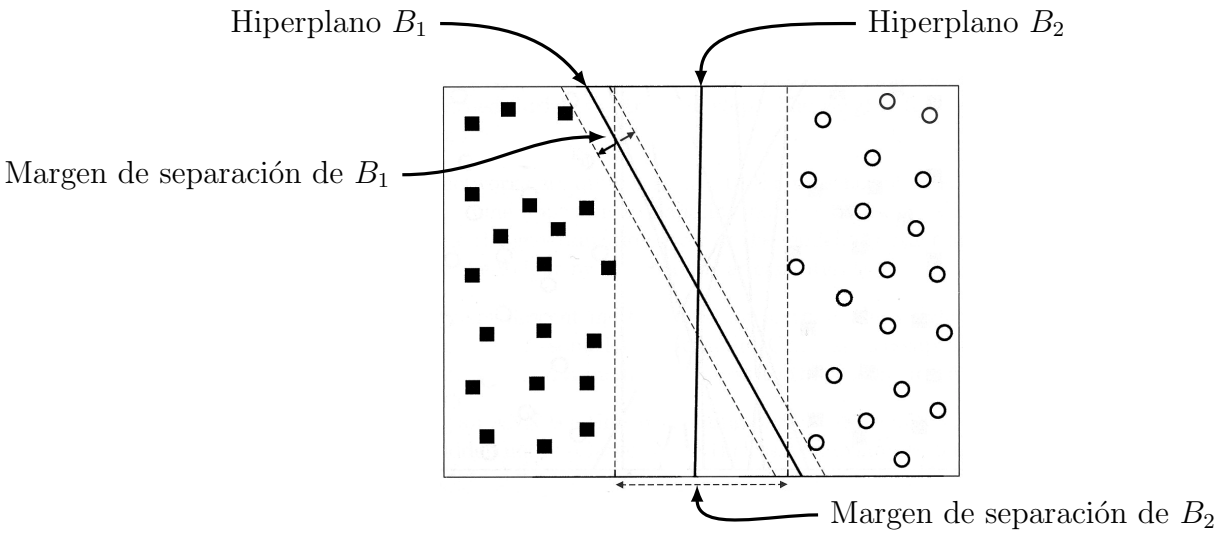


Figura 5.24: Separación de dos clases.

Los SVMs calculan el *hiperplano* con el mayor margen de separación a través del siguiente problema de optimización:

$$\min_{w} \frac{\| w \|^2}{2} \tag{5.4}$$

$$\text{sujeto a } y_i(w \cdot \Phi(x_i) + b) \geq 1, i = 1, 2, \cdots, N.$$

Donde w es un vector ortogonal al *hiperplano* que divide a un atributo x_i con etiqueta de clase y_i. La función Φ permite convertir el atributo x_i desde un espacio de coordenadas

a otro para una separación perceptible y b es un parámetro del modelo. Esta clase de problema se puede resolver a través de programas de cálculos numéricos como GNU Octave[4] o MATLAB[5]. Sin embargo en OpenCV gracias a la clase *SVM* también existen herramientas que permiten calcular el *hiperplano* con mayor margen de separación a partir de un conjunto de muestras.

Una restricción que impone la clase *SVM* de OpenCV es que las muestras deben estar representadas como vectores. Por ello antes de comenzar la construcción del modelo de clasificación es necesario transformar las muestras a vectores. Convenientemente en nuestro caso, las muestras están construidas a partir de vectores por lo cual no existe problema alguno. Pero, para evitar que los valores grandes de algún atributo dominen a los valores pequeños se decidió normalizar los vectores. Esto incluso evita el manejo de números muy grandes que ocasionen dificultades en los cálculos.

Habiendo preparado las muestras, solo resta elegir la función Φ para iniciar el cálculo del *hiperplano* óptimo. OpenCV ofrece tres funciones Φ para elegir:

- **Polinomial**. $K(x_i, x_j) = (\gamma x_i^T x_j + c)^e, \gamma > 0$
- **Sigmoid**. $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + c)$
- **Base Radial (RBF)**. $K(x_i, x_j) = e^{-\gamma ||x_i - x_j||^2}, \gamma > 0$

Considerando la recomendación en la documentación para la clase *SVM* de OpenCV[6] se optó por utilizar la función kernel *RBF*.

La función Φ RBF únicamente requiere un valor para γ. Sin embargo, dado un conjunto de muestras es complicado determinar cuál es el valor adecuado. Una estrategia para resolver este problema consiste en construir varios modelos (asignando un rango de valores a γ) y posteriormente elegir aquel con el *hiperplano* de mayor margen de separación. Esto se puede realizar a través de la *validación cruzada*. Esta técnica divide el total de muestras en pequeños subconjuntos de un mismo tamaño. Posteriormente, con un número de subcojuntos pequeño se construye un modelo inicial, el cual se evalúa con los subconjuntos restantes; sucesivamente se incrementa el número de subconjuntos para la construcción del modelo y a la par se reducen los subconjuntos de prueba.

[4]http://www.gnu.org/software/octave/ (accedida en julio de 2012)
[5]http://www.mathworks.com/products/matlab/ (accedida en julio de 2012)
[6]http://opencv.itseez.com/modules/ml/doc/support_vector_machines.html (accedida en mayo de 2012)

La *validación cruzada* en OpenCV se encuentra implementada en la función *train_auto()* de la clase *SVM*. Esta función construye indefinidamente distintos modelos de clasificación (constantemente evaluados) hasta que se alcance un valor máximo de iteraciones; se regresa el modelo con el mayor margen de separación. Como es de esperarse, obtener un modelo de clasificación mediante la validación cruzada no es precisamente un procedimiento rápido. Con tan solo tres gestos, con 16 muestras cada uno, se requiere al rededor de 20 minutos para construir el modelo de clasificación.

5.3.3 Clasificación de movimientos

La tarea *clasificación de movimientos* se realiza posterior a la construcción del modelo. Una vez que se ha identificado una postura de puño y registrado sus movimientos, se comparan las direcciones de la mano contra el modelo de clasificación a través de la función *predict()* de la clase *SVM*. Si los movimientos elaborados coinciden en un 80 % con algún patrón del modelo, entonces se reconoce como un gesto y se ejecuta una actividad asociada al mismo, en caso contrario se descartan los movimientos.

La figura 5.25 muestra el reconocimiento del gesto para la calibración de la superficie táctil. Cuando el usuario realiza una postura de puño junto con un movimiento circular el sistema reconoce que el usuario desea iniciar el proceso para calibrar la superficie táctil.

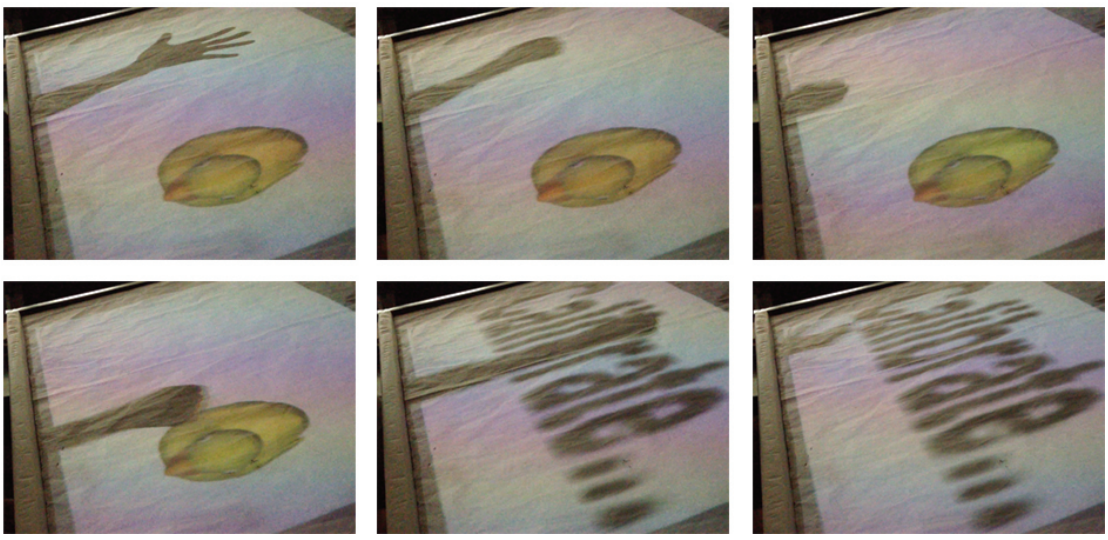


Figura 5.25: Gesto para la calibración de la superficie táctil.

Resumen

Este capítulo presentó los componentes de software relacionados con la construcción y uso de modelos de clasificación para la detección de rostros, manos y gestos. NEKO emplea estos modelos de clasificación para analizar la interacción del usuario sobre la *tabletop*, permitiendo fundamentalmente, construir mecanismos para la manipulación de objetos virtuales 2D y 3D.

El modelo de clasificación para la detección de rostros es la base para acoplar el entorno virtual a la perspectiva del usuario. En uso, el modelo se compara con cada marco de video capturado por la cámara web para identificar la posición del rostro del usuario y así calcular una perspectiva concorde al usuario. En nuestro caso se usó la técnica *cascada de clasificadores con características Haar* propuesta por Paul Viola y Michael Jones para construir el modelo de clasificación.

La detección de manos es una tarea imprescindible en NEKO, en especial porque las manos son el principal método de interacción sobre el prototipo de *tabletop*. Se desarrollaron dos mecanismos para realizar el reconocimiento de manos. El primero se realizó mediante la biblioteca OpenNI y NITE. Sin embargo debido a que NITE no está diseñado para funcionar con marcos de video que contengan únicamente manos, se construyó otro modelo de clasificación para identificar las manos del usuario sobre los tableros del prototipo de *tabletop* de NEKO. Al igual que en la detección de rostros, en la detección de manos se usó la técnica propuesta por Paul Viola y Michael Jones para construir el modelo.

Finalmente la detección de gestos evalúa si existe algún pauta, es decir una silueta o pose, en las manos del usuario. De ser este el caso, se registran los movimientos de la mano y posteriormente se verifica con los movimientos implícitos en algún gesto. Para nuestro propósito se utilizó las máquinas de vectores de apoyo para construir un modelo de clasificación capaz de discernir los movimientos de las manos del usuario.

Capítulo 6

Interacción con objetos virtuales

"Si algo va a ser mejor, es nuevo y si es nuevo estarás enfrentando problemas y desafíos que no tienen referencias."

Jonathan Ive

En tareas para la manipulación de objetos virtuales, una interfaz que utilice como método de entrada dispositivos como el teclado y el ratón no es ni intuitiva ni fácil de operar. Para este objetivo, la interacción con las manos es un método ideal. Debido a que la manipulación directa con las manos es una de las modalidades de interacción de mayor impacto en el mundo real, es lógico pensar que interfaces que usen esta modalidad faciliten la manipulación de objetos virtuales.

La interacción con objetos virtuales es el componente de software de NEKO que integra la interacción que realiza un usuario, entre los tableros del prototipo de *tabletop*, sobre un entorno virtual. Gracias a los avances en algoritmos de visión por computadora y de los controles de videojuegos como Kinect, es posible construir técnicas para interactuar en un entorno virtual sin la necesidad de dispositivos de control artificial que actúen como interpretes.

Las interfaces de usuario diseñadas para la manipulación de objetos virtuales deben proporcionar medios para lograr al menos una de tres tareas básicas: selección, traslación y rotación [Bowman et al., 2004]. En este capítulo se describen dos mecanismos para la manipulación de objetos virtuales que emplean estas tareas básicas. Sin mayor preámbulo este capítulo comienza describiendo las tareas que componen a la interacción 2D. Seguidamente se describen las tareas que integran la interacción 3D.

6.1 Interacción 2D

La interacción 2D se enfoca en la manipulación de objetos virtuales en dos dimensiones a través de las tareas de traslación, rotación y escalado. Como se mencionó en el capítulo 4, el prototipo de *tabletop* de NEKO está compuesto de dos tableros, cada uno con un cristal de 90 x 60 centímetros. Semejante a un dispositivo multitáctil la interacción 2D se realiza sobre el cristal claro del tablero inferior, de forma similar al trabajo descrito en [Wilson, 2010]. En este trabajo, se emplea el sensor de profundidad del Kinect para simular una superficie táctil. Dado que los marcos que captura el sensor de profundidad reportan la distancia de los elementos de una escena, el autor creó un modelo inicial de una superficie con el cual compara el contacto por parte del usuario.

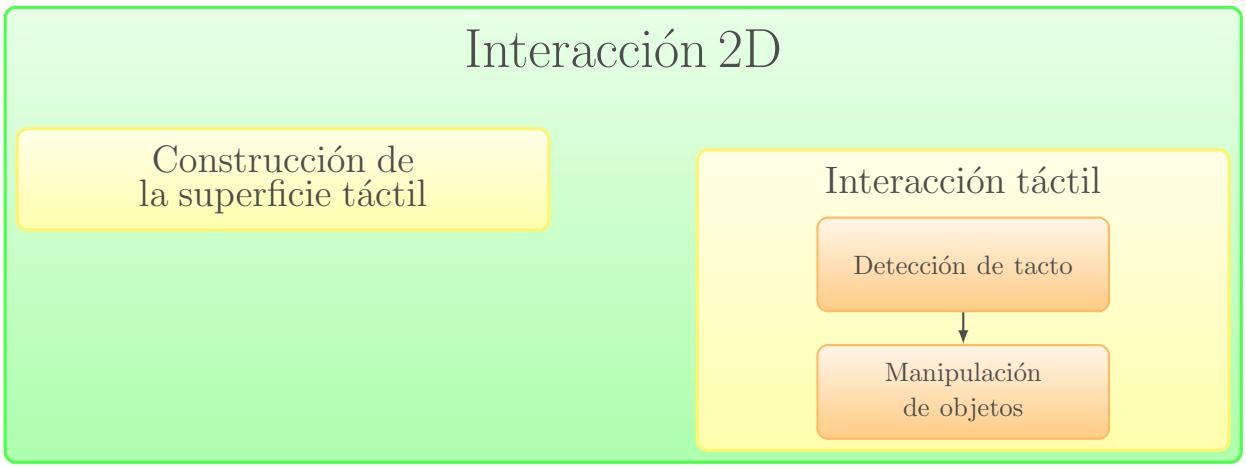


Figura 6.1: Tareas involucradas en la interacción 2D.

La interacción 2D en NEKO se compone de las tareas *construcción de la superficie táctil* e *interacción táctil* (ver figura 6.1). A continuación se describe en detalle cada una de ellas.

6.1.1 Construcción de la superficie táctil

La *construcción de la superficie táctil* es el proceso donde, a través de los pixeles de los marcos de profundidad, se crea un modelo de una superficie interactiva, es decir del cristal claro del tablero inferior del prototipo de *tabletop* de NEKO. Este proceso se inicia cada vez que el usuario realiza el gesto de calibración de la superficie táctil (descrito la sección 5.3) y consta de tres pasos:

1. Modelado de la superficie táctil. Normalmente los rayos infrarrojos emitidos por el

proyector infrarrojo del Kinect a traviesan cualquier superficie transparente, por ello una vez identificado el gesto de calibración se coloca un objeto opaco sobre el cristal claro en aras de construir un modelo acorde a su superficie. En nuestro caso se decidió emplear una hoja de papel del tamaño del cristal claro para construir el modelo.

2. Reducción de ruido. Como se mencionó en la sección 5.2.4, los marcos de profundidad padecen de ruido, lo que ocasiona variabilidad en la información aun en marcos de video consecutivos. Por consiguiente en el afán de mitigar el ruido en el modelo de la superficie táctil, se decidió aplicar un *filtro mediana* a los marcos de profundidad. La aplicación de filtros es una tarea común en el procesamiento de imágenes que permite amplificar o reducir determinadas características en una imagen. Para nuestro caso en particular el filtro mediana elimina valores *outliers*[1] de los marcos de profundidad. Así este filtro "suaviza" los marcos de profundidad mediante la sustitución de sus pixeles por el valor de la mediana respecto a los vecinos de cada pixel. Por lo tanto la variabilidad que existe en los marcos de profundidad se reduce en gran medida.

3. Captura del modelo de la superficie táctil. Finalmente solo resta almacenar el modelo del cristal claro del tablero inferior. Debido a que los rayos del proyector infrarrojo del Kinect no traspasan la hoja de papel es imposible detectar gestos para indicarle al sistema que almacene el modelo de la superficie. Por consiguiente, aprovechando la sombra que producen las manos sobre el papel, se empleó la detección de manos para solucionar esta problemática. Así cuando el usuario sobrepone la palma de sus dos manos sobre la hoja de papel se almacena un marco de profundidad como el modelo del cristal claro.

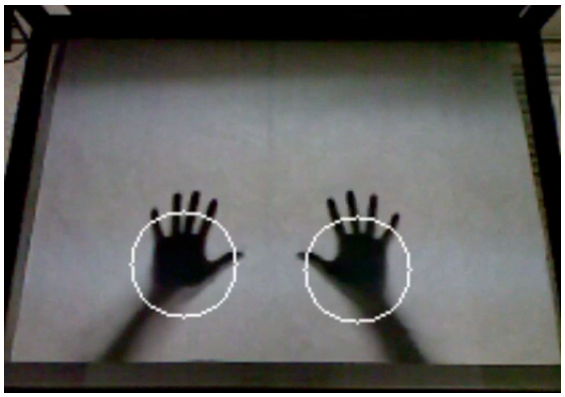


Figura 6.2: Construcción de la superficie táctil.

[1]Datos atípicos que tienen un valor inusual respecto a sus semejantes

La figura 6.2 muestra la sombra de dos manos sobre la hoja de papel ubicada sobre el cristal claro. Cuando el usuario posiciona sus manos sobre la hoja de papel, se guarda un marco de profundidad como el modelo de la superficie interactiva.

6.1.2 Interacción táctil

Para identificar el contacto de las manos del usuario sobre una superficie interactiva en [Wilson, 2010] comparan cada marco de video capturado por el sensor de profundidad contra un modelo de la superficie. En NEKO se utiliza esta misma idea para sustentar la interacción 2D, es decir mediante el modelo almacenado en la tarea anterior (*construcción de la superficie táctil*) se compara con la distancia de los pixeles correspondientes a cualquier cuerpo opaco que haga contacto con el cristal del tablero inferior.

La figura 6.3 muestra el mecanismo empleado para identificar dicho contacto. Una vez que se han detectado los dedos de la mano del usuario (ver sección 5.3.1) se evalúa si algún dedo se encuentran dentro de un umbral de distancia respecto al modelo de la superficie. De ser este el caso, se reconoce que el dedo hace contacto con el cristal.

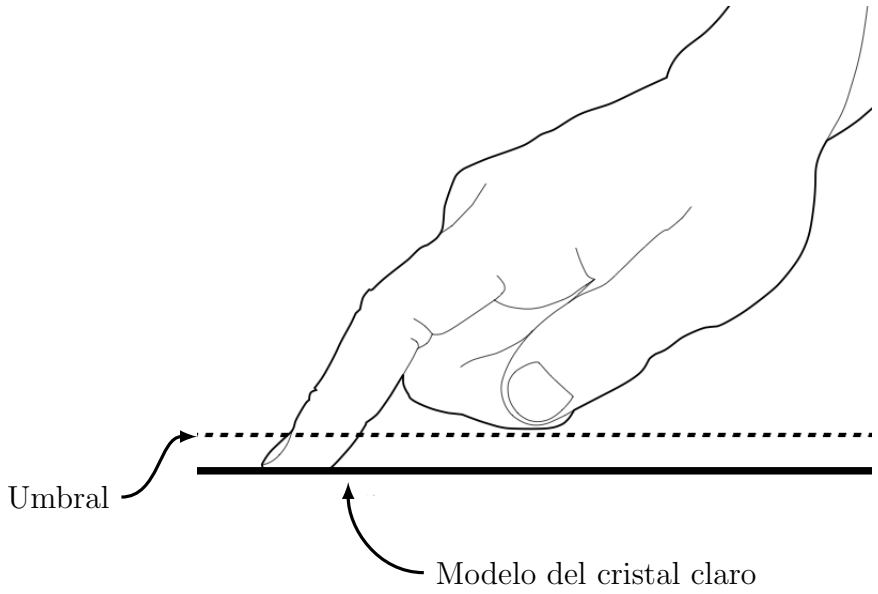


Figura 6.3: Detección de tacto sobre el cristal claro.

Posterior a la detección de un contacto con el cristal claro, se identifica si algún dedo hace "contacto" con un objeto virtual, en cuyo caso, si el dedo se mueve, se mueve el objeto virtual acordemente. Para identificar el contacto entre objetos virtuales se utiliza una técnica

llamada coordenadas baricéntricas. Las coordenadas baricéntricas permiten parametrizar (es decir representar mediante valores llamados parámetros) un plano a partir de varios puntos de referencia. Una aplicación típica de las coordenadas baricéntricas es la parametrización de triángulos. Puesto que los objetos virtuales 2D en NEKO están compuestos de triángulos[2], esta técnica es muy útil para identificar si un dedo hace contacto con algún objeto virtual.

Considerando cada triángulo de un objeto virtual como tres puntos no colineales A, B y C, cualquier punto P en el plano de los puntos se puede expresar como:

$$P = uA + vB + wC \tag{6.1}$$

donde u, v, w son las coordenadas baricéntricas de P con respecto de A, B y C. Si $u, v, w \in [0, 1]$ y $u + v + w = 1$ entonces el punto P se encuentra dentro del triángulo. En caso contrario el punto se encuentra fuera del triángulo.

Sin embargo, como la posición de los dedos está en relación a las coordenadas de los marcos de video del dispositivo Kinect y la posición de los objetos virtuales está en relación a las coordenadas de OpenGL (la biblioteca empleada en NEKO para construir el entorno virtual) es necesario homogeneizar tales coordenadas. La figura 6.4 muestra el proceso para convertir las coordenadas del dispositivo Kinect a coordenadas OpenGL. Desde la posición de la cámara de OpenGL que captura el entorno virtual, se calcula con el ángulo de visión de dicha cámara las dimensiones de un plano (ubicado en la base del entorno virtual) que representa el área de un marco de profundidad del dispositivo Kinect. Posteriormente con las dimensiones obtenidas se calculan las coordenadas de los dedos a sus respectivas coordenadas OpenGL mediante una regla de tres.

En el algoritmo 5 se describe el procedimiento desarrollado para interactuar de forma táctil en NEKO. Primeramente se evalúa la distancia que existe del modelo del cristal claro a cada dedo del usuario entre los tableros del prototipo de *tabletop* de NEKO. Si la distancia de un dedo es menor que un umbral entonces se reconoce que el dedo hace contacto sobre la superficie del cristal (líneas 2-6). Enseguida con las dimensiones de los marcos de video y las dimensiones del plano base del entorno virtual se calculan las coordenadas OpenGL del dedo (líneas 7-14). Posteriormente con estas coordenadas OpenGL se identifican los dedos

[2]En general, un objeto virtual se construye a partir de primitivas geométricas simples tales como puntos, lineas y/o triángulos.

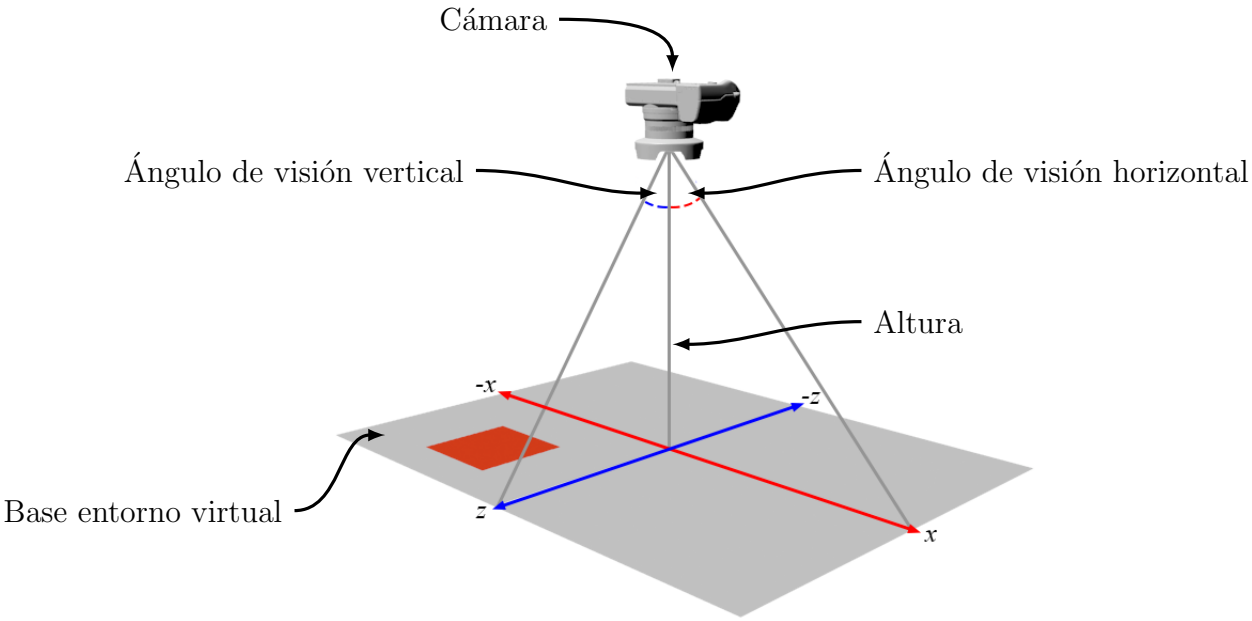


Figura 6.4: Coordenadas Kinect a coordenadas OpenGL.

que hacen contacto con los objetos virtuales. Para cada triángulo de cada objeto virtual se evalúa, mediante las coordenadas baricéntricas, si las coordenadas OpenGL de un dedo están dentro algún triángulo (líneas 15-36). De ser este el caso, el objeto implicado se manipula acorde al número de dedos que lo tocan. Cuando solo un dedo hace contacto, el objeto se traslada a las coordenadas del dedo involucrado (líneas 38-41). En cambio cuando más de un dedo hace contacto, el objeto se rota y/o escala (líneas 42-46); se emplea la ecuación de la pendiente (6.2) y la distancia que existe entre los dedos para calcular los valores de rotación y escalado.

$$m = atan(\frac{y_2 - y_1}{x_2 - x_1}) \tag{6.2}$$

Donde (x_1, y_1) y (x_2, y_2) son las coordenadas de los puntos.

6.2 Interacción 3D

La interacción 3D se enfoca en la manipulación de objetos virtuales en tres dimensiones. A diferencia de la interacción 2D que se realiza sobre el cristal claro, la interacción 3D se realiza sobre el espacio que existe entre los tableros del prototipo de *tabletop* de NEKO. Por ello para mantener la correspondencia entrada/salida (mencionada en la sección 4.1) los

Algorithm 5 Manipulación de objetos virtuales 2D

Require: *dedos, objetosVirtuales, superficieTactil, umbralTacto, dimensionesMarcosProfundidad, dimensionesEntornoVirtual*

Ensure: Manipulación de objetos virtuales 2D

1: *contactos* ← vector vacío
2: **for all** *dedo* ∈ *dedos* **do**
3: *coordenadasDedoKinect* ← coordenadas de *dedo*
4: $profundidadDedo$ ← profundidad de *dedo*
5: $profundidadSuperficie$ ← profundidad $superficieTactil$ con $coordenadasDedoKinect$
6: $distanciaDedoSuperficie \leftarrow profundidadSuperficie - profundidadDedo$
7: **if** $distanciaDedoSuperficie$ es menor que $umbralTacto$ **then**
8: $xMed \leftarrow dimensionesMarcosProfundidad.x/2$
9: $yMed \leftarrow dimensionesMarcosProfundidad.y/2$
10: $x \leftarrow ((coordenadasDedoKinect.x - (xMed)) * dimensionesEntornoVirtual.x)/xMed$
11: $z \leftarrow ((coordenadasDedoKinect.z - (yMed)) * dimensionesEntornoVirtual.z)/yMed$
12: Agrega x, z en *contactos*
13: **end if**
14: **end for**
15: **for all** $objeto \in objetosVirtuales$ **do**
16: **for all** $triangulo \in$ malla de $objetoVirtual$ **do**
17: $A \leftarrow$ vértice 0 de $triangulo$
18: $B \leftarrow$ vértice 1 de $triangulo$
19: $C \leftarrow$ vértice 2 de $triangulo$
20: *dedos* ← vector vacío
21: **for all** *dedo* ∈ *contactos* **do**
22: $v0 \leftarrow C - A$
23: $v1 \leftarrow B - A$
24: $v2 \leftarrow dedo - A$
25: $pp00 \leftarrow v0$ producto punto $v0$
26: $pp01 \leftarrow v0$ producto punto $v1$
27: $pp02 \leftarrow v0$ producto punto $v2$
28: $pp11 \leftarrow v1$ producto punto $v1$

29: $pp12 \leftarrow v1$ producto punto $v2$
30: $aux \leftarrow 1/(pp00 * pp11 - pp01 * pp01)$
31: $u \leftarrow (pp11 * pp02 - pp01 * pp12) * aux$
32: $v \leftarrow (pp00 * pp12 - pp01 * pp02) * aux$
33: **if** u, v son menores que cero y $u + v$ mayor que uno **then**
34: Agrega *dedo* en *dedos*
35: **end if**
36: **end for**
37: **if** *dedos* diferente de vacío **then**
38: *pseudoPulgar* ← extrae primer elemento de *dedos*
39: **if** *dedos* contiene un elemento **then**
40: Traslada *objeto* a la posición de *pseudoPulgar*
41: **else**
42: *pseudoIndice* ← promedio de elementos de *dedos*
43: *pendiente* ← calcula pendiente con *pseudoPulgar* y *pseudoIndice*
44: Rota *objeto* con *pendiente*
45: *distanciaDedos* ← calcula distancia entre *pseudoPulgar* y *pseudoIndice*
46: Escala *objeto* con *distanciaDedos*
47: **end if**
48: **end if**
49: **end for**
50: **end for**

objetos virtuales se proyectan sobre el cristal obscuro del tablero superior, entre el trayecto que existe de las manos y el rostro del usuario.

En general, la interacción 3D en NEKO emplea los marcos de video que captura el sensor de profundidad del dispositivo Kinect para evaluar la "distancia" que existe entre los cuerpos físicos (generalmente las manos del usuario) y los objetos virtuales. Cuando un cuerpo físico está lo suficientemente cerca, el objeto virtual es manipulado acordemente a la interacción inducida por el movimiento del cuerpo físico.

La figura 6.5 muestra las tareas que componen la interacción 3D. Como se puede observar, se desarrollaron dos técnicas, y las tareas correspondientes, para la interacción con los objetos virtuales. La primera técnica, *manipulación de objetos virtuales con los dedos*, fue diseñada con la idea de aminorar en lo posible los recursos involucrados en la interacción con objetos virtuales. En cambio la *manipulación de objetos virtuales a través de partículas* se construyó como una manipulación alternativa que involucra mayor cantidad de recursos pero que a su vez ofrece mejores resultados en términos de fidelidad de interacción a la vida real. Además, se desarrolló una tarea para simular *fuerzas físicas* básicas entre los mismos objetos virtuales y su entorno. A continuación se describe cada una de estas tareas.

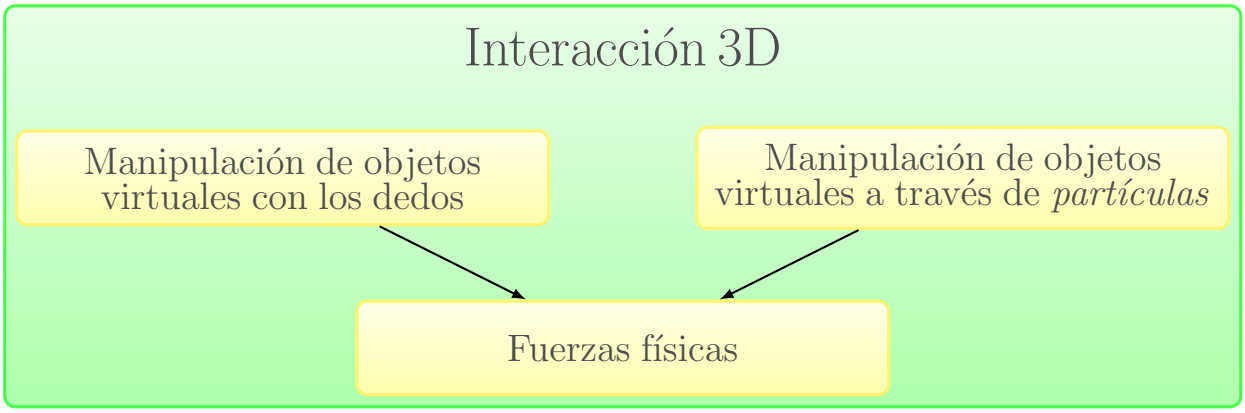


Figura 6.5: Tareas involucradas en la interacción 3D.

6.2.1 Manipulación de objetos virtuales con los dedos

Los dedos son un mecanismo sencillo y rápido para manipular objetos virtuales 3D. Los seres humanos manipulamos objetos a nuestro alrededor principalmente con los dedos. Esta cualidad es una ventaja evolutiva muy notable que facilita la interacción con nuestro entorno. Bajo esta misma ideología se decidió construir en NEKO una técnica similar que permitiera

al usuario interactuar con objetos virtuales mediante los dedos.

Si se analiza con detalle como un ser humano agarra un objeto de proporciones pequeñas con sus manos, se pude observar que emplea el pulgar en conjunto de algún otro dedo, generalmente el índice y/o medio. Por ello para manipular los objetos virtuales con los dedos de forma semejante a la vida real primeramente identificamos los pulgares de las manos del usuario.

Examinando la estructura de una mano extendida se puede apreciar que el falange distal (es decir la punta del dedo) del dedo pulgar, en comparación con los demás dedos, se encuentra a menor distancia de la muñeca. Incluso, cuando los dedos están estirados, el pulgar no rebasa la tercera parte de la distancia que existe entre la muñeca y el dedo medio. Por lo tanto se determinó reconocer un dedo como pulgar cuando cumple con estas características.

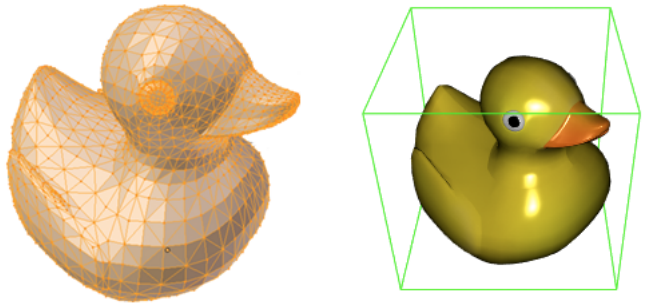


Figura 6.6: Cubo contenedor de la malla de un objeto.

Por otro lado para manipular los objetos virtuales mediante los dedos es imperativo identificar cuando el usuario hace contacto con el objeto. Usualmente la malla de un objeto virtual está compuesto por un gran número de triángulos, sin embargo con la idea de usar una representación más simple se decidió calcular un *cubo contenedor* acorde a las dimensiones de la malla (ver figura 6.6). En consecuencia no se requieren demasiados recursos para representar y manipular a un objeto virtual. Simplemente basta con que un dedo toque el cubo contenedor para simular el contacto con la malla del objeto.

El algoritmo 6 describe el procedimiento construido para la manipulación de objetos virtuales mediante los dedos. Primeramente, de los dedos obtenidos mediante el algoritmo 4 (sección 5.3.1) se identifica un pulgar y un dedo índice de acuerdo a la distancia que existe de la muñeca a cada dedo (líneas 4-14). Posteriormente para cada objeto residente en el entorno virtual se calcula un cubo contenedor con la malla del objeto a través de la función

calcBoundingBox() de la biblioteca Cinder (líneas 15-16). Enseguida, se obtienen las coordenadas del cubo contenedor (que representan el "centro" del objeto) y se calculan dos vectores: uno hacia la posición del dedo pulgar y otro hacia la posición del dedo índice (líneas 17-19). Si los componentes de ambos vectores son menores a las dimensiones del cubo contenedor entonces se identifica que el objeto está siendo "sujetado" por los dedos del usuario (línea 20). Por lo tanto, se calcula la nueva posición del objeto con la posición promedio del dedo pulgar y el dedo índice (línea 21). Asimismo se obtiene la pendiente entre la posición de los dedos con la ecuación (6.2) para los ejes X e Y (líneas 22-23). Finalmente se rota el objeto y se traslada a la nueva posición (líneas 24-25).

Algorithm 6 Interacción 3D mediante dedos

Require: *objetosVirtuales, dedos*

Ensure: Manipulación de objetos virtuales 3D con los dedos

basePalma ← último elemento de dedos
pulgar ← vector vacío
indice ← vector vacío
numeroDedos ← 0
for all *dedo* ∈ *dedos* **do**
  distanciaDedo ← calcula la distancia de *dedo* a la base de *elipsePalma*
  if *distanciaDedo* es menor que un rango **then**
    pulgar ← *dedo*
  else
    indice ← *indice* + *dedo*
    numeroDedos ← *numeroDedos* + 1
  end if
end for
indice ← *indice*/*numeroDedos*
for all *objetoVirtual* ∈ *objetosVirtuales* **do**
  cuboContenedor ← calcula cubo contenedor de la malla de *objetoVirtual*
  centroObjeto ← posición del *cuboContenedor*
  vectorPulgar ← *centroObjeto* − *pulgar*
  vectorIndice ← *centroObjeto* − *indice*
  if componentes de *vectorPulgar* y *vectorIndice* son menores a las dimensiones de *cuboContenedor* **then**
    nuevaPosicionObjeto ← (*pulgar* + *indice*) ∗ 0.5
    ejeX calcula pendiente para eje X con (6.2)
    ejeY calcula pendiente para eje Y con (6.2)
    Traslada *objetoVirtual* a *nuevaPosicionObjeto*
    Rota *objetoVirtual* con *ejeX* y *ejeY*
  end if
end for

6.2.2 Manipulación de objetos virtuales a través de *partículas*

La *manipulación de objetos virtuales con los dedos* tiene la ventaja de que no requiere demasiados recursos computacionales para efectuar la interacción con el entorno virtual. No obstante, es necesario resaltar que mediante este método la interacción está restringida exclu-

sivamente con los dedos. ¿Por qué no construir un mecanismo alterno sin una manipulación de objetos tan restrictiva que a la par no merme demasiado la velocidad del sistema? La *manipulación de objetos virtuales a través de partículas* es el mecanismo alterno que provee dicha manipulación. A diferencia de la manipulación mediante los dedos, en este mecanismo se buscó permitir una interacción con objetos virtuales mediante cualquier cuerpo físico del mundo real, no solo con los dedos o manos del usuario.

La construcción de un mecanismo de interacción entre cualesquiera objetos físicos indiscutiblemente conlleva construir una representación o modelo de dichos objetos sobre el entorno virtual. Esto puede ser complicado y costoso computacionalmente hablando. Con la idea de simplificar la representación de los objetos físicos se decidió emplear algo muy común en la naturaleza: cúmulos de "partículas". Usualmente, cuando un rayo de luz atraviesa una ventana se alcanzan a distinguir pequeñas partículas de polvo dentro del rayo (ver figura 6.7[3] (accedida en julio 2012)). De forma análoga, los objetos físicos capturados en los marcos de profundidad se pueden representar como cúmulos de pequeñas esferas suspendidas en el aire.

Figura 6.7: Partículas de polvo sobre el aire.

Ciertamente estos cúmulos por sí solos no ofrecen mayor ventaja para manipular los objetos virtuales. Sin embargo, las cosas se tornan interesantes cuando a cada partícula se le agrega una fuerza de repulsión o atracción. En general, para manipular un objeto físico en la vida real básicamente aplicamos una fuerza. Utilizando este mismo principio para cada partícula que toque la malla de un objeto virtual se puede construir una técnica para empujar o sujetar un objeto virtual.

La figura 6.8 muestra la idea esencial detrás de la manipulación de objetos virtuales a

[3]Imagen extraída de http://robotcosmonaut.tumblr.com

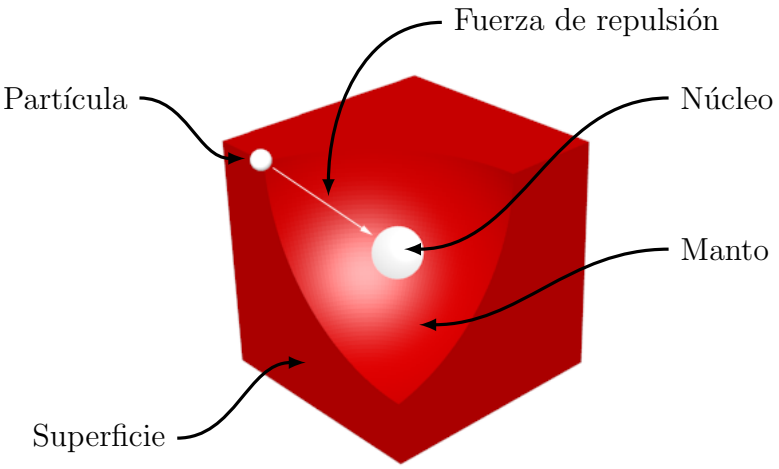


Figura 6.8: Interacción objeto virtual-partícula.

través de partículas. Basta con que una partícula toque la superficie del objeto virtual para calcular una fuerza de repulsión entre la partícula y el núcleo del objeto. Esto permite empujar al objeto acorde a la dirección de la fuerza de repulsión obtenida. En cambio, para sujetar el objeto se requiere que al menos dos partículas toquen en diferentes puntos la superficie del objeto y que además el ángulo entre las normales de la superficie donde tocan los puntos sea mayor a 120°. Por ejemplo, si un par de partículas tocaran dos caras opuestas entre sí del cubo mostrado en la figura 6.8, se formaría un ángulo entre sus normales de 180°. Cabe mencionar, que para mantener la impenetrabilidad[4] del objeto virtual, se calcula una fuerza de repulsión a toda partícula que se encuentra sobre el manto del objeto.

Asimismo en la manipulación a través de partículas los objetos virtuales se pueden rotar sobre los ejes X, Y y Z. Anteriormente en la manipulación mediante los dedos se calculaba la pendiente de la línea que existía entre el dedo pulgar y el dedo índice. Lamentablemente es difícil calcular una pendiente para un cúmulo de partículas. Para solucionar este problema, se decidió empotrar virtualmente una línea al cúmulo de partículas y así calcular la pendiente. El ajuste de una línea a un conjunto de puntos es un problema clásico en matemáticas en el cual se minimiza la suma de las distancias para cada punto, o en nuestro caso para cada partícula; una técnica muy utilizada para el ajuste de líneas a un conjunto de puntos es *M-estimador* [Meer et al., 1991].

Al igual que en la *manipulación de objetos virtuales con los dedos*, en la *manipulación de objetos virtuales a través de partículas* se requiere identificar el contacto con la superficie

[4]Resistencia que opone un cuerpo a que otro ocupe, simultáneamente, su lugar. En pocas palabras, ningún cuerpo puede ocupar al mismo tiempo el lugar de otro

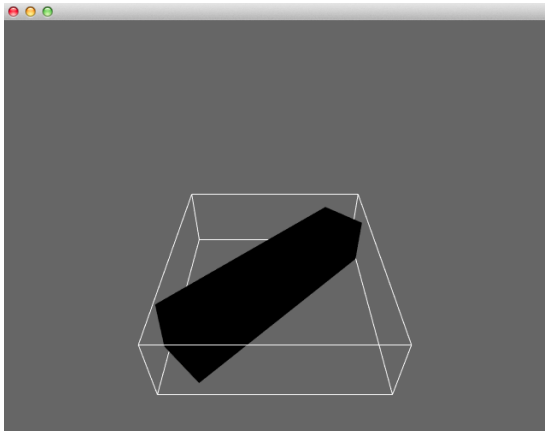


Figura 6.9: Cubo contenedor para objetos con dimensiones desiguales.

de los objetos virtuales. Después de analizar la interacción con los objetos virtuales mediante los dedos se determinó que el cubo contenedor de la malla es muy útil con objetos que contienen proporciones similares en sus dimensiones. Sin embargo, con objetos irregulares el cubo contenedor queda bastante "holgado" a la malla. Sobre todo cuando el usuario gira el objeto sobre alguno de los ejes (ver figura 6.9). Debido a semejante discordancia entre la malla y el cubo contenedor, los dedos manipulaban el objeto aun cuando realmente no tocaban su malla. A fin de solucionar la problemática que ocasiona un cubo contenedor con objetos virtuales irregulares en la *manipulación de objetos virtuales a través de partículas* se decidió construir un procedimiento capaz de evaluar (de forma rápida y que no involucre muchos recursos) si realmente existe contacto con la malla del objeto virtual.

Una técnica muy conocida para identificar la intersección con elementos de un entorno virtual es la *emisión de rayo* (ray casting) [Appel, 1968]. En general, esta técnica consiste en lanzar un rayo, con cierta dirección y desde un origen particular, sobre una escena para identificar si en la trayectoria de dicho rayo se intersecta algún elemento de interés. De hecho, es mediante la técnica emisión de rayos que también se construyen las partículas de los objetos físicos. La ecuación del rayo en función de la distancia t está definida en (6.4), donde o es el punto origen del rayo y d es la dirección.

$$R(t) = o + td,\ t > 0 \tag{6.3}$$

Asumiendo que la base del entorno virtual es el tablero inferior del prototipo de *tabletop* de NEKO, se transforman los pixeles de un marco de profundidad a partículas a través de la

emisión de rayos como se muestra en la figura 6.10. Cada marco de profundidad se sobrepone a la cámara de OpenGL que captura el entorno virtual y, desde la posición de la cámara, se lanza un rayo a cada uno de los pixeles del marco de video hasta intersectar con el plano base del entorno virtual. De esta forma, los pixeles se "proyectan" en el plano base y se obtienen las partículas de los objetos físicos sobre el entorno virtual.

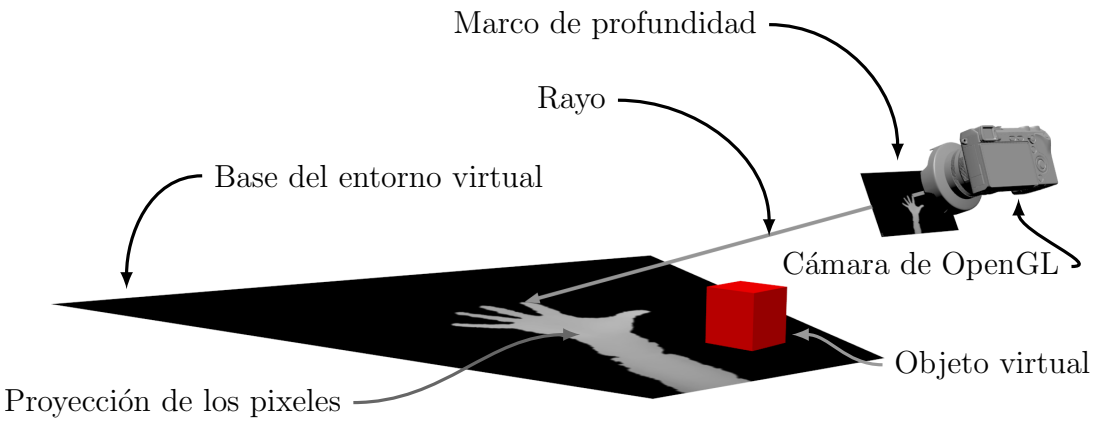


Figura 6.10: Construcción de partículas.

Posteriormente con la técnica emisión de rayos se identifica si una partícula hace contacto con la superficie de algún objeto virtual. Para ello, se lanza un rayo desde cada partícula hacia el núcleo de cada objeto objeto virtual. Si la distancia de intersección del rayo con la malla de algún objeto es menor que cero entonces se considera que la partícula toca la superficie del objeto. Como se mencionó con anterioridad, es necesario señalar que la malla de un objeto virtual puede estar formada por cientos o incluso miles de triángulos. Por ello, en afán de disminuir cálculos innecesarios, se optó por construir una esfera contenedora, a partir de la malla del objeto virtual, con la cual se evalúe inicialmente el contacto partícula-objeto virtual. Si la partícula hace contacto con la esfera contenedora entonces se evalúa el contacto de la partícula con la malla del objeto virtual.

La interacción con objetos virtuales a través de partículas se realiza mediante el algoritmo 7. Para la malla de cada objeto virtual desplegado sobre el prototipo de *tabletop* de NEKO, se calcula una esfera contenedora a través de la función de cinder *calculateBoundingSphere()* (líneas 3-9). Posteriormente, se identifica si el rayo de algún pixel del marco de video del sensor de profundidad (del cual previamente se eliminaron los pixeles de los elementos no relevantes) intersecta la esfera contenedora (líneas 10-14). De ser este el caso, se crea una partícula y se construye un rayo desde la posición de ésta con dirección hacía el centro de la esfera contenedora (líneas 15-19). Enseguida, se evalúa la distancia de intersección del

rayo con los triángulos de la malla; si es menor o igual a cero se calcula la fuerza que ejerce la partícula sobre el objeto (líneas 20-30). Luego se calcula la normal del triángulo donde intersectó el rayo y se compara con las normales de otros triángulos donde otras partículas han hecho contacto para identificar si el usuario está sujetando el objeto (líneas 31-38). Si se determinó que el usuario está sujetando el objeto entonces se atrae su malla hacia el cúmulo de partículas, en caso contrarío la malla se empuja (líneas 46-51). Finalmente se ajusta una línea al cúmulo de partículas para calcular la rotación del objeto con la función (6.2).

Algorithm 7 Interacción 3D mediante partículas

Require: *objetosVirtuales, marcoProfundidad, fuerzaManto, fuerzaSuperficie*

Ensure: Manipulación de objetos virtuales 3D mediante partículas

```
1: matricesTransformacion ← contenedor vacío
2: dimensionesMarco ← dimensiones de marcoProfundidad
3: for all objetoVirtual ∈ objetosVirtuales do
4:    manto ← vector vacío
5:    superficie ← vector vacío
6:    normales ← contenedor vacío
7:    particulas ← contenedor vacío
8:    objetoSujetado ← no
9:    esferaContenedora ← esfera contenedora de la malla de objetoVirtual
10:   for all fila ∈ marcoProfundidad do
11:      for all pixel ∈ fila do
12:         if pixel es mayor que cero then
13:            rayo ← genera rayo con la posición de pixel/dimensionesMarco
14:            if rayo interseca esferaContenedora then
15:               particula ← calcula coordenadas OpenGL de pixel
16:               centroObjeto ← posición del centro de esferaContenedora
17:               direccionParticula ← centroObjeto − particula
18:               distanciaParticula ← calcula distancia de direccionParticula
19:               rayo ← genera rayo con origen particula y direccionParticula
20:               for all triangulo ∈ malla de objetoVirtual do
21:                  if rayo interseca triangulo then
22:                     distanciaTriangulo ← calcula distancia entre particula y triangulo
23:                     if distanciaParticula es menor que distanciaTriangulo then
24:                        if distanciaTriangulo es igual que cero then
25:                           superficie ← superfice + direccionParticula ∗ fuerzaSuperficie
26:                        else
27:                           if distanciaTriangulo es menor que cero then
28:                              manto ← manto + direccionParticula ∗ fuerzaManto
29:                           end if
30:                        end if
31:                        normalTriangulo ← calcula la normal de triangulo
```

```
                    for all normal ∈ normales do
                        if producto punto entre normalTriangulo y normal es menor que coseno de 120° then
                            objetoSujetado ← si
                        end if
                    end for
                    Agrega normal a normales
                    Agrega particula a particulas
                end if
            end if
        end for
    end if
end if
end for
nuevaPosicionObjeto ← vector vacío
if objetoSujetado es verdad then
    nuevaPosicionObjeto ← manto − superficie
else
    nuevaPosicionObjeto ← manto + superficie
end if
linea ← ajusta línea a particulas
rotacionObjeto ← calcula ángulos de linea
Traslada objetoVirtual a nuevaPosicionObjeto
Rota objetoVirtual con rotacionObjeto
end for
```

6.2.3 Fuerzas físicas

Los objetos virtuales no solo están gobernados por las fuerzas de atracción y repulsión que suministran los objetos físicos. También actúan bajo una fuerza de gravedad y por la impenetrabilidad entre objetos.

La fuerza de gravedad es un mecanismo indispensable para la manipulación de un entorno virtual. Sobre todo si se cuenta con un mecanismo para sujetar objetos. En aras de evitar la suspensión indefinida de objetos por la carencia de gravedad, se decidió proveer al entorno de una fuerza de gravedad.

De modo similar a la gravedad de nuestro planeta tierra la fuerza de aceleración para un objeto en el entorno virtual es de 9.81 m/s^2. Así pues cuando se libera un objeto en el "aire" se calcula la distancia recorrida d en un tiempo t del objeto hasta que toque la base del entorno virtual; g es la constante de gravedad y v_0 es la velocidad inicial.

$$d = v_0 t + \frac{gt^2}{2} \tag{6.4}$$

Por otra parte para evitar que un objeto virtual ocupe al mismo tiempo el lugar de otro objeto se optó por construir un procedimiento que asemeje la impenetrabilidad entre objetos. Idealmente, los vértices de la malla de un objeto son la mejor opción para construir una fuerza de separación entre los objetos del entorno. Así, para cada vértice de una malla se podría calcular la distancia que existe con respecto a los triángulos de la malla de otro objeto; si la distancia para algún vértice es menor que cero entonces se ejercería una fuerza de separación entre las mallas involucradas.

Como es de esperarse, evaluar la distancia para los cientos o miles de vértices que puede contener una malla es un proceso costoso. En cambio, de forma semejante a la manipulación de objetos mediante los dedos, si se representa la malla de un objeto virtual a través de una esfera contenedora se puede evaluar de forma rápida la impenetrabilidad entre objetos. Solamente se requiere calcular la distancia que existe entre los centros de las esferas para identificar si dos objetos están ocupando el mismo lugar.

El algoritmo 8 describe el procedimiento para calcular la fuerza de gravedad y la impenetrabilidad entre objetos. Primeramente, se obtiene el tiempo actual de la computadora y

la posición de cada objeto virtual en su entorno (líneas 1-3). Enseguida se verifica si algún objeto está siendo sujetado por el usuario (línea 4). De ser así se almacena el tiempo obtenido como el tiempo inicial de caída de dicho objeto (línea 5). Sino, mientras el objeto esté en el aire, se calcula la distancia recorrida entre el tiempo inicial de caída y el tiempo actual (líneas 7-11). Posteriormente para cada objeto virtual se identifica si su esfera contenedora colisiona con la esfera contenedora de otro objeto (líneas 13 -19). En caso de que existe alguna colisión se verifica si alguna de las esferas involucradas está por encima de otra (líneas 20-21). Cuando esto ocurre, la esfera que está por encima debe mantenerse apilada a la otra esfera (líneas 22-23). Si las esferas no están encimadas entonces se calcula una fuerza de repulsión entre entre los objetos virtuales (líneas 25-30).

Algorithm 8 Fuerzas físicas básicas para objetos virtuales

Require: *objetosVirtuales, fuerzaSeparacion, direccionEjeVertical*

Ensure: Gravedad e impenetrabilidad entre objetos

1: **for all** $objetoVirtual \in objetosVirtuales$ **do**
2: $tiempoTranscurrido \leftarrow$ tiempo del sistema
3: $posicionObjetoVirtual \leftarrow$ posición de $objetoVirtual$
4: **if** $objetoVirtual$ está siendo sujetado **then**
5: $tiempoInicial \leftarrow tiempoTranscurrido$
6: **else**
7: **if** $posicionObjetoVirtual.altura$ es mayor que base de $objetoVirtual$ **then**
8: $segundos \leftarrow tiempoTranscurrido - tiempoInicial$
9: $posicionObjetoVirtual.altura \leftarrow posicionObjetoVirtual.altura - 4.905f * (segundos * segundos)$
10: **end if**
11: **end if**
12: $radioObjetoVirtual \leftarrow$ radio de la esfera contenedora de $objetoVirtual$
13: **for all** $objetoCercano \in objetosVirtuales$ **do**
14: **if** $objetoVirtual$ es diferente de $objetoCercano$ **then**
15: $radioObjetoCercano \leftarrow$ radio de la esfera contenedora de $objetoCercano$
16: $posicionObjetoCercano \leftarrow$ posición de $objetoCercano$
17: $direccionObjetoCercano \leftarrow posicionObjetoCercano - posicionObjetoVirtual$
18: $distanciaObjetoCercano \leftarrow$ calcula distancia de $direccionObjetoCercano$
19: **if** $distanciaObjetoCercano$ es menor que $radioObjetoCercano + radioObjetoVirtual$ **then**
20: Normaliza $direccionObjetoCercano$
21: **if** producto punto entre $direccionObjetoCercano$ y $direccionEjeVertical$ es mayor que coseno de 45° **then**
22: $nuevaBase \leftarrow posicionObjetoVirtual.altura + radioObjetoVirtual$
23: Actualiza base de $objetoCercano$ con $nuevaBase$
24: **end if**
25: $fuerzaRepulsion \leftarrow direccionObjetoCercano * fuerzaSeparacion$
26: Actualiza posición de $objetoCercano$ con $fuerzaRepulsion$
27: **end if**
28: **end if**
29: **end for**
30: **end for**

Resumen

En este capítulo se presentó el componente de software que integra la interacción de un usuario entre los tableros del prototipo de *tabletop* de NEKO sobre un entorno virtual. El propósito principal en el desarrollo de este componente de software fue proveer al usuario de una interacción directa con las manos, es decir una manipulación del entorno virtual sin el uso de dispositivos de control artificial. Se desarrolló un mecanismo para la interacción 2D y otro mecanismo para la interacción 3D.

La interacción 2D se enfoca en la manipulación de objetos virtuales en dos dimensiones. Semejante a un dispositivo multitáctil, la interacción 2D se realiza sobre el cristal claro del tablero inferior del prototipo de *tabletop* de NEKO. Para identificar el contacto por parte del usuario sobre el cristal claro se empleó una técnica similar a la que reportan en [Wilson, 2010]. Asimismo se hizo uso de las coordenadas baricéntricas para identificar si los dedos del usuario hacen contacto con algún objeto virtual.

La interacción 3D se enfoca en la manipulación de objetos virtuales en tres dimensiones y se lleva a cabo sobre el espacio que existe entre los tableros del prototipo de *tabletop* de NEKO. Se desarrollaron dos técnicas, y las tareas correspondientes, para la interacción con los objetos virtuales. La primera técnica se realiza exclusivamente con los dedos y fue diseñada con la idea de aminorar en lo posible los recursos involucrados en la interacción con objetos virtuales. En cambio la segunda se realiza mediante partículas y se construyó como una manipulación alternativa que involucra mayor cantidad de recursos pero que a su vez ofrece mejores resultados.

Capítulo 7

Resultados

"No existe tal cosa como el fracaso. Solo existen resultados."
Anthony Robbins

Los dos capítulos anteriores se centraron en el diseño, organización y algoritmos del módulo *interacción con objetos virtuales* que forma parte de la arquitectura de software de NEKO. En dichos capítulos se mostraron imágenes que evidencian la funcionalidad de los algoritmos para la detección de rostros, manos y gestos. Además se describieron los algoritmos involucrados para la manipulación de objetos virtuales 2D y 3D.

Desafortunadamente, por falta de tiempo, no se logró desarrollar una aplicación completa que refleje el gran potencial de NEKO con la interacción con objetos virtuales. No obstante, se desarrollaron un par de pruebas en las cuales se demuestran que este trabajo de investigación ha logrado exitosamente su objetivo, es decir, construir un prototipo de *tabletop* que permite la visualización y manipulación de objetos virtuales 2D y 3D de forma semejante a la vida real. En este capítulo se describen los resultados de tales pruebas y además se provee enlaces a videos que muestran en detalle lo que se puede hacer actualmente con NEKO; asimismo se describen algunas posibles aplicaciones para el uso de NEKO.

7.1 Interacción 2D

Como se mencionó en el capítulo anterior, la interacción 2D se enfoca en la manipulación de objetos virtuales en dos dimensiones sobre el cristal claro del tablero inferior del prototipo de *tabletop* de NEKO. En general la interacción 2D emplea los algoritmos 4 y 5 (descritos

con anterioridad en las secciones 5.3.1 y 6.1.2 respectivamente) para manipular los objetos virtuales. El algoritmo 4 permite identificar los dedos extendidos de una mano, mientras que el algoritmo 5 identifica si cualquiera de los dedos toca el cristal claro y algún objeto virtual.

Para evaluar la funcionalidad de esta interacción, se desplegó un menú virtual (mostrado en la figura 7.1) sobre el cristal claro del tablero inferior. De forma semejante a una pantalla multitáctil, se evaluó el contacto sobre los elementos del menú para realizar las siguientes actividades:

- cargar un objeto virtual
- cargar una imagen
- eliminar todos los elementos del entorno virtual
- ocultar el menú

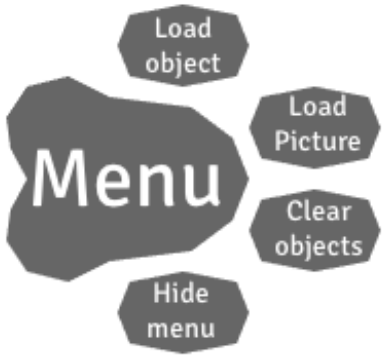


Figura 7.1: Menú construido para la interacción 2D.

La figura 7.3 muestra una secuencia de marcos donde se manipulan objetos virtuales 2D sobre el tablero inferior del prototipo de *tabletop* de NEKO. Mediante el proyector se despliega el menú (compuesto de múltiples objetos virtuales construidos a partir de triángulos[1]) sobre la esquina inferior izquierda del cristal claro. Cuando el usuario hace contacto con algún elemento del menú se efectúa una tarea asociada a dicho elemento; por ejemplo, cargar un objeto virtual o eliminar todos los elementos del entorno. Asimismo cuando un dedo toca y arrastra un objeto virtual 2D, éste se traslada a las coordenadas del dedo. En cambio cuando múltiples dedos hacen contacto con un objeto se calcula un valor de rotación y escalado.

[1]Se recordará que en NEKO todos los elementos del entorno virtual están construidos a partir de triángulos.

Figura 7.2: Objeto virtual constituido de dos triángulos.

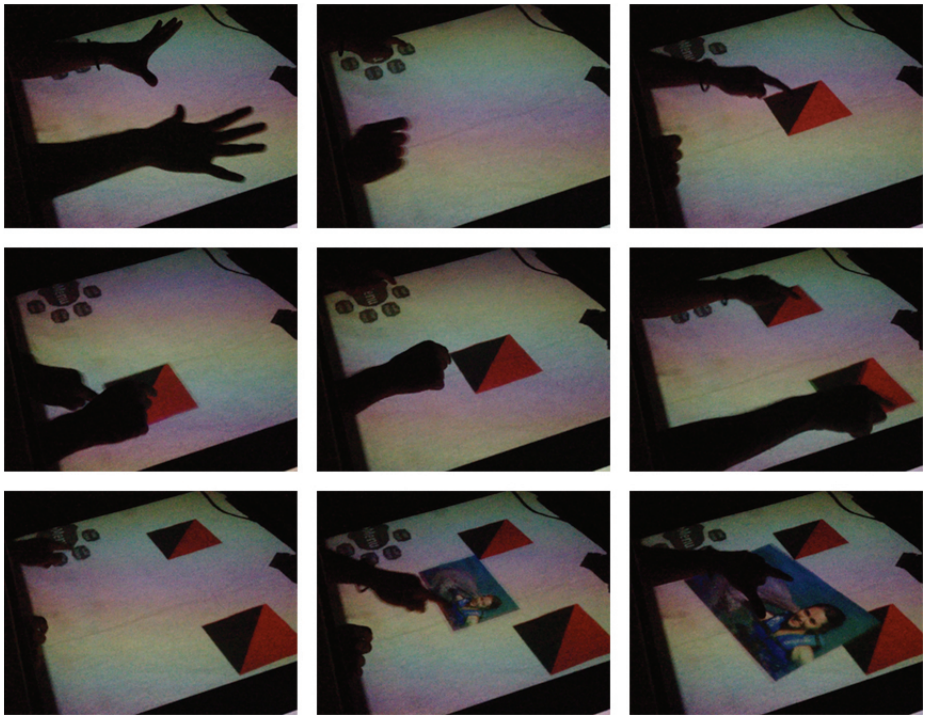


Figura 7.3: Manipulación de objetos 2D.

Con el afán de evidenciar los resultados obtenidos mediante esta sencilla prueba, se decidió mostrar un video donde se manipula un cuadro virtual constituido de dos triángulos (ver figura 7.2) y una imagen capturada por una cámara digital. En el video, el cual se puede acceder mediante la dirección **http://youtu.be/_rBYvhrN8V0**, se aprecia como al tocar el cuadro virtual con un dedo, el cuadro se traslada de posición conforme al movimiento inducido por el dedo. De igual forma, al tocar la imagen con dos dedos, ésta se rota y escala dependiendo de la posición y separación de los dedos.

La detección táctil en NEKO no es tan precisa como la de un teléfono celular o una tableta, pero permite manipular objetos virtuales 2D con suficiente exactitud. Además, si se compara los costos implicados en la construcción de la superficie táctil de NEKO con los de una pantalla del mismo tamaño, se puede observar que es mucho menor.

7.2 Interacción 3D

La interacción 3D se enfoca en la manipulación de objetos virtuales en tres dimensiones mediante las manos. Esta interacción depende intrínsecamente de la detección de rostro, manos y gestos del usuario. La detección de rostro permite desplegar los objetos virtuales acordemente a la perspectiva que corresponde a la posición del rostro del usuario. Mientras que la detección de manos y gestos son la base para la manipulación del entorno virtual.

Para evaluar la funcionalidad de la interacción 3D se decidió construir un entorno virtual con objetos de prueba simples y con objetos un poco más elaborados. Como se describió en el capítulo 6, se desarrollaron dos técnicas para la interacción con objetos virtuales. A continuación se describen los resultados obtenidos con tales técnicas.

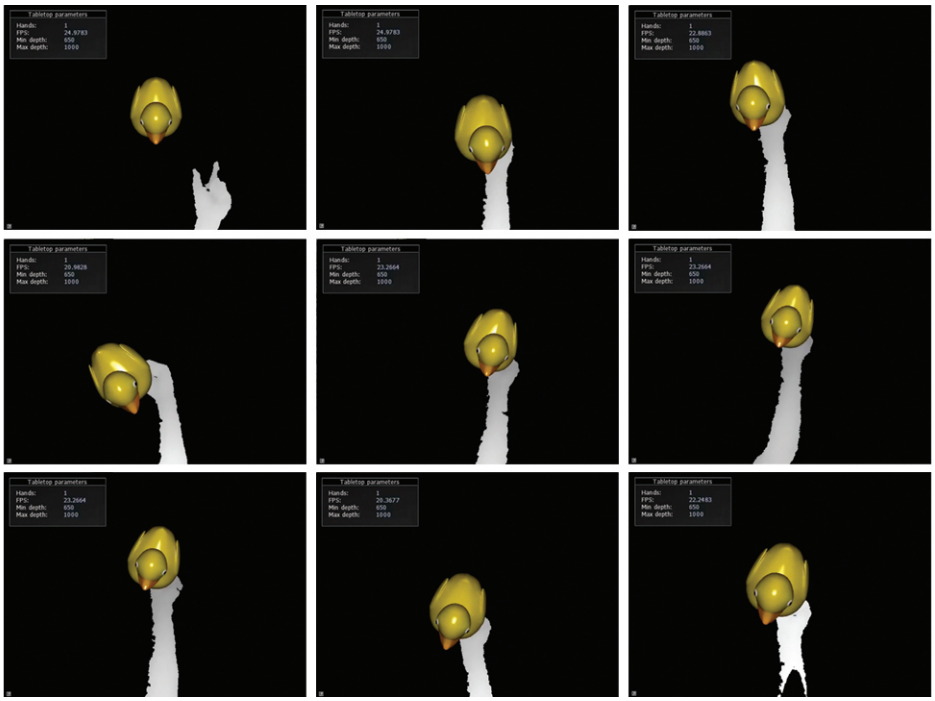


Figura 7.4: Manipulación de un objeto virtual mediante los dedos.

7.2.1 Manipulación de objetos virtuales con los dedos

Como se describió en la sección 6.2.1, esta técnica emplea exclusivamente los dedos como mecanismo de interacción. Asimismo en esta técnica los objetos virtuales se representan a través de un cubo contenedor. En la figura 7.4 se muestra una secuencia de marcos donde se manipula un pato de goma virtual mediante los dedos. A través de un cubo contenedor se

identifican si algún dedo hace contacto con el pato de goma. Cuando los dedos pulgar e índice tocan el cubo contenedor, el pato de goma se manipula conforme a la posición e inclinación de los dedos que el sistema detecta. En los marcos de la segunda fila se puede apreciar con claridad la rotación inducida al pato de goma sobre el eje Y; en los marcos de la fila inferior se observa la traslación sobre el eje Z e Y.

El video que muestra con mayor detalle la manipulación de objetos virtuales con los dedos se encuentra en la siguiente dirección: **http://youtu.be/ZCF9F4_hD7w**. En dicho video se puede observar como un objeto virtual se puede trasladar y rotar sobre los ejes X, Y y Z mediante los dedos pulgar e índice.

Esta técnica de interacción no requiere demasiados recursos computacionales para manipular los objetos virtuales. Sin embargo se restringe sobremanera la interacción de NEKO exclusivamente con los dedos. Otra de los inconvenientes de esta técnica de interacción es la "holgura" que existe entre el cubo contenedor y la malla de objetos virtuales irregulares (ver sección 6.2.2).

7.2.2 Manipulación de objetos virtuales a través de *partículas*

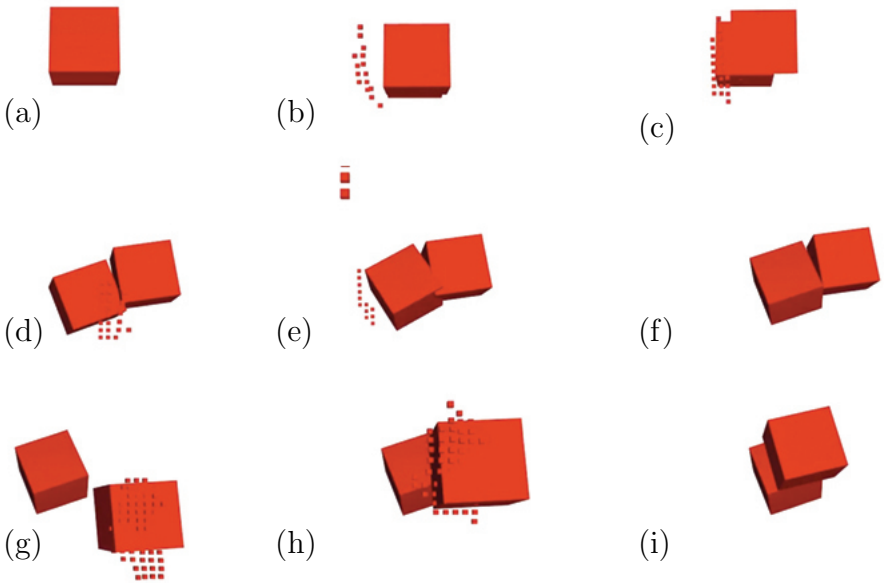


Figura 7.5: Fuerzas físicas entre objetos virtuales.

En la manipulación de objetos virtuales a través de *partículas* los objetos físicos capturados por el dispositivo Kinect se representan como cúmulos de pequeñas esferas suspendidas en el aire. Para comprobar la funcionalidad de esta técnica se decidió construir un entorno

virtual con objetos simples tales como cubos.

La figura 7.5 muestra una secuencia de marcos donde se manipulan dos cubos. Inicialmente en (a) la impenetrabilidad entre objetos mantiene un cubo posicionado por encima del otro. Posteriormente en (b)-(c) el cubo superior es movido hacia la derecha hasta que en (d) cae a un costado del otro cubo. En (e)-(f) el cubo izquierdo es empujado hacia la derecha, por lo que el cubo derecho también es empujado en la misma dirección. Finalmente el cubo de la derecha se levanta en (g)-(h) y se posiciona nuevamente sobre el otro cubo en (i). El video donde se demuestra los resultados de esta interacción con los cubos se encuentra en **http://youtu.be/3Vm2dlDDVg8**.

A partir de la manipulación anterior de los cubos se observó que NEKO es capaz de facilitar una interacción con objetos virtuales muy semejante a la vida real. Por ello se decidió construir un entorno con objetos virtuales de mayor grado de realismo. Para fines prácticos se construyó un tablero de ajedrez con dos peones. En la figura 7.6 se muestra una secuencia de imágenes donde se manipula uno de los peones, posicionado sobre un tablero de ajedrez, mediante la técnica de cúmulo de partículas. En (b) y (c) se puede observar como un cúmulo de partículas (que representan la mano del usuario) se acerca al peón de la derecha. Cuando el cúmulo de partículas hace contacto con la malla del peón, éste se manipula conforme a la interacción inducida por el cúmulo; en (d) el peón está siendo sujetado y levantado. En (e) y (f) se puede observar como el peón se deja de sujetar, por lo que el peón cae nuevamente sobre el tablero de ajedrez.

La manipulación mediante partículas permite identificar el contacto entre la malla de un objeto virtual y un objeto físico (representado mediante un cúmulo de partículas) de forma rápida y con un costo computacional no tan elevado. Sin embargo, una limitante de esta técnica es que la velocidad del sistema se reduce con objetos virtuales que contengan una malla con varios miles de triángulos. El video de la manipulación del peón sobre el tablero de ajedrez se puede observar en el siguiente enlace **http://youtu.be/IBn77o0Yfvs**.

7.3 Aplicaciones

Las aplicaciones que se pueden realizar sobre NEKO son realmente variadas. NEKO permite la manipulación de objetos 2D y 3D *en general* y por esta razón puede considerarse como una base en la cual se pueden construir aplicaciones de manipulación específicas. Por

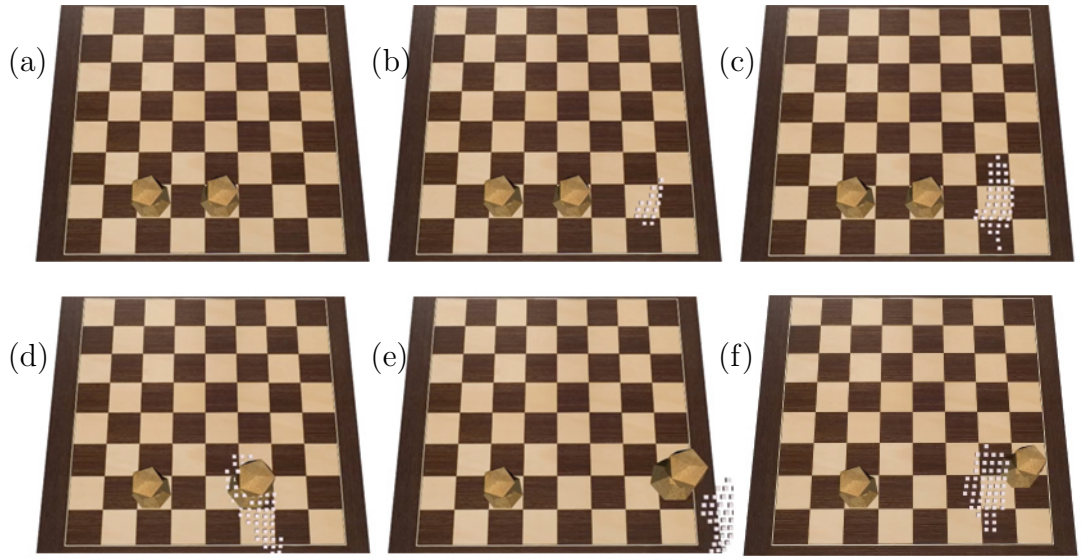


Figura 7.6: Manipulación de un objeto virtual mediante partículas.

ejemplo aplicaciones enfocadas a las siguientes actividades:

- Pruebas con prototipos de hardware. Mediante NEKO, un diseñador de productos de hardware podría manipular el modelo virtual de su producto en construcción. En consecuencia el diseñador podría evaluar a priori aspectos de dicho producto sin la necesidad de construirlo físicamente.

- Manejo de equipo peligroso, complejo y/o frágil. En el sector salud, muchas hospitales preparan a sus médicos para realizar operaciones quirúrgicas mediante la observación. En NEKO a través de una aplicación, los futuros médicos podrían además entrenar sus habilidades de forma práctica sin comprometer la vida de una persona. De hecho se podría construir aplicaciones para la capacitación de personal en gran cantidad de maquinaria, incluso para tareas de ensamblaje/desensamblaje.

- Asistiendo a profesores en los salones de clases. La atención de los estudiantes podría incrementarse mediante una aplicación que facilite la presentación de material didáctico o multimedia interactiva. Mediante NEKO el profesor sería capaz de manipular objetos relacionados a algún tema (como los planetas de nuestro sistema solar) facilitando su labor en el aula.

- Juegos. Gran cantidad de juegos 2D y 3D son posibles desarrollar sobre NEKO, desde simples rompecabezas hasta juegos RPG[2]. Los usuarios en lugar de usar controles

[2]Role-playing game. Son juegos en donde el jugador controla un personaje virtual

físicos, podrían interactuar directamente con una interfaz 2D sobre el cristal claro o incluso manipular objetos 3D entre el espacio de los tableros.

Estas son solo algunos ejemplos de las múltiples aplicaciones donde se podría utilizar NEKO. Como se puede observar existen muchas áreas donde la manipulación de objetos virtuales mediante las manos aportaría grandes beneficios para la interacción con aplicaciones de software.

Resumen

En este capítulo se describieron los resultados de las herramientas de NEKO que permiten la manipulación de objetos 2D y 3D. A pesar de que el tiempo no permitió construir una aplicación completa y enfocada a una necesidad particular, las pruebas realizadas con varios objetos virtuales posibilitan concluir que actualmente NEKO permite realizar una manipulación de objetos virtuales 2D y 3D de manera semejante a la vida real. Particularmente, los videos que se enlazan en este capítulo dan una idea de la utilidad y precisión de NEKO. De entre las aplicaciones que se pueden construir sobre NEKO se encuentran entrenamiento de personal en empresas, pruebas con prototipos de hardware, enseñanza y juegos.

Capítulo 8

Conclusiones

El presente trabajo de tesis presentó un prototipo de *tabletop* para la manipulación de objetos virtuales en dos y tres dimensiones. NEKO (del inglés *Natural Environment for Kinect-based Object interaction*) es el nombre del prototipo de *tabletop*, desarrollado con hardware comercial de bajo costo y software libre, que permite el despliegue y manipulación de un entorno virtual acorde a la percepción del usuario.

El principal objetivo en la construcción de NEKO fue crear un sistema que permitiera al usuario manipular objetos virtuales de una forma semejante a como se haría sobre una mesa convencional, sin la necesidad de instrumentos o dispositivos de control artificial de por medio. Para cumplir con esta meta se construyó un prototipo de mesa interactiva instrumentada con el dispositivo Kinect, una cámara web y un proyector. Asimismo se desarrolló un sistema de visión por computadora para identificar la interacción realizada por el usuario.

El prototipo de mesa de NEKO, a diferencia de una mesa convencional, posee una estructura constituida de dos tableros superpuestos a cierta distancia entre sí. A su vez cada tablero está compuesto de un marco y un cristal con dimensiones acorde a las especificaciones del dispositivo Kinect. De igual forma la estructura de la mesa se diseñó en relación a la estatura promedio de la población mexicana (164 centímetros).

Por otra parte el sistema de visión por computadora se diseñó en base a una arquitectura de software con componentes modulares en aras de facilitar la administración y el control del sistema. Cada módulo se estructuró por actividades específicas relacionadas entre sí, lo cual permitió organizar el sistema en tareas con menor grado de complejidad. El requerimiento primordial tomado en cuenta en el diseño del sistema fue identificar la interacción del usuario con un tiempo de respuesta corto.

Además se propuso un modelo de clasificación para el desplegado del entorno virtual acorde a la perspectiva del usuario. Tal y como sucede en la vida real la perspectiva de los objetos cambia mientras trasladamos nuestra cabeza a una nueva ubicación. Para simular semejante comportamiento con el entorno virtual se construyó y empleó un modelo de clasificación para discernir el rostro del usuario, mientras interactua sobre la *tabletop*, sobre los marcos capturados por la cámara web. Así se calcula una perspectiva del entorno virtual acorde a la posición del rostro del usuario.

Debido a los problemas encontrados con el *middleware* NITE también se construyó un modelo de clasificación para la detección de manos. A diferencia de NITE este modelo no requiere de un gesto inicial para identificar la posición de la mano. Incluso permite reconocer la silueta de una mano en una sombra o sobre una hoja de papel.

Con el interés de identificar movimientos significativos de las manos se desarrolló un mecanismo para el reconocimiento de gestos. Se determinó que un puño es la pose ideal para iniciar el reconocimiento de gestos pues es fácil de realizar y sobretodo permite discriminar los movimientos de la mano sobre la manipulación de objetos virtuales. De igual forma, se construyó un modelo de clasificación para verificar la correspondencia de los movimientos acorde a dos funcionalidades: moverse entre espacios de trabajo y construcción de un modelo para la detección de tacto.

Particularmente en este trabajo se exploraron dos mecanismos para la interacción con objetos virtuales. El primero se enfocó en la manipulación de objetos en dos dimensiones (interacción 2D) de forma semejante a una pantalla multitáctil. En general, en la interacción 2D se identifica el contacto de los dedos del usuario, sobre la superficie del cristal ubicado sobre el tablero inferior de la *tabletop*, a partir de un modelo construido con los marcos de profundidad del dispositivo Kinect. Debido al ruido residente sobre los marcos de profundidad la detección del tacto no es tan precisa como una pantalla táctil.

El segundo mecanismo se enfocó en la manipulación de objetos en tres dimensiones (interacción 3D). Se desarrollaron dos metodologías para la interacción 3D: mediante los dedos y a través de partículas. La interacción mediante los dedos no requiere demasiados recursos para manipular los objetos virtuales. Sin embargo, se restringe la interacción exclusivamente con los dedos. En contraste, la interacción a través de partículas otorga mayor flexibilidad pero a su vez involucra mayor cantidad de recursos.

8.1 Trabajo a futuro

Aun falta un gran trecho por recorrer y todavía existe mucho por mejorar sobre NEKO. Sin embargo, el tiempo fue una de las principales limitaciones durante el desarrollo de este trabajo de investigación. Entre las mejoras que se pueden estructurar para ampliar este proyecto como trabajo a futuro se encuentran:

- Diseñar un nuevo prototipo de *tabletop* más ligero y con componentes ajustables. La estructura rígida del prototipo actual y así como sus dimensiones dificultan el transporte de un lugar a otro. Además un diseño con componentes móviles y materiales ligeros como el aluminio permitiría ajustar el espacio de interacción acorde a la estatura de los usuarios.

- Adicionar otro proyector y otro dispositivo Kinect. Actualmente se emplea un solo proyector para desplegar los objetos 2D y 3D de forma simultanea. No obstante la visualización del entorno virtual se mejoraría con el desplegado por separado de los objetos 2D sobre el tablero inferior y los objetos 3D sobre el tablero superior. Asimismo mediante la añadidura de otro dispositivo Kinect se reduce la posibilidad de oclusión entre las manos del usuario (es decir que una mano obstruya la visibilidad al dispositivo Kinect de la otra mano) mientras manipula los objetos 3D.

- Emplear un mecanismo de calibración de la superficie táctil con los dedos. Actualmente se emplea una hoja de papel para construir el modelo de la superficie interactiva. Sin embargo se podría calcular un modelo mediante el contacto de determinadas partes del tablero inferior. Esto facilitaría la construcción del modelo de la superficie y evitaría el uso de la hoja de papel.

- Diseñar una arquitectura de software para la ejecución de tareas simultaneas. En general el desempeño del sistema podría beneficiarse de las bondades del cómputo paralelo. Sobre todo en la detección de colisiones con la maya de un objeto virtual. Incluso valdría la pena investigar el desarrollo de algoritmos sobre la GPU.

- Agregar nuevas fuerzas físicas al entorno virtual. El solo desarrollo de un componente de software para la simulación de fuerzas físicas es digno de un trabajo de investigación. Por ejemplo en tareas para deformar objetos virtuales acorde a la fuerza inducida por el usuario o incluso para la manifestación de energía entre objetos (como calor, fricción, etc.).

- Agregar interacción mediante voz. El dispositivo Kinect está integrado de varios micrófonos, lo cual permite el desarrollo de aplicaciones para esta modalidad. Desafortunadamente las bibliotecas utilizadas en NEKO para la comunicación con el dispositivo Kinect aun no contaban con la funcionalidad de uso de micrófonos. Posiblemente las nuevas versiones de estas bibliotecas ya permiten el uso de micrófonos.

- Desarrollar herramientas que permitan construir aplicaciones para el control de software. Existen muchas áreas donde la manipulación de objetos virtuales mediante las manos aportaría grandes beneficios para la interacción con aplicaciones de software, como en los salones de clases asistiendo a profesores a presentar material didáctico. Por ello sería ideal el desarrollo de herramientas de software que permitan construir aplicaciones sobre NEKO acorde a las necesidades de los diversos usuarios. Por ejemplo, editores de presentaciones o software para la construcción de multimedia interactiva. Incluso valdría la pena evaluar la posibilidad de integrar a NEKO sobre un sistema operativo móvil como Android y aprovechar el nicho de aplicaciones existente.

Apéndice A

Detector de manos basado en OpenNI/NITE

En el capítulo 5, en la sección detección de manos, se mencionó que se utilizó la biblioteca OpenNI y el *middleware* NITE para desarrollar un detector de manos sobre NEKO. Como se indicó anteriormente, se descartó el uso de dicho detector porque el *middleware* NITE no está diseñado para funcionar con marcos de profundidad que contengan únicamente manos y porque los tableros del prototipo de *tabletop* de NEKO ocasionan problemas con el mecanismo de rastreo que emplea NITE. Sin embargo para fines meramente educacionales se decidió describir cómo es que se desarrolló dicho detector.

Los seres humanos manipulamos objetos a nuestro alrededor principalmente con los dedos, en especial con el dedo pulgar e índice. Por ello se decidió identificar los dedos de las manos sobre los marcos de video del dispositivo Kinect y asemejar la interacción con objetos virtuales a la vida real. El resultado de este detector se puede observar sobre la sucesión de marcos de videos mostrado en la Figura A.1. En la esquina inferior izquierda de cada marco de video se observa la elipse ajustada al contorno de la mano. Los puntos superiores son los dedos del usuario, mientras que el punto inferior es la base de la palma.

El algoritmo 9 describe el proceso para identificar las manos y dedos del usuario entre los tableros de la *tabletop*. Este proceso requiere, por imposición del *middleware* NITE, que el usuario primeramente realice un gesto con sus manos para iniciar su rastreo. Las utilidades de NITE encargadas identificar los gestos realizados por el usuario son los generadores de manos y de gestos. Con la información implícita en el contexto del dispositivo Kinect, se inicializan estos generadores y se define el espacio de búsqueda sobre los marcos del sensor de profundidad (líneas 1-4). Enseguida se examinan los movimientos realizados por el

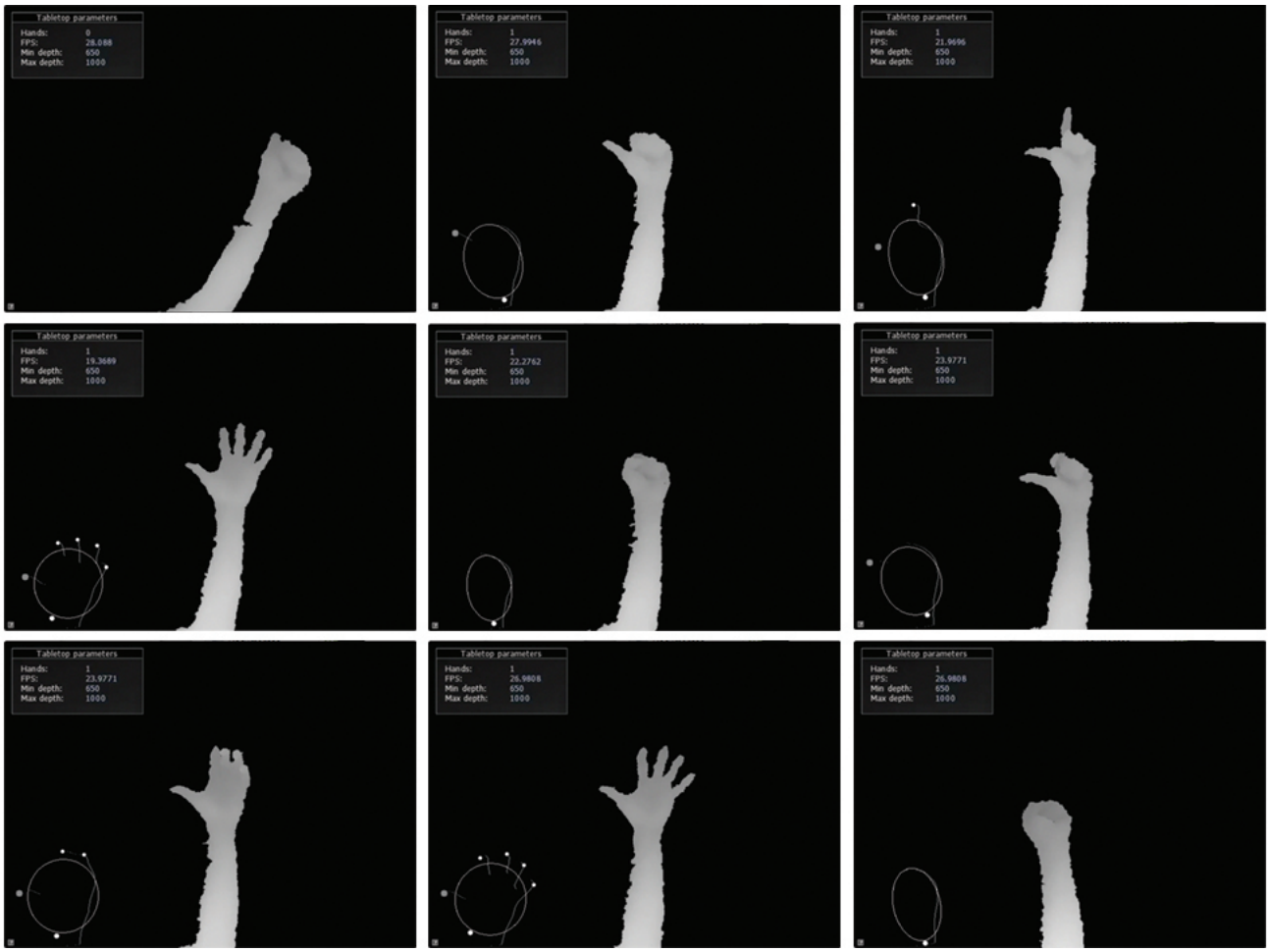


Figura A.1: Detección de manos y dedos sobre los marcos del Kinect

usuario y si alguno de ellos se reconoce como un gesto conocido se inicia el rastreo de la mano (líneas 5-6). Después se obtiene la posición de la mano que realizó el gesto y se extrae una imagen con esta posición (líneas 7-9). Mediante la imagen extraída se obtiene el contorno de la mano con la función de OpenCV *findContours()* (línea 10). Posteriormente se ajusta una elipse al contorno de la mano mediante la función *fitEllipse()*, e inmediatamente el contorno se aproxima a un polígono con la función *approxPolyDP()* (líneas 11-12). Si alguno de los puntos del polígono está afuera de la elipse entonces se considera como un dedo (líneas 14-15). Finalmente, se obtiene la distancia de la base de la elipse a cada uno de los dedos y si alguno está dentro de un rango definido se considera como un pulgar (líneas 16-20).

Algorithm 9 Detección de manos y dedos sobre los marcos del Kinect

Require: *contextoOpenNI*

Ensure: Dedos

Inicializa *generadorManos* con la información de *contextoOpenNI*
Inicializa *generadorGestos* con la información de *contextoOpenNI*
Define espacio de búsqueda para *generadorGestos*
Inicia captura de *generadorManos* y *generadorGestos*
for all *movimiento* ∈ *generadorGestos* **do**
  if *movimiento* es algún gesto **then**
    posicionMano ← obtén posición de *generadorManos*
    Convierte *posicionMano* a coordenadas proyectivas
    Extrae *recuadroMano* a partir de *posicionMano*
    Extrae *contornoMano* de *recuadroMano*
    Ajusta *elipsePalma* con *contornoMano*
    Aproxima a un polígono *contornoMano*
    dedos ← contenedor vacío
    for all *punto* ∈ *contornoMano* **do**
      if *punto* está afuera de *elipsePalma* **then**
        distanciaBase ← calcula la distancia de *punto* a la base de *elipsePalma*
        if *distanciaBase* es menor que un rango **then**
          Considera a *punto* como pulgar
        end if
        Agrega *punto* a *dedos*
      end if
    end for
    puntoMedio ← promedio de dedos que no son pulgar
  end if
end for

Apéndice B

Detector de pose basado en SVM

En el capítulo 5, en la sección detección de gestos, se mencionó que inicialmente se había desarrollado un mecanismo capaz de identificar las posturas de la mano mediante la técnica SVM, en esta sección se describe en detalle dicho mecanismo. Cabe mencionar que el detector de pose que utiliza NEKO está descrito en la sección 5.3.1.

Aprovechando el potencial de la técnica SVM se decidió construir un modelo capaz de identificar las posturas de las manos. De forma semejante a la mayoría de técnica de clasificación, el proceso para construir un modelo de clasificación con SVMs consta de dos tareas: *creación de muestras* y *aprendizaje del modelo*. La creación de muestras, como su nombre lo indica, consiste en aglomerar imágenes de manos con posturas específicas. Para esta tarea se utilizó el algoritmo 3, descrito en la sección *detección de manos* del capítulo 5, pero en lugar de almacenar las dimensiones de cada muestra se guarda una etiqueta de clase asociada a la postura. La Figura B.1 presenta algunas muestras, con sus respectivas etiquetas de clase, de un conjunto de posturas ejemplo.

0 1 2

Figura B.1: Muestras ejemplo de posturas de mano.

Posteriormente, de cada muestra capturada se extrae información relevante que facilite

la construcción del modelo de clasificación para la detección de pose (ver Algoritmo 10). Para cada muestra se extraen los puntos que constituyen la silueta de la mano a través de la función de OpenCV *findContours()* (líneas 1-2). Enseguida, acorde a dichos puntos, se ajusta una elipse y se calcula un polígono mediante las funciones *fitEllipse()* y *approxPolyDP()* respectivamente (líneas 3-4). El ángulo implícito en la elipse indica la rotación actual de la mano (y por ende del polígono) sobre la imagen, por ello para homogeneizar la rotación de todas las muestras, se contrarresta la rotación de cada polígono con el ángulo de la elipse (línea 5). Finalmente se construye una nueva muestra con el conjunto de vectores que forman el perímetro del polígono (ver Figura B.2) y la etiqueta de clase asociada a la postura de la mano (líneas 7-11).

Figura B.2: Conjunto de direcciones que conforman a un polígono de una mano.

Algorithm 10 Creación de muestras para el modelo de poses

Require: *muestras, etiquetas*

Ensure: Dedos

1: **for all** $muestra \in muestras$ **do**
2: $contornoMano \leftarrow$ contorno de $muestra$
3: $elipsePalma \leftarrow$ ajuste de elipse a $contornoMano$
4: $contornoMano \leftarrow$ aproximación a un polígono $contornoMano$
5: Rota $contornoMano$ con el valor negativo del ángulo de $elipsePalma$
6: $muestrasDireccion \leftarrow$ contenedor vacío
7: **for all** $punto \in contornoMano$ **do**
8: $siguientePunto \leftarrow punto + 1$
9: Agrega $siguientePunto - punto$ a $muestrasDireccion$
10: **end for**
11: Agrega valor $etiqueta$ de $muestra$ a $muestrasDireccion$
12: **end for**

Una vez generadas las muestras se construye el modelo de clasificación, es decir se encuentra un hiperplano óptimo de separación entre las diferentes posturas de mano mediante la clase *SVM* de *OpenCV*. Considerando la recomendación en la documentación para la clase *SVM* el cálculo del *hiperplano* se realizó mediante la función kernel **Base Radial** $K(x_i, x_j) = e^{-\gamma||x_i - x_j||^2}, \gamma > 0$. La Figura B.3 muestra un conjunto de marcos donde se identifica la postura de la mano mediante el modelo de clasificación construido.

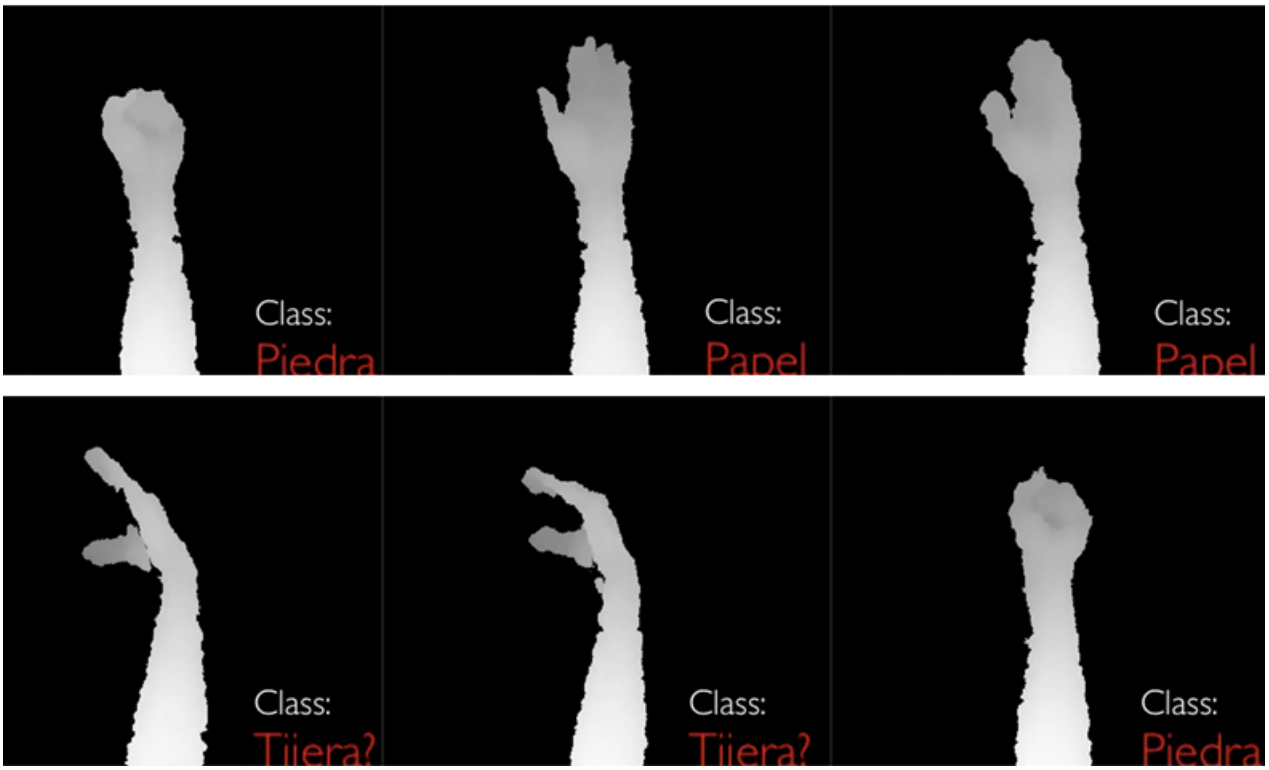


Figura B.3: Detección de posturas de manos mediante SVM.

Bibliografía

[Hew, 1992] (1992). Acm sigchi curricula for human-computer interaction. Technical report, New York, NY, USA.

[Appel, 1968] Appel, A. (1968). Some techniques for shading machine renderings of solids. In *AFIPS Spring Joint Computing Conference*, pages 37–45, Atlantic City, NJ, USA. Thomson Book Company.

[Barnes, 1997] Barnes, S. B. (1997). Douglas carl engelbart: Developing the underlying concepts for contemporary computing. *IEEE Annals of the History of Computing*, 19(3):16–26.

[Benko et al., 2008] Benko, H., Wilson, A. D., and Balakrishnan, R. (2008). Sphere: multi-touch interactions on a spherical display. In Cousins, S. B. and Beaudouin-Lafon, M., editors, *Proceedings of the 21st annual ACM symposium on User interface software and technology*, UIST 2008, pages 77–86, Monterey, CA, USA. ACM.

[Bennett and Rajlich, 2000] Bennett, K. H. and Rajlich, V. T. (2000). Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, pages 73–87, Limerick, Ireland. ACM.

[Bowman et al., 2004] Bowman, D. A., Kruijff, E., LaViola, J. J., and Poupyrev, I. (2004). *3D User Interfaces: Theory and Practice*. Addison Wesley Longman Publishing Co., Inc.

[Breiman et al., 1984] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth.

[Buxton et al., 1985] Buxton, W., Hill, R., and Rowley, P. (1985). Issues and techniques in touch-sensitive tablet input. In Cole, P., Heilman, R., and Barsky, B. A., editors, *Annual Conference on Computer Graphics and Interactive Techniques*, pages 215–224. ACM.

[Chao, 2009] Chao, G. (2009). Human-computer interaction: Process and principles of human-computer interface design. In *Computer and Automation Engineering, 2009. ICCAE '09. International Conference on*, pages 230 –233, Bangkok Thailand. IEEE.

[Favalora, 2005] Favalora, G. E. (2005). Volumetric 3d displays and application infrastructure. *IEEE Computer*, 38(8):37 – 44.

[Febretti et al., 2011] Febretti, A., Mateevitsi, V., Chau, D., Nishimoto, A., McGinnis, B., Misterka, J., Johnson, A., and Leigh, J. (2011). The omegadesk: Towards a hybrid 2d and 3d work desk. In Bebis, G., Richard D. Boyle, B. P., Koracin, D., Wang, S., Kim, K., Benes, B., Moreland, K., Borst, C. W., DiVerdi, S., Chiang, Y.-J., and Ming, J., editors, *Advances in Visual Computing*, volume 6939, pages 13–23, Las Vegas, NV, USA. Springer Berlin.

[Fitzgibbon et al., 1999] Fitzgibbon, A. W., Pilu, M., and Fisher, R. B. (1999). Direct least square fitting of ellipses. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(5):476–480.

[Frati and Prattichizzo, 2011] Frati, V. and Prattichizzo, D. (2011). Using kinect for hand tracking and rendering in wearable haptics. In Lynette A. Jones, Matthias Harders, Y. Y., editor, *World Haptics Conference (WHC), 2011 IEEE*, pages 317 –321. IEEE Computer Society.

[Freund and Schapire, 1996] Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In Saitta, L., editor, *Machine Learning, Proceedings of the Thirteenth International Conference*, pages 148–156, Bari, Italy. Morgan Kaufmann.

[Gama et al., 2012] Gama, A. D., Chaves, T., Figueiredo, L., and Teichrieb, V. (2012). Poster: Improving motor rehabilitation process through a natural interaction based system using kinect sensor. In *3DUI*, pages 145–146, Mesa, CA, USA. IEEE.

[Gregg and Tarrier, 2007] Gregg, L. and Tarrier, N. (2007). Virtual reality in mental health. *Social Psychiatry and Psychiatric Epidemiology*, 42:343–354.

[Han, 2005] Han, J. Y. (2005). Low-cost multi-touch sensing through frustrated total internal reflection. In Baudisch, P., Czerwinski, M., and Olsen, D. R., editors, *UIST*, pages 115–118, Seattle, WA, USA. ACM.

[Hartmann et al., 2009] Hartmann, B., Morris, M. R., Benko, H., and Wilson, A. D. (2009). Augmenting interactive tables with mice & keyboards. In Wilson, A. D. and Bérard, F., editors, *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, UIST 2009, pages 149–152, Victoria, BC, Canada. ACM.

[Henry et al., 2010] Henry, P., Krainin, M., Herbst, E., Ren, X., and Fox, D. (2010). Rgbd mapping: Using depth cameras for dense 3d modeling of indoor environments. In *In RGB-D: Advanced Reasoning with Depth Cameras Workshop in conjunction with RSS*.

[Hilliges et al., 2009] Hilliges, O., Izadi, S., Wilson, A. D., Hodges, S., Garcia-Mendoza, A., and Butz, A. (2009). Interactions in the air: adding further depth to interactive tabletops. In Wilson, A. D. and Bérard, F., editors, *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, UIST 2009, pages 139–148, Victoria, BC, Canada. ACM.

[Hutchins et al., 1985] Hutchins, E. L., Hollan, J. D., and Norman, D. A. (1985). Direct manipulation interfaces. *Human-Computer Interaction*, 1(4):311–338.

[Johnson et al., 1989] Johnson, J., Roberts, T. L., Verplank, W., Smith, D. C., Irby, C. H., Beard, M., and Mackey, K. (1989). The xerox star: A retrospective. *Computer*, 22(9):11–26, 28–29.

[Kaltenbrunner et al., 2006] Kaltenbrunner, M., Jorda, S., Geiger, G., and Alonso, M. (2006). The reactable*: A collaborative musical instrument. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2006. WETICE '06. 15th IEEE International Workshops on*, pages 406–411.

[Kim, 2005] Kim, G. J. (2005). *Designing virtual reality systems - the structured approach*. Springer.

[Kjeldsen et al., 2002] Kjeldsen, R., Pinhanez, C. S., Pingali, G., Hartman, J., Levas, T., and Podlaseck, M. (2002). Interacting with steerable projected displays. In Steven Feiner, J. A. L., editor, *Proceedings of the Fifth IEEE International Conference on Automatic Face and Gesture Recognition*, FGR 2002, pages 402–, Washington, DC, USA. IEEE Computer Society.

[Laurel and Mountford, 1990] Laurel, B. and Mountford, S. J., editors (1990). *The Art of Human-Computer Interface Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Letessier and Bérard, 2004] Letessier, J. and Bérard, F. (2004). Visual tracking of bare fingers for interactive surfaces. In Steven Feiner, J. A. L., editor, *Proceedings of the 17th annual ACM symposium on User interface software and technology*, UIST 2004, pages 119–122, Santa Fe, NM, USA. ACM.

[Lienhart et al., 2003] Lienhart, R., Kuranov, A., and Pisarevsky, V. (2003). Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In Michaelis, B. and Krell, G., editors, *DAGM-Symposium*, pages 297–304, Magdeburg, Germany. Springer.

[Liu, 2010] Liu, W. (2010). Natural user interface- next mainstream product user interface. In *Computer-Aided Industrial Design Conceptual Design (CAIDCD), 2010 IEEE 11th International Conference on*, volume 1, pages 203–205, Yiwu, China. IEEE.

[Marshall et al., 2008] Marshall, J., Pridmore, T., Pound, M., Benford, S., and Koleva, B. (2008). Pressing the flesh: Sensing multiple touch and finger pressure on arbitrary surfaces. In *Proceedings of the 6th International Conference on Pervasive Computing*, Pervasive '08, pages 38–55. Springer-Verlag.

[Meer et al., 1991] Meer, P., Mintz, D., Rosenfeld, A., and Kim, D. Y. (1991). Robust regression methods for computer vision: A review. *International Journal of Computer Vision*, 6(1):59–70.

[NUIGroup, 2010] NUIGroup (2010). *Multitouch Technologies*. Community Release.

[Papageorgiou et al., 1998] Papageorgiou, C., Oren, M., and Poggio, T. (1998). A general framework for object detection. In *ICCV*, pages 555–562.

[Preece and Benyon, 1993] Preece, J. and Benyon, D. (1993). *A guide to usability - Human Factors in Computing*. Addison Wesley.

[Rauterberg et al., 1997] Rauterberg, M., Bichsel, M., Krueger, H., and Meier, M. (1997). Pattern recognition as a key technology for the next generation of user interfaces. In Smith, M. J., Salvendy, G., and Koubek, R. J., editors, *International Conference on Human-Computer Interaction*, volume 2, pages 929–932, San Francisco, CA, USA. Elsevier.

[Scharstein and Szeliski, 2003] Scharstein, D. and Szeliski, R. (2003). High-accuracy stereo depth maps using structured light. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 195–202, Madison, WI, USA. IEEE Computer Society.

[Schöning et al., 2008] Schöning, J., Brandl, P., Daiber, F., Echtler, F., Hilliges, O., Hook, J., Löchtefeld, M., Motamedi, N., Muller, L., Olivier, P., Roth, T., and von Zadow, U. (2008). Multi-Touch Surfaces: A Technical Guide. Technical report.

[Smith, 1993] Smith, D. C. (1993). Pygmalion: an executable electronic blackboard. pages 19–48.

[Sutherland, 2003] Sutherland, I. E. (2003). Sketchpad: A man-machine graphical communication system. Technical Report UCAM-CL-TR-574, University of Cambridge, Computer Laboratory.

[Suzuki and Abe, 1985] Suzuki, S. and Abe, K. (1985). Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46.

[Tan et al., 2005] Tan, P.-N., Steinbach, M., and Kumar, V. (2005). *Introduction to Data Mining*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, "1.edition.

[van Dam, 2000] van Dam, A. (2000). Beyond wimp. *IEEE Computer Graphics and Applications*, 20(1):50–51.

[Vanderheeren, 2011] Vanderheeren, M. (2011). Tabletops and their advantage in a practical scenario. Master's thesis, Catholic University of Leuven, Belgium.

[Viola and Jones, 2001] Viola, P. A. and Jones, M. J. (2001). Rapid object detection using a boosted cascade of simple features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 511–518, Kauai, HI, USA. IEEE Computer Society.

[Wellner, 1991] Wellner, P. (1991). The digitaldesk calculator: tangible manipulation on a desk top display. In Rhyne, J. R., editor, *Proceedings of the 4th Annual ACM Symposium on User Interface Software and Technology*, UIST 1991, pages 27–33, Hilton Head, SC, USA. ACM.

[Wellner et al., 1993] Wellner, P., Mackay, W. E., and Gold, R. (1993). Computer-augmented environments: Back to the real world - introduction to the special issue. *Commun. ACM*, 36(7):24–26.

[Wigdor and Wixon, 2011] Wigdor, D. and Wixon, D. (2011). *Brave NUI World: Designing Natural User Interfaces for Touch and Gesture*. Morgan Kaufmann.

[Wilson, 2010] Wilson, A. D. (2010). Using a depth camera as a touch sensor. In Krüger, A., Schöning, J., Wigdor, D., and Haller, M., editors, *ACM International Conference on Interactive Tabletops and Surfaces*, pages 69–72, Saarbrücken, Germany. ACM.