Escuela de Computación

Maestría en Computación con énfasis en Ciencias de la Computación

LaGeR: Lenguaje para descripción de gestos bidimensionales y tridimensionales

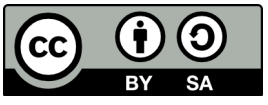
Tesis
para el grado de

Magister Scientiæ en Computación

Autor
Andrés Odio Vivi

Asesor
Erick Mata Montero, Ph.D.

mayo 2015



Dedicada a Milena

Agradecimientos

Agradezco a mi esposa Milena, a mis padres y al resto de mi familia y amigos por todo su apoyo y paciencia durante tantas largas noches. Agradezco también a mi profesor, el Dr. Erick Mata, por toda su ayuda en este proceso tan enriquecedor.

Índice general

Índice de cuadros

Índice de figuras

Abreviaturas

API: Application Programming Interface, es un conjunto de interfaces que permite integrar un sistema informático con otro.

DTW: Dynamic Time Warping, es un algoritmo para medir la similitud entre dos series de tiempo. Las series pueden diferir en tiempo o velocidad.

EBNF: Extended Backus-Naur Form, es una técnica de notación para gramáticas libres de contexto.

SDK: Software Development Kit, es un paquete de código y/o herramientas para desarrollar software.

SVG: Scalable Vector Graphics, es un formato de imágenes vectoriales basado en XML.

Resumen

El reciente auge de la realidad virtual está fomentando la proliferación de dispositivos de entrada gestuales tales como el Razer Hydra, el Leap Motion 3D Controller y el Microsoft Kinect. Esto dificulta la labor de los académicos y desarrolladores de aplicaciones, ya que deben programar con un Application Programming Interface (API) diferente para cada dispositivo.

El objetivo principal de la presente investigación fue definir e implementar un lenguaje para la representación e interpretación de gestos bidimensionales y tridimensionales. Por medio de este lenguaje, los desarrolladores podrán utilizar el lenguaje para definir gestos a ser detectados sin importar el dispositivo o las API de por medio.

Investigamos distintas formas de representar y reconocer gestos, y luego creamos el lenguaje LaGeR — *Language for Gesture Representation* (Lenguaje para la Representación de Gestos). Para probar su factibilidad, escribimos un conjunto de herramientas y bibliotecas de *software* para convertir movimientos de sensores a hileras de LaGeR, reconocer las mismas como gestos, visualizarlos y comunicar su detección a programas suscriptores. Además, creamos un programa de prueba que utiliza dicho *framework* para controlar el navegador *web* Google Chrome por medio de gestos.

Probamos la aplicación con usuarios, y comprobamos que LaGeR es un lenguaje útil y efectivo para representar gestos y desarrollar aplicaciones controladas por medio de ellos.

Palabras clave: *Pointing devices, Virtual reality, Gestural input, Grammars and context-free languages, Publish-subscribe / event-based architectures*

Abstract

The recent boom in virtual reality is fostering the proliferation of gestural input devices such as the Razer Hydra, the Leap Motion 3D Controller, and the Microsoft Kinect. This hinders the work of researchers and application developers, since they must write their programs using a different API for each device.

The main goal of this research was to define and implement a language for the representation and interpretation of bidimensional and tridimensional gestures. Through it, developers will be able to use the language to define gestures to be detected regardless of the device or the underlying APIs.

We investigated different ways of representing and recognizing gestures, and then created LaGeR — *Language for Gesture Representation*. In order to test its feasibility, we wrote a set of software tools and libraries for converting sensor movements into LaGeR strings, recognizing them as gestures, visualizing them, and notifying subscribed programs of their detection. We also created a test program that uses said framework to control the Google Chrome web browser through gestures.

We tested the application with users and showed that LaGeR is a useful and effective language for representing gestures and developing gesture-based applications.

Keywords: *Pointing devices, Virtual reality, Gestural input, Grammars and context-free languages, Publish-subscribe / event-based architectures*

www.tec.ac.cr

TEC | Tecnológico de Costa Rica
Programa de Maestría en Computación
Teléfono: 2550-2402

APROBACIÓN DE LA TESIS

"LaGeR: Lenguaje para descripción de gestos bidimensionales y tridimensionales"

TRIBUNAL EXAMINADOR

Ph.D. Erick Mata Montero
Profesor Asesor

Ph.D. Carlos González Alvarado
Profesor Lector

Ph.D. Jorge Monge Fallas
Profesional Externo

Dr. Roberto Cortés Morales
Coordinador
Programa de Maestría en Computación

Mayo, 2015

Introducción

El resurgimiento de la realidad virtual como método de interacción computacional está llevando a una proliferación cada vez mayor de dispositivos de entrada gestuales bidimensionales y tridimensionales (e.g., Microsoft Kinect, Razer Hydra, Leap Motion 3D Controller). Aún no existe un estándar para dichos dispositivos, por lo que los desarrolladores de software deben adaptar sus programas a los API de cada uno.

La presente investigación propone solucionar este problema por medio de un sistema que le permita a los desarrolladores hacer que sus programas reaccionen a gestos sin tener que preocuparse por el dispositivo de entrada por medio del cual se detectan los gestos.

El resultado será un ahorro considerable de tiempo y dinero al tomar en cuenta el efecto multiplicador del número de aplicaciones relevantes, el número de dispositivos de entrada y el costo de las horas-persona requeridas para soportar cada uno.

El costo de la presente investigación es bastante modesto en relación a sus beneficios, especialmente en el ámbito del software interactivo AAA[1]. Dichos juegos suelen tener un costo de entre 50 y 100 millones de dólares (Superannuation, 2014), por lo que un ahorro de tan solo 0.5 % en *una* de estas aplicaciones superaría la inversión de esta investigación en un orden de magnitud.

Más allá del beneficio a grandes estudios de software, nuestro sistema también facilitará la creación de aplicaciones gestuales por parte de estudiantes de computación, desarrolladores independientes y pequeños estudios.

[1]La designación AAA se refiere a los videojuegos con los presupuestos y nivel de calidad más altos en la industria.

Capítulo 1

Generalidades de la Investigación

1.1. Planteamiento del Problema

En años recientes, los métodos de entrada tradicionales como el mouse, el teclado y el control de videojuegos clásico se han venido complementando con una serie de alternativas novedosas basadas en gestos. Si bien ya existían métodos de entrada gestuales en la academia y la industria (e.g., VPL DataGlove, Immersion CyberGlove), estos eran costosos y de limitada disponibilidad, por lo que su impacto había sido limitado fuera de dichos contextos.

La masificación de los métodos de entrada gestuales para la electrónica de consumo se fue consolidando a través de hitos tales como el lanzamiento del Nintendo Wii con sus controles de movimiento (2006), el Apple iPhone con su pantalla táctil (2007) y el Microsoft Kinect con su cámara tridimensional y rastreo de movimientos (2010), entre otros. Dispositivos tales como el Razer Hydra (2011) y el Leap Motion 3D Controller (2013) han provisto a las computadoras tradicionales de métodos de entrada gestuales de alta precisión a bajo costo.

Aunado a esto, el uso de métodos de entrada gestuales es cada vez mayor. Durante décadas, el mouse y el teclado han sido suficientes para la mayoría de tareas realizadas en computadoras con pantallas bidimensionales. Sin embargo se avizora una revolución en la manera de interactuar con la información digital por medio de la realidad virtual. Las gafas de realidad virtual Oculus Rift están por lanzarse al mercado, y su potencial es tal que a principios del 2014 Facebook compró Oculus por una suma cercana a los $2 mil millones (Welch, 2014). Asumiendo que Facebook logre venderle un Rift al 1 % de sus 1300 millones de usuarios (Hamburger, 2014) en los próximos 5 años, el número de usuarios de realidad virtual sería de al menos 13 millones.

El paradigma de la realidad virtual aún no ha sido explotado a gran escala, por lo que hay una gran variedad de nuevos problemas por resolver. Es dentro de este contexto que cobran protagonismo los métodos de entrada basados en gestos bidimensionales y tridimensionales más cercanos a nuestra interacción con el mundo real.

Cabe destacar que existen varios niveles de interpretación para el concepto de "gestos". El más básico es el de los gestos primarios o primitivas en el espacio tridimensional, las cuales consisten en movimientos de una posición (X, Y, Z) hasta otra posición (X', Y', Z'). Al otro lado del espectro están los gestos como forma de expresión no-verbal de sentimientos humanos: "Movimiento del rostro, de las manos o de otras partes del cuerpo con que se expresan diversos afectos del ánimo." (Real Academia Española, 2012).

La presente investigación se enfocó en los gestos de la interacción humano-computador, los cuales suelen representar un punto intermedio y pueden codificar acciones propias del dominio de un programa informático (e.g., "aumentar", "volver atrás", "rotar").

En dicho contexto, cada programa puede codificar diferentes tipos de acciones con gestos propios, o bien codificar las mismas acciones de manera diferente a otros programas. A su vez, los gestos se obtienen por medio de dispositivos de entrada de varias índoles.

Aunado a esto, el panorama se complica por dos motivos:

1. El interés por la realidad virtual está creciendo aceleradamente, por lo que cada vez hay más compañías anunciando sus propios dispositivos de entrada gestuales,

2. El mercado es tan incipiente que no existen estándares para la gran variedad de métodos de entrada que están surgiendo.

El resultado es que los desarrolladores de sistemas operativos y aplicaciones para realidad virtual se encuentran en la difícil posición de apoyar un subconjunto reducido de gestos y dispositivos – impactando la usabilidad y excluyendo gran parte de sus potenciales usuarios – o invertir grandes cantidades de recursos para apoyar la mayoría de dispositivos. Es por eso indispensable la definición de un lenguaje de descripción de gestos que permita a las demás piezas de software recibir eventos de entrada de forma agnóstica[1] al dispositivo sensor.

1.2. Objetivos

1.2.1. Objetivo General

Definir, implementar y probar un lenguaje para describir y reconocer gestos bidimensionales y tridimensionales de usuario detectados por sensores, de tal manera que el lenguaje pueda ser utilizado para reconocer gestos a nivel de sistema independientemente del tipo de sensor. Dicho lenguaje permitirá que un programa o servicio lo utilice, por ejemplo, a nivel de sistema operativo, o para entregarle eventos de entrada a aplicaciones.

[1] Sin conocimiento de la naturaleza de los componentes con los cuales interactúa

1.2.2. Objetivos Específicos

1. Plantear una forma general de representar el movimiento – y de ser posible, la posición – de dos puntos (ej. dedos índice de las manos de un usuario, posición de un control o puntero, etc.) según los datos proveídos por un sensor de movimiento (ej. touchpad, Leap Motion 3D Controller, Razer Hydra, etc.).

2. Definir un lenguaje que utilice dicha representación para codificar un conjunto de gestos, por medio de una representación vectorial en formato Scalable Vector Graphics (SVG) o por medio de una gramática libre de contexto en Extended Backus-Naur Form (EBNF) cuyas producciones resulten en secuencias de terminales correspondientes a gestos.

3. Identificar al menos una forma de adaptar datos inexactos (e.g., el usuario intenta mover el dispositivo de entrada en línea recta pero no tiene un pulso perfecto) al lenguaje antes descrito.

4. Demostrar la factibilidad y la utilidad del lenguaje por medio de una prueba de concepto. Esta consistirá en la creación de un programa que utilice el lenguaje para codificar y reconocer los gestos. El programa se debe enlazar con el sistema operativo para que otras piezas de software se puedan registrar y recibir eventos correspondientes a gestos específicos.

1.2.3. Hipótesis

Es posible codificar gestos por medio de un lenguaje que represente el movimiento de dos puntos correspondientes a las lecturas de un sensor de entrada gestual en espacio bidimensional y tridimensional. Dicho lenguaje puede ser utilizado por un sistema operativo para proveer eventos de gestos a sus aplicaciones de forma agnóstica al dispositivo de entrada.

Capítulo 2

Revisión de Literatura

2.1. Visión de conjunto

Este capítulo resume la literatura consultada y cómo los resultados de esas investigaciones fueron usados (si lo fueron) en este trabajo para lograr cada objetivo específico.

La sección 2.2 presenta los artículos más relevantes sobre heterogeneidad de las API de sensores, que es un reto importante que buscamos resolver con esta investigación (Objetivo Específico 4). La sección 2.3 se enfoca en artículos que describen diferentes formas de representación de movimientos, sentando los cimientos para el resto de la investigación (Objetivo Específico 1). La sección 2.4 presenta artículos acerca de lenguajes específicos, lo cual fue útil para definir nuestro propio lenguaje (Objetivo Específico 2). La sección 2.5 investiga artículos sobre soluciones de software y hardware que nos ayudaron a crear una prueba de concepto con la capacidad de describir y reconocer gestos a partir de los movimientos inexactos de un usuario (Objetivos Específicos 3 y 4).

2.2. Heterogeneidad de API de sensores

Actualmente, cada sensor de entrada provee su propio método para obtener y utilizar sus datos. Por ejemplo, Microsoft provee un Software Development Kit (SDK) para el Kinect que permite crear aplicaciones que se enlazan con el dispositivo por medio de un API que expone tanto gestos y posturas predefinidas como datos crudos para mayor control (Microsoft Corporation, 2014). También existe una gran variedad de herramientas tanto gratuitas (ej. Kinetic Space (Wölfel, 2012)) como de pago (ej. GesturePak (Franklin, 2012)) que ofrecen funcionalidad avanzada tal como el entrenamiento de reconocedores de gestos personalizados. Por su parte, Sixsense provee un SDK que permite crear juegos y otras aplicaciones que utilizan la posición (X, Y, Z) y orientación (*pitch, roll, yaw* — cabeceo, alabeo, guiño) de los controles del Razer Hydra. A su vez Leap Motion provee un SDK (Leap Motion, Inc, 2014) que provee una nube de puntos tridimensional de lo detectado por la cámara de su dispositivo así como un modelo esquelético de las manos del usuario.

La anterior no es una lista completa de todos los dispositivos de entrada gestuales y las interfaces que proveen. Lo importante de observar es que cada uno de esos dispositivos provee sus datos de una forma distinta. Si un desarrollador escribe su aplicación para que detecte gestos manuales a través del SDK de Microsoft, sus usuarios solamente podrán utilizarla con un Kinect. En caso de que desee

soportar a los usuarios del Leap Motion 3D Controller, deberá re-escribir parte de su aplicación o reestructurarla de modo que utilice plugins específicos para el SDK de cada dispositivo.

2.3. Representación y análisis de movimientos

El objetivo general de nuestra investigación fue encontrar una forma de evitar estos problemas por medio de un lenguaje de descripción de gestos, lo suficientemente general como para poder proveer una biblioteca que sea agnóstica al dispositivo de entrada. Afortunadamente, ha habido un corpus interesante de investigación relacionada.

Nuestro primer objetivo específico fue plantear una forma general de representar el movimiento — y de ser posible, la posición — de dos puntos (ej. dedos índice de las manos de un usuario). Un buen punto de partida fue "Gesture Recognition: A Survey" (Mitra & Acharya, 2007), el cual presenta un sondeo muy completo de técnicas para el reconocimiento de gestos.

Más específicamente, (Lee, Cakmak & DePalma, 2008) comparó una serie de métodos de aprendizaje en su "Gesture Recognition with Temporally Local to Global Representations", entre los cuales se destaca el uso de máquinas vectoriales de apoyo. Estas clasifican cuadros de imágenes representadas por características visuales simples, lo cual consideramos adaptar a nuestras necesidades para detectar primitivas de movimientos. Sin embargo, decidimos en contra de este enfoque puesto que se basa en técnicas que dependen de imágenes. Si bien sería útil para las entradas de sensores ópticos como el Kinect o el Leap Motion, no funcionaría con dispositivos directamente manipulables (e.g., Razer Hydra).

Sin embargo, el concepto de analizar imágenes de video cuadro por cuadro nos llevó a pensar en la discretización de los datos del sensor, definiendo un espacio vectorial en R3 donde el sensor describiría la posición de puntos con una periodicidad constante. Luego obtendríamos una secuencia con la diferencia entre cada uno de los puntos, y asignaríamos terminales de nuestro lenguaje a cada posible dirección de los vectores en x, y, z. Por ejemplo un movimiento diagonal arriba-derecha podría ser representado por (+x+y)*. Partiendo de dicha representación pensamos en aprovechar la investigación de máquinas de estado finito para reconocimiento de gestos manuales (Mitra & Acharya, 2007), donde los gestos se representan como una serie de movimientos correspondientes a estados (ej. comienzo/arriba/abajo/izquierda/derecha). Este habría sido un enfoque válido, pero con

una representación demasiado primitiva como para representar gestos complejos de forma económica.

Un desafío adicional que surgió de este enfoque fue el de dividir los gestos de un solo trazo en varios componentes significativos. Para esto nos referimos a "Iterative Incremental Clustering of Time Series" (Lin, Vlachos, Keogh & Gunopulos, 2004), el cual describe un novedoso algoritmo de partición de *clusters* (cúmulos). Dicho algoritmo podría ser útil para discretizar una serie de tiempo de movimientos detectados por el sensor, definiendo k *clusters* a ser identificados.

Los clusters podrían corresponder a puntos de inflexión en la trayectoria de un dedo, por ejemplo. Cuando el dedo va en línea recta se mueve rápidamente, pero cuando va a cambiar de dirección sufre una desaceleración de la trayectoria actual y una aceleración para tomar la nueva trayectoria. Esta transición se correlaciona con una densidad más alta de puntos por unidad de tiempo, la cual identificaríamos como un *cluster*.

Cabe destacar que la implementación final de esta investigación no utilizó *clustering*. Esto porque descubrimos que los gestos podían ser descritos y reconocidos de forma efectiva como una sola unidad, lo cual simplificó el sistema considerablemente. Con la implementación actual, pudimos distinguir gestos consistentemente con un nivel de correspondencia de ∼70-80 % (ver Capítulo 6).

Otro pilar teórico que investigamos es el *paper* original de Dynamic Time Warping (DTW) (Ratanamahatana & Keogh, 2004). En él se investigan las series de tiempo y se describe cómo compararlas con DTW. Entre otras cosas, se considera la comparación de series de diferente tamaño, lo cual podría ser útil a la hora de comparar gestos de formas equivalentes pero con magnitudes diferentes. Se consideró el uso de estas técnicas para para convertir las entradas de los sensores a datos discretizados, los cuales posteriormente convertiríamos a una representación o lenguaje estandarizado. Si bien la implementación final lleva a cabo el reconocimiento directamente a partir de hileras en LaGeR, es posible que el uso de técnicas avanzadas de DTW permita mejorar el desempeño en un trabajo futuro.

Por último, consideramos la posibilidad de representar y comparar los gestos directamente a partir de su representación en hileras de LaGeR. Primero, los movimientos del sensor se convertirían en una hilera de LaGeR, la cual se compararía con las hileras de los gestos almacenados. Seguidamente, la hilera con menor distancia determinaría el gesto reconocido. Para este fin, investigamos el algoritmo de distancia de hileras Damerau-Levenshtein, el cual cuenta el número de operaciones de inserción, borrado, sustitución y transposición de caracteres necesario

para convertir una hilera en otra. Dicho algoritmo ha sido utilizado exitosamente para diversas aplicaciones tales como la corrección ortográfica (Bard, 2007) y la medición de variaciones en ADN (Majorek et al., 2014). En nuestro caso, el tipo de operaciones tomadas en cuenta por Damerau-Levenshtein se adecuó satisfactoriamente a las diferencias que pueden existir entre una hilera almacenada y una producida por un ser humano sosteniendo un sensor.

2.4. Lenguajes

Nuestro segundo objetivo específico fue la definición del lenguaje. Una de las formas de describir los gestos que se analizó fue por medio de la especificación 1.1 de SVG (Dengler et al., 2011). Esta provee un elemento *path* (camino) que nos permitiría representar el contorno de un polígono por medio de caminos. Sin embargo, tiene la limitación importante de que no permite representar figuras tridimensionales.

Posteriormente, consideramos el *paper* de Hachaj y Ogiela titulado "Rule-based approach to recognizing human body poses and gestures in real time" (Hachaj & Ogiela, 2014), el cual propone un clasificador de poses y gestos corporales que almacena las reglas de una base de conocimiento en *scripts* especiales utilizando un "Gesture Description Language" (Lenguaje de Descripción de Gestos) reconocido por un parser basado en una gramática libre de contexto LALR-1. Dicho lenguaje contempla gestos tridimensionales, pero es específico para poses esqueléticas de cuerpo entero (e.g., codo derecho, cuello, etc.), por lo que no es lo suficientemente general como para describir gestos arbitrarios tales como el movimiento circular de un dedo índice. Sin embargo, la formalización del GDL en sí en el apéndice del *paper* proveyó un buen punto de partida a la hora de definir nuestro propio lenguaje.

Otro de nuestros objetivos fue identificar una forma de lidiar con datos inexactos. Hay varios antecedentes en esta área que lo abordan desde el punto de vista del lenguaje, como por ejemplo el *paper* "Hand Gesture Recognition Using Haar-Like Features and a Stochastic Context-Free Grammar" (Chen, Georganas & Petriu, 2008). El uso de características pseudo-Haar se limita al reconocimiento de imágenes, por lo que su aplicabilidad a nuestra investigación se vio limitada por el mismo motivo que (Lee et al., 2008). Además, la gramática de (Chen et al., 2008) se enfoca en gestos a nivel de las manos, por lo que seguía produciendo un lenguaje demasiado específico para nuestras necesidades. Sin embargo, el concepto de una

gramática libre de contexto inherentemente estocástica como tal podría ser objeto de estudios futuros para mejorar o complementar nuestra investigación.

Adicionalmente, analizamos el enfoque planteado por (Han, Everding & Wee, 2004) en "Graph Matching for Object Recognition and Recovery" y (Bai, Yang, Yu & Latecki, 2008) en "Skeleton-Based Shape Classification Using Path Similarity", el cual permite clasificar y comparar formas por medio de esqueletos simplificados de sus contornos en forma de gráficos, árboles o hileras. Estas lecturas se complementaron con "Curves vs Skeletons in Object Recognition" (Sebastian & Kimia, 2005), donde se comparan los esqueletos con otras maneras de representar formas, tomando en cuenta su desempeño. Dichos enfoques son interesantes por su elegancia conceptual, pero tienen la limitante de que solo tomarían en cuenta la forma final de un gesto, sin importar el orden en que fue formado.

2.5. Implementación

2.5.1. Software

Por último, el cuarto objetivo específico de esta investigación fue implementar una prueba de concepto. Para esto teníamos que crear una biblioteca e integrarla con un sistema operativo. Inicialmente consideramos integrarla con el servidor de despliegue Wayland de manera tal que las aplicaciones pudieran registrarse para ser notificadas con eventos para gestos específicos. El año pasado Forrest Reiling del California Polytechnic State University terminó su tesis "Toward General Purpose 3D User Interfaces: Extending Windowing Systems to Three Dimensions" (Reiling, 2014), la cual requirió este mismo tipo de integración. Estuvimos en contacto con Reiling y, aunque se mostró anuente a colaborar con nuestra investigación, finalmente optamos por una integración basada en el servidor de despliegue X.Org, dado que su uso es prácticamente universal en las distribuciones de Linux, mientras que el de Wayland aún es incipiente.

Este tipo de software, también conocido como *middleware* , sería la "goma" responsable de mediar entre los sensores de entrada y las aplicaciones del sistema. Actualmente existen ejemplos de *middleware* con este mismo propósito, como por ejemplo MiddleVR. MiddleVR es un *plugin* genérico de realidad virtual que permite que aplicaciones 3D utilicen diferentes dispositivos de entrada tridimensionales tales como el Microsoft Kinect, el Razer Hydra, el Leap Motion y el TrackIR, entre otros (MiddleVR, 2015c). Una de sus desventajas es que por el momento

solo está disponible al público en la forma de un *plugin* para el motor de videojuegos Unity. Además la versión gratuita obliga a incluir una marca de agua en los programas, mientras que las demás versiones no son de fácil acceso para el público en general, dado que solo se pueden obtener directamente del fabricante y su precio no es divulgado públicamente (MiddleVR, 2015a). Sin embargo, para efectos de esta investigación, la desventaja más grande de MiddleVR es que no soporta la definición y análisis de gestos. Si bien su creador ha expresado el deseo de implementar dicha funcionalidad (Boger, 2013), en la actualidad solo se soporta el reporte de coordenadas y movimiento de ejes de los dispositivos de entrada (MiddleVR, 2015b). Recientemente contactamos a Sebastien Kuntz — creador de MiddleVR — para conocer sus planes a futuro, y nos confirmó que "no [añadieron] ningún reconocimiento de gestos en MiddleVR" (Kuntz, 2014).

Buena parte del soporte a dispositivos provisto por MiddleVR proviene de la biblioteca VRPN (*Virtual-Reality Peripheral Network*). VRPN "es un conjunto de clases adentro de una biblioteca y un conjunto de servidores diseñados para implementar una interfaz transparente a la red entre programas de aplicaciones y el conjunto de dispositivos físicos (rastreador etc.) utilizados en un sistema de realidad virtual (VR)" (Taylor II, Yang, Weber & Hudson, 2014a). Tal y como en el caso de MiddleVR, el problema con VRPN es que no le permite a las aplicaciones definir y tener acceso a la detección de gestos, sino que se limita a proveer datos de rastreo (Taylor II, Yang, Weber & Hudson, 2014b).

No obstante las limitaciones de VRPN, sus características fueron muy útiles para construir nuestra propio sistema sobre él. Una de sus principales ventajas es que es de código abierto y gratuito, lo cual nos facilitó modificarlo e integrarlo en nuestra solución. La otra ventaja es que automáticamente nos permitió soportar más de 50 dispositivos de entrada distintos — además de ratones y teclados en general — sin tener que escribir código específico para el *driver* o SDK de cada dispositivo (Taylor II et al., 2014a). Por medio del objeto de aplicación vrpn_Tracker_Remote, tuvimos acceso a datos con marca de tiempo (*"timestamped"*) de posición y orientación para casi cualquier dispositivo de entrada conectado a la computadora (Taylor II, Yang, Weber & Hudson, 2014c).

Luego de obtener los datos crudos de un sensor a través de VRPN, es necesario suavizarlos con el fin de eliminar fuentes de error tales como ruido y *jitter* (fluctuaciones). Para este fin, VRPN provee una implementación del 1€ Filter (Casiez, Roussel & Vogel, 2012), un filtro de paso bajo de frecuencia adaptativa. A bajas velocidades, utiliza una frecuencia de corte baja para reducir *jitter*, mientras que a

altas velocidades eleva la frecuencia de corte para reducir *lag* (retraso). De acuerdo con Casiez et al, este filtro no solo es simple de implementar y de ajustar, sino que también resulta en un menor grado de *lag* en comparación a filtros tales como el reconocido Kalman. Por estas razones, el 1€ Filter fue de gran utilidad para pre-procesar los datos del sensor.

Por último, uno de los antecedentes que más se acercan a nuestra visión de lo que debe producir esta investigación es el *paper* reciente de Lu, Fogarty y Li titulado "Gesture Script: Recognizing Gestures and their Structure using Rendering Scripts and Interactively Trained Parts" (Lü, Fogarty & Li, 2014). En él se presenta un sistema para reconocimiento de gestos que permite entrenar primitivas (ej. círculo, arco, línea) que luego se secuencian en scripts para describir gestos de un solo trazo. Por ejemplo, una flecha podría describirse como una línea recta seguida de una cabeza triangular — *draw(Line), draw(Head)* — o bien como una secuencia de cuatro líneas rectas — *draw(Line), draw(Line), draw(Line), draw(Line).*

Una de las desventajas del sistema de Lü et al es que corre sobre una aplicación de alto nivel en Java, la cual requiere de entrenamiento manual en vivo por parte de los desarrolladores para definir las primitivas. Esto porque los literales del lenguaje (círculo, línea, etc.) no tienen un significado específico sino que dependen del entrenamiento y el etiquetado que se les de durante el entrenamiento. En otras palabras, existe un estrecho acoplamiento entre la semántica del lenguaje y el entrenamiento de un reconocedor.

En contraposición a eso, nuestro sistema busca desacoplar el lenguaje del reconocimiento, y de esta manera permitir la definición de gestos de forma más explícita y precisa. No obstante, es notable la facilidad que Gesture Script provee a la hora de definir los gestos, puesto que puede tornarse difícil visualizar mentalmente y describir gestos de manera puramente simbólica. Esto nos motivó a crear lager_viewer, una herramienta para visualizar los gestos correspondientes a hileras de LaGeR (ver Subsección 5.4.9.2). El uso de medios gráficos para *definir* los gestos podría ser un enfoque valioso para trabajos futuros.

2.5.2. Hardware

Dado que decidimos utilizar VRPN como *middleware* (ver Subsección 2.5.1), la elección de dispositivo de entrada no estuvo limitada por la calidad del SDK de los dispositivos, sino por consideraciones de costo, conveniencia y funcionalidad. Fue por eso que elegimos el Razer Hydra como dispositivo de entrada para llevar a cabo el desarrollo y las pruebas de concepto con las que validamos la utilidad

de LaGeR. Dicho sistema consiste de una base a la que se conectan dos *wands* o sensores en forma de barra (Figura 2.1). El usuario sujeta ambos sensores con sus manos y los puede mover libremente en el espacio tridimensional, teniendo acceso además a *joysticks* de pulgar, gatillos y varios botones (Figura 2.2). El dispositivo rastrea los movimientos de las barras emitiendo campos magnéticos a partir de tres bobinas en la esfera central de la base (Figura 2.3), los cuales son detectados por tres bobinas en cada sensor (Figura 2.4). Las bobinas de emisión y recepción están instaladas en orientaciones ortogonales, lo cual permite obtener lecturas para cada uno de los ejes X, Y y Z (Feiner, 2014).

FIGURA 2.1: Razer Hydra (Hardware.Info, 2011)

La escogencia del Razer Hydra se dio por varios motivos:

- **Exactitud:** El sistema de rastreo del Razer Hydra permite rastrear la posición y orientación absoluta de sus sensores con exactitud de 1 mm y 1 °, respectivamente. Además, no sufre de problemas de oclusión de linea de vista como es el caso de dispositivos ópticos como el Microsoft Kinect (Sixsense Entertainment, Inc., 2012). Si bien su precisión puede verse afectada por interferencias magnéticas, esto se puede subsanar por medio de algoritmos de filtrado tales como el 1€ Filter (Casiez et al., 2012).

- **Costo:** Al comienzo de esta investigación, pudimos obtener un Razer Hydra nuevo por tan solo $40. Esto abarató los costos de la investigación, a la vez que facilitaría que otros investigadores pudieran replicar nuestros resultados. Si bien el Hydra fue descontinuado recientemente, la empresa que lo diseñó planea lanzar un sucesor en el 2015. El costo será mayor, pero tendrá ventajas

FIGURA 2.2: Los dos sensores del Razer Hydra en las manos de un usuario. (Castle, 2011)

FIGURA 2.3: Bobinas en la base del Razer Hydra (Matthews, 2013)

importantes como sensores inalámbricos y mucha mayor precisión y exactitud (Sixsense Entertainment, Inc., 2014).

- **Popularidad:** Dadas sus características y bajo costo, el Razer Hydra ha sido considerado el estándar *de facto* en dispositivos de entrada del reciente *boom* de la realidad virtual (Lang, 2013).

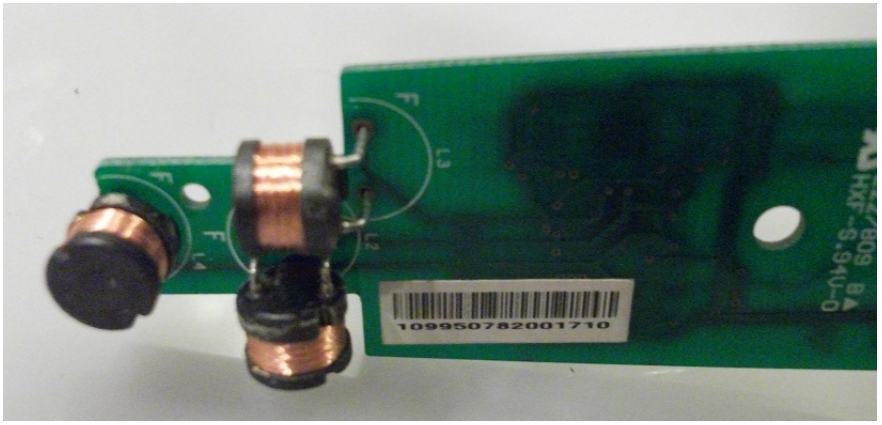


FIGURA 2.4: Bobinas en un sensor del Razer Hydra (WiredEarp, 2011)

- **Escenarios de uso:** El Razer Hydra puede ser utilizado cómodamente en un escritorio o en los regazos frente a un monitor, mientras que un dispositivo óptico como el Microsoft Kinect requiere que el usuario esté a una distancia de al menos 137 cm de la cámara (Chacksfield, 2013).

Capítulo 3

Marco Metodológico

3.1. Metodología

La presente investigación contiene un componente teórico y uno práctico. Por lo tanto, la metodología cubre la formulación de soluciones de ambas índoles para validar la hipótesis.

En primer lugar, deseábamos definir un lenguaje para representar los movimientos de puntos detectados por un sensor. Para ello analizamos el SDK del Razer Hydra, el Leap Motion y el Microsoft Kinect, así como las soluciones de *middleware*[1] VRPN y MiddleVR para encontrar la forma más simple y flexible de integrar dispositivos de entrada. Una vez definida, estudiamos el formato de salida con el que representa los movimientos (e.g., tira de posiciones en el tiempo). Luego analizamos la literatura para encontrar un lenguaje que pudiéramos reaprovechar o utilizar como base para definir un lenguaje nuevo que represente gestos formados a partir de esos movimientos.

Seguidamente llevamos a cabo una prueba de concepto. Esta consistió de los siguientes elementos:

1. Creamos un programa en `C++` (*lager_viewer*) que convierte hileras de LaGeR en representaciones gráficas de gestos por medio de OpenGL. Esta se utilizó para facilitar la depuración de los demás componentes, y posteriormente será de utilidad para que los desarrolladores de aplicaciones la utilicen como herramienta de visualización a la hora de definir y ajustar gestos en LaGeR.

2. Utilizamos el navegador de poliedros Great Stella para crear una representación plana de un rombicuboctaedro. Esta podrá ser impresa y plegada para formar un modelo físico tridimensional con los literales de LaGeR (a-z) en las caras correspondientes a sus ángulos ϕ y θ. Dicho modelo facilitará la visualización de LaGeR para desarrolladores y usuarios finales a la hora de describir o llevar a cabo gestos en LaGeR.

3. Creamos una biblioteca en `C++` (*lager_convert*) que provea una clase para tomar los datos provistos por un Razer Hydra por medio de VRPN y codificarlos en hileras de LaGeR.

4. Creamos una biblioteca en `C++` (*lager_recognize*) que provea una clase para tomar una hilera de LaGeR y compararla con un conjunto de hileras almacenadas, devolviendo el gesto reconocido más cercano.

[1] Capa de software que conecta componentes de software o aplicaciones entre sí

5. Creamos una biblioteca en `C++` (*lager_connect*) que provea clases y funciones para el registro de gestos y notificación de detecciones a suscriptores por medio de colas de mensajes.

6. Creamos un programa en `C++` (*lager_recognizer*) que utilice las bibliotecas *lager_convert* y *lager_recognize* para convertir movimientos de un Razer Hydra en LaGeR y reconocer un conjunto de gestos predeterminados.

7. Convertimos *lager_recognizer* en un *daemon* (demonio) del sistema operativo Linux. El demonio utiliza la biblioteca *lager_connect* para escuchar una cola de mensajes por medio de la cual permite que otras piezas de software se registren para recibir eventos de entrada para los gestos detectados.

8. Creamos un programa en `C++` (*lager_gesture_manager*) que permita administrar un archivo de definición de gestos. Permite añadir, remover y editar gestos por medio de hileras de LaGeR o por medio de sensores de movimiento, utilizando la biblioteca *lager_convert*.

9. Creamos una aplicación de prueba en `C++` (*lager_injector*) que utilice la biblioteca *lager_connect* para comunicarse con *lager_recognizer* y así enlazar uno o más gestos con una de sus funciones. Dichos gestos son definidos por la aplicación utilizando las hileras de LaGeR contenidas en un archivo. Los gestos hacen que el programa inyecte pulsaciones de teclado al sistema operativo por medio del *X Window System*.

10. Llevamos a cabo pruebas con un conjunto predeterminado de gestos por medio de un Razer Hydra conectado a nuestro sistema, para así comprobar que estos fueran codificados correctamente en el lenguaje y que la aplicación recibiera las notificaciones correspondientes.

    La primera prueba — "Prueba 1" — tuvo como objetivo comprobar el funcionamiento básico del sistema en condiciones controladas. Para ello, definimos diez gestos primitivos (compuestos de formas básicas) y le pedimos a un usuario que los reprodujera con el dispositivo de entrada. Únicamente evaluamos la tasa de éxito de reconocimiento del sistema. Los gestos se evaluaron en el siguiente orden:

    - línea horizontal
    - línea vertical
    - triángulo

- cuadrado
- círculo
- línea hacia adelante
- línea hacia atrás
- apartar dos sensores simultáneamente
- juntar dos sensores simultáneamente

La segunda prueba — "Prueba 2" — se enfocó en replicar condiciones más realistas para evaluar la utilidad del sistema como un todo. En este caso, se definió un conjunto de gestos más complejos correspondientes a acciones comunes para navegar por la Web a través de Google Chrome. Además de utilizar el reconocedor, lo enlazamos a Google Chrome para que el usuario realmente pudiese controlar el navegador y ver el resultado concreto de sus acciones en pantalla. La prueba se estructuró como diez repeticiones de la siguiente secuencia de acciones:

- Abrir Google Chrome (trazar G mayúscula en el aire)
- Abrir nueva pestaña (trazar T mayúscula en el aire)
- Cargar CNN (trazar CNN en mayúsculas en el aire)
- *Zoom in* (apartar sensores horizontalmente)
- *Zoom out* (juntar sensores horizontalmente)
- Refrescar página (trazar círculo en dirección de las manecillas del reloj en el aire)
- Cerrar pestaña (trazar X mayúscula en el aire)
- Abrir nueva pestaña (trazar T mayúscula en el aire)
- Cargar Google (trazar G mayúscula en el aire)
- Cerrar pestaña (trazar X mayúscula en el aire)

Ambas pruebas se llevaron a cabo con un usuario novato ajeno al investigador, a fin de evitar sesgos causados por efectos de entrenamiento y adaptación. Los detalles de implementación de dichas pruebas se encuentran en la Subsección 5.4.10.

Capítulo 4

El Lenguaje LaGeR

4.1. Introducción

Este capítulo describe el lenguaje LaGeR, tanto desde el punto de vista sintáctico como semántico y pragmático. La Sección 4.2 presenta una discusión sobre el origen conceptual del lenguaje, a partir de la definición de movimientos en 2D con la tradicional rosa de los vientos. La Sección 4.3 presenta la sintaxis de LaGeR por medio de una GLC (gramática libre de contexto). LaGeR es un lenguaje regular, pero se presenta una GLC para que sea más compacta y fácil de leer. La Sección 4.4 resume la semántica de LaGeR a partir de la discusión de la Sección 4.2 y guiados por la sintaxis presentada en la Sección 4.3. La Sección Sección 4.5 presenta aspectos pragmáticos de LaGeR. En particular, se describe cómo se enfrenta la invariabilidad de escala en los movimientos, la tolerancia a gestos imprecisos, y la invariabilidad rotativa de gestos cerrados.

4.2. Origen conceptual del lenguaje

Nuestro lenguaje se definirá en función de su objetivo: la representación de gestos producto de sensores de entrada bidimensionales y tridimensionales.

Por lo tanto, partimos de la representación de los datos provistos por la biblioteca VRPN. Esta representa los movimientos de uno o más punteros como una secuencia de coordenadas cartesianas en R^3.

> Ej.
> [x, y, z]
> (0, 0, 0) → (1, 0, 0) → (1, 1, 0)

Esto correspondería a un vector de magnitud 1 hacia la derecha, seguido por un vector de magnitud 1 hacia adelante.

Si pensáramos puramente en términos bidimensionales, podríamos representar una aproximación de los movimientos por medio de 8 vectores en intervalos de 45°, algo similar a un compás o "rosa de los vientos" con norte, noreste, este, sureste, sur, suroeste, oeste y noroeste (Figura 4.1).

Si expandimos el concepto a una esfera en el espacio tridimensional, obtenemos 26 direcciones que podemos ver como flechas que apuntan desde el núcleo de un rombicuboctaedro hacia el centro de cada una de sus caras (Figura 4.2).

Cada una de estas direcciones se puede representar por medio de coordenadas esféricas $\{r, \theta, \phi\}$ donde r es el radio, θ el ángulo polar, y ϕ el ángulo azimutal (Figura 4.3).

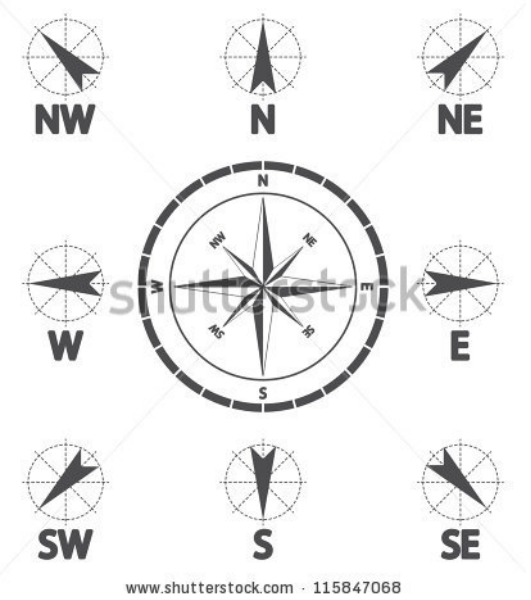


FIGURA 4.1: Rosa de los vientos (Shutterstock, 2014)

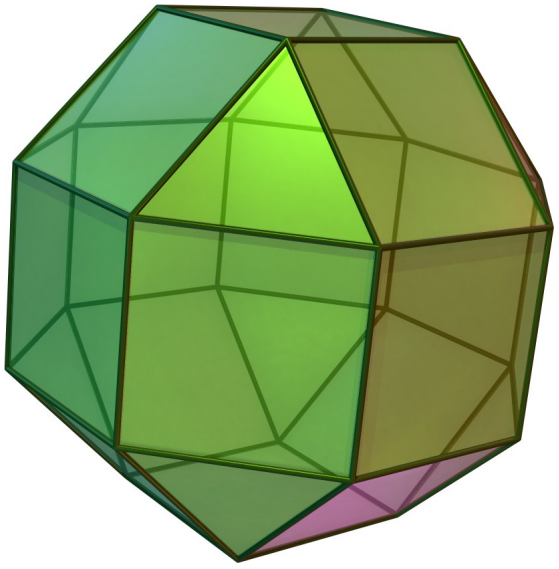


FIGURA 4.2: Rombicuboctaedro (Wikipedia, s.f.-a)

4.2.1. Literales de movimiento

En el caso de LaGeR, representamos los gestos como una secuencia de movimientos normalizados en cuanto a magnitud, velocidad y duración. Dada la normalización, omitimos el radio de las coordenadas esféricas y utilizamos únicamente los ángulos θ y ϕ para describir los movimientos (Figura 4.4).

Coincidentemente, el abecedario inglés consiste de 26 letras, las cuales convenientemente utilizamos para representar cada una de las direcciones (Figura 4.5).

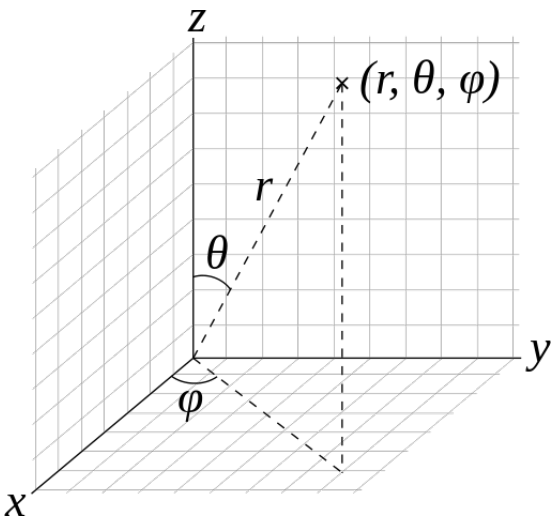


FIGURA 4.3: Coordenadas esféricas (Wikipedia, s.f.-b)

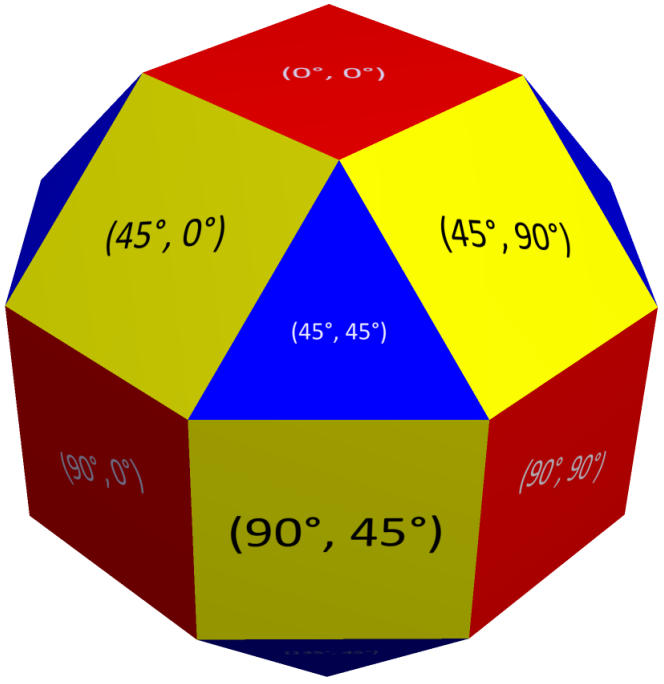


FIGURA 4.4: Coordenadas esféricas en caras de rombicuboctaedro

El polo norte lo etiquetamos con el literal 'a', y las demás caras de arriba con 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i' (Figura 4.6).

Las caras del medio las etiquetamos con los literales 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q' (Figura 4.7)

Por último, etiquetamos las caras de abajo con los literales 'r', 's', 't', 'u', 'v', 'w', 'x', 'y' y el polo sur con el literal 'z' (Figura 4.8).

Dicho etiquetado de direcciones lo podemos resumir por medio de la tabla en el Apéndice A.

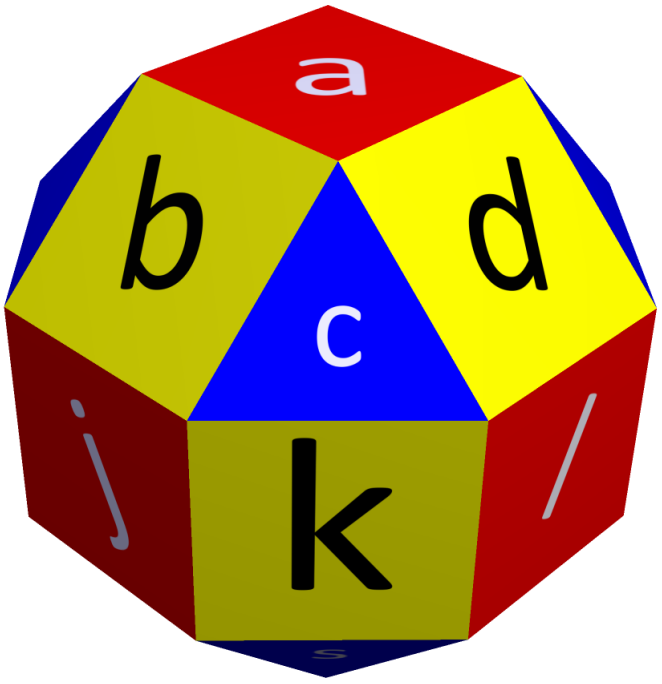


FIGURA 4.5: Literales de LaGeR en caras de rombicuboctaedro

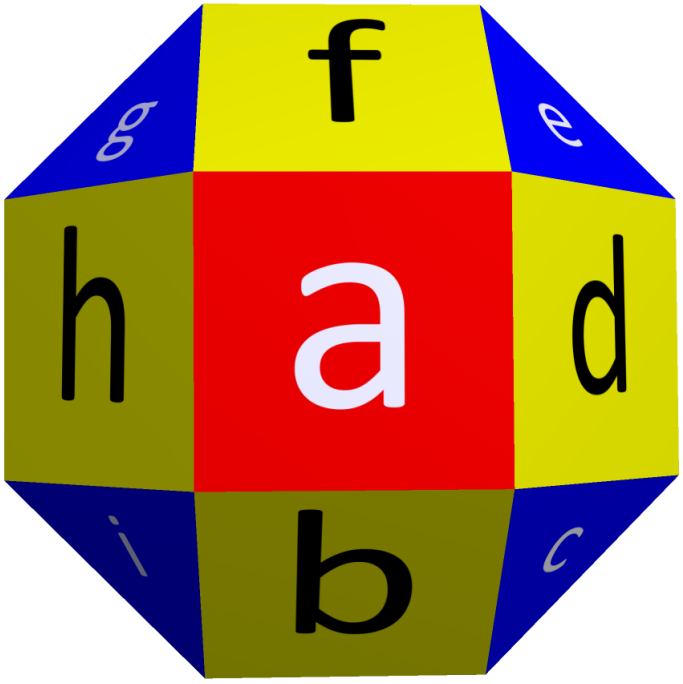


FIGURA 4.6: Literales de LaGeR en caras de arriba de un rombicuboctaedro

4.2.2. Literal de agrupamiento

Además de los literales de movimiento, LaGeR utiliza el literal punto ('.'). Este tiene la función de agrupar los movimientos de los sensores por intervalos de tiempo, en orden ascendente de sensor. De esta manera, podemos representar el

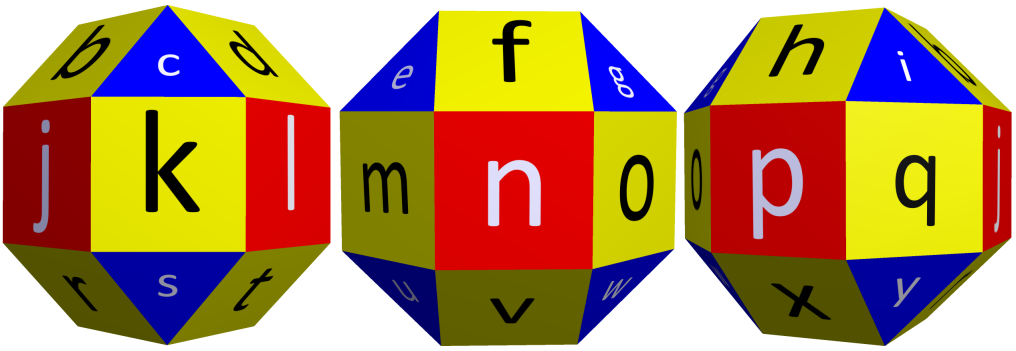


FIGURA 4.7: Literales de LaGeR en caras del lado de un rombicuboctaedro

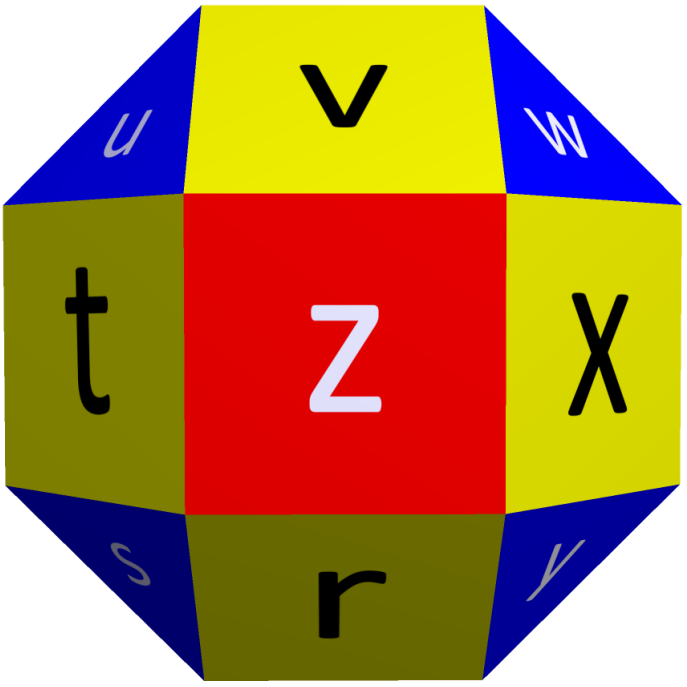


FIGURA 4.8: Literales de LaGeR en caras de abajo de un rombicuboctaedro

movimiento de dos sensores simultáneamente escribiendo el literal correspondiente al movimiento del sensor 0, seguido del literal correspondiente al movimiento del sensor 1, terminando con un punto. El punto es necesario dado que el número de sensores puede ser variable; de lo contrario podríamos haberlo omitido del lenguaje.

Por ejemplo, imaginemos que deseamos representar el movimiento de dos sensores en línea recta en direcciones opuestas. Este movimiento comúnmente se conoce como *pinch-to-zoom*, o "pellizcar para ampliar". Para ello, podemos mover el sensor izquierdo hacia la izquierda (en la dirección $(90°, 270°)$), y el sensor derecho hacia la derecha (en la dirección $(90°, 90°)$). Dichas direcciones corresponden a los literales de LaGeR 'p' y 'l', respectivamente. La representación del gesto resultante es por lo tanto pl.pl.pl.pl.pl.pl.pl.pl. (ver Figura 4.9).

FIGURA 4.9: *Pinch-to-zoom* representado por la hilera pl.pl.pl.pl.pl.pl.pl.pl. (visualización simplificada)

4.2.3. Literal de inmovilidad

El conjunto de literales de LaGeR lo completa el guión bajo, o *underscore*, '_'. En caso de que no haya habido movimiento de uno de los sensores en un intervalo de tiempo dado, representamos dicha falta por medio del '_'.

Por ejemplo, imaginemos que un usuario mueve el sensor izquierdo de un Razer Hydra una unidad hacia la derecha y dos hacia el frente (Figura 4.10). En LaGeR, el movimiento hacia la derecha correspondiente a la dirección $(90°, 90°)$ es 'l', y el movimiento hacia el frente correspondiente a la dirección $(90°, 180°)$ es 'n'. El sensor derecho permanece inmóvil, lo cual se representa con el literal '_'. Por lo tanto, en LaGeR el gesto antes descrito se expresa por medio de la hilera l_.n_.n_. (ver Figura 4.10).

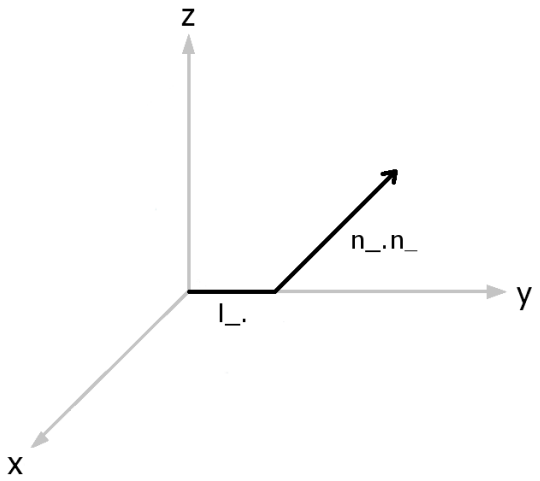


FIGURA 4.10: Flecha representada por la hilera l_.n_.n_.

4.3. Sintaxis de LaGeR

La siguiente gramática libre de contexto define la sintaxis del lenguaje LaGeR. Como se indicó anteriormente, LaGeR es un lenguaje regular, pero se presenta una gramática libre de contexto para que sea más compacta:

$G_{LaGeR} = (N_{LaGeR}, T_{LaGeR}, P_{LaGeR}, < gesto >)$

donde
$N_{LaGeR} = < movimiento >, < movimientos >, < grupo >, < gesto >$
$T_{LaGeR} = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, ., \_, \#\}$

y las producciones de P_{LaGeR} son:
$< gesto > \rightarrow < grupo > \# \mid < grupo >< gesto >$
$< grupo > \rightarrow < movimientos > .$
$< movimientos > \rightarrow < movimiento > \mid < movimiento >< movimientos >$
$< movimiento > \rightarrow a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|\_$

4.4. Semántica de LaGeR

La semántica de LaGeR es bastante simple. En esta sección la resumimos por medio de una descripción guiada por su sintaxis.

$< movimiento >$ corresponde a un vector que apunta desde el núcleo de un rombicuboctaedro hacia el centro de cada una de sus 26 caras (ver Figuras 4.5, 4.6, 4.7 y 4.8). El terminal "_" denota la ausencia de movimiento. Cada $< movimiento >$ está asociado a un sensor.

$< movimientos >$ es una secuencia de una o más instancias de $< movimiento >$ que ocurren al mismo tiempo [1]. En esta investigación, se usaron dos sensores, por lo cual $< movimientos >$ siempre consiste de dos movimientos.

$< grupo >$ tiene el mismo sentido que $< movimientos >$. Su diferencia es el marcador sintáctico ".".

$< gesto >$ corresponde a una secuencia de $< movimientos >$ que han ocurrido en orden cronológico ascendente y se espera que tengan unidad gestual, es decir, que correspondan a uno de los gestos que el intérprete de LaGeR comprende. En esta investigación, el marcador de final de un gesto puede ser temporal (ha pasado suficiente tiempo sin que el usuario haga más movimientos) o táctil (el usuario ha apretado un botón del dispositivo de entrada). En ambos casos, se añade un marcador de fin de gesto a la hilera: "#".

[1] Dicho agrupamiento se determina según el Algoritmo 2 en la Sección 5.2

4.5. Aspectos pragmáticos

4.5.1. Invariabilidad de escala

Una de las propiedades deseables de un sistema informático que utilice LaGeR para representar gestos es la invariabilidad de escala. Gestos como los de la Figura 4.11 deberían ser equivalentes en la mayoría de los escenarios de reconocimiento. De lo contrario, un usuario no solo tendría que llevar a cabo un gesto con la forma correcta sino también adivinar el tamaño exacto que el programa espera recibir.

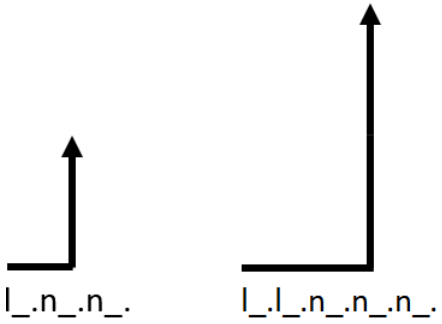


FIGURA 4.11: Gestos equivalentes (visualización simplificada)

Inicialmente, pensamos en solucionar este problema a nivel del lenguaje: LaGeR siempre tendría que representar los gestos en su forma más compacta posible. Para ello, llevaríamos a cabo una especie de reducción utilizando el máximo común divisor del número de repeticiones consecutivas de todos los literales en la hilera. Por ejemplo, dividiríamos el número de repeticiones en "l_.l_.n_.n_.n_.n_." entre 2 y obtendríamos "l_.n_.n_.".

Sin embargo, descubrimos que este enfoque tiene tres problemas importantes.

Primero, significa mezclar detalles del diseño del lenguaje con detalles de la implementación del sistema de reconocimiento de gestos. El diseño del lenguaje debe definirse en función de su poder para representar gestos de forma efectiva y portátil, no necesariamente en función de un método de reconocimiento en particular.

Segundo, implica limitar de forma artificial el poder de expresión de LaGeR. Si bien es cierto que muchos de los casos de uso no exigen una diferenciación de los gestos según su tamaño, es posible que existan casos donde sí sea necesario hacer dicha distinción. Por lo tanto, no podemos prohibir las hileras "no compactas".

Por último, la reducción solo es práctica en casos ideales donde se desee representar gestos exactos con formas grandes y regulares. En la mayoría de casos, no se podría llevar a cabo una reducción porque no habría un máximo común divisor

mayor a 1 para el número de repeticiones consecutivas de todos los literales. Bastaría con que la representación de un gesto contenga un solo literal aislado para que este sea "indivisible", y que por lo tanto la hilera entera no se pueda reducir.

La solución a estos problemas fue permitir que LaGeR represente gestos sin restricciones de compactibilidad, y abordar la invariabilidad de escala a nivel del algoritmo de reconocimiento.

Sea una "hilera de entrada" la hilera de LaGeR correspondiente al gesto hecho por un usuario, e "hilera almacenada" la hilera de LaGeR correspondiente a los gestos que el sistema puede identificar. En el flujo normal del sistema, el usuario produce un gesto a través de un dispositivo de entrada, el cual se convierte en una hilera de entrada para ser comparada a una o más hileras almacenadas que representan gestos candidatos. En caso de que la hilera de entrada y la hilera almacenada sean de diferente longitud, podemos expandir ambas por diferentes factores hasta llegar a su mínimo común múltiplo.

Por ejemplo, si el gesto almacenado es "a_.b_.b_.c_.c_.c_." (longitud 18) y el gesto de entrada es "a_.b_.c_.c_."(longitud 12), el mínimo común múltiplo es 36 (18*2 y 12*3). Por lo tanto expandimos la primera hilera por 2 y la segunda por 3, obteniendo hileras de longitud 36:

a_.b_.b_.c_.c_.c_. * 2 =
a_.a_.b_.b_.b_.b_.c_.c_.c_.c_.c_.c_.

a_.b_.c_.c_. * 3 =
a_.a_.a_.b_.b_.b_.c_.c_.c_.c_.c_.c_.

Luego de ser expandidas, las hileras están listas para ser comparadas sin importar la diferencia en sus tamaños originales. Con esto logramos el objetivo de la invariabilidad de escala a la hora de reconocer gestos.

Debemos hacer notar que este enfoque puede llegar a expandir las hileras considerablemente, incrementando el uso de memoria y el número de comparaciones de literales a ser efectuadas por el algoritmo de reconocimiento. El peor caso sería con gestos cuyas longitudes sean números primos, dado que su mínimo común múltiplo siempre sería su producto. Por ejemplo, dos hileras medianas de longitud 17 y 11 se expandirían hasta 17*11 = 187 literales.

Afortunadamente, este peor caso nunca se daría en LaGeR ya que todas sus hileras consisten de múltiples muestras de la misma longitud. Esto significa que una hilera con dos o más muestras nunca tendrá una longitud prima. Por ejemplo, cada muestra del Razer Hydra consiste de dos literales (uno por sensor) y un punto,

por lo que la longitud siempre sería un múltiplo de 3. Si bien el resultado puede seguir siendo grande, sigue siendo menor que en el caso de longitudes primas. Por ejemplo, el mínimo común múltiplo de una hilera con longitud 17*3=51 y una con longitud 11*3=33 es 561, el cual es inferior al producto de las longitudes (51*33=1683). En todo caso, existe un crecimiento considerable de las hileras a comparar, el cual podría motivar la investigación de optimizaciones en un trabajo futuro (ver Capítulo 7).

4.5.2. Tolerancia a gestos imprecisos

Un aspecto a tomar en cuanta es que los gestos descritos por LaGeR usualmente serán realizados por seres humanos. Esto implica un cierto grado de imprecisión que podría dificultar la detección exitosa de los gestos.

A nivel del lenguaje, nos limitamos a representar gestos de la forma más exacta y precisa posible. Por ejemplo, digamos que el sistema tiene almacenado un gesto que consiste de un trazo hacia la derecha y luego dos hacia el frente. El usuario intentará reproducirlo moviendo el sensor y, como su pulso no es perfecto, la hilera resultante contendrá literales correspondientes a pequeñas desviaciones. Por ejemplo, la Figura 4.12 muestra los gestos correspondientes a la hilera de entrada "l_.l_.l_.l_.l_.l_.l_.m_.n_.n_.n_.n_.n_.n_.n_.n_.n_.q_." y la hilera almacenada ".l_.l_.l_.l_.l_.l_.l_.n_.n_.n_.n_.n_.n_.n_.n_.n_.n_.".

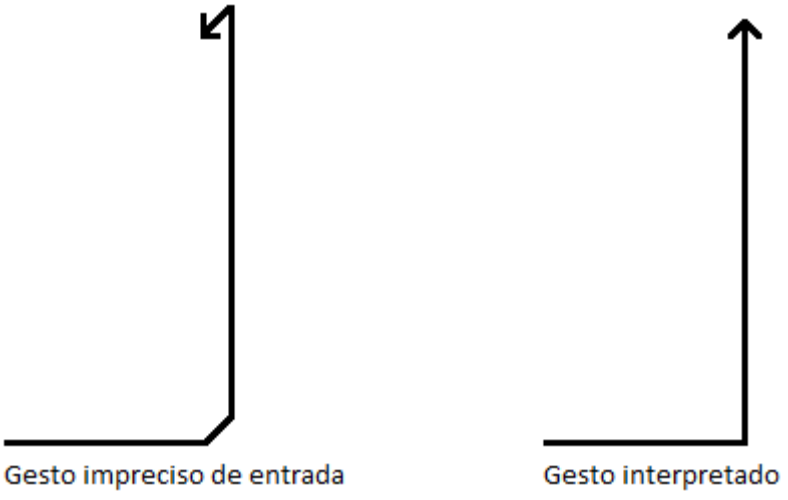


FIGURA 4.12: Hilera de gesto almacenado vs. hilera de gesto impreciso de entrada (visualización simplificada)

A pesar de los errores del usuario, el sistema debe poder reconocer el gesto siempre y cuando este tenga un cierto grado de parecido al trazo deseado. Este

es un aspecto que resolvimos a nivel del algoritmo de reconocimiento, el cual describiremos en la sección 5.3.

4.5.3. Invariabilidad rotativa de gestos cerrados

Finalmente, evaluamos la necesidad de que el sistema considere como equivalentes aquellos gestos cerrados que se inicien en posiciones distintas. Por ejemplo, un gesto circular que comience desde arriba versus un gesto circular que comience desde el lado derecho (Figura 4.13).

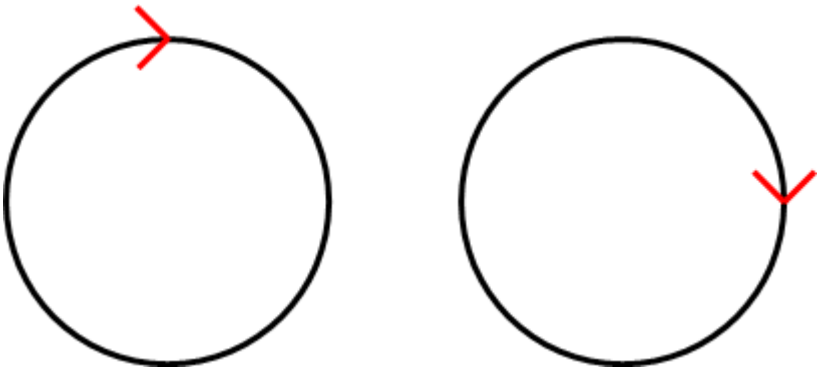


FIGURA 4.13: Gestos cerrados equivalentes

Si bien algunos casos de uso se beneficiarían de la invariabilidad rotativa (e.g., inserción de círculos en programa de dibujo), decidimos que esta no es una propiedad que deba ser parte del lenguaje. Al igual que con la invariabilidad de escala y la tolerancia a gestos imprecisos, es preferible que LaGeR se limite a representar los gestos lo más fielmente posible, y dejarle al reconocedor la opción de soportar o no la propiedad. En el Capítulo 7 discutiremos brevemente una forma de soportar la invariabilidad rotativa.

Capítulo 5

Implementación

5.1. Conversión a LaGeR

LaGeR es solo un lenguaje, y como tal no presupone una implementación específica de la conversión de gestos a hileras del lenguaje. Dicho esto, describimos nuestro propio algoritmo a modo de ejemplo. Este se monta sobre las API de VRPN para obtener muestras consecutivas de la posición de los sensores, calcular el movimiento resultante, y obtener el literal de LaGeR correspondiente (ver Algoritmo 1).

La distancia mínima entre muestras (MIN_SAMPLE_DIST) permite ajustar la sensibilidad de la conversión, estableciendo la magnitud real mínima que representará cada literal de LaGeR. En nuestro caso, utilizamos una distancia mínima de 4 mm, la cual nos permitió obtener una buena granularidad de movimientos convertidos, a la vez que pudimos evitar la detección espuria de pequeños movimientos involuntarios.

Algoritmo 1 Algoritmo de conversión de movimientos a LaGeR

Require: Muestra de la posición de un sensor
1: **if** Distancia euclidiana entre las muestras > MIN_SAMPLE_DIST **then**
2: Calcular los ángulos θ y ϕ correspondientes al movimiento desde la posición anterior hasta la posición más reciente
3: Redondear los ángulos hacia el intervalo de 45° más cercano
4: Utilizar dichos ángulos para consultar una tabla de conversión y obtener el literal de LaGeR correspondiente (ver Apéndice A)
5: **return** Literal de LaGeR
6: **else**
7: Ignorar la muestra
8: **end if**

5.2. Agrupamiento de literales de LaGeR

Dado que LaGeR agrupa las muestras de todos los sensores en cada intervalo de tiempo, fue necesario idear un mecanismo que hiciera esto con las muestras provistas por VRPN.

En VRPN, las muestras de cada sensor llegan por aparte a nuestro *callback* de movimiento. Afortunadamente, cada muestra viene con una marca de tiempo, lo cual nos permitió agrupar aquellas muestras de sensores distintos que coincidieran en la misma ventana de tiempo.

Para ello, ejecutamos un algoritmo de agrupamiento propio cada vez que se recibe una muestra para el movimiento de un sensor. Los literales correspondientes se agrupan cada vez que ocurran a una distancia temporal menor o igual a la constante MOVEMENT_GROUPING_TIME_MILLISECONDS (e.g., 200 ms), según se observa en el Algoritmo 2.

Algoritmo 2 Algoritmo de agrupamiento de literales a LaGeR

Require: Muestra de la posición de un sensor
1: Obtener literal de LaGeR para la muestra (ver Algoritmo 1)
2: Almacenar la marca de tiempo y el literal de la muestra
3: **if** El otro sensor se ha movido desde la última vez en que sus movimientos fueron agrupados **and** tiempo transcurrido desde dicho movimiento < MOVEMENT_GROUPING_TIME **then**
4: Almacenar las marcas de tiempo de los últimos movimientos agrupados para ambos sensores
5: Remplazar el último grupo de literales de la hilera de entrada por el literal de movimiento para el sensor 0, seguido del literal para el sensor 1, seguido del separador '.'
6: **else**
7: Concatenar a la hilera de entrada el literal del sensor actual y un '_' en la posición del literal del otro sensor, seguido del separador '.'
8: **end if**

5.3. Reconocimiento de Gestos

Uno de los objetivos más importantes de esta investigación fue comprobar la viabilidad y utilidad real de LaGeR como lenguaje para la representación de gestos. Para ello, no solo tuvimos que definir LaGeR como lenguaje, sino también comprobar que una representación de LaGeR podría ser utilizada para reconocer gestos.

Anteriormente, describimos cómo nuestra implementación continuamente convierte movimientos de sensores a LaGeR. Dicho proceso concatena literales de movimiento y agrupamiento a una hilera, la cual continúa creciendo hasta que el gesto termine.

Nuestro software delimita y separa los gestos a través de uno de dos mecanismos, según se configure:

1. **Tiempo:** El gesto termina si no se han detectado movimientos de sensor mayores al umbral de distancia en un lapso de tiempo mayor al umbral de pausa (e.g., 500 ms).

2. **Botones:** El gesto comienza al presionar un botón del sensor, y termina al soltarlo.

Una vez detectado el término de un gesto, se invoca el algoritmo de reconocimiento (ver Algoritmo 3) y se limpia la hilera antes de continuar con la conversión de movimientos a LaGeR.

Algoritmo 3 Algoritmo de reconocimiento de gestos en LaGeR

Require: Hilera de LaGeR para gesto de entrada
1: **for all** Hileras de gestos candidatos **do**
2: Expandir hilera de entrada e hilera de gesto candidato a MCM (ver subsección 4.5.1)
3: Calcular distancia Damerau-Levenshtein entre hilera de entrada e hilera de gesto candidato
4: Convertir distancia a porcentaje de la longitud de las hileras
5: **end for**
6: **if** Menor distancia < MIN_DISTANCE_PCT **then**
7: **return** Gesto candidato correspondiente
8: **else**
9: **return false**
10: **end if**

Nótese que MIN_DISTANCE_PCT, el umbral de distancia para reconocer gestos, es mayor conforme se incrementa el número de sensores que se movieron para producir el gesto (e.g., 20 % para un sensor, 30 % para dos sensores). Esto porque entre más sensores requiera un gesto, más literales pueden contener discrepancias, y viceversa. Por ejemplo, las hileras correspondientes a gestos de una sola mano tenderían a contener el literal '_' en todas las posiciones correspondientes al sensor no utilizado. El caso contrario sería un gesto de dos manos, donde las hileras utilizarían dos posiciones por grupo para representar movimientos. En nuestra implementación hicimos la distinción entre estos tipos de gestos iterando por las hileras y revisando si había al menos un literal de movimiento para cada sensor.

5.4. Software

5.4.1. Visión de Conjunto

La prueba de concepto de LaGeR consiste de una serie de bibliotecas y programas que ayudan a validar la hipótesis de que es posible codificar gestos a través de LaGeR para notificarle a las aplicaciones cuando se haya detectado el gesto deseado.

Para ello, nos apoyamos en los siguientes recursos:

- **GitHub**: Repositorio de código
- **Doxygen**: Generador de documentación
- **VRPN**: *Middleware* para dispositivos de entrada
- **1€ Filter**: Filtro para reducción de *jitter*
- **Boost**: Bibliotecas de C++ para hilos, comunicación entre procesos y operaciones matemáticas
- **Great Stella**: Programa para visualización y manipulación de poliedros

Y escribimos tres bibliotecas:

- **liblager_connect**: Intercambio de mensajes entre programas
- **liblager_convert**: Conversión de gestos a LaGeR
- **liblager_recognize**: Reconocimiento de gestos

Las cuales nos permitieron crear cuatro programas:

- **lager_recognizer**: Reconocedor de gestos
- **lager_injector**: Cliente del reconocedor
- **lager_gesture_manager**: Manejador de gestos
- **lager_viewer**: Visualizador de gestos

A continuación describiremos cómo implementamos cada componente y cómo encajaron entre sí para producir la prueba de concepto.

5.4.2. Repositorio para el código

Todo el código fuente relacionado a esta investigación se encuentra en el sitio de hospedaje de proyectos de software GitHub, en la dirección https://github.com/andresodio/lager.

En él podemos encontrar todos los *commits* de la implementación, los cuales está documentados con comentarios para facilitar la comprensión del historial de cambios.

5.4.3. Documentación y compilación

Como parte de los contenidos del repositorio en GitHub, incluimos tres archivos principales de documentación:

- **README.md**: Punto de partida para el desarrollador

- **dependencies.txt**: Instrucciones detalladas acerca de cómo instalar todas las dependencias del proyecto a fin de poder construirlo (*build*)
- **Doxyfile**: Archivo de configuración para el programa Doxyfile, el cual examina los archivos de C++ y genera documentación en HTML a partir de etiquetas especiales que incluimos en el código fuente. El resultado se encuentra en el directorio *docs*.

El *build* se lleva a cabo a través del script *buildall*, que se encarga de compilar e instalar todos los módulos de la prueba de concepto en el orden correcto, además de generar la documentación de Doxygen.

5.4.4. *Middleware* para dispositivos de entrada

Primero que todo, clonamos el repositorio de Git del proyecto VRPN en GitHub (https://github.com/vrpn/vrpn.git) y compilamos la versión más reciente (pre-07.34). Luego, descomentamos la siguiente línea en el archivo de configuración /usr/local/etc/vrpn.cfg para habilitar el uso del Razer Hydra como dispositivo de entrada (Tracker0):

```
#vrpn_Tracker_RazerHydra Tracker0
```

A lo largo de la investigación, utilizamos VRPN para obtener las coordenadas de los sensores en la biblioteca liblager_convert, la cual los convertía a literales de LaGeR. Para ello, creamos un objeto *vrpn_Tracker_Remote* y registramos *callbacks* para sus eventos por medio del método *register_change_handler*.

5.4.5. Suavizado de datos

Al comienzo del desarrollo de la prueba de concepto, utilizamos la salida cruda del dispositivo de entrada provisto por VRPN, *Tracker0*. Sin embargo, vimos que la salida era demasiado sucia, resultando en hileras de LaGeR llenas de literales espurios a causa del *jitter* en las coordenadas de entrada.

Fue por esto que recurrimos al filtro *1€ Filter*, el cual es provisto por VRPN para suavizar los datos.

Para utilizarlo, descomentamos la siguiente línea en el archivo de configuración /usr/local/etc/vrpn.cfg:

```
#vrpn_Tracker_FilterOneEuro Filter0 *Tracker0 2 1.15 1.0 1.2
1.5 5.0 1.2
```

Esto configuró el filtro de la siguiente manera:

- **Nombre del dispositivo filtrado**: Filter0
- **Nombre del dispositivo local a filtrar**: Tracker0
- **Número de sensores a filtrar**: 2

Para vectores

- **Frecuencia de corte mínima del filtro de paso bajo**: 1.15 Hz
- **Factor de pendiente de corte (β)**: 1.0
- **Frecuencia de corte de la derivada**: 1.2 Hz

Para cuaterniones

- **Frecuencia de corte mínima del filtro de paso bajo**: 1.5 Hz
- **Factor de pendiente de corte (β)**: 5.0
- **Frecuencia de corte de la derivada**: 1.2 Hz

Una vez configurado el filtro fue bastante sencillo utilizarlo en nuestro código. Bastó con inicializar VRPN utilizando el dispositivo *Filter0* (filtrado) en vez de *Tracker0* (crudo).

5.4.6. Visualización de equivalencia entre movimientos y literales de LaGeR

La definición de LaGeR se basa en una equivalencia entre direcciones de movimiento y literales, la cual se puede visualizar en términos de un rombicuboctaedro (ver Sección 4.2). Para facilitar dicha visualización, instalamos al programa Great Stella (Webb, 2015). Great Stella es un navegador de poliedros con el cual pudimos etiquetar las caras de un rombicuboctaedro y manipularlo directamente con el ratón para rotarlo, aumentar y disminuir el *zoom*. Además, el programa produjo un desarrollo del poliedro que se puede imprimir y armar para formar un modelo físico de papel (ver Figura 5.1).

5.4.7. Bibliotecas externas

Como parte de nuestra implementación, aprovechamos parte del conjunto de bibliotecas de C++ Boost:

- **libboost_serialization**: Provee clases tales como *boost::archive::text_iarchive* y *boost::archive::text_oarchive* , con las cuales serializamos y des-serializamos mensajes de suscripción y notificación de gestos en liblager_connect.

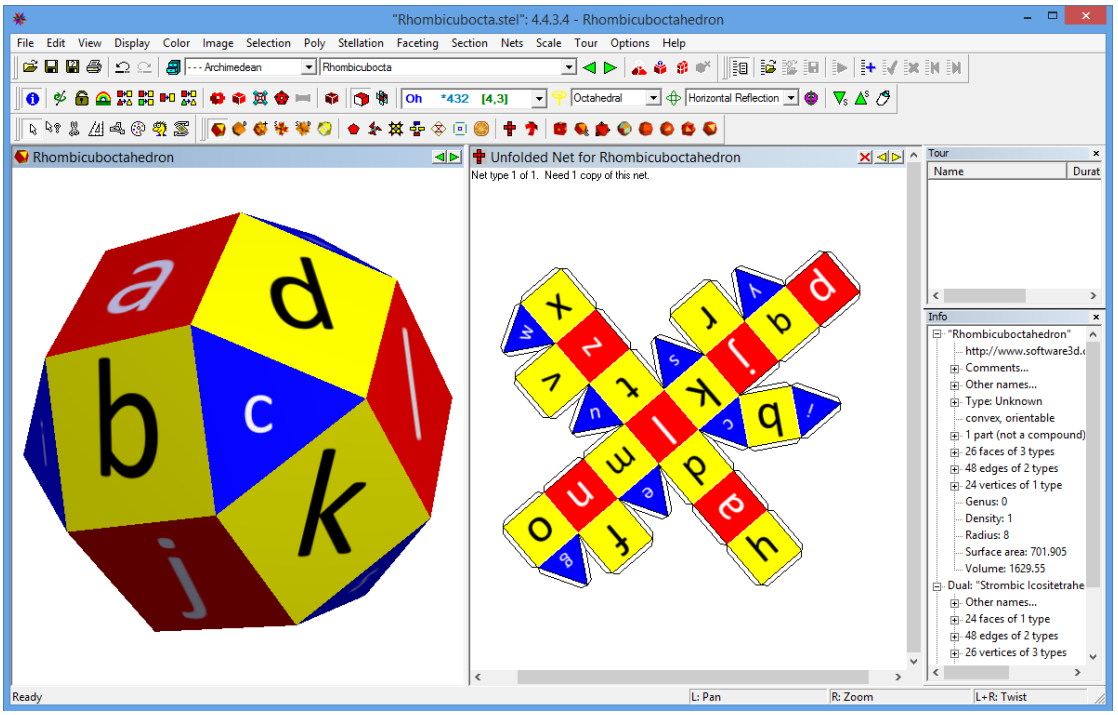


FIGURA 5.1: Rombicuboctaedro etiquetado con literales de LaGeR en Great Stella

- **libboost_system**: Provee una serie de funciones matemáticas tales como *boost::math::lcm (Least Common Multiple)*, la cual utilizamos en liblager_recognize para expandir gestos al mínimo común múltiplo de sus longitudes antes de compararlos.
- **libboost_thread**: Provee la clase *boost::thread*, la cual utilizamos para lanzar hilos para el procesamiento de datos (ej. revisión de cola de mensajes en liblager_recognize, conversión de movimientos a LaGeR en liblager_convert).

Además de Boost, también utilizamos libXtst en lager_injector para inyectar eventos de entrada al sistema operativo por medio del manejador de ventanas X Server con funciones tales como XTestFakeKeyEvent.

Por último, el visualizador de gestos lager_viewer está enlazado contra las bibliotecas gráficas de OpenGL tales como GLFW (creación y manejo de eventos de entrada en ventanas), GLM (operaciones matemáticas para gráficos) y GLEW (detección en tiempo de ejecución de soporte para extensiones).

5.4.8. Bibliotecas internas

Las bibliotecas internas de LaGeR están documentadas por medio de comentarios especialmente etiquetados en el código. Estos se utilizan durante el proceso

de *build* para generar documentación detallada por medio de Doxygen, la cual se encuentra en docs/index.html.

A continuación presentamos un resumen acerca de cada biblioteca.

5.4.8.1. liblager_connect

La principal función de la biblioteca liblager_connect es facilitar el intercambio de mensajes entre los programas que utilizan LaGeR. Esta provee una cola de mensajes y los mecanismos necesarios para la suscripción y notificación de mensajes de gestos. Es a través de ella que el reconocedor de gestos y sus clientes se comunican.

Las siguientes clases son los principales componentes de liblager_connect:

- **SubscribedGesture**: Gestos suscritos
- **DetectedGestureMessage**: Mensajes para notificar la detección de un gesto
- **GestureSubscriptionMessage**: Mensajes para suscribirse a un gesto

Además se proveen funciones para crear una cola de mensajes, recibir y enviar mensajes a través de ella, etc.

5.4.8.2. liblager_convert

La principal función de liblager_convert es tomar los datos producidos por los movimientos de un dispositivo de entrada y convertirlos en la hilera de LaGeR correspondiente. En nuestra implementación, los datos se obtienen por medio de VRPN y se convierten a través del Algoritmo 1 y el Algoritmo 2.

El principal componente expuesto por liblager_convert es la clase *singleton* LagerConverter. Entre otros, provee un método público BlockingGetLagerString que permite que el llamador bloquee hasta que un gesto haya terminado y su hilera correspondiente haya sido retornada.

5.4.8.3. liblager_recognize

La principal función de liblager_recognize es tomar una hilera de LaGeR, compararla contra un conjunto de gestos almacenados, identificar y retornar el gesto más similar. Para ello, utiliza las bibliotecas externas de Boost y la biblioteca interna liblager_connect. El reconocimiento de gestos se lleva a cabo según el Algoritmo 3.

El principal componente expuesto por liblager_recognize es la clase *singleton* LagerRecognizer. Entre otros, provee un método público RecognizeGesture que permite que el llamador provea una hilera de LaGeR y reciba el gesto correspondiente (o el valor *false* en caso de que no haya ninguno).

5.4.9. Programas internos

Los programas internos de LaGeR están documentados por medio de comentarios especialmente etiquetados en el código. Estos se utilizan durante el proceso de *build* para generar documentación detallada por medio de Doxygen, la cual se encuentra en docs/index.html.

A continuación presentamos un resumen acerca de cada programa.

5.4.9.1. lager_recognizer

La principal función de lager_recognizer es recibir suscripciones a gestos descritos en LaGeR para luego notificar a los suscriptores en caso de que un gesto suficientemente parecido haya sido producido a través de un dispositivo de entrada. Esto se logra a través del uso de todas las bibliotecas internas de LaGeR:

- **liblager_connect**: Para recibir suscripciones y enviar notificaciones
- **liblager_convert**: Para convertir los movimientos de un dispositivo de entrada en una hilera de LaGeR
- **liblager_recognize**: Para obtener el gesto suscrito más similar a la hilera de LaGeR de entrada

lager_recognizer no expone una API externa, sino que se comunica con otros procesos (e.g., los suscriptores) por medio de una cola de mensajes. Puede ser invocado por el sistema operativo de forma automática al iniciar el sistema, o bien ser invocado directamente desde la línea de comandos para ver bitácoras de debugueo (ver Figura 5.2). Además, lager_recognizer provee una opción de debugueo *–draw_gestures* que invoca lager_viewer cada vez que se detecta un gesto para representar gráficamente su hilera de LaGeR y la del candidato más cercano.

5.4.9.2. lager_viewer

El objetivo principal de lager_viewer es proveer una forma de visualizar hileras de gestos que han sido representados en LaGeR. Para ello, recibe una hilera de LaGeR de la línea de comandos y convierte cada literal de movimiento en un

```
 _____________________________
|                             |
|       COLLECTING DATA...    |
|_____________________________|

 _____________________________
|                             |
|            INPUT            |
|_____________________________|

Input:
  _a._q._p._p._p._x._x._z._t._t._t._l._l._l._l.

 _____________________________
|                             |
|         RECOGNIZING...      |
|_____________________________|

OpenChrome
  _p._p._p._y._y._z._z._z._t._t._l._l._l.
Distance: 16.2393% (95 D-L ops)

NewTab
  _l._l._l._l._l._l._p._p._p._z._z._z._z.
Distance: 33.3333% (195 D-L ops)

OpenCNN

_p._p._x._x._z._t._t._l._l._l._d._a._a._a._t._t._t._t._t._d._a._a._a._t._t._t._t._t._a._a._a._a.
Distance: 30.8333% (444 D-L ops)

OpenGoogle
  _p._p._p._y._x._y._y._z._t._t._t._l._l._l._d._a._h._p._p._p.
Distance: 32.2222% (58 D-L ops)

CloseTab
  _x._x._x._x._x._x._x._a._a._a._a._a._t._t._t._t._t._t._t.
Distance: 28.7719% (246 D-L ops)

ZoomIn
  pl.pl.pl.pl.pl.pl.
Distance: 57.7778% (52 D-L ops)

ZoomOut
  lp.lp.lp.lp.lp.lp.
Distance: 60% (54 D-L ops)

RefreshTab

_l._l._l._t._t._t._t._t._z._z._z._z._z._z._x._x._x._x._x._x._p._p._p._p._p._p._h._h._h._h._h.
_h._a._a._a._a._a._a._d._d._d._d._d._d._l._l._l.
Distance: 27.7778% (200 D-L ops)

 _____________________________
|                             |
|         MATCH FOUND!        |
|_____________________________|


Closest gesture:      OpenChrome

Distance:             16.2393% (95 D-L ops)
Threshold:            20%

Recognition time:     53ms
```

FIGURA 5.2: Bitácora de lager_recognizer

segmento normalizado en la dirección correspondiente, y luego dibuja todos los segmentos para formar un trazo.

En caso de que haya más de un sensor, puede utilizar diferentes colores para diferenciarlos. Además, utiliza gradientes para indicar la dirección del movimiento, comenzando el trazo de forma oscura y aclarándolo cada vez más (ver Figura 5.3). Los gestos se dibujan en una ventana que le permite al usuario desplazarse la vista linealmente con las flechas del teclado y rotarla con el *mouse* (ver Figura 5.4).

FIGURA 5.3: Lager Viewer mostrando gesto con dos sensores

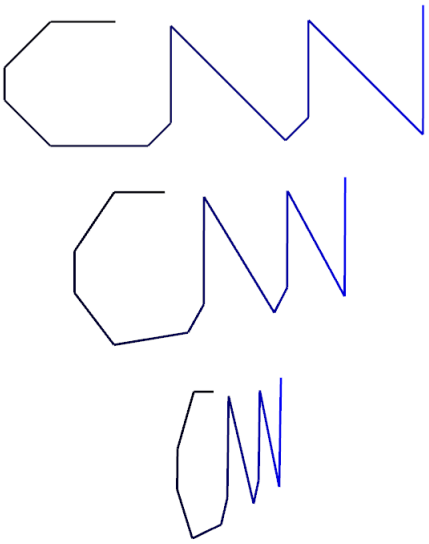


FIGURA 5.4: Lager Viewer mostrando CNN desde diferentes ángulos

5.4.9.3. lager_gesture_manager

La principal función del lager_gesture_manager es proveer una forma conveniente de manejar un archivo de gestos representados en LaGeR. Para ello, provee un menú interactivo en la línea de comandos desde el cual los desarrolladores pueden añadir gestos (tanto directamente en hileras de LaGeR como por medio de los movimientos de un sensor), visualizarlos, editarlos y borrarlos (ver Figura 5.5). Los gestos se traducen a LaGeR por medio de la biblioteca liblager_convert y se visualizan invocando lager_viewer.

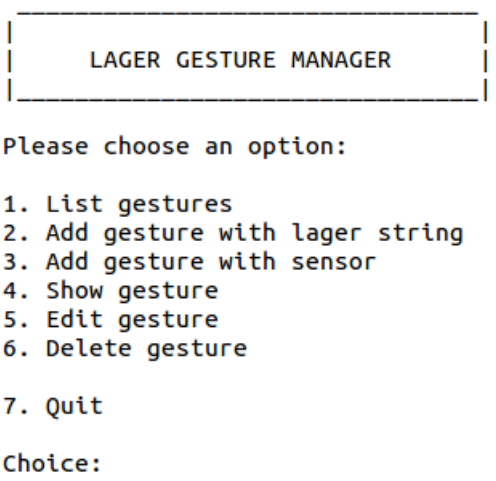


FIGURA 5.5: Menú principal de lager_gesture_manager

5.4.9.4. lager_injector

A diferencia de los demás programas internos de LaGeR, lager_injector no es estrictamente parte del sistema de representación y reconocimiento de gestos en sí, sino que se incluye como prueba de concepto.

La principal función de lager_injector es proveer un ejemplo para demostrar que los desarrolladores pueden escribir aplicaciones controladas por gestos de forma sencilla y efectiva utilizando LaGeR. Para ello, carga un archivo de gestos previamente creado por lager_gesture_manager y utiliza la biblioteca liblager_connect para enviarle mensajes de suscripción a lager_recognizer para cada uno de ellos. Al ser notificado de un gesto, utiliza la biblioteca libXtst para inyectar eventos de entrada de teclado al X Server para controlar el manejador de ventanas.

En nuestra implementación, los gestos permiten abrir el navegador web Google Chrome y llevar a cabo varias tareas tales como abrir una nueva pestaña, cargar el sitio de CNN, hacer *zoom in* y *zoom out*, refrescar y cerrar una pestaña.

5.4.10. Pruebas

A fin de comprobar la hipótesis, llevamos a cabo dos pruebas de usuario que utilizaron todos los componentes del sistema de descripción y reconocimiento de gestos por medio de hileras de LaGeR.

5.4.10.1. Prueba 1

La primera prueba tuvo como objetivo comprobar el funcionamiento básico del sistema en condiciones controladas. Para ello diseñamos el protocolo descrito

en el Capítulo 3, el cual requiere que un usuario novato realice diez repeticiones por gesto de un conjunto de primitivas (líneas, polígonos, círculo, gestos de dos sensores).

La implementación de esta prueba consistió de los siguientes pasos:

1. **Conectar sensor**: Primero que todo, conectamos un dispositivo de entrada tridimensional (Razer Hydra) al puerto USB de la computadora.

2. **Arrancar VRPN**: Luego, iniciamos el servidor de VRPN, el cual convierte los movimientos del dispositivo de entrada en coordenadas X, Y y Z:

   ```
   sudo vrpn_server -f /usr/local/etc/vrpn.cfg
   ```

3. **Definir gestos**: Una vez levantado VRPN, la opción "Add gesture with sensor" de lager_gesture_manager nos permitió utilizar el dispositivo de entrada para definir los gestos primitivos a ser reconocidos por el sistema. Internamente, lager_gesture_manager utilizó la biblioteca liblager_convert para convertir las coordenadas de sensores reportadas por VRPN en hileras de LaGeR, las cuales se almacenaron en el archivo gestures.dat.

4. **Refinar gestos**: Para asegurarnos de que las hileras resultantes representaran los gestos de manera adecuada, visualizamos los trazos resultantes gráficamente a través de lager_viewer, el cual invocamos con la opción "Show gesture" de lager_gesture_manager. En algunos casos, esto nos llevó a retocar las hileras de LaGeR directamente con la ayuda del rombicuboctaedro etiquetado de Great Stella y la opción "Edit gesture" de lager_gesture_manager.

5. **Invocar reconocedor**: Una vez definidos y refinados los gestos, arrancamos el reconocedor lager_recognizer, el cual utiliza liblager_convert para convertir las coordenadas provistas por VRPN en literales de LaGeR, y luego utiliza liblager_recognize para encontrar la hilera de LaGeR más cercana en gestures.dat. Al ejecutar lager_recognizer, pasamos argumentos de entrada para producir bitácoras detalladas y utilizar el archivo de gestos (gestures.dat):

   ```
   lager_recognizer --print_updates --use_gestures_file
   ```

6. **Realización de gestos**: Por último, instruimos al sujeto a tomar los bastones del dispositivo de entrada y realizar cada gesto diez veces, observando las bitácoras de lager_recognizer.

Los resultados de la Prueba 1 se encuentran en el Apéndice B, y se analizan en la Sección 6.1.

5.4.10.2. Prueba 2

La segunda prueba tuvo como objetivo replicar condiciones más realistas para evaluar la utilidad del sistema como un todo. Para ello diseñamos el protocolo descrito en el Capítulo 3, el cual requiere que un usuario novato realice diez veces una secuencia de acciones para navegar por la Web con Google Chrome. A diferencia de la Prueba 1, la Prueba 2 involucró el uso de un cliente suscriptor de gestos descritos en LaGeR y todo el andamiaje relacionado.

La implementación de esta prueba consistió de los siguientes pasos:

1. **Conectar sensor**: Igual a la Prueba 1
2. **Arrancar VRPN**: Igual a la Prueba 1
3. **Definir gestos**: Similar a la Prueba 1, solo que en este caso se añadieron hileras de LaGeR para gestos de navegación Web.
4. **Refinar gestos**: Igual a la Prueba 1
5. **Invocar reconocedor**: Similar a la Prueba 1, pero en este caso ejecutamos lager_recognizer omitiendo la bandera –use_gestures_file. Esto ocasionó que el programa utilizara la biblioteca liblager_connect para esperar mensajes de suscripción de gestos, en vez de utilizar un archivo de gestos propio.
6. **Arrancar inyector**: La Prueba 2 requirió de un cliente LaGeR que pudiera utilizar la detección de gestos de lager_recognizer para manipular el sistema operativo. Para ello, arrancamos lager_injector, el cual utilizó liblager_connect para enviar mensajes a la cola de suscripción de gestos de lager_recognizer, y libXtst para enviar eventos de entrada (presión de teclas) al sistema operativo a través del manejador de ventanas.
7. **Realización de gestos**: Por último, instruimos al sujeto a tomar los bastones del dispositivo de entrada y realizar cada secuencia de gestos diez veces. En este caso el usuario pudo observar en tiempo real el efecto concreto de sus gestos.

Los resultados de la Prueba 2 se encuentran en el Apéndice C, y se analizan en la Sección 6.2.

Capítulo 6

Análisis de Resultados

6.1. Prueba 1

Los datos crudos (∼300 páginas) y filtrados (∼25 páginas) de la Prueba 1 (gestos primitivos) están disponibles al lector por medio de consulta directa al autor. Los resultados procesados se encuentran en el Apéndice B. Estos se dividen según el gesto evaluado y las 10 corridas que se llevaron a cabo para cada uno. Para cada corrida tenemos el candidato más cercano reportado por el sensor, si el sistema indicó una correspondencia, el porcentaje de la distancia Damerau-Levenshtein entre las hileras versus su longitud expandida, el umbral de distancia y el tiempo de procesamiento.

En general, los resultados fueron positivos, ya que la mayoría de los gestos fueron detectados exitosamente con tiempos de respuesta rápidos.

La principal excepción fue el círculo, que no se detectó en ninguna de las corridas (ver Cuadro B.5 y Figura B.5). Esto puede ser debido a que el círculo está compuesto por una sola curva continua, lo cual plantea un reto para la relativamente limitada granularidad de los literales de LaGeR (intervalos de 45°). Además, el sujeto expresó que era muy difícil reproducir un círculo en el aire, añadiendo que en general es difícil reproducir círculos de forma uniforme, inclusive con papel y lápiz. La otra excepción fueron las líneas hacia adelante y hacia atrás. Estas se detectaron mucho mejor que el círculo, pero en todo caso presentaron 2 y 4 fallos, respectivamente (ver Cuadros B.6 y B.7, y Figuras B.6 y B.7). Notamos que el sujeto tuvo dificultad en reproducir los gestos en el aire de forma exacta, al igual que con el círculo.

Los resultados del círculo y las líneas hacia adelante y hacia atrás nos indican tres posibles soluciones:

1. Los desarrolladores deben ser cuidadosos para no requerir que el usuario haga gestos excesivamente complicados o incómodos. Algunos gestos que pueden parecer simples se vuelven difíciles de reproducir fielmente en espacio tridimensional al mover un sensor en el aire sin un apoyo o punto de referencia visual.

2. Los umbrales de detección deben aumentarse. Aún las corridas de gestos exitosos estuvieron cerca del umbral, por lo que fácilmente podrían fallar con otros usuarios o variaciones de uso. Y en general, la gran mayoría de las corridas harían sido exitosas con solo elevar el umbral 5 %, llegando a una tasa de éxito de 100 % con solo subir el umbral a 30 %.

3. La comparación de hileras por parte del sistema se podría hacer más precisa si la distancia entre literales tomara en cuenta la similitud de los movimientos que representan (ver Capítulo 7).

6.2. Prueba 2

Los datos crudos (∼400 páginas) y filtrados (∼30 páginas) de la Prueba 2 (utilización de Google Chrome por medio de gestos) están disponibles al lector por medio de consulta directa al autor. Los resultados procesados se encuentran en el Apéndice C. Estos incluyen el consolidado de las 10 corridas de la tarea, mostrando el tiempo total para completar cada una, así como el número de errores. Además se presenta el desglose de los datos para la ejecución de cada gesto involucrado en las corridas de la tarea.

Observamos que el tiempo tendió hacia la baja a lo largo de la prueba, probablemente debido al aprendizaje del sujeto (ver Figura C.1). La tasa de errores también tendió hacia la baja, pero no con tanta claridad (ver Figura C.2). Estos resultados, junto con nuestras observaciones cualitativas, indican que el factor que limitó el desempeño del sujeto no fue el funcionamiento del sistema de reconocimiento de gestos, sino su memorización de los pasos a seguir y la fluidez con que encadenó un gesto tras otro. En general, la tasa de errores fue relativamente baja y el sujeto reportó sentirse satisfecho con su desempeño.

Un detalle interesante es que tanto el gesto circular de la Prueba 1 (Circle) como el gesto para refrescar la página de la Prueba 2 (RefreshTab) eran idénticos (círculo en el aire), pero tuvieron tasas de éxito radicalmente diferentes. Mientras que todos los intentos por reproducirlo en la Prueba 1 fallaron (ver Figura B.5), todos fueron exitosos en la Prueba 2 (ver Figura C.8). Al analizar los datos crudos nos dimos cuenta de que, debido a una pulga en el sistema, el umbral de distancia para el gesto RefreshTab se elevó a 30 % versus el 20 % de Circle. Esto ocurrió gracias a que la secuencia de la Prueba 2 estableció que el gesto RefreshTab debía seguir al gesto ZoomOut, el cual se llevó a cabo con dos sensores y por lo tanto se evaluó con el umbral más alto. Por algún motivo, el último movimiento del segundo sensor para ese gesto se trasladó al comienzo del LaGeR para RefreshTab, por lo que se le asignó el mismo umbral. En cambio, el gesto para abrir Google (OpenGoogle) falló la mayor parte del tiempo a pesar de tener una forma muy similar a RefreshTab (”G”mayúscula en el aire), ya que siguió a un gesto de un solo sensor y se evaluó con el umbral de 20 % (ver Figura C.10). Estos resultados

proveyeron una demostración fortuita de los beneficios de elevar el umbral de detección, lo cual sugerimos en la sección 6.1.

Capítulo 7

Conclusiones y Trabajo Futuro

7.1. Conclusiones

La presente investigación probó la hipótesis de que es posible codificar gestos por medio de un lenguaje que represente el movimiento de dos puntos correspondientes a las lecturas de un sensor de entrada gestual en espacio bidimensional y tridimensional. Dicho lenguaje pudo ser utilizado por un sistema operativo para proveer eventos de gestos a sus aplicaciones de forma agnóstica al dispositivo de entrada.

Los objetivos planteados se alcanzaron de la siguiente manera:

1. **Objetivos 1**: Planteamos una forma general de representar el movimiento de dos puntos por medio de la metáfora de una rosa de los vientos en tres dimensiones.

2. **Objetivo 2**: Formalizamos la metáfora diseñando LaGeR, un lenguaje que codifica gestos en hileras.

3. **Objetivo 3**: Diseñamos un algoritmo de reconocimiento de gestos tolerante a datos inexactos, basándonos en la distancia Damerau-Levenshtein entre las hileras de LaGeR que representan los gestos.

4. **Objetivo 4**: Diseñamos e implementamos un sistema completo para la creación de aplicaciones controladas por gestos descritos en LaGeR, el cual probamos con un usuario de forma exitosa.

7.1.1. Trabajo Futuro

Si bien el sistema que implementamos alcanzó los objetivos de la investigación, identificamos varias mejoras que se pueden hacer como parte de investigaciones futuras:

1. **Umbrales de distancia**: Durante los experimentos vimos que el factor más significativo en el éxito del reconocimiento de gestos fue el umbral máximo de diferencia entre hileras. Con solo haberlo elevado de 20 % a 30 %, habríamos obtenido una tasa de éxito casi total. Esta mejora es notable puesto que es la más sencilla de llevar a cabo (basta con modificar una constante en liblager_convert). Sin embargo, es importante evaluar cuidadosamente el equilibrio entre facilidad de reconocimiento (la cual se maximiza elevando el umbral) y capacidad para discriminar entre gestos (la cual se maximiza bajando el umbral). En nuestro caso, establecimos el umbral más bajo posible

durante la fase de desarrollo, y probablemente lo bajamos demasiado debido a un efecto de auto-entrenamiento que nos permitió reproducir gestos con más exactitud de la que tendría un usuario principiante o intermedio.

2. **Expansión de hileras**: La expansión de hileras que utilizamos en esta investigación es propensa a generar hileras relativamente largas (ver Subsección 4.5.1), lo cual puede impactar la velocidad del algoritmo de reconocimiento. Una posible mejora es no expandir las hileras si la diferencia de longitud entre ellas es menor a un cierto porcentaje de sus longitudes. En caso de que la diferencia sea mayor a dicho umbral, podemos evaluar truncar una o ambas hileras para asegurarnos de que su longitud no sea prima, lo cual reducirá su mínimo común múltiplo y por ende su longitud expandida.

3. **Comparación de literales**: Actualmente, el algoritmo de reconocimiento deriva la distancia de Damerau-Levenshtein tratando a todos los literales como equivalentes a la hora de calcular el número de cambios de caracteres que se necesitarían para ir de una hilera a otra. Esto no es óptimo puesto que los literales en realidad corresponden a direcciones de movimiento, y evidentemente algunas direcciones de movimiento son más cercanas entre sí. Esta mejora puede ser relativamente compleja de implementar, y hay que tener cuidado para no impactar la velocidad de respuesta del sistema, pero podría mejorar notablemente la precisión del reconocimiento de gestos.

4. **Entrenamiento de gestos**: El diseño de los experimentos estableció que el usuario debía reproducir gestos previamente definidos por el programador por medio de una única hilera de LaGeR para representar cada gesto. Una forma de mejorar las probabilidades de detección exitosa sería que el usuario entrene los gestos antes de usar los programas. Para ello, el utilitario lager_gesture_manager ya permite añadir entradas al archivo de gestos (hileras de LaGeR) de los programas. Dicha funcionalidad también se podría incorporar directamente en los programas por medio de la biblioteca liblager_convert, ya sea como una actividad explícita o como parte de un mecanismo de entrenamiento automático y continuo. La detección también se podría mejorar añadiendo múltiples entradas para cada gesto, lo cual ayudaría a tomar en cuenta las variaciones naturales del usuario cada vez que lleva a cabo el gesto. Al igual que otras mejoras, estas elevarían el tiempo de detección a cambio de una mejora en la exactitud de la misma.

5. **Invariabilidad rotativa**: En la Subsección 4.5.3 explicamos los motivos por los que LaGeR como lenguaje no debería implementar esta propiedad. Sin embargo, podría ser útil que el reconocedor lo haga. Para ello, podríamos detectar si un gesto es cerrado almacenando las coordenadas del sensor a la hora de comenzar el gesto, y luego comparándolas con las coordenadas a la hora de finalizar el gesto. El gesto se consideraría cerrado en caso de que la distancia euclideana entre dichas coordenadas esté por debajo de un cierto umbral. Una vez identificados como cerrados, las hileras de LaGeR de los gestos se podrían comparar de manera similar a hileras corridas tales como "palabra" vs. "abrapal", por medio de alguna biblioteca preexistente que tolere corrimientos.

6. **Sistema de suscripciones**: Actualmente, los programas que se suscriben a gestos (e.g., lager_injector) a través de lager_recognizer solo pueden crear nuevas suscripciones. Se podría crear un mecanismo para cancelar las suscripciones por solicitud del cliente, así como un mecanismo para cancelarlas automáticamente en caso de que el programa cliente muera inesperadamente.

Apéndice A

Mapeo de Coordenadas Esféricas a Literales de LaGeR

θ	ϕ	Literal de LaGeR
0	*	a
180	*	z
360	*	a

θ	ϕ	Literal de LaGeR
45	0	b
45	45	c
45	90	d
45	135	e
45	180	f
45	225	g
45	270	h
45	315	i
45	360	b
90	0	j
90	45	k
90	90	l
90	135	m
90	180	n
90	225	o
90	270	p
90	315	q
90	360	j
135	0	r
135	45	s
135	90	t
135	135	u
135	180	v
135	225	w
135	270	x
135	315	y
135	360	r

θ	ϕ	Literal de LaGeR
225	0	v
225	45	w
225	90	x
225	135	y
225	180	r
225	225	s
225	270	t
225	315	u
225	360	v
270	0	n
270	45	o
270	90	p
270	135	q
270	180	r
270	225	s
270	270	t
270	315	u
270	360	n
315	0	f
315	45	g
315	90	h
315	135	i
315	180	j
315	225	k
315	270	l
315	315	m
315	360	f

CUADRO A.1: Mapeo de coordenadas esféricas a literales de LaGeR

Apéndice B

Resultados de Prueba 1

Gesto	Corrida	Candidato	Correspondencia	Distancia	Umbral	Tiempo (ms)
Línea horizontal	1	Línea horizontal	Sí	14.67%	20%	225
	2	Línea horizontal	Sí	15.38%	20%	111
	3	Línea horizontal	Sí	17.86%	20%	23
	4	Línea horizontal	Sí	16.67%	20%	71
	5	Línea horizontal	Sí	3.51%	20%	143
	6	Línea horizontal	Sí	12.70%	20%	40
	7	Línea horizontal	Sí	10.61%	20%	48
	8	Línea horizontal	Sí	14.04%	20%	215
	9	Línea horizontal	Sí	16.67%	20%	10
	10	Línea horizontal	Sí	14.49%	20%	251

CUADRO B.1: Resultados de gestos para primitiva de línea horizontal

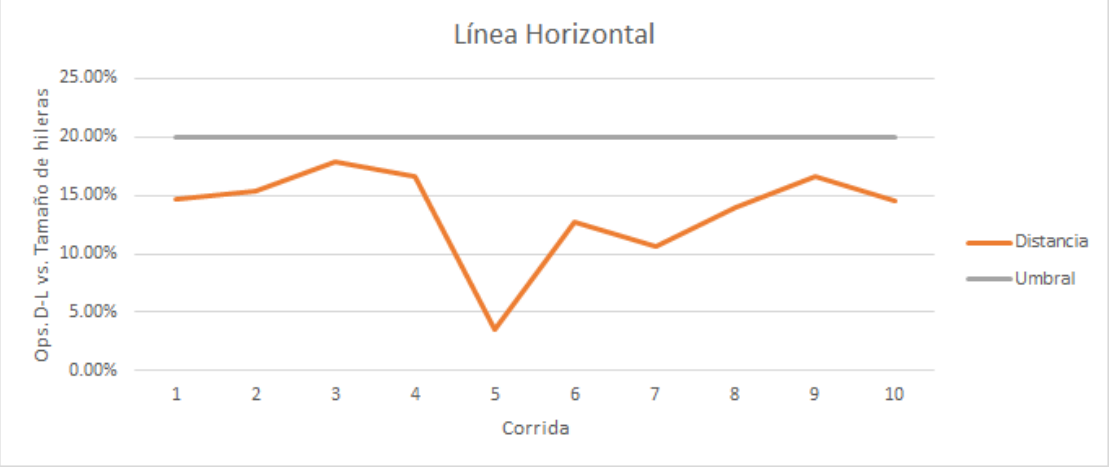


FIGURA B.1: Distancia y umbral vs. corridas de gestos de línea horizontal

Gesto	Corrida	Candidato	Correspondencia	Distancia	Umbral	Tiempo (ms)
Línea vertical	1	Línea vertical	Sí	8.77%	20%	239
	2	Línea vertical	Sí	15.87%	20%	38
	3	Línea vertical	Sí	14.29%	20%	64
	4	Línea vertical	Sí	18.67%	20%	209
	5	Línea vertical	Sí	19.75%	20%	87
	6	Línea vertical	Sí	13.04%	20%	268
	7	Línea vertical	Sí	13.33%	20%	19
	8	Línea vertical	Sí	15.79%	20%	136
	9	Línea vertical	Sí	5.26%	20%	135
	10	Línea vertical	No	22.22%	20%	36

CUADRO B.2: Resultados de gestos para primitiva de línea vertical

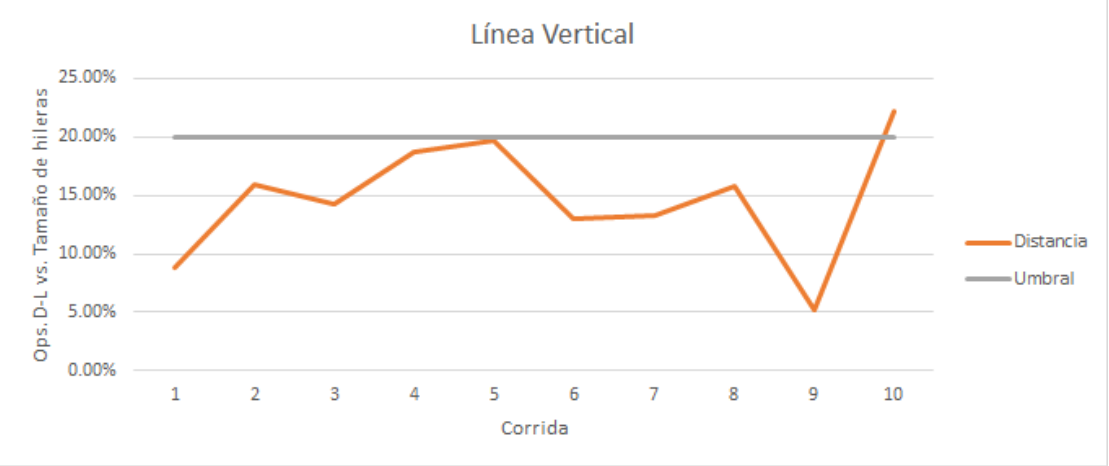


FIGURA B.2: Distancia y umbral vs. corridas de gestos de línea vertical

Gesto	Corrida	Candidato	Correspondencia	Distancia	Umbral	Tiempo (ms)
Triángulo	1	Triángulo	Sí	18.36%	20%	1688
	2	Triángulo	Sí	19.37%	20%	104
	3	Triángulo	Sí	17.93%	20%	545
	4	Triángulo	Sí	18.83%	20%	507
	5	Triángulo	Sí	19.55%	20%	54
	6	Triángulo	Sí	17.91%	20%	691
	7	Triángulo	Sí	18.79%	20%	54
	8	Triángulo	No	23.70%	20%	134
	9	Triángulo	No	20.93%	20%	783
	10	Triángulo	No	20.18%	20%	135

CUADRO B.3: Resultados de gestos para primitiva de triángulo

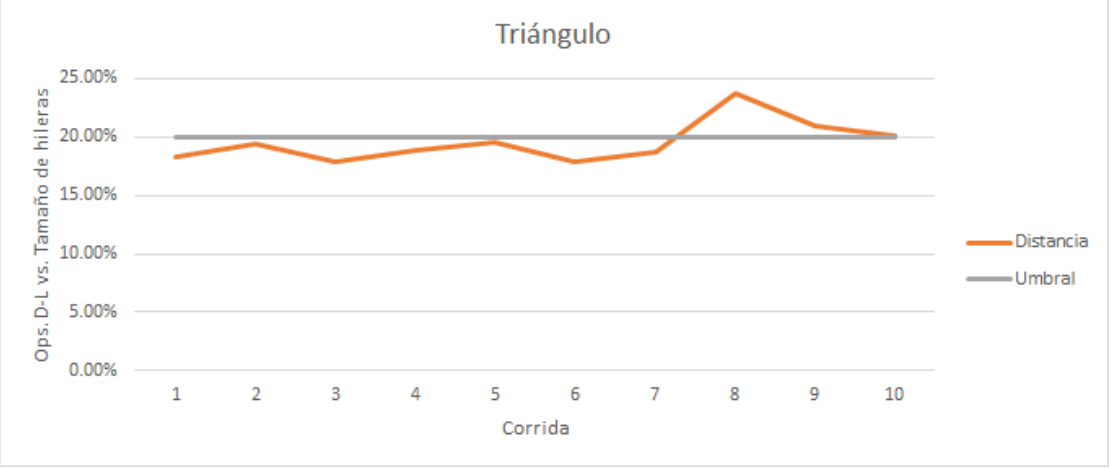


FIGURA B.3: Distancia y umbral vs. corridas de gestos de triángulo

Gesto	Corrida	Candidato	Correspondencia	Distancia	Umbral	Tiempo (ms)
Cuadrado	1	Cuadrado	Sí	5.00%	20%	207
	2	Cuadrado	Sí	8.97%	20%	130
	3	Cuadrado	Sí	10.09%	20%	208
	4	Cuadrado	Sí	7.50%	20%	17
	5	Cuadrado	Sí	11.71%	20%	508
	6	Cuadrado	Sí	8.78%	20%	508
	7	Cuadrado	Sí	12.16%	20%	519
	8	Cuadrado	Sí	10.00%	20%	18
	9	Cuadrado	Sí	11.11%	20%	27
	10	Cuadrado	Sí	8.56%	20%	521

CUADRO B.4: Resultados de gestos para primitiva de cuadrado

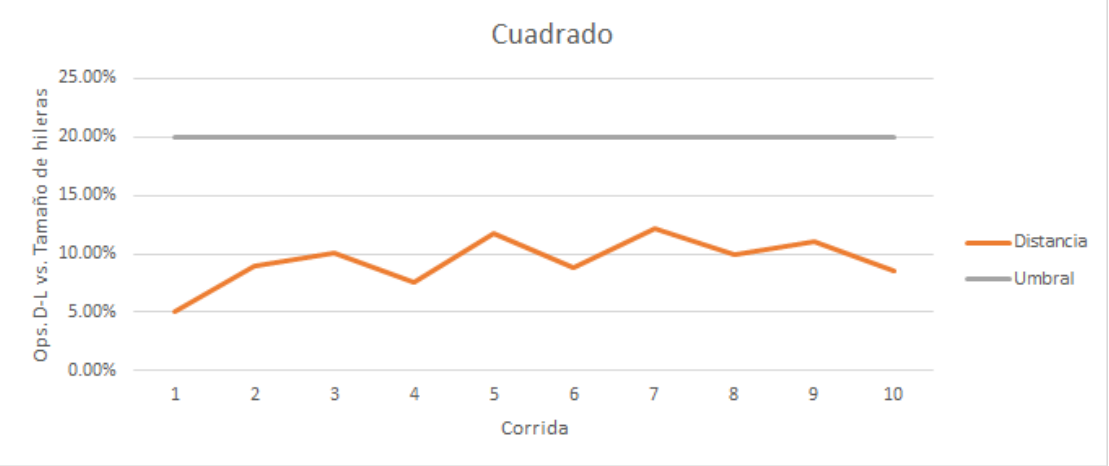


FIGURA B.4: Distancia y umbral vs. corridas de gestos de cuadrado

Gesto	Corrida	Candidato	Correspondencia	Distancia	Umbral	Tiempo (ms)
Círculo	1	Círculo	No	24.34%	20%	632
	2	Círculo	No	21.17%	20%	904
	3	Círculo	No	24.80%	20%	368
	4	Círculo	No	26.47%	20%	153
	5	Círculo	No	24.04%	20%	462
	6	Círculo	No	20.22%	20%	113
	7	Círculo	No	21.73%	20%	418
	8	Cuadrado	No	25.90%	20%	517
	9	Círculo	No	27.08%	20%	18
	10	Círculo	No	23.50%	20%	703

CUADRO B.5: Resultados de gestos para primitiva de círculo

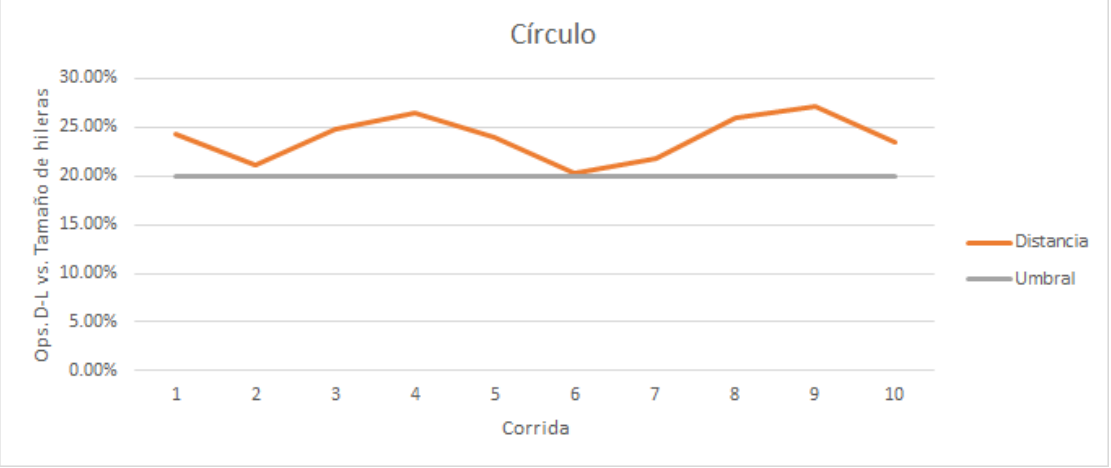


FIGURA B.5: Distancia y umbral vs. corridas de gestos de círculo

Gesto	Corrida	Candidato	Correspondencia	Distancia	Umbral	Tiempo (ms)
Línea adelante	1	Línea adelante	Sí	16.67%	20%	12
	2	Línea adelante	Sí	5.88%	20%	145
	3	Línea adelante	Sí	12.50%	20%	3
	4	Línea adelante	No	22.22%	20%	9
	5	Línea adelante	No	27.08%	20%	3
	6	Línea adelante	Sí	6.25%	20%	13
	7	Línea adelante	Sí	4.44%	20%	32
	8	Línea adelante	Sí	11.76%	20%	110
	9	Línea adelante	Sí	2.22%	20%	34
	10	Línea adelante	Sí	4.76%	20%	65

CUADRO B.6: Resultados de gestos para primitiva de línea hacia adelante

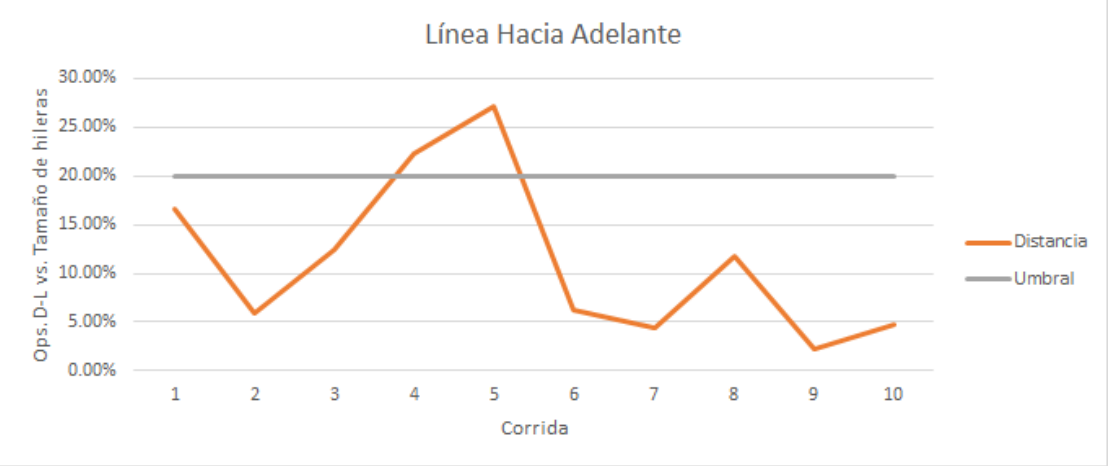


FIGURA B.6: Distancia y umbral vs. corridas de gestos de línea hacia adelante

Gesto	Corrida	Candidato	Correspondencia	Distancia	Umbral	Tiempo (ms)
Línea atrás	1	Línea atrás	Sí	11.76%	20%	112
	2	Línea atrás	No	29.41%	20%	110
	3	Línea atrás	No	29.41%	20%	108
	4	Línea atrás	No	31.25%	20%	3
	5	Línea atrás	No	25.00%	20%	4
	6	Línea atrás	Sí	16.67%	20%	8
	7	Línea atrás	Sí	4.44%	20%	15
	8	Línea atrás	Sí	6.67%	20%	23
	9	Línea atrás	Sí	6.67%	20%	18
	10	Línea atrás	Sí	11.76%	20%	115

CUADRO B.7: Resultados de gestos para primitiva de línea hacia atrás

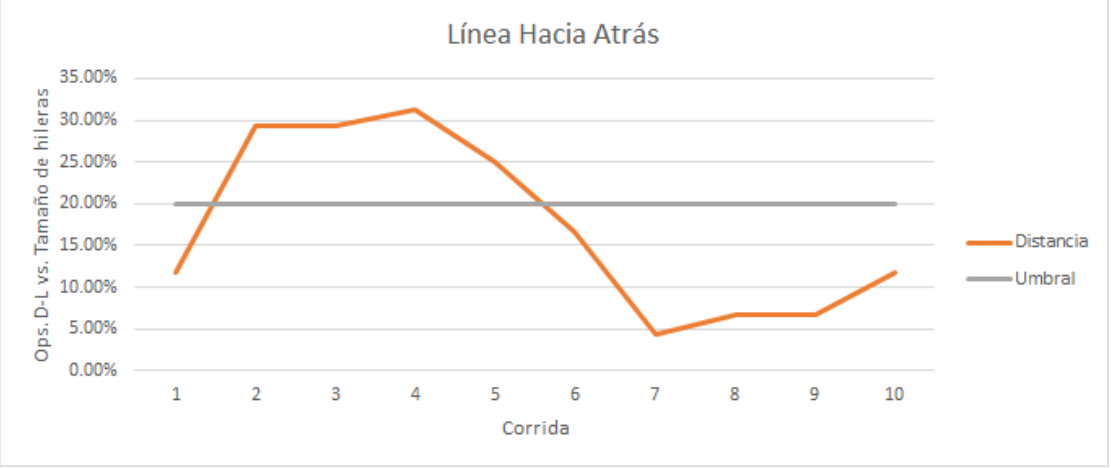


FIGURA B.7: Distancia y umbral vs. corridas de gestos de línea hacia atrás

Gesto	Corrida	Candidato	Correspondencia	Distancia	Umbral	Tiempo (ms)
Separar sensores	1	Separar sensores	Sí	16.67%	20%	5
	2	Separar sensores	Sí	14.29%	20%	51
	3	Separar sensores	Sí	11.11%	20%	15
	4	Separar sensores	Sí	9.53%	20%	26
	5	Separar sensores	Sí	17.65%	20%	109
	6	Separar sensores	Sí	15.38%	20%	103
	7	Separar sensores	Sí	14.29%	20%	20
	8	Separar sensores	Sí	13.33%	20%	38
	9	Separar sensores	Sí	16.67%	20%	3
	10	Separar sensores	Sí	13.33%	20%	9

CUADRO B.8: Resultados de gestos para primitiva de separar sensores

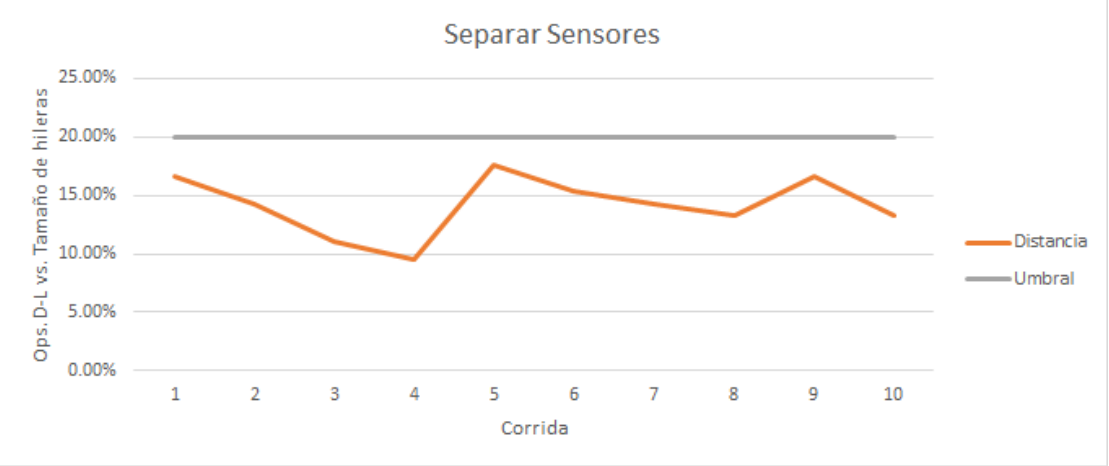


FIGURA B.8: Distancia y umbral vs. corridas de gestos de separar sensores

Juntar sensores	1	Juntar sensores	Sí	12.50%	20%	3
	2	Juntar sensores	Sí	12.82%	20%	126
	3	Juntar sensores	Sí	19.05%	20%	18
	4	Juntar sensores	Sí	19.05%	20%	21
	5	Juntar sensores	Sí	20.00%	20%	9
	6	Juntar sensores	Sí	17.78%	20%	26
	7	Juntar sensores	Sí	16.67%	20%	36
	8	Juntar sensores	Sí	12.82%	20%	62
	9	Juntar sensores	Sí	19.05%	20%	39
	10	Juntar sensores	No	24.44%	20%	18

CUADRO B.9: Resultados de gestos para primitiva de juntar sensores

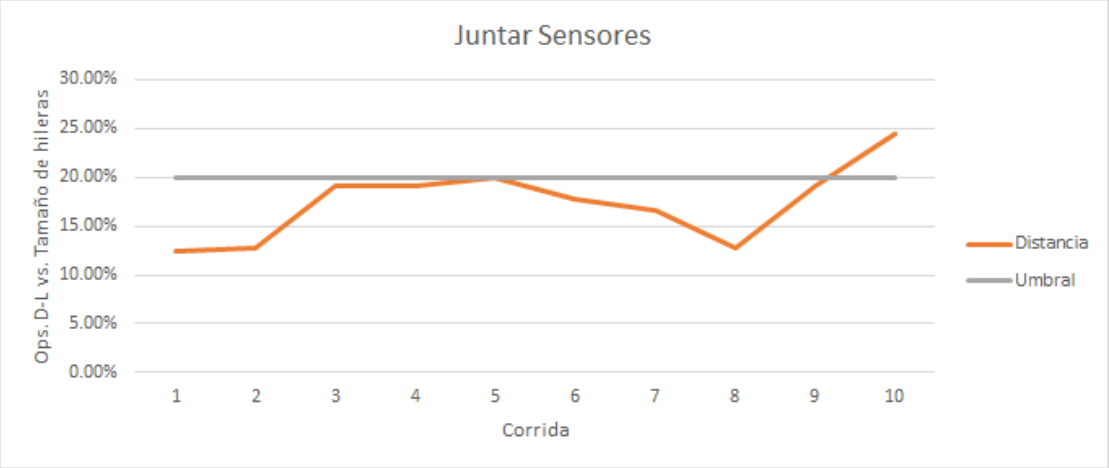


FIGURA B.9: Distancia y umbral vs. corridas de gestos de juntar sensores

Apéndice C

Resultados de Prueba 2

Corrida	Tiempo (s)	Errores
1	111	0
2	89	5
3	80	3
4	70	3
5	54	1
6	52	1
7	53	2
8	46	1
9	58	3
10	48	1

CUADRO C.1: Resultados consolidados de corridas de operación de navegador

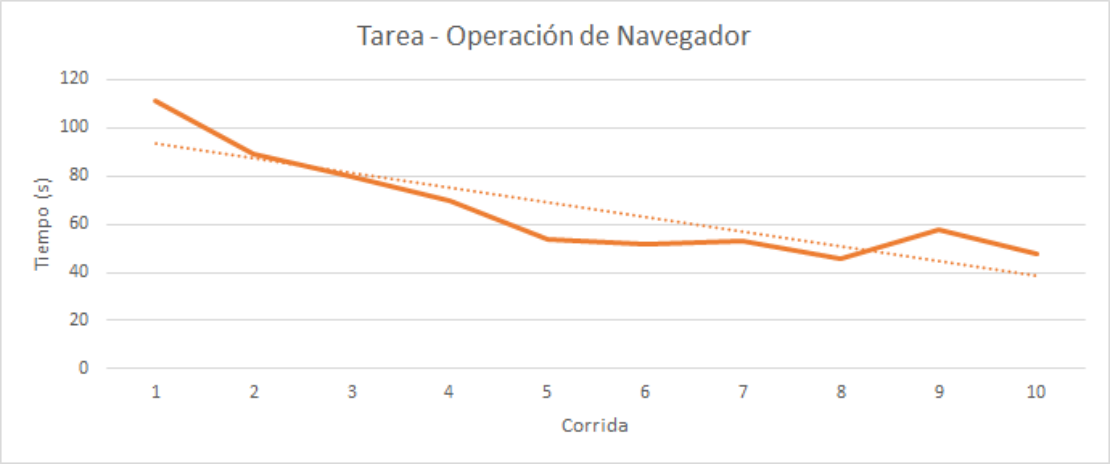


FIGURA C.1: Tiempo por cada corrida de operación de navegador

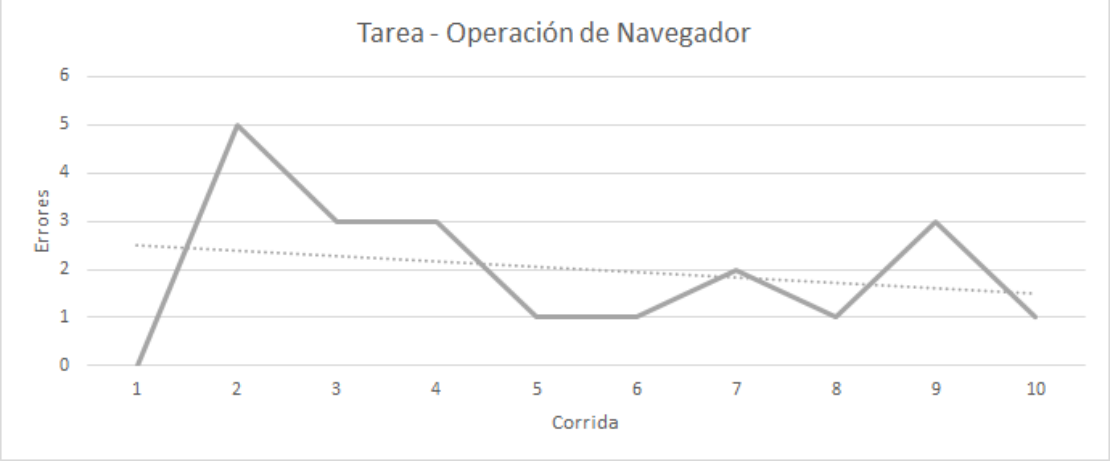


FIGURA C.2: Errores por cada corrida de operación de navegador

Gesto	Intento	Candidato	Correspondencia	Distancia	Umbral	Tiempo (ms)
OpenChrome	1	OpenChrome	Sí	14.81%	20%	469
	2	OpenChrome	Sí	19.34%	20%	50
	3	OpenChrome	Sí	19.40%	20%	259
	4	OpenChrome	No	21.13%	20%	385
	5	OpenChrome	No	21.28%	20%	40
	6	OpenChrome	Sí	15.73%	20%	124
	7	OpenChrome	No	25.64%	20%	191
	8	OpenChrome	Sí	16.24%	20%	53
	9	OpenChrome	Sí	16.24%	20%	114
	10	OpenChrome	No	20.28%	20%	59
	11	OpenChrome	No	22.39%	20%	58
	12	OpenChrome	Sí	14.45%	20%	83
	13	OpenChrome	Sí	15.13%	20%	106
	14	OpenChrome	Sí	15.65%	20%	217
	15	OpenGoogle	No	21.67%	20%	112
	16	OpenChrome	No	24.20%	20%	27
	17	OpenChrome	Sí	18.52%	20%	45

CUADRO C.2: Resultados de gestos para OpenChrome

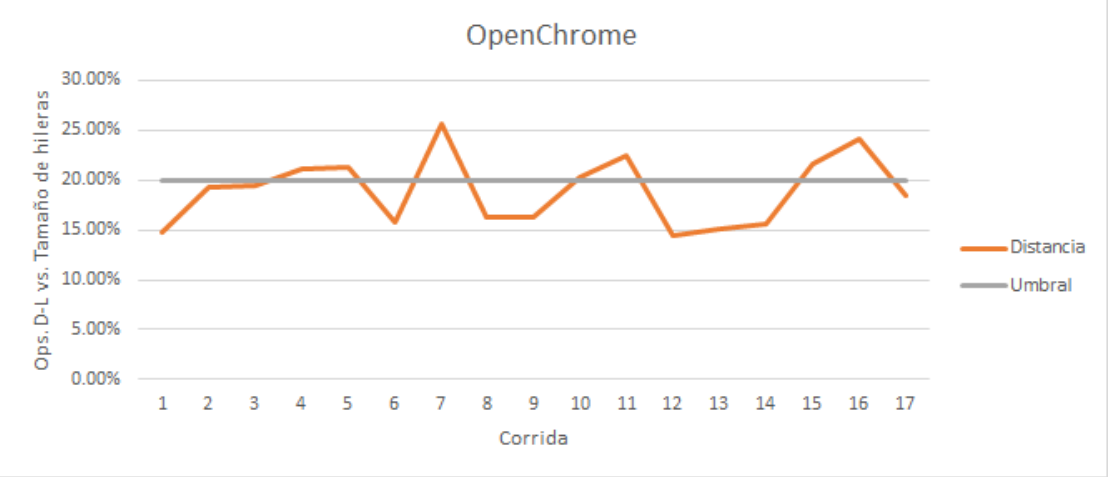


FIGURA C.3: Distancia y umbral vs. corridas de gestos de OpenChrome

Gesto	Intento	Candidato	Correspondencia	Distancia	Umbral	Tiempo (ms)
NewTab	1	NewTab	Sí	5.63%	20%	160
	2	NewTab	Sí	6.78%	20%	470
	3	NewTab	Sí	8.89%	20%	102
	4	NewTab	Sí	15.19%	20%	205
	5	NewTab	Sí	11.71%	20%	257
	6	NewTab	Sí	15.08%	20%	279
	7	NewTab	Sí	13.08%	20%	45
	8	NewTab	Sí	7.41%	20%	192
	9	NewTab	Sí	2.31%	20%	45
	10	NewTab	Sí	3.59%	20%	42
	11	NewTab	Sí	6.18%	20%	144
	12	NewTab	Sí	6.67%	20%	47
	13	NewTab	Sí	5.13%	20%	41
	14	NewTab	Sí	7.83%	20%	39
	15	NewTab	Sí	3.26%	20%	87
	16	NewTab	Sí	2.71%	20%	142
	17	NewTab	Sí	5.45%	20%	20
	18	NewTab	Sí	10.02%	20%	60
	19	NewTab	Sí	5.13%	20%	90
	20	NewTab	Sí	10.02%	20%	61

CUADRO C.3: Resultados de gestos para NewTab

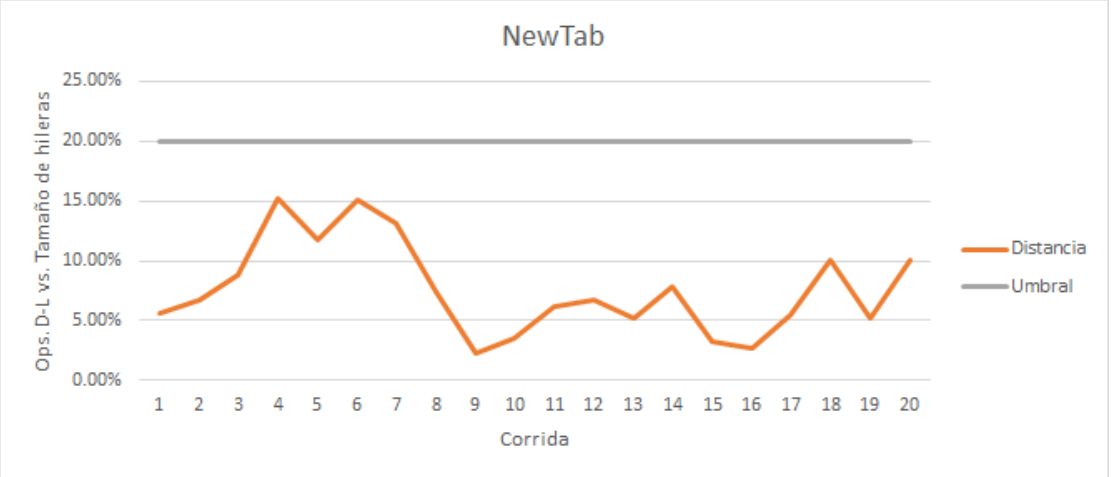


FIGURA C.4: Distancia y umbral vs. corridas de gestos de NewTab

Gesto	Intento	Candidato	Correspondencia	Distancia	Umbral	Tiempo (ms)
OpenCNN	1	OpenCNN	Sí	17.79%	20%	3157
	2	OpenCNN	Sí	14.58%	20%	507
	3	OpenCNN	Sí	16.71%	20%	1233
	4	OpenCNN	Sí	18.46%	20%	597
	5	OpenCNN	Sí	19.07%	20%	1209
	6	OpenCNN	Sí	17.90%	20%	1347
	7	OpenCNN	Sí	16.80%	20%	1359
	8	OpenCNN	Sí	19.31%	20%	187
	9	OpenCNN	No	22.67%	20%	1067
	10	OpenCNN	No	20.79%	20%	1151
	11	OpenCNN	Sí	17.27%	20%	945
	12	OpenCNN	Sí	15.43%	20%	697

CUADRO C.4: Resultados de gestos para OpenCNN

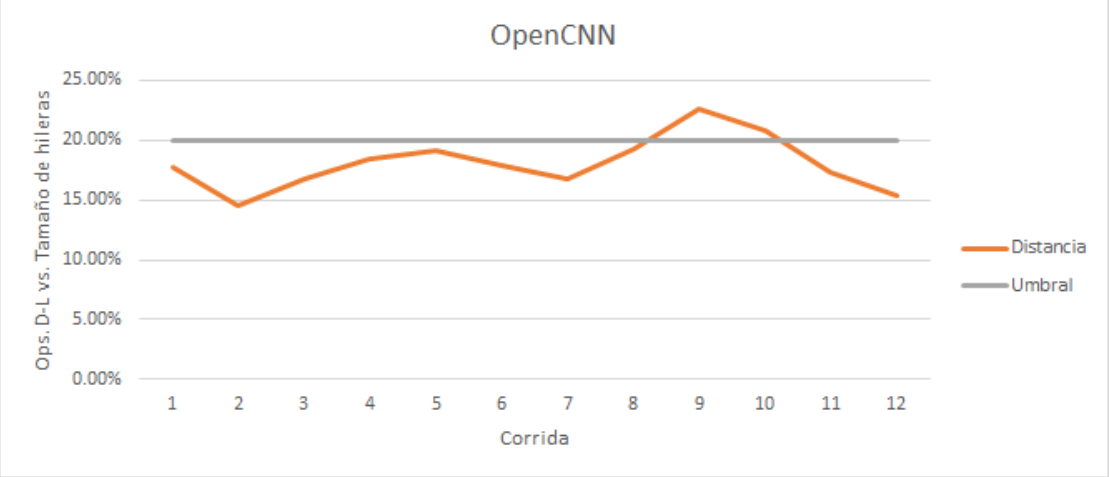


FIGURA C.5: Distancia y umbral vs. corridas de gestos de OpenCNN

Gesto	Intento	Candidato	Correspondencia	Distancia	Umbral	Tiempo (ms)
ZoomIn	1	ZoomIn	Sí	17.95%	30%	102
	2	ZoomIn	No	30.56%	30%	16
	3	ZoomIn	Sí	18.18%	30%	106
	4	ZoomIn	Sí	15.38%	30%	83
	5	ZoomIn	Sí	16.67%	30%	37
	6	ZoomIn	Sí	18.18%	30%	59
	7	ZoomIn	Sí	21.21%	30%	76
	8	ZoomIn	Sí	13.33%	30%	25
	9	ZoomIn	Sí	21.21%	30%	61
	10	ZoomIn	Sí	16.67%	30%	17
	11	ZoomIn	Sí	22.22%	30%	22

CUADRO C.5: Resultados de gestos para ZoomIn

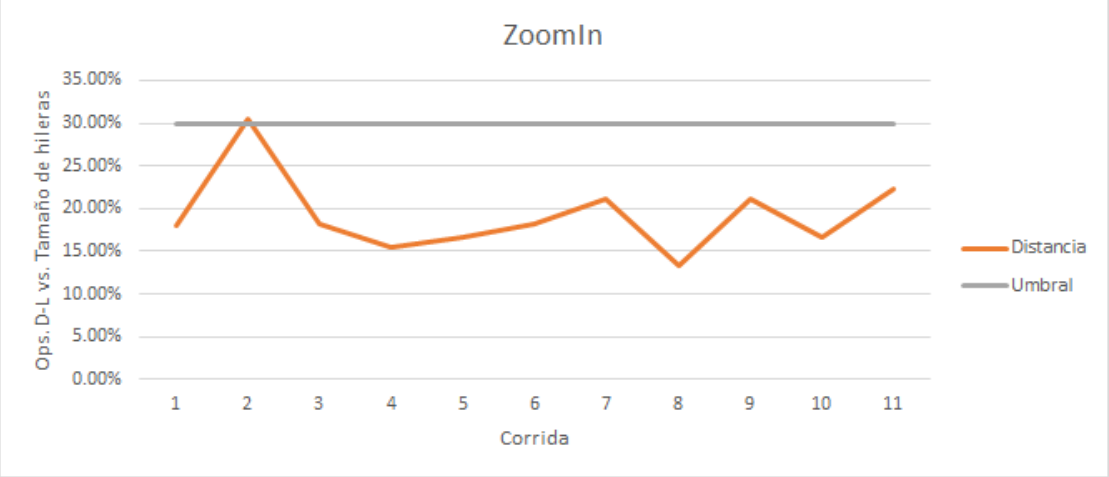


FIGURA C.6: Distancia y umbral vs. corridas de gestos de ZoomIn

Gesto	Intento	Candidato	Correspondencia	Distancia	Umbral	Tiempo (ms)
ZoomOut	1	ZoomOut	Sí	21.21%	30%	64
	2	ZoomOut	Sí	15.15%	30%	59
	3	ZoomOut	Sí	10.00%	30%	52
	4	ZoomOut	Sí	20.00%	30%	19
	5	ZoomOut	Sí	20.51%	30%	76
	6	ZoomOut	Sí	11.11%	30%	24
	7	ZoomOut	Sí	9.09%	30%	59
	8	ZoomOut	Sí	16.67%	30%	19
	9	ZoomOut	Sí	6.67%	30%	22
	10	ZoomOut	Sí	11.11%	30%	22
	11	ZoomOut	Sí	8.33%	30%	11

CUADRO C.6: Resultados de gestos para ZoomOut

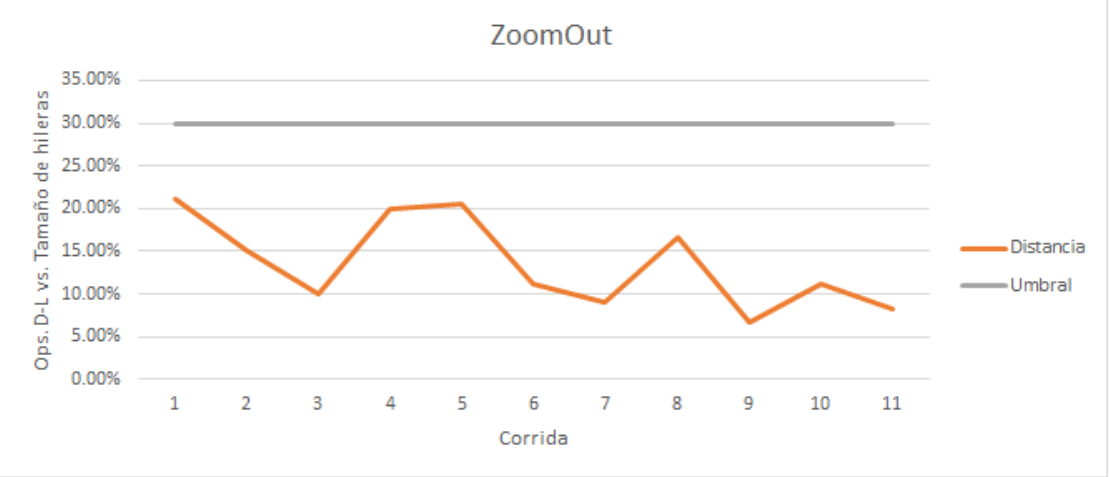


FIGURA C.7: Distancia y umbral vs. corridas de gestos de ZoomOut

Gesto	Intento	Candidato	Correspondencia	Distancia	Umbral	Tiempo (ms)
RefreshTab	1	RefreshTab	Sí	16.67%	30%	84
	2	RefreshTab	Sí	24.85%	30%	192
	3	RefreshTab	Sí	27.88%	30%	101
	4	RefreshTab	Sí	27.55%	30%	188
	5	RefreshTab	Sí	18.94%	30%	94
	6	RefreshTab	Sí	22.92%	30%	21
	7	RefreshTab	Sí	27.50%	30%	58
	8	RefreshTab	Sí	22.54%	30%	92
	9	RefreshTab	Sí	22.92%	30%	29
	10	RefreshTab	Sí	22.70%	30%	166

CUADRO C.7: Resultados de gestos para RefreshTab

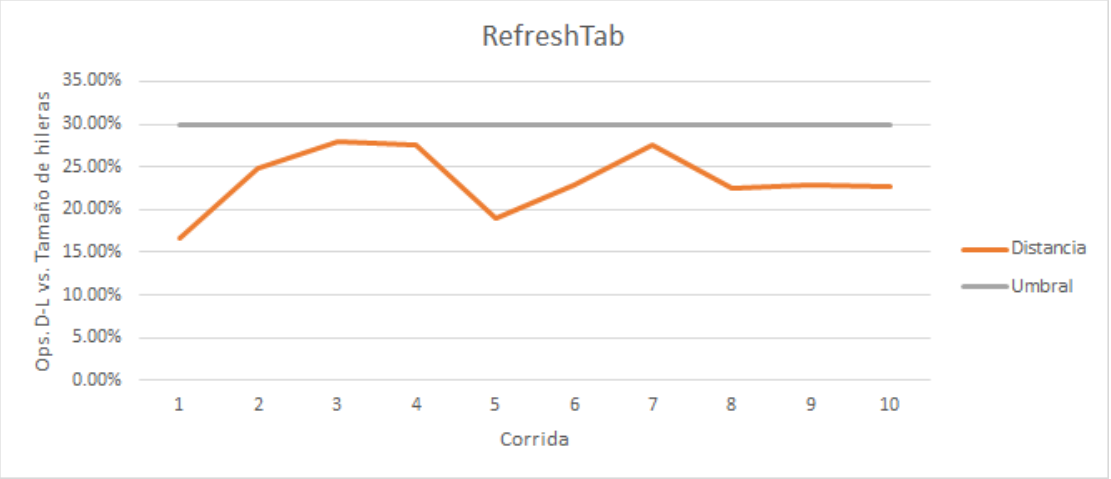


FIGURA C.8: Distancia y umbral vs. corridas de gestos de RefreshTab

Gesto	Intento	Candidato	Correspondencia	Distancia	Umbral	Tiempo (ms)
CloseTab	1	CloseTab	Sí	7.89%	20%	202
	2	CloseTab	Sí	4.39%	20%	136
	3	CloseTab	Sí	12.28%	20%	119
	4	CloseTab	Sí	11.71%	20%	677
	5	CloseTab	Sí	7.53%	20%	469
	6	CloseTab	Sí	8.23%	20%	352
	7	CloseTab	Sí	13.21%	20%	114
	8	CloseTab	Sí	12.61%	20%	713
	9	CloseTab	Sí	7.16%	20%	46
	10	CloseTab	Sí	12.22%	20%	156
	11	CloseTab	Sí	8.89%	20%	104
	12	CloseTab	Sí	4.55%	20%	85
	13	CloseTab	Sí	10.14%	20%	255
	14	CloseTab	Sí	10.26%	20%	95
	15	CloseTab	Sí	8.01%	20%	272
	16	CloseTab	Sí	11.53%	20%	115
	17	CloseTab	Sí	7.33%	20%	139
	18	NewTab	No	21.87%	20%	188
	19	CloseTab	Sí	9.09%	20%	85
	20	CloseTab	Sí	13.33%	20%	40
	21	CloseTab	Sí	8.09%	20%	38

CUADRO C.8: Resultados de gestos para CloseTab

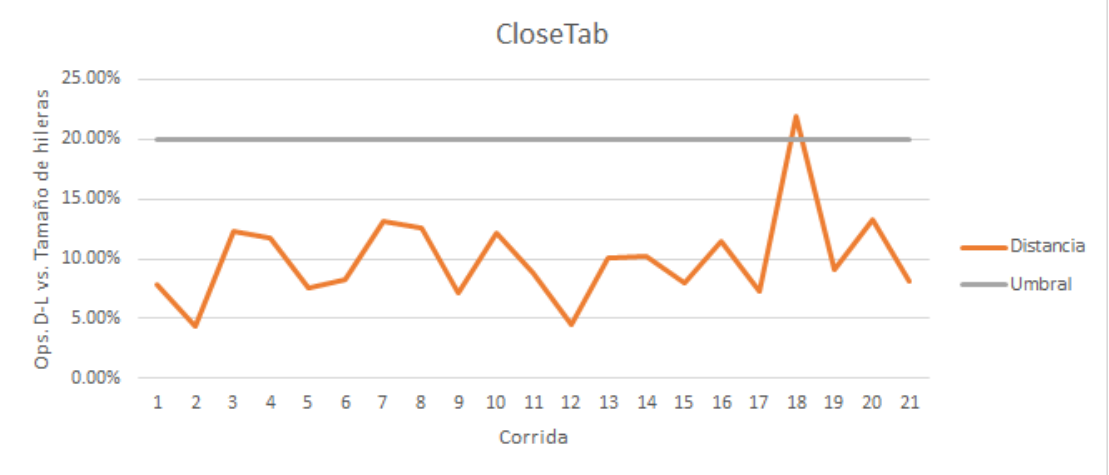


FIGURA C.9: Distancia y umbral vs. corridas de gestos de CloseTab

Gesto	Intento	Candidato	Correspondencia	Distancia	Umbral	Tiempo (ms)
OpenGoogle	1	OpenGoogle	Sí	16.02%	20%	809
	2	OpenGoogle	No	24.51%	20%	209
	3	OpenChrome	No	27.23%	20%	131
	4	OpenGoogle	No	20.42%	20%	89
	5	OpenGoogle	Sí	18.70%	20%	112
	6	OpenGoogle	No	21.11%	20%	98
	7	OpenGoogle	No	20.77%	20%	121
	8	OpenGoogle	No	23.46%	20%	266
	9	OpenGoogle	Sí	19.39%	20%	283
	10	OpenGoogle	No	20.59%	20%	673
	11	OpenGoogle	Sí	20.00%	20%	132
	12	OpenGoogle	Sí	13.67%	20%	272
	13	OpenGoogle	No	21.67%	20%	98
	14	OpenGoogle	Sí	19.74%	20%	162
	15	OpenGoogle	Sí	19.05%	20%	138
	16	OpenGoogle	No	20.54%	20%	464
	17	OpenGoogle	Sí	19.82%	20%	201
	18	OpenGoogle	Sí	19.13%	20%	114
	19	OpenGoogle	Sí	16.59%	20%	112

CUADRO C.9: Resultados de gestos para OpenGoogle

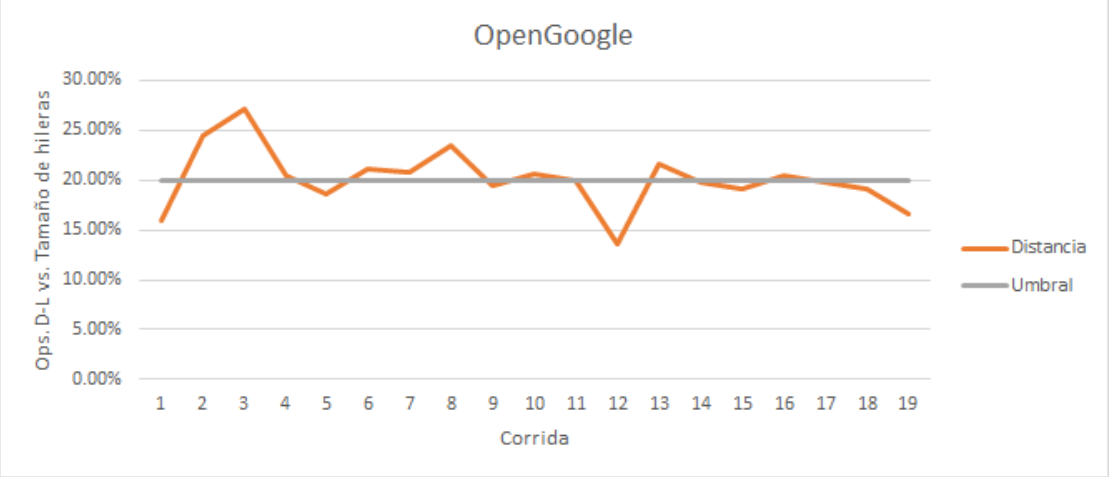


FIGURA C.10: Distancia y umbral vs. corridas de gestos de OpenGoogle

Referencias Bibliográficas

Bai, X., Yang, X., Yu, D. & Latecki, L. J. (2008). Skeleton-based shape classification using path similarity. *International Journal of Pattern Recognition and Artificial Intelligence*, *22*(04), 733-746. Recuperado desde http://knight.cis.temple.edu/~latecki/Papers/IJPRAI08.pdf

Bard, G. V. (2007, enero). Spelling-error tolerant, order-independent pass-phrases via the damerau-levenshtein string-edit distance metric. *Proceedings of the Fifth Australasian Symposium on ACSW Frontiers*. Recuperado desde http://crpit.com/confpapers/CRPITV68Bard.pdf

Boger, Y. (2013, 3 de noviembre). An interview with sebastien kuntz, CEO of i'm in VR.

Casiez, G., Roussel, N. & Vogel, D. (2012, 5 de mayo). 1 € filter: a simple speed-based low-pass filter for noisy input in interactive systems. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. doi:10.1145/2207676.2208639

Castle, A. (2011, 12 de septiembre). Razer hydra review. MaximumPC. Recuperado desde http://www.maximumpc.com/article/reviews/razer_hydra_review

Chacksfield, M. (2013, 24 de octubre). Microsoft reveals just how cosy you can get with kinect 2.0. TechRadar.

Chen, Q., Georganas, N. D. & Petriu, E. M. (2008). Hand gesture recognition using haar-like features and a stochastic context-free grammar. *IEEE Transactions on Instrumentation and Measurement*, *57*(8), 1562-1571. Recuperado desde http://www.discover.uottawa.ca/~qchen/my_papers/I%5C&M_published.pdf

Dengler, P., Grasso, A., Lilley, C., McCormack, C., Schepers, D. & Watt, J. (2011). Scalable vector graphics (SVG) 1.1 (second edition). W3C. Recuperado el 25 de junio de 2014, desde http://www.w3.org/TR/SVG11/

Feiner, S. (2014). *COMS w4172: 3d tracking technologies*. Columbia University. Recuperado desde http://monet.cs.columbia.edu/courses/csw4172/classes/3DtrackingNonoptical-14.pdf

Franklin, C. (2012). GesturePak. Recuperado el 21 de agosto de 2014, desde http://www.franklins.net/gesturepak.aspx

Hachaj, T. & Ogiela, M. R. (2014). Rule-based approach to recognizing human body poses and gestures in real time. *Multimedia Systems*, *20*(1), 81-99. Recuperado desde http://link.springer.com/content/pdf/10.1007/s00530-013-0332-2.pdf

Hamburger, E. (2014, 23 de julio). Facebook's new stats: 1.32 billion users, 30 percent only use it on their phone [The verge]. Recuperado el 21 de agosto de 2014, desde http://www.theverge.com/2014/7/23/5930743/facebooks-new-stats-1-32-billion-users-per-month-30-percent-only-use-it-on-their-phones

Han, C. Y., Everding, B. & Wee, W. G. (2004). Graph matching for object recognition and recovery. *Pattern Recognition*, *37*(7), 1557-1560. Recuperado desde http://www.cipprs.org/papers/VI/VI2003/papers/S8/S8_he_132.pdf

Hardware.Info. (2011). Razer hydra + portal 2. Hardware.Info. Recuperado desde http://us.hardware.info/productinfo/131486/razer-hydra-+-portal-2

Kuntz, S. (2014, 14 de agosto). [Re: master's thesis related to gesture recognition middleware (comunicación personal)].

Lang, B. (2013, 15 de mayo). Razer hydra 50 % off sale extended to end of may. Recuperado desde http://www.roadtovr.com/razer-hydra-50-off-vr-promo-sale-extended-may/

Leap Motion, Inc. (2014). Leap motion c++ SDK documentation. Recuperado desde https://developer.leapmotion.com/documentation/skeletal/cpp/index.html

Lee, J., Cakmak, M. & DePalma, N. (2008). Gesture recognition with temporally local to global representations. *Atlantic*, 1-6. Recuperado desde http://www.mendeley.com/download/public/2829371/3769593392/fde19992241d33f9d577e01abba7b1eb96b7c213/dl.pdf

Lin, J., Vlachos, M., Keogh, E. & Gunopulos, D. (2004). Iterative incremental clustering of time series. En *Extending database technology (EDBT)*. Recuperado desde http://www.cs.ucr.edu/~eamonn/Lin_wavelet_EDBT.pdf

Lü, H., Fogarty, J. & Li, Y. (2014). Gesture script: recognizing gestures and their structure using rendering scripts and interactively trained parts. CHI 2014. Toronto, Ontario, Canada. Recuperado el 13 de agosto de 2014, desde http://yangl.org/pdf/gscript.pdf

Majorek, K. A., Steczkiewicz, K., Muszewska, A., Nowotny, M., Ginalski, K. & Bujnicki, J. M. (2014, abril). The RNase h-like superfamily: new members, comparative structural analysis and evolutionary classification. *Nucleic Acids Research*, *42*(7), 4160-79. Recuperado desde http://nar.oxfordjournals.org/content/42/7/4160.long

Matthews, H. (2013, 29 de mayo). Razer hydra teardown part 2 [H's blog]. Recuperado desde http://howiem.com/wordpress/index.php/2013/05/29/razer-hydra-teardown-part-2/

Microsoft Corporation. (2014). Kinect for windows programming guide. Recuperado desde http://msdn.microsoft.com/en-us/library/hh855348.aspx

MiddleVR. (2015a). MiddleVR. Recuperado el 4 de julio de 2014, desde http://www.middlevr.com/middlevr-for-unity/

MiddleVR. (2015b). MiddleVR user guide - 6.10.4: accessing wand data. Recuperado desde http://www.middlevr.com/doc/current/#accessing-wand-data

MiddleVR. (2015c). Products comparison. Recuperado desde http://www.middlevr.com/middlevr-for-unity/product-comparison/

Mitra, S. & Acharya, T. (2007). Gesture recognition: a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, *37*(3), 311-324.

Ratanamahatana, C. A. & Keogh, E. (2004). Everything you know about dynamic time warping is wrong. 10th ACM SIGKDD international conference on knowledge discovery and data mining (KDD-2004). Seattle, WA. Recuperado desde http://www.cs.ucr.edu/~eamonn/DTW_myths.pdf

Real Academia Española. (2012). *Gesto*. En *Diccionario de la real academia española* (22.ª ed.).

Reiling, F. (2014). *Toward general purpose 3d user interfaces: extending windowing systems to three dimensions* (Tesis doctoral, California Polytechnic State University). Recuperado desde https://github.com/evil0sheep/MastersThesis/blob/master/thesis.pdf?raw=true

Sebastian, T. B. & Kimia, B. B. (2005). Curves vs skeletons in object recognition. *Signal Processing*, *85*(2), 247-263. Recuperado desde http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.64.8757%5C&rep=rep1%5C&type=pdf

Shutterstock. (2014). Rose of the winds. Recuperado desde http://image.shutterstock.com/display_pic_with_logo/356785/115847068/stock-vector-compass-wind-rose-wind-icon-vector-illustration-115847068.jpg

Sixsense Entertainment, Inc. (2012). Razer hydra FAQ — sixsense. Recuperado desde http://sixense.com/hydra_faq

Sixsense Entertainment, Inc. (2014). STEM system — sixsense. Recuperado desde http://sixense.com/wireless

Superannuation. (2014, 15 de enero). How much does it cost to make a big video game? [Kotaku]. Recuperado el 21 de agosto de 2014, desde http://kotaku.com/how-much-does-it-cost-to-make-a-big-video-game-1501413649

Taylor II, R. M., Yang, R., Weber, H. & Hudson, T. (2014a). Virtual reality peripheral network. Recuperado desde http://www.cs.unc.edu/Research/vrpn/

Taylor II, R. M., Yang, R., Weber, H. & Hudson, T. (2014b). VRPN from the application's point of view. Recuperado desde http://www.cs.unc.edu/Research/vrpn/app_point_of_view.html

Taylor II, R. M., Yang, R., Weber, H. & Hudson, T. (2014c). Vrpn_tracker_remote. Recuperado desde http://www.cs.unc.edu/Research/vrpn/vrpn_Tracker_Remote.html

Webb, R. (2015). Stella: polyhedron navigator. Recuperado desde http://www.software3d.com/Stella.php

Welch, C. (2014, 25 de marzo). Facebook buying oculus VR for $2 billion [The verge]. Recuperado el 1 de agosto de 2014, desde http://www.theverge.com/2014/3/25/5547456/facebook-buying-oculus-for-2-billion

Wikipedia. (s.f.-a). *Rhombicuboctahedron*. En *Wikipedia*. Recuperado desde https://en.wikipedia.org/wiki/Rhombicuboctahedron

Wikipedia. (s.f.-b). *Spherical coordinate system*. En *Wikipedia*. Recuperado desde http://en.wikipedia.org/wiki/Spherical_coordinate_system

WiredEarp. (2011, 28 de noviembre). Razer hydra teardown. [Meant to be seen]. Recuperado desde http://www.mtbs3d.com/phpBB/viewtopic.php?f=120%5C&t=14036#p67251

Wölfel, M. (2012). Kinetic space. Recuperado el 21 de agosto de 2014, desde https://code.google.com/p/kineticspace/