



PROYECTO FINAL

Gestos en dispositivos móviles

Tutor

GÓMEZ, Leticia

Integrantes

MATA SUÁREZ, Andrés

MERCHANTE, Mariano

Agosto 2014

ÍNDICE DE CONTENIDOS

1. Introducción

2. Trabajos relacionados

3. Descubrimiento de gestos en entorno UCD

3.1. Objetivo

3.2. Diseño

3.2.1. Diseño del primer experimento

3.2.2. Videojuegos utilizados y descartados

3.3. Implementación

3.3.1. Aplicación mobile

3.3.2. Arquitectura general

3.3.3. Formato de escenarios

3.4. Análisis y evaluación

3.4.1. Procesamiento de gestos obtenidos

3.4.2. Procesamiento de entrevistas

3.5. Resultados

3.5.1. Gestos propuestos por escenario

3.5.2. Clasificación de gestos

4. Reconocimiento de gestos propuestos por usuarios

4.1. Objetivo

4.2. Diseño del segundo experimento

4.3. Implementación

4.3.1. Estandarización de gestos ejemplares

4.3.2. Limitaciones de los sensores

4.3.3. Dynamic Time Warping

4.3.4. Aplicación mobile

4.3.5. Reconocimiento offline

4.3.6. Conjuntos de entrenamiento

4.3.7. Utilidades

4.4. Resultados

4.4.1. Resultados y matriz de confusión

- 4.4.2. Dimensionalidad de la distancia mínima
- 4.4.3. Filtrado de señales
- 4.4.4. Gesto Shake
- 4.4.5. Gesto Orbital

5. Conjuntos de gestos no conflictivos y aplicación demostrativa

- 5.1. Conjuntos no conflictivos
- 5.2. Aplicación demostrativa

6. Conclusiones y trabajo futuro

7. Bibliografía

8. Apéndice

- 8.1. Archivo de escenarios utilizado por SensorLogger
- 8.2. Formato de archivo de salida de SensorLogger
 - 8.2.1. Ubicación
 - 8.2.2. Contenido
- 8.3. Repositorio
- 8.4. Arquitectura de las aplicaciones

1. INTRODUCCIÓN

Debido a la continua mejora y accesibilidad de la tecnología móvil, en los últimos años ha surgido un fuerte interés en nuevas formas de interacción con dispositivos móviles. En particular, y aprovechando los avances en los sensores integrados que estos dispositivos ofrecen, los gestos de movimiento han obtenido gran popularidad por la facilidad de uso para el usuario. Si bien los gestos de movimiento son un tema actual de investigación y desarrollo, en general los usuarios finales no han sido involucrados para la determinación de dichos gestos.

Para que los usuarios puedan utilizar gestos en forma natural al interactuar con cierta aplicación de software es fundamental descubrir precisamente cuáles serían los gestos que a ellos les resultan más intuitivos. Esto implica incorporar en forma temprana a un grupo de usuarios representativos de la comunidad que usará luego dicha aplicación y planificar cuidadosamente los escenarios de interacción que se les propondrá.

Es de particular interés utilizar el área de videojuegos como contexto para la investigación de gestos móviles, ya que actualmente es un área muy prometedora por la cantidad creciente de usuarios que interactúa con videojuegos, y no hay suficientes propuestas sobre nuevas interacciones usuario-dispositivo en este tipo de aplicaciones.

Frecuentemente, los videojuegos suelen ser aquellas plataformas en las cuales se ponen a prueba más seguido las capacidades intuitivas y creativas de los jugadores para resolver situaciones repentinas de manera eficiente. El jugador está en contacto con varios tipos de agentes y debe tomar decisiones, a menudo rápidamente, para poder sortear los obstáculos que se le presentan. Es por esto que se considera que la pantalla táctil ofrece poca flexibilidad al usuario, ya que se interpone entre el usuario y su fin, que es interactuar con el videojuego, y que debido a la velocidad de la interacción que un videojuego exige, deben encontrarse y utilizarse otros medios de interacción.



Figura 1. Ejemplo de videojuego móvil (DEAD TRIGGER) en el que la interface se superpone a la jugabilidad, complicando la interacción del usuario con el juego. Destacados están los botones táctiles que el usuario puede (y debe) presionar, ya sea intencionalmente o por error.

Dado que existe un espacio sin explorar en la interacción con videojuegos móviles, en el presente trabajo se plantea utilizar al Diseño Centrado en el Usuario (UCD) para el descubrimiento de los gestos. El diseño centrado en el usuario se caracteriza por considerar al usuario como parte del desarrollo, en un proceso iterativo en el que el usuario y el desarrollador interactúan continuamente. El usuario trabaja como evaluador del producto y devuelve feedback, ya sea explícita o implícitamente, mediante procesos de evaluación establecidos por el desarrollador (entrevistas, grabaciones, etc.).

Muchas veces, el diseño centrado en el usuario ignora las limitaciones técnicas que puedan llegar a influenciar el comportamiento del usuario. De esta forma, busca encontrar la forma de interacción humano-dispositivo más natural posible; es decir, asume que el humano posee una intuición que puede generalizarse. Las interfaces desarrolladas con esta premisa son denominadas Natural User Interface (NUI), y la interacción mediante gestos móviles caen dentro de esta categorización, ya que el movimiento gestual de un individuo es natural. Es por esto que se busca que los usuarios propongan, mediante un experimento UCD, los gestos más naturales e intuitivos para ciertas tareas.

El presente trabajo tiene la siguiente estructura. En la sección 2 se discutirán trabajos relacionados. En la sección 3 se discutirá cómo se utilizó la metodología UCD para descubrir los gestos que a los usuarios representativos les pareció intuitivos para su interacción con videojuegos. La sección 4 explica los algoritmos utilizados para poder identificar los gestos, las dificultades encontradas y soluciones propuestas, junto con los resultados del reconocimiento de dichos gestos. La sección 5 presenta una aplicación demostrativa de los resultados obtenidos. Por último, las conclusiones y reflexiones del trabajo, junto las posibles mejoras a futuro se describen en la sección 6.

2. TRABAJOS RELACIONADOS

Aunque las formas más comunes para ingresar información (input) a un dispositivo son teclado y mouse, existen otras alternativas que son adoptadas muy naturalmente por los usuarios. En el caso específico de los smartphones, sus superficies táctiles y su equipamiento con cámaras y sensores (acelerómetros, giróscopos, entre otros) ofrecen posibilidades de nuevos tipos de interacciones. Sin embargo, los usuarios reducen su uso a sólo cambiar la orientación de la pantalla.

En 1985 Hutchins et al. [1] en su publicación fundacional analizaron que cuando un objeto se representa en una computadora, su manipulación a través del dispositivo debe ser lo más similar posible a la manipulación que los usuarios harían sobre ese objeto en el mundo real. Eso da a los usuarios la sensación de manipulación directa. Cuando un usuario interactúa con un dispositivo lo hace a través del lenguaje que propone la interface, no el suyo. Cuanto menor es esa distancia, mayor es la sensación de manipulación directa. Por ejemplo, la "distancia" es menor cuando el usuario introduce una fórmula compleja a través de la selección de objetos y conectando la salida de uno con la entrada de otro (al estilo workflow) que cuando tiene que codificar la misma fórmula en un lenguaje de programación por medio de sentencias anidadas.

Asimismo analizaron las ventajas y desventajas de su uso. En particular para tareas que requieran mucha precisión o tareas repetitivas, la manipulación indirecta por medio de un lenguaje de programación o un script resultaría mejor. Para juegos, en cambio, el uso de manipulación directa se vuelve muy atractiva.

Ruiz et al. [2] en el 2011 analizaron el uso de sensores de movimiento en smartphones para detectar gestos tridimensionales realizados "explícitamente" por el usuario para reemplazar el uso de comandos (excluyendo a las superficies táctiles que detectan gestos bidimensionales). Para ello, condujeron una serie de experimentos en donde les propusieron a veinte participantes la realización de un conjunto predeterminado de tareas clasificadas en acciones (por ejemplo: atender la llamada, cortar la llamada, etc.) y navegación (zoom in, zoom out, ir a lo anterior, etc). Los resultados, por un lado, mostraron que efectivamente existe una correspondencia natural entre ciertos gestos y acciones. Esto los llevó a presentar heurísticas

de diseño y una taxonomía con el fin de facilitar la decisión de cuáles gestos resultan más útiles para determinados comandos, evitando que los diseñadores definan gestos propios para cada aplicación. Su trabajo se centró en aplicaciones generales que no requieran comandos específicos.

Negulescu et al. [3] realizaron estudios del uso de gestos en dispositivos móviles en situaciones de distracción, es decir cuando el usuario debe dividir su atención entre su propio movimiento físico y la interacción con el celular. Tal es el caso de chequeo de e-mails y mensajes de texto mientras se está manejando un auto o caminando. Su trabajo se centra en comparar gestos de movimiento y gestos en pantallas táctiles bajo la premisa de usuarios en distracción. Más aún, analizan la efectividad del uso de gestos de movimientos cuando el dispositivo no está visible para el usuarios (por ejemplo, se manipula dentro del bolsillo de un saco). Para delimitar el comienzo de los gestos de movimiento que usan acelerómetro (debido a la imperfección de la medición) usaron un gesto delimitador (double tap). La intensidad, tiempo, dirección, ángulo del dispositivo son parámetros que influenciaron la precisión de la detección del movimiento. Resaltaron que los parámetros varían según los usuarios y es preciso "detectar" los parámetros que mejor se ajustan por usuario. Si bien su estudio se centró en un grupo de usuarios expertos (entrenados) y los parámetros observados permitieron construir un modelo preciso para este tipo de usuarios, aconsejaron construir nuevos modelos si se usa con otros usuarios, para mejorar la performance de reconocimiento de los gestos.

Respecto al diseño de experimentos, la incorporación de usuarios a veces fue sólo usada para etapa de testeo y no para el diseño en sí de los gestos. Schwesig et al. [4,5] proponen un dispositivo físicamente deformable equipado con sensores de deformación. El mismo es usado para proporcionar acciones (zoom in, zoom out, etc.) y entrada de datos (selección de un carácter entre varios despleables en la pantalla). Aplicaciones como navegación en mapas se pueden beneficiar de esta forma de interacción. Sin embargo, los autores primero analizan cuáles son los gestos que pueden generarse con estos dispositivos y establecen a priori el mapeo entre gesto y comando. Luego, incorporan a usuarios para medir la aceptación de lo propuesto. Su objetivo consistió en observar cuán fácil le resultaba a un usuario entender la interacción propuesta de deformación. Los experimentos comenzaban explicándoles en 2 o 3 minutos cómo se usaba la interface deformable Gummi para navegación, zoom e ingreso de texto y los dejaban luego interactuar con la misma.

Descubrieron que a los usuarios les resultaba más intuitivo el uso de comandos tipo zoom o navegación que para ingreso de texto. Pero este descubrimiento fue realizado luego de haber determinado cuáles gestos serían mapeados a cuáles comandos. Es decir, su trabajo está centrado en tecnología y no en usuario.

En el 2009 Wobbrock et al. [6] prepararon un entorno de pruebas de usuarios provenientes de ámbitos no técnicos en donde a cada uno se le presentaba el efecto de un gesto de superficie (por ejemplo, el desplazamiento de un círculo de izquierda a derecha) y se le pedía que especifique el gesto para provocar dicho resultado. De estas pruebas se dedujo, entre otras cosas, que hay ciertas acciones que no provocan ningún acuerdo gestual implícito y que en ese caso son necesarios elementos UI. Su trabajo fue fuertemente motivado por UCD.

En el 2010, Sang-Su et al [7] analizan cómo podrían usarse superficies deformables para introducir gestos. Lo novedoso no es sólo que incorporan a usuarios desde el principio sino que les soliciten que sean ellos los que indiquen cuáles acciones les parecen que podrían hacerse con gestos de deformación. Es decir los usuarios son los que sugieren el mapeo entre acciones y gestos. Más aún, no usan ningún dispositivo electrónico que limite tecnológicamente el estudio, sino que analizan si el nivel de flexibilidad o deformación podría influenciar los gestos propuestos, por lo cual el estudio lo realizan usando tres tipos de materiales: plástico, papel y tela elástica.

Nuestro acercamiento al diseño de las pruebas con usuarios sigue la motivación UCD o User-Centered Design. Posterior a la clasificación de las acciones según su aplicabilidad en los distintos géneros de videojuegos en dispositivos móviles, se pensaron escenarios de ejecución de dichas acciones y se mostró un video ejemplar al usuario de algún juego en donde se esté realizando la acción para que visualice el efecto.

Debido a la clasificación de acciones en géneros de videojuegos, nuestra propuesta asegura que en muy raras ocasiones dos gestos iguales para acciones de distinta clase se encuentren en un mismo juego, por lo tanto el usuario no necesita visualizar todos los escenarios previamente para asociar gestos y acciones por preferencia, como ocurre en las pruebas de Wobbrock et al. [6].

Respecto del procesamiento de los datos de sensado en sí, Salvador et al. [8]

mencionan que el algoritmo DTW ordinario tiene una complejidad temporal cuadrática y su complejidad espacial limita su uso a volúmenes de datos pequeños. De todas las aproximaciones, FastDTW es la más precisa, por lo que optamos por ella al momento de hacer reconocimiento *realtime* en el dispositivo.

3. DESCUBRIMIENTO DE GESTOS EN ENTORNO UCD

3.1 OBJETIVO

En la primera etapa del proyecto, y utilizando las premisas del diseño centrado en el usuario, se busca encontrar un conjunto de gestos de movimiento intuitivos y naturales para usuarios de videojuegos.

Para esto se definió una secuencia de escenarios existentes en distintos tipos de juegos, donde se proponen acciones que no poseen un gesto asociado por defecto, para evitar ser influenciados por costumbres preexistentes. Luego se diseñó un experimento de diseño centrado en el usuario en el que se entrevistaron usuarios de videojuegos móviles, para preguntarles cómo harían, utilizando los sensores con los que los celulares vienen equipados, para interactuar con el videojuego y ejecutar comandos o acciones dentro del mismo.

En este trabajo se entiende por "gesto" a una secuencia de movimientos y/o interacciones con el dispositivo con el fin de comunicarle a la aplicación un comando o acción. Dado que los dispositivos móviles actuales poseen varios sensores que pueden aportar información útil, inicialmente un gesto podría estar compuesto por una señal de cualquier sensor, no necesariamente de movimiento. Sensores ejemplares de esto son el sensor de proximidad, sensor de humedad, sensor de temperatura o incluso de iluminación. Sin embargo, como se verá luego, en la práctica se terminaron utilizando pocos de estos sensores adicionales.

Debido a la naturaleza compleja de los gestos, una descripción verbal no es suficiente para poder definirlos y reconocerlos, por lo que utilizando una aplicación móvil desarrollada en la plataforma Android se registraron todas las señales generadas por cada usuario. Luego se analizaron y clasificaron con el objetivo de encontrar un consenso colectivo e intuitivo acerca de los gestos que representan cada acción.

3.2 DISEÑO

3.2.1 DISEÑO DEL PRIMER EXPERIMENTO

El experimento tuvo una fase de diseño inicial, donde se definieron los pasos a seguir y las herramientas que serían utilizadas. Son dos los dispositivos móviles que se decidieron usar, Galaxy Nexus y Nexus S, ambos donados por Google al Departamento de Ingeniería Informática del ITBA. Estos dos dispositivos poseen tres sensores: el acelerómetro, giroscopio y sensor de proximidad. Tiene sistema operativo Android Versión Gingerbread 2.3.

Un factor importante a considerar es el tipo de individuo que sería entrevistado. Debido a que el contexto de estos experimentos es el área de videojuegos, se consideró que hombres y mujeres entre 18 y 25 años serían adecuados para los experimentos, ya que son uno de los principales consumidores de videojuegos actuales. Es deseable que el usuario posea, al menos, un conocimiento mínimo de videojuegos y las formas de interactuar más comunes, ya que de esta forma no tendrán que enfrentarse a la curva de aprendizaje que cada género de videojuego posee.

Se convocó de manera individual a cada participante al aula donde se realizarían las pruebas. A cada uno, se le dio una breve descripción verbal del proyecto con la descripción de la motivación del mismo, el uso de la aplicación, y en particular, una explicación de cómo funcionan los tres sensores con los que se encuentran equipados los dispositivos. Además, se mostraron visualmente gestos poco conocidos para demostrar la flexibilidad del sistema; por ejemplo, un doble tap en la parte inferior derecha del celular.

Esta etapa es importante para evitar el sesgo que todos los usuarios familiares con tecnologías móviles tienen y ofrecerles un conjunto de primitivas gestuales más amplio.

La aplicación móvil **SensorLogger** fue instalada previamente en los dispositivos móviles de prueba que nos fueron brindados por el ITBA y a cada usuario se le otorgó uno para que realizara las pruebas.

Luego, se ejecutó la aplicación en el dispositivo, y a medida que el jugador avanzaba con cada escenario, se explicó el videojuego que representa, la acción que el jugador debe

ejecutar, y el contexto que sea necesario. Es importante notar que la aplicación desarrollada sólo indica el videojuego actual en formato textual, sin feedback visual de lo que el jugador estaría haciendo. Es por esto que se utilizaron videos para presentarle a cada jugador, en una computadora aparte, la jugabilidad de cada videojuego. Además, se le ofreció a cada jugador la posibilidad de proponer dos gestos distintos para la misma acción, ya que puede suceder que ciertas acciones posean varias soluciones instintivas simultáneamente.

Dieciséis estudiantes de la carrera de Ingeniería Informática se propusieron para hacer las pruebas. Se estimó que cada estudiante iba a tomar aproximadamente treinta minutos para grabar todos los gestos, por lo que se organizó un esquema de horarios adecuado, ya que es importante que cada usuario esté aislado y no vea los gestos que otras personas proponen, para evitar cualquier tipo de sugestión.

Cada usuario fue grabado con una cámara digital durante toda su participación. Este registro del accionar del usuario fue usado para realizar la clasificación de los gestos. Las preguntas que cada usuario formuló también se tuvieron en cuenta en el análisis.

3.2.2 VIDEOJUEGOS UTILIZADOS Y DESCARTADOS

Para el experimento es necesario definir un conjunto de escenarios en los que el usuario propone gestos. Para esto, se construyó una lista de juegos en los que existen acciones posibles de ser trasladadas a gestos, como saltar, disparar, moverse, etc.

La selección de dichos juegos es importante, ya que el objetivo es cubrir la mayor superficie de jugabilidad existente. Por esto, la lista está compuesta por los principales juegos de los géneros más comunes y traducibles al contexto móvil. Existen muchos géneros de videojuegos que actualmente son incompatibles con gestos móviles; esto principalmente se debe a la necesidad del mouse como principal forma de interacción, y al poco espacio utilizable en las pantallas táctiles actuales. Un ejemplo de esto son los videojuegos de estrategia en tiempo real, que requieren cierta destreza para manipular muchos controles, botones o indicadores en pantalla. El uso de gestos móviles en este tipo de juegos sería disruptivo con la jugabilidad planteada por este tipo de juegos (Ver figura 3.3.2.1).

Esto está cambiando gracias al avance y popularización de las tabletas y mayores tamaños de pantallas táctiles. Sin embargo, estos videojuegos serán descartados debido a que en este proyecto se utilizan celulares, que se caracterizan por pantallas relativamente chicas.

Se utilizaron juegos fácilmente reconocibles y, en lo posible, suficientemente sencillos como para que los usuarios puedan entenderlos sin problema. De cada uno de estos juegos se seleccionó un video representativo de la mecánica de juego y de la acción que requerimos que el usuario ejecute.

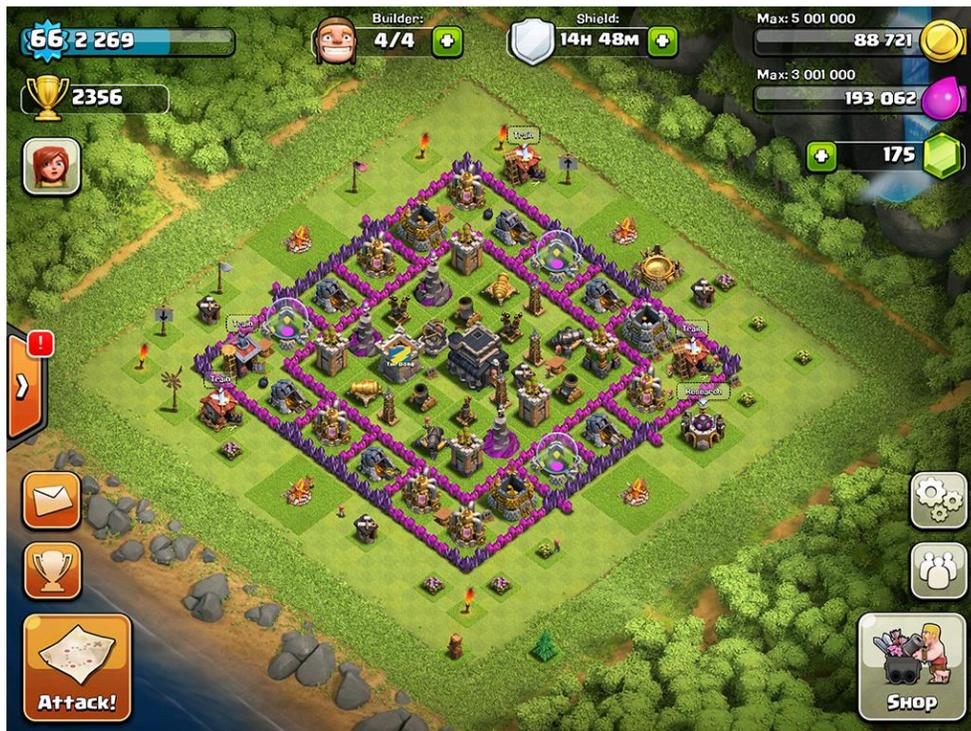


Figura 3.3.2.1. Clash of Clans.

A continuación se describe la lista de juegos que finalmente quedó seleccionada y utilizada en los experimentos. Cada videojuego posee una pequeña descripción del mismo y el género que representa.



Sidescroller 2D de pelea

Cadillacs & Dinosaurs

<https://www.youtube.com/watch?v=GohiKAhIm98>

El juego consiste en mover un personaje en un mundo 2D con una vista isométrica. El personaje puede moverse horizontalmente y también en el eje Z, o sea, profundidad.

Acciones

Mover al personaje en el eje de profundidad.



Juego de pelea 2D

Street Fighter

<https://www.youtube.com/watch?v=XLVoMW-OVyg>

El jugador tiene el control de un personaje. Puede saltar, moverse horizontalmente y atacar de distintas formas.

Acciones

Moverse horizontalmente.
Saltar.



Pinball 2D

Pinball

https://www.youtube.com/watch?v=Ww4s_c2qCzs

El jugador debe sumar la máxima cantidad de puntos evitando que la pelota caiga por un hueco, utilizando dos paletas.

Acciones

Lanzar la pelota.

Usa paletas.
Pegarle al tablero (tilt).



Juego de plataformas

Wonder Boy

<https://www.youtube.com/watch?v=Bvim7O9RvK0>

El jugador debe llegar al final del nivel sin morir, esquivando y/o matando enemigos. Puede obtener una patineta, que acelera su movimiento pero complica el juego.

Acciones

Frenar/acelerar personaje cuando usa skate.



First-person shooter

Dead Trigger

<https://www.youtube.com/watch?v=qILZwQsKiFM>

El jugador debe sobrevivir a hordas de zombies utilizando diversas armas de fuego.

Acciones

Recargar arma.
Cambiar de arma activa.



Shoot 'em up

Space Invaders

https://www.youtube.com/watch?v=437Ld_rKM2s

El jugador tiene el control de una nave que debe defenderse de enemigos invasores. Puede moverse horizontalmente y disparar, pero muere si recibe daño.

Acciones

Disparar.



Survival Horror

Amnesia

<https://www.youtube.com/watch?v=iXkCx4NyvFI>

El jugador debe resolver un misterio sin ser alcanzado por abominaciones y/o demonios.

Acciones

Mover la cámara.



Carreras de autos

Asphalt 8

<https://www.youtube.com/watch?v=L8F8DCsQYrA>

El jugador tiene el control de un vehículo y debe ganar carreras para juntar puntos y obtener nuevos autos.

Acciones

Manejar la caja de cambios.



Tenis

Wii Sports

<https://www.youtube.com/watch?v=grDpefPv0AQ>

El jugador tiene el control de un deportista de tenis y debe intentar ganar partidos.

Acciones

Mover la raqueta/Pegarle a la pelota. Lanzar y sacar.



Juegos de billar

3D Pool Game

<https://www.youtube.com/watch?v=oCJCtvAGP0Y>

Acciones

Disparar.

Juegos de mesa/cartas/dados

Truco

Acciones

Barajar cartas.

Lanzar dados.

Juegos de ingenio

Ajedrez

<https://www.youtube.com/watch?v=wPm9k6ul9EI>

Acciones

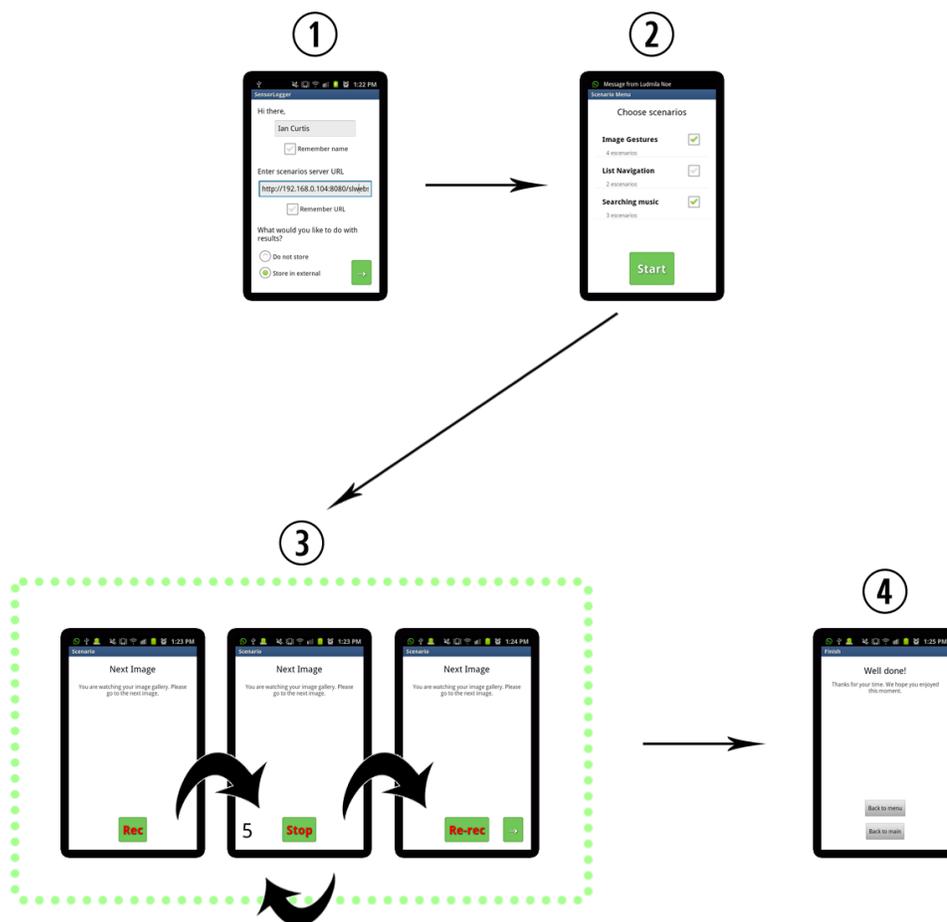
Pasar (terminar turno).

3.3 IMPLEMENTACIÓN

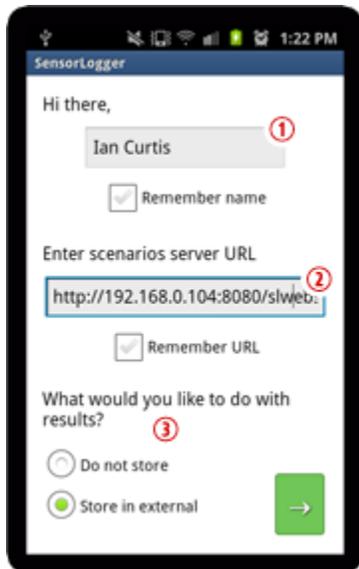
3.3.1 APLICACIÓN MOBILE

La aplicación **SensorLogger** fue desarrollada para utilizar en esta primera etapa de pruebas con usuarios. Su objetivo es automatizar el proceso de proposición de gestos de los usuarios para los escenarios elegidos. Éstos serían servidos a la aplicación a través de una URL, que se encargaría de presentarlos en secuencia a los usuarios, grabar el input y almacenarlo en la memoria externa del dispositivo en el cual se esté ejecutando. La aplicación no realizaría ningún tipo de procesamiento; tendría una función puramente colectora.

El flujo de **SensorLogger** es como se detalla a continuación:



Pantalla de bienvenida



Lógica MainActivity.java

Vista activity_main.xml

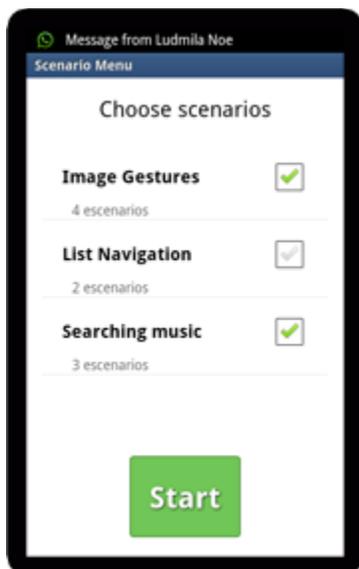
Se piden los datos básicos necesarios para comenzar a ejecutar la aplicación.

(1) Nombre de usuario. Para poder discriminar los archivos de output de cada usuario que ejecute la aplicación en un mismo dispositivo.

(2) URL del archivo de escenarios. El servidor de escenarios proveerá el archivo XML del cual se cargarán los escenarios de prueba.

(3) Persistencia de archivos de output. El usuario puede decidir almacenar los resultados en la memoria externa del dispositivo o no hacer nada.

Pantalla de selección de escenarios



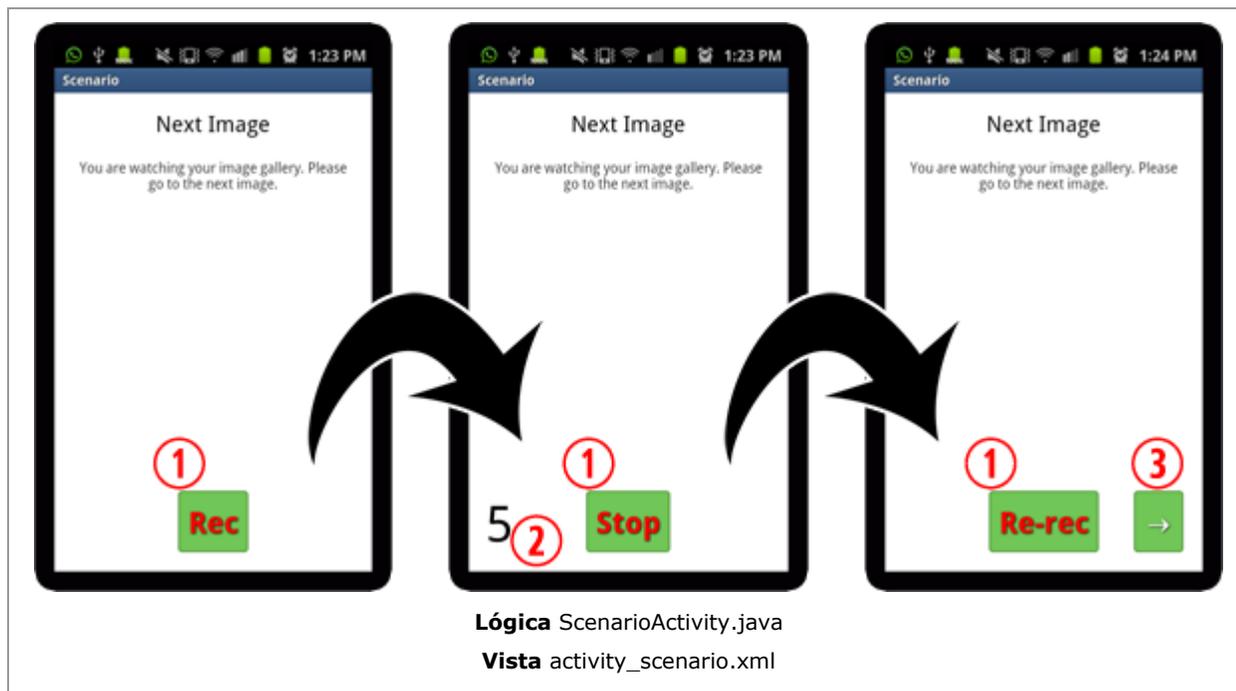
Lógica ScenarioMenuActivity.java

Vista activity_scenari_menu.xml

Una vez cargados los escenarios, se muestran al usuario clasificados en grupos para que elija cuáles ejecutar. Una vez conforme con su selección, dará paso al comienzo de las pruebas.

El botón de Start (comienzo de las pruebas) no se habilita hasta que el usuario no haya seleccionado al menos un grupo de escenarios a ejecutar.

Pantalla de ejecución de escenarios de prueba



Seleccionados los grupos de escenarios en la pantalla anterior, se procede a ejecutar cada escenario secuencialmente. Se muestra el grupo de pertenencia y la descripción de cada uno para que el usuario no se desoriente.

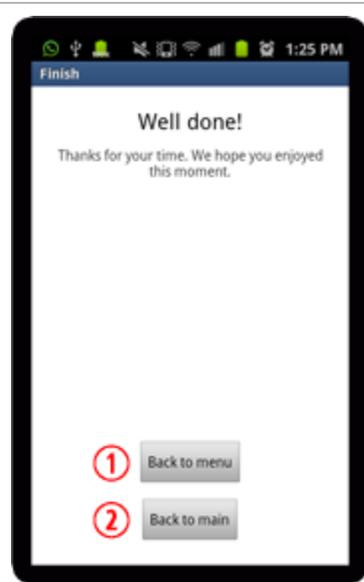
(1) Rec/Stop/Re-rec. Al presionar el botón **Rec**, el usuario está iniciando la grabación de todos los movimientos que los sensores habilitados para ese escenario detecten. En cualquier momento, el usuario puede interrumpir la grabación presionando **Stop**, y volver a comenzarla con **Re-rec**.

(2) Duración del escenario. Una vez comenzada la grabación, se habilitará en la pantalla del dispositivo una cuenta regresiva que representa la duración especificada en el atributo **duration** para el escenario en cuestión en el archivo de definiciones de escenarios. Una vez que la cuenta llegue a cero (o bien, si el usuario decidiera pausar la grabación con el botón **Stop**), se considerará como finalizada la recolección de datos de los sensores y queda a criterio del usuario regrabar o seguir adelante con el resto de los escenarios.

(3) Siguiente escenario. Si el usuario decidiese avanzar con las pruebas, este proceso

se repetirá de manera secuencial para cada escenario de cada grupo que se seleccionó en la pantalla anterior.

Pantalla de finalización



Lógica FinishActivity.java

Vista activity_finish.xml

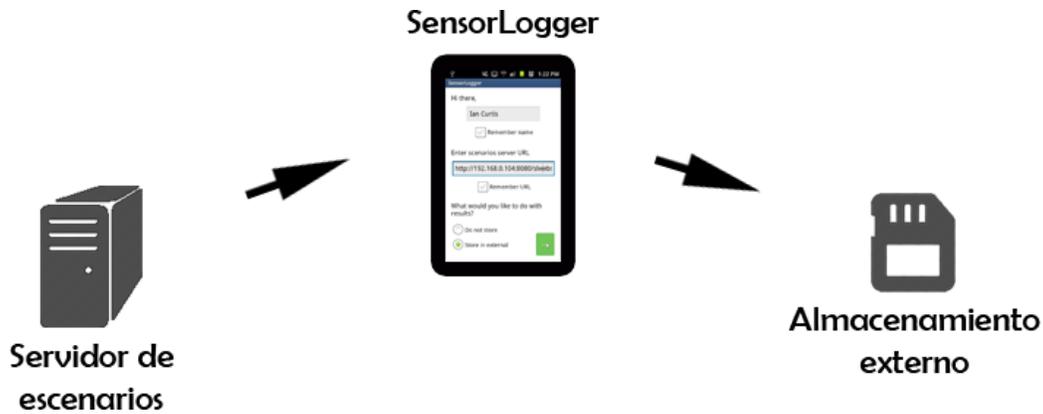
Finalizados todos los escenarios de prueba, se agradece al usuario por su cooperación y se le da la opción de volver al menú de selección de escenarios **(1)** o al principal **(2)**.

En caso de haber elegido almacenar la salida y una vez terminado el proceso, los datos del usuario quedan en los archivos de salida en la raíz de la memoria externa, en el directorio llamado SensorLogger.

3.3.2 ARQUITECTURA GENERAL

La arquitectura de la aplicación **SensorLogger** es similar a la de cualquier aplicación estándar de Android. La interfaz de usuario y los recursos se definen en archivos XML. Cada *activity* es una clase Java que representa una pantalla del flujo (que es inflada por algún archivo de layout) y contiene su lógica.

Se distinguen dos componentes principales con los cuales interactúa la aplicación:

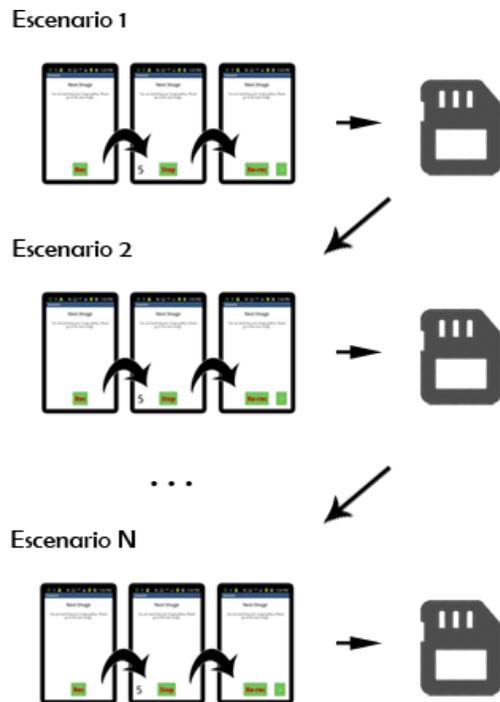


El servidor de escenarios provee el archivo XML con los escenarios que serán utilizados a lo largo del ciclo de la aplicación. La aplicación recibe la URL del servidor en la pantalla de bienvenida. El archivo XML debe respetar el formato detallado en la sección 3.3.3.

El servidor de escenarios es consultado una única vez a lo largo de todo el proceso. Una vez cargado, se mantiene una referencia global en memoria al conjunto de escenarios con la cual se sigue trabajando. Cada instancia de ScenarioActivity pasa a la siguiente el índice correspondiente en el conjunto de escenarios. En ningún momento se vuelve a consultar al servidor.

A medida que el usuario avanza por los escenarios, la aplicación va almacenando en memoria externa en streams el resultado de las grabaciones comprimido en DEFLATE. No se realiza un proceso batch de almacenamiento; cada escenario es almacenado antes de pasar al siguiente. Si el usuario decidiera rehacer la grabación, los datos ya almacenados se descartan y se comienza de nuevo.

La siguiente imagen describe a grandes rasgos el proceso de almacenamiento por escenario:



Los archivos de salida de la aplicación con la información de las pruebas se almacenan según [Apéndice 8.2.1] y siguen el formato especificado en [Apéndice 8.2.2].

3.3.3 FORMATO DE ESCENARIOS

Los escenarios de prueba se definen en un archivo XML con las siguientes características:

- El elemento raíz se denomina **scenarios**.
- **scenarios** contiene uno o más elementos **group**, que representan los grupos de escenarios, con los siguientes atributos obligatorios:
 - **id**: identificador único de grupo.
 - **name**: nombre del grupo.
- Cada **group** contiene uno o más elementos **scenario**, que representan cada escenario de prueba en cuestión, con los siguientes atributos obligatorios:
 - **id**: identificador único de escenario.
 - **name**: nombre del escenario.
 - **duration**: duración del escenario de prueba.

A su vez, un elemento **scenario** puede contener los siguientes hijos:

- **description**: instrucciones de lo que se debe realizar en el escenario.
- uno o más **sensor**: sensores involucrados en el escenario en cuestión. Los siguientes son todos los posibles valores para un elemento de este tipo:
 - ✓ ALL: todos los sensores disponibles en el dispositivo en el cual se ejecutarán las pruebas.
 - ✓ ACCELEROMETER
 - ✓ AMBIENT_TEMPERATURE
 - ✓ GRAVITY
 - ✓ GYROSCOPE
 - ✓ LIGHT
 - ✓ LINEAR_ACCELERATION
 - ✓ MAGNETIC_FIELD
 - ✓ ORIENTATION
 - ✓ PRESSURE
 - ✓ PROXIMITY
 - ✓ RELATIVE_HUMIDITY
 - ✓ ROTATION_VECTOR
 - ✓ TEMPERATURE

Por su parte, cada elemento **sensor** puede contener opcionalmente el atributo **delay** indicando el retardo que registrará durante las pruebas, con alguno de los siguientes posibles valores (estas constantes son *device-dependent*, por lo cual no tienen valores numéricos específicos asignados), listados de mayor a menor retardo:

- ✓ NORMAL: retardo ideal para cambios de orientación en la pantalla.
Valor por defecto.
- ✓ UI: retardo para actualización de la interfaz de usuario.
- ✓ GAME: retardo para aplicaciones de juegos.
- ✓ FASTEST: mínimo retardo, es decir, máxima velocidad de sensado.

En [Apéndice 8.1] se halla el archivo de escenarios que fue utilizado en estas pruebas de usuario.

Para cada una de las acciones de cada videojuego, se escribió un elemento **<scenario>** en el archivo de escenarios de la aplicación. Dicho elemento se agregó dos

veces al archivo para que el usuario tenga la oportunidad de proponer dos gestos distintos para una misma acción, si así lo quisiera.

3.4 ANÁLISIS Y EVALUACIÓN

3.4.1 PROCESAMIENTO DE GESTOS OBTENIDOS

Cada gesto propuesto por los usuarios consiste en una señal de varios sensores compuestos, registrado en un archivo comprimido por el dispositivo y luego analizado offline. Una vez recolectados todos los gestos, se utilizaron utilidades para inspeccionar empíricamente las formas de las señales, sus amplitudes y frecuencias, para tener una idea clara de qué tipo de gestos habría que reconocer en la segunda etapa del proyecto.

Sin embargo, para la clasificación de los gestos se encontró que utilizar las grabaciones captadas por la cámara digital resultaron mucho más eficaces, ya que se contaba con el feedback visual y auditivo de lo que cada usuario comentaba mientras proponía gestos.

Por último, en la segunda etapa del proyecto y con los gestos ejemplares ya definidos, se optó por realizar nuevas grabaciones con usuarios que no hayan participado del primer experimento. Los gestos propuestos por los primeros usuarios poseían muchas pequeñas variaciones (debido a que improvisaban los gestos en el momento) que les impedían ser utilizadas en un conjunto de entrenamiento, por lo que fue necesario obtener nuevas grabaciones.

3.4.2 PROCESAMIENTO DE ENTREVISTAS

Las entrevistas consistieron principalmente de la recolección de los sensores del dispositivo y comentarios que los usuarios realizaron espontáneamente y resultaron de interés al momento de clasificar los gestos. Muchas veces sucedió que un usuario grababa un gesto de una forma particular, pero realmente quería hacer otra cosa y luego lo comentaba. En particular, el gesto que consiste en pegarle al dispositivo en la parte inferior era problemático para el usuario, ya que perdía el control firme del dispositivo.

Este tipo de observaciones fueron consideradas al clasificar. Otro ejemplo claro es el gesto de Orbitar, un gesto que fue propuesto casi unánimemente para explorar un mundo virtual, y que los usuarios tenían problemas en definirlo con el dispositivo, pero que podían describir verbalmente sin problema. Debido a su naturaleza, este gesto es analizado más adelante.

Más allá de estas consideraciones, las grabaciones registradas con la cámara digital fueron de gran importancia, ya que es el principal medio por el cual se revisaron y clasificaron todos los gestos propuestos. Cada entrevista fue registrada en un video, y al momento de recolectar y clasificar cada gesto se revisaron todos los videos.

3.5 RESULTADOS

3.5.1 GESTOS PROPUESTOS POR ESCENARIO

A continuación mostraremos el glosario de gestos propuestos por los usuarios:

ID	Gesto
g1	Tilt hacia adelante
g2	Movimiento hacia arriba
g3	Movimiento hacia la persona
g4	Tilt hacia la derecha
g5	Movimiento hacia la derecha
g6	Rotación hacia la derecha
g7	Tilt hacia la izquierda
g8	Tap arriba/izquierda
g9	Mover horizontalmente
g10	Levantar dispositivo hacia arriba
g11	Levantar/Tilt hacia la persona
g12	Movimiento/Tilt hacia la persona
g13	Tilt hacia arriba
g14	Tilt/Tap rápido arriba
g15	Movimiento/Tilt hacia arriba
g16	Swing hacia arriba
g17	Tap abajo
g19	Movimiento/Tilt hacia direccion de salto
g20	Tilt desde abajo del celular
g21	Movimiento/Tilt hacia la persona y vuelta
g22	Swing lateral hacia la derecha
g23	Movimiento/Tilt desde arriba hacia abajo

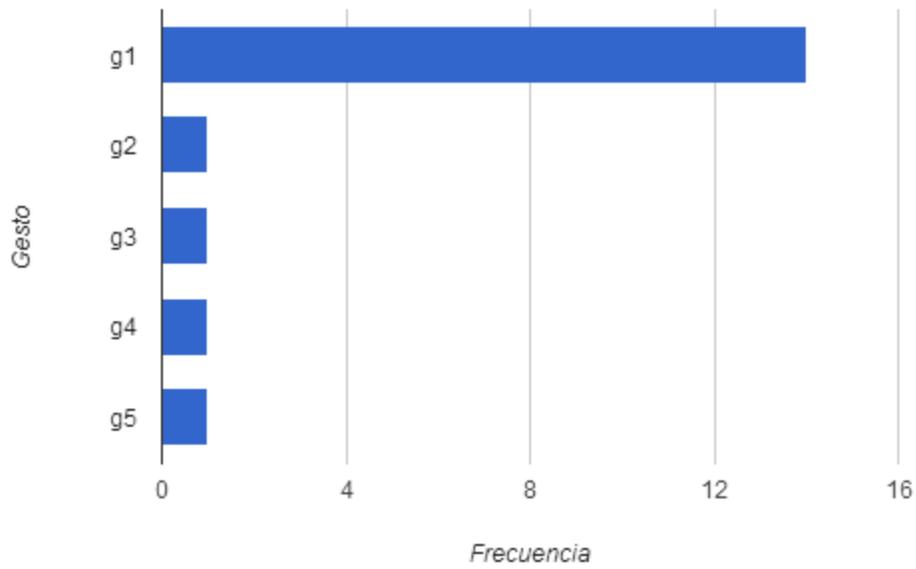
g24	Tilt hacia arriba, luego hacia abajo
g25	Movimiento hacia atrás, luego hacia adelante
g26	Swing hacia derecha
g27	Tap arriba/derecha
g28	Rotacion rapida en eje X
g29	Movimiento derecha/izquierda rápido en eje X
g30	Shake de movimiento lateral
g31	Shake de movimiento hacia la persona
g32	Movimiento hacia arriba/abajo
g33	Shake rápido en rotacion
g34	Doble tap lateral izquierda
g35	Doble tap lateral derecha
g36	Swing hacia donde se quiere tiltear el tablero
g37	Tap lateral derecha
g38	Shake/Pivot derecho
g39	Shake/tilt hacia persona
g40	Mover el celular hacia el fondo, mirando a la persona
g41	Rotacion hacia la persona
g42	Movimiento hacia la izquierda
g43	Rotación hacia la izquierda
g44	Swipe con la mano izquierda sobre el sensor de proximidad
g45	Tap atras/derecha
g46	Doble tap atrás
g47	Movimiento hacia abajo
g48	Rotacion/pivot con la mano derecha
g48	Tilt hacia abajo
g50	Tap atrás/izquierda
g51	Sensor de proximidad
g52	Swing hacia abajo/arriba pero hacia un lado (izquierda o derecha)
g53	Swing corto hacia arriba
g54	Tap atrás/centro
g55	Varios tilts, casi un shake rapido hacia persona
g56	Tilt hacia adelante varias veces
g57	Orbitar

g58	Doble tap atrás/derecha
g59	Doble tilt hacia la izquierda
g60	Tilt hacia adelante/atrás
g61	Doble tilt hacia persona
g62	Movimiento hacia adelante
g63	Movimiento/Tilt hacia abajo
g64	Swing de abajo hacia adelante/arriba
g65	Movimiento/Tilt hacia adelante
g66	Swing de abajo/izquierda hacia arriba/derecha
g67	Tilt de arriba hacia abajo
g68	Swing hacia adelante
g69	Tilt hacia atrás/adelante
g70	Párbola hacia adelante
g72	Movimiento hacia arriba y luego swing hacia abajo
g73	Swing hacia arriba y luego hacia un costado/abajo
g74	Movimiento adelante/atrás varias veces, terminando adelante
g75	Shake horizontal y movimiento hacia adelante
g77	Movimiento hacia la persona y luego hacia afuera
g78	Movimiento hacia la persona y luego hacia afuera, con shake previo
g79	Shake y luego tirar hacia adelante
g80	Shake y luego tirar hacia abajo
g81	Shake de tilt hacia adelante
g82	Movimiento hacia arriba/abajo rápido
g83	Swing lateral
g84	Swing/Pivot hacia adelante con mano izquierda
g85	Movimiento arriba, luego swing hacia abajo
g86	Shake con mano derecha
g87	Girar el dispositivo 180° y orientarlo contra el piso
g88	Shake y luego movimiento hacia adelante
g89	Shake y luego swing adelante
g90	Pegarle a la otra mano con el dispositivo
g91	Pegarle a la otra mano con el dispositivo, pero rotado (mas vertical), como si fuesen cartas
g92	Swing horizontal, con un leve freno en la segunda mano

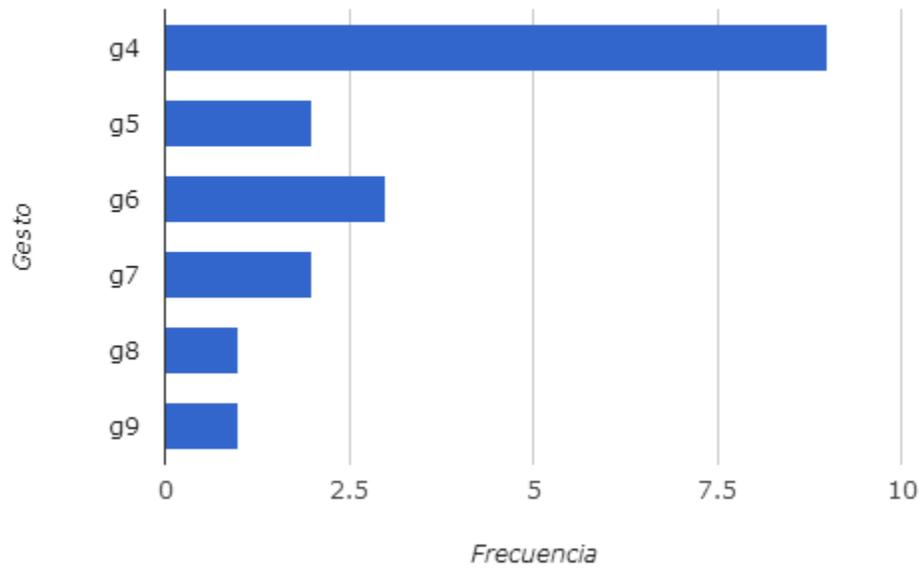
g93	Shake horizontal
g94	Shake hacia la persona
g95	Shake hacia la otra mano
g96	Shake rotación izquierda
g97	Shake vertical
g98	Shake de movimiento vertical con tap atrás
g99	Múltiples tilts hacia arriba
g100	Múltiples tap con la mano izquierda
g101	Shake como si estuviese tirando las cartas con la mano derecha, hacia abajo
g103	Dos rotaciones hacia adelante
g104	Parábola hacia un costado
g105	Lento movimiento/tilt vertical hacia la persona, luego rápido hacia abajo
g106	Lento movimiento/tilt vertical hacia la persona, luego rápido hacia abajo pero mas horizontal y sin tilting
g107	Movimiento hacia afuera de la persona, con el dispositivo mirando hacia arriba
g108	Movimiento hacia afuera de la persona, con el dispositivo mirando hacia arriba pero mirando hacia la persona
g109	Movimiento/Tilt arriba/abajo rápido
g110	Pequeño golpe abajo/derecha

A continuación se puede visualizar la frecuencia de los gestos elegidos por los usuarios en cada uno de los juegos.

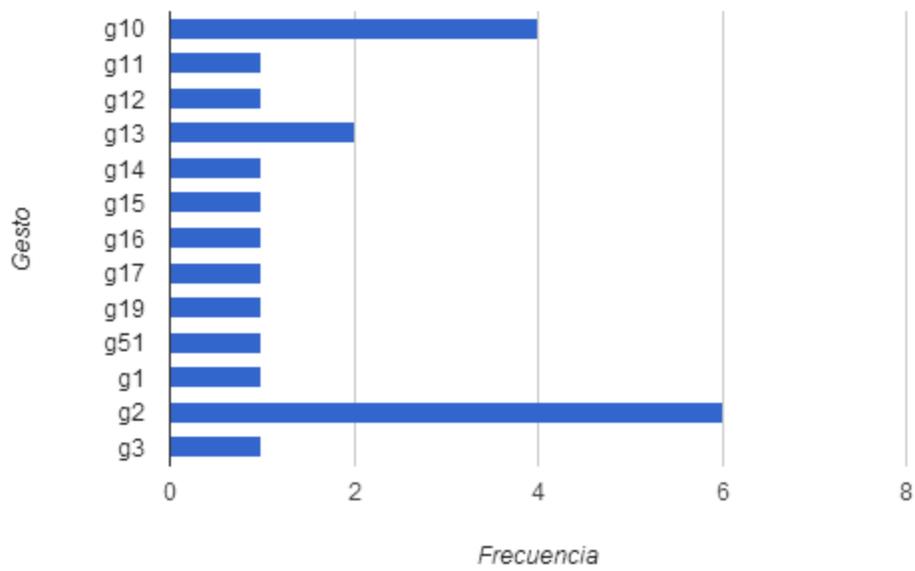
Cadillacs & Dinosaurs: Moverse en profundidad



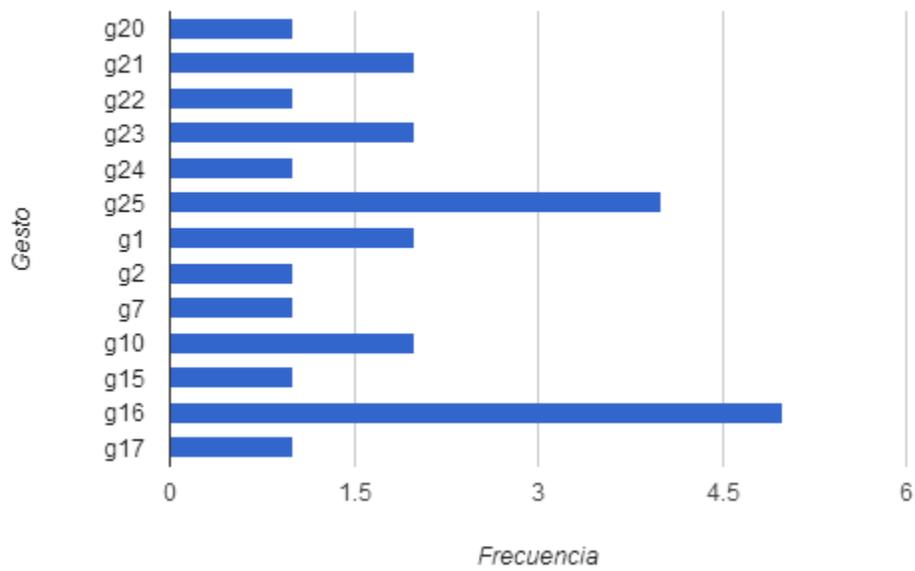
Street Fighter: Mover horizontalmente



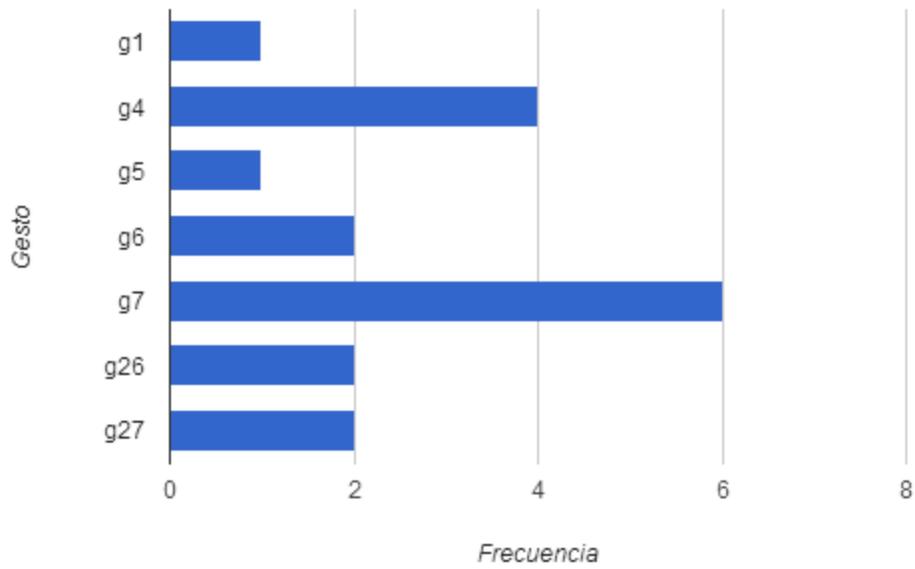
Street Fighter: Movimiento hacia arriba



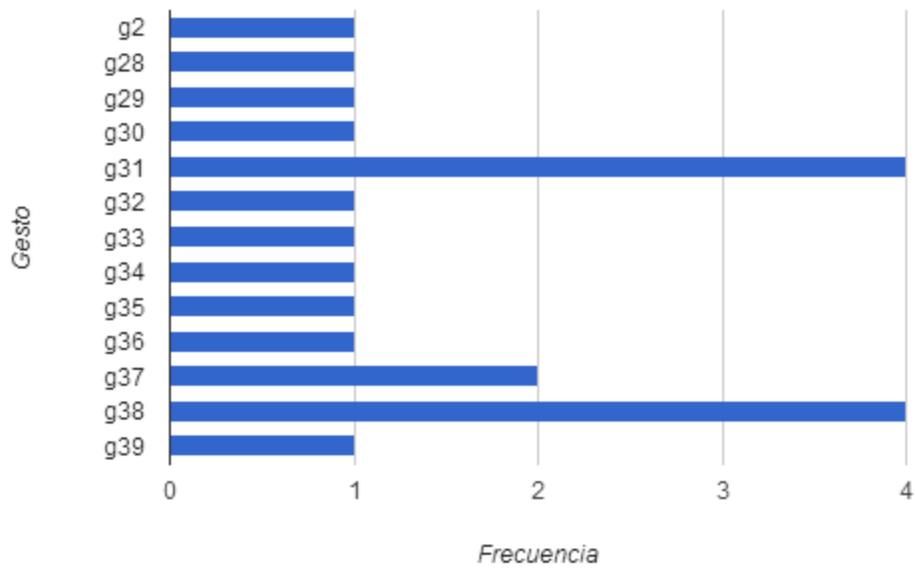
Pinball: Lanzar pelota



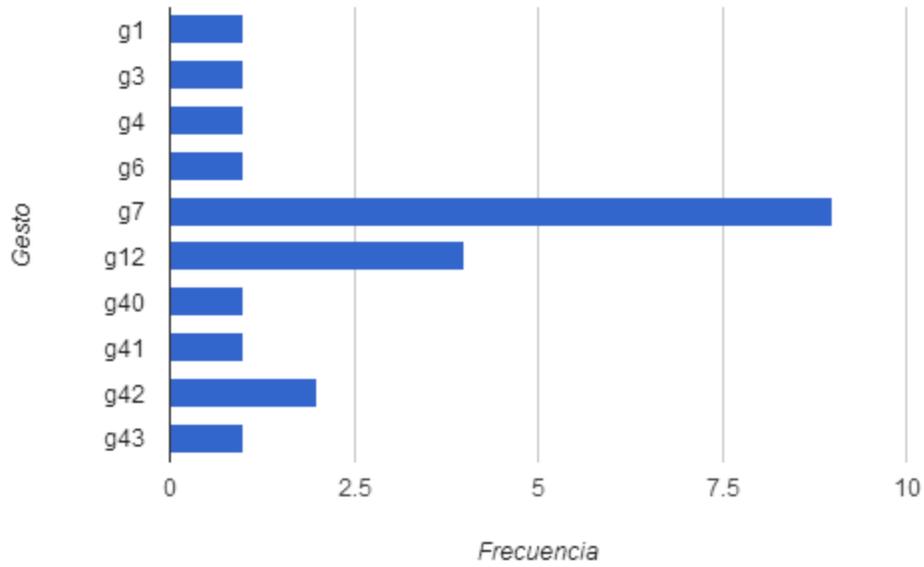
Pinball: Usar paleta (paleta derecha)



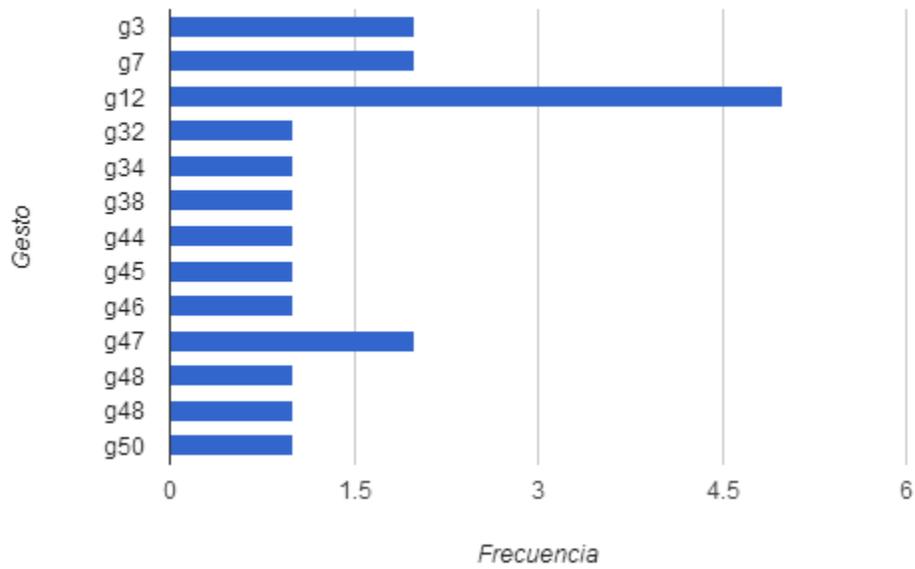
Pinball: Pegarle al tablero



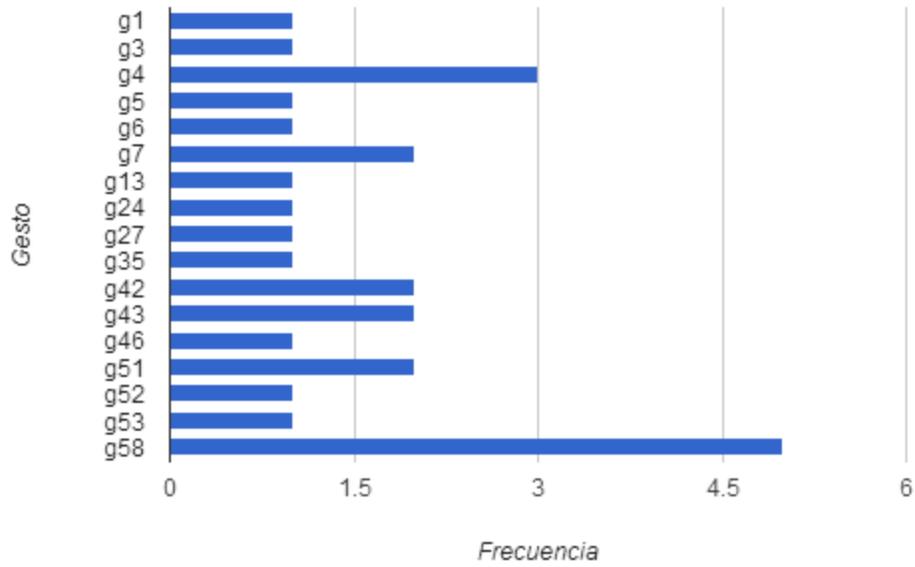
Wonder Boy: Frenar/Acelerar personaje



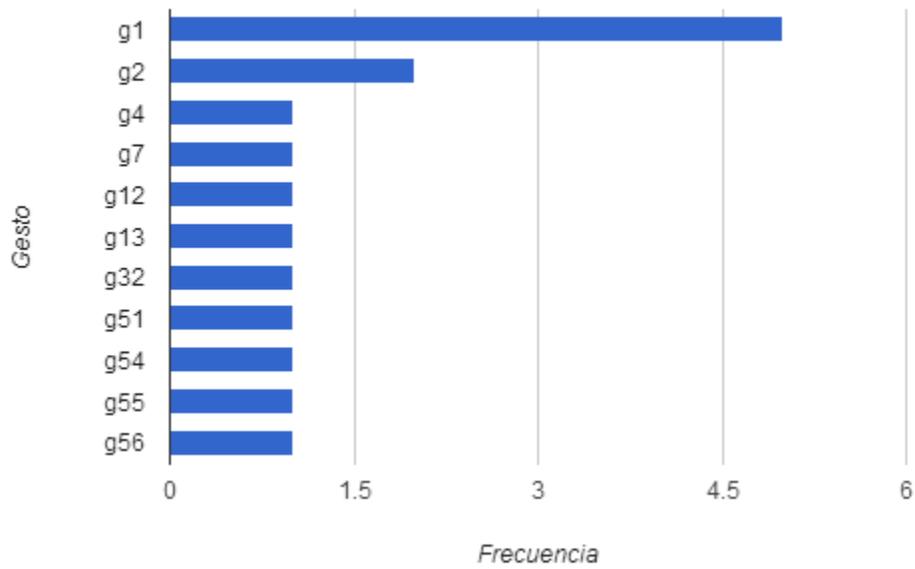
Dead Trigger: Recargar arma



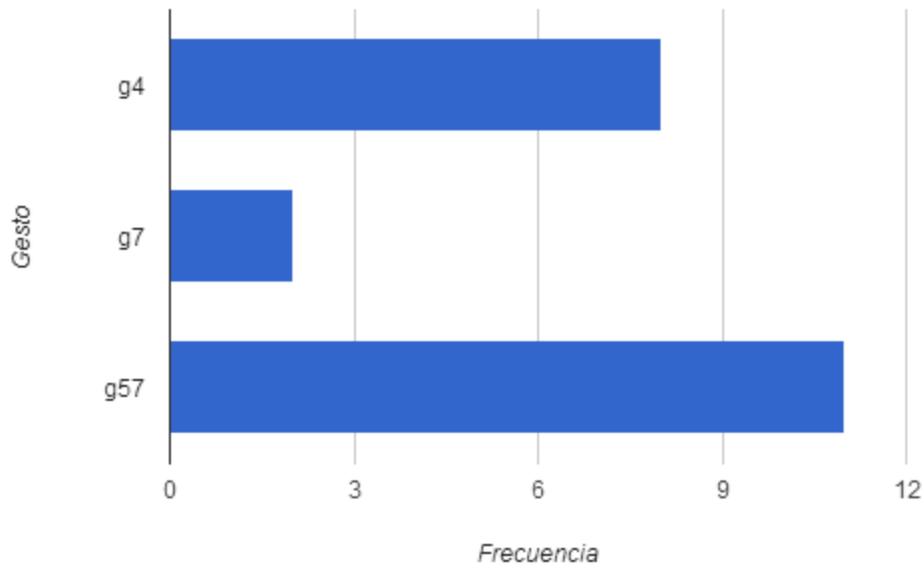
Dead Trigger: Cambiar de arma activa



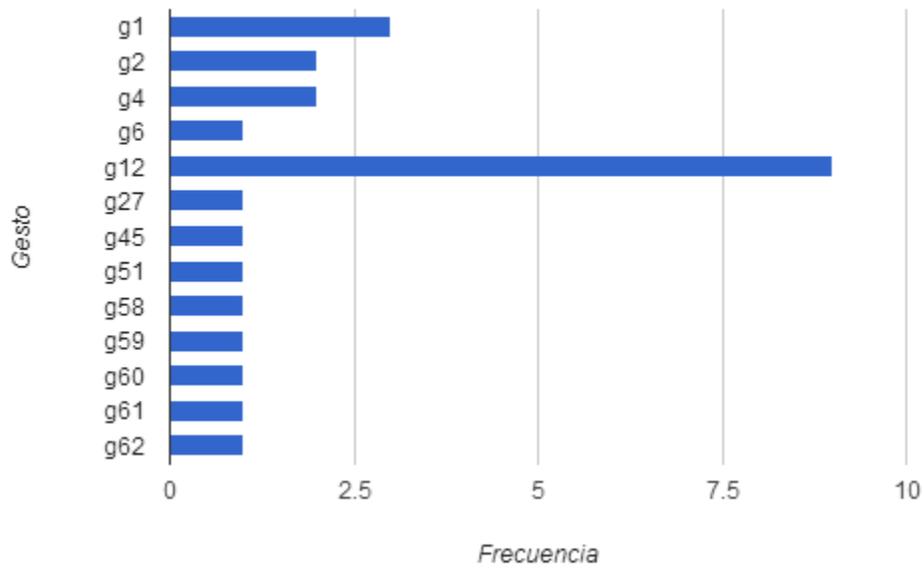
Space Invaders: Disparar



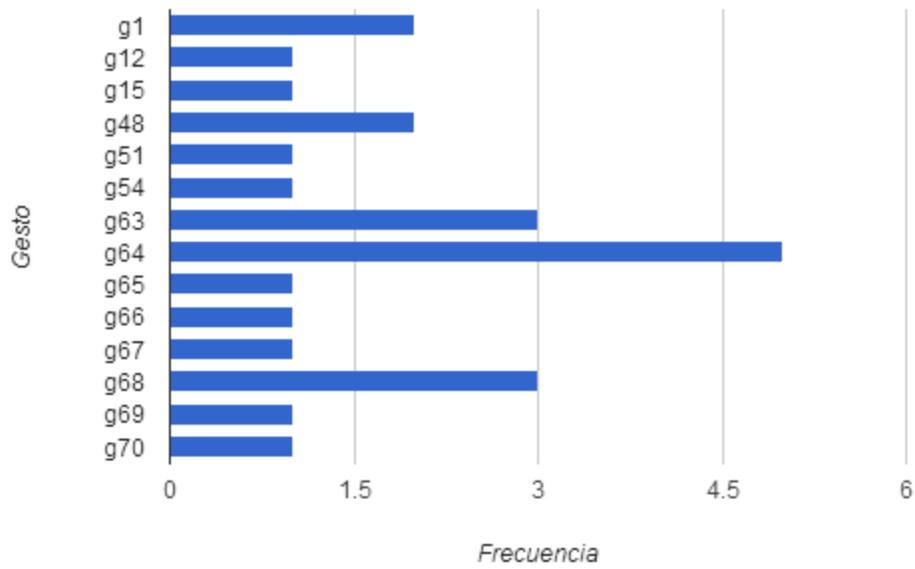
Amnesia: Mover la cámara



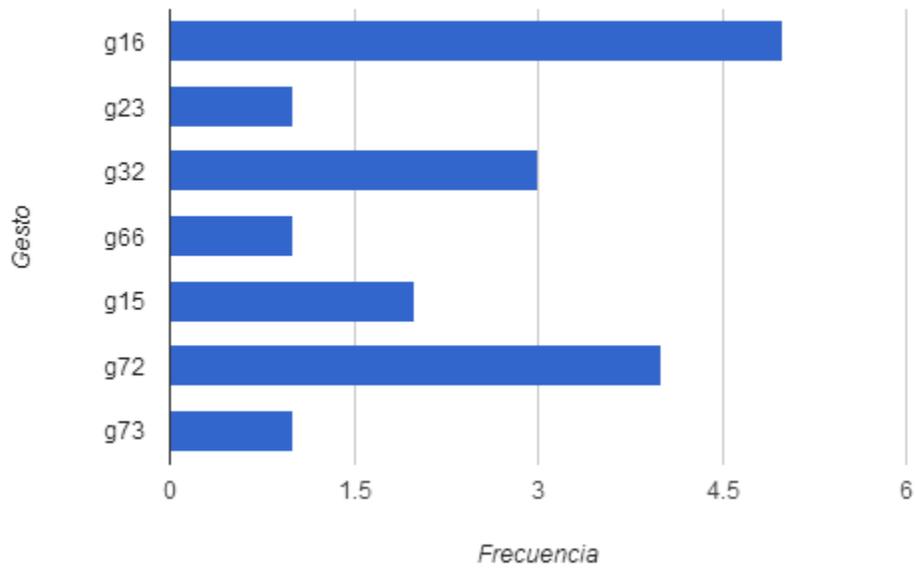
Asphalt 8: Manejar caja de cambios



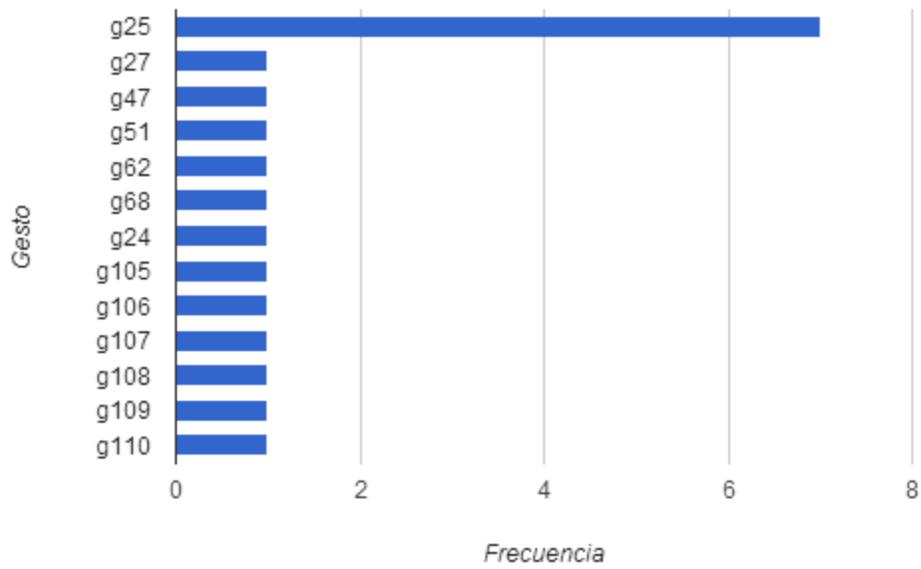
Wii Sports: Pegarle a la pelota



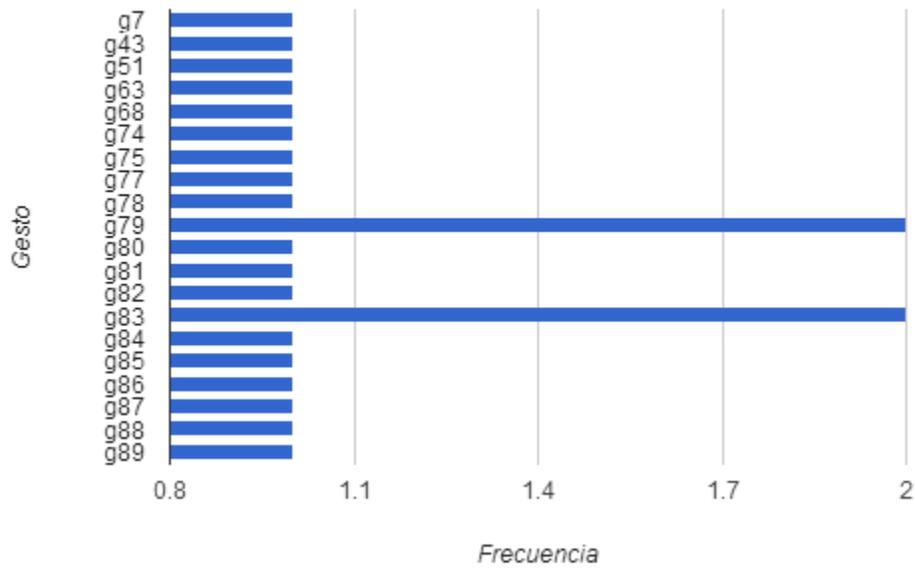
Wii Sports: Lanzar y sacar



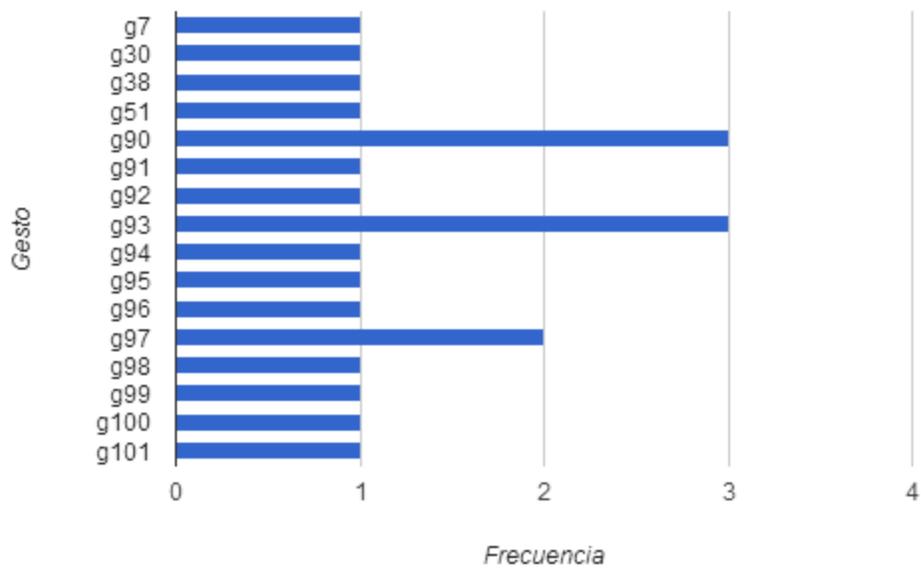
3D Pool Game: Disparar



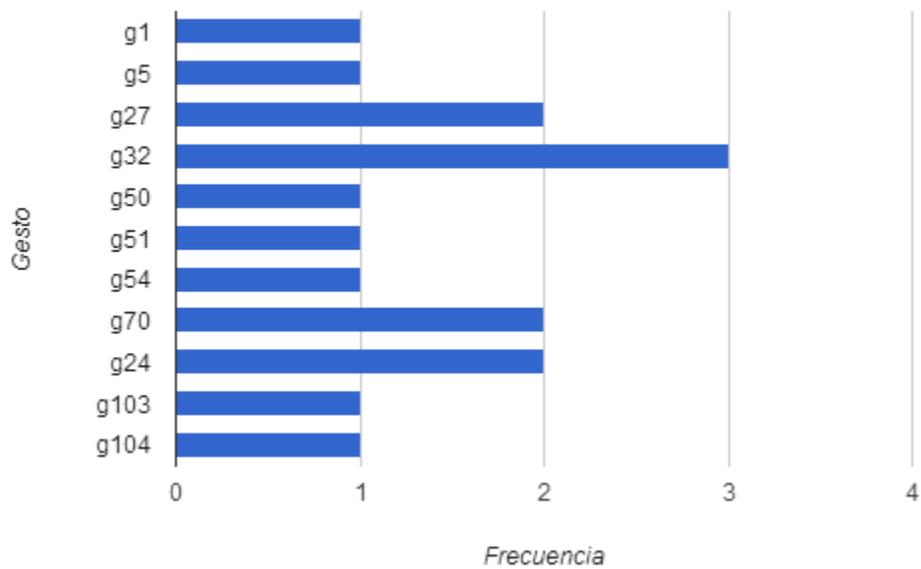
Juegos de cartas/dados: Lanzar dados



Juegos de cartas/dados: Barajar mazo



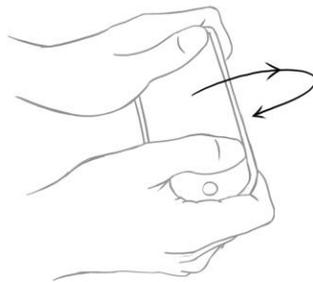
Ajedrez: Pasar turno



3.5.2 CLASIFICACIÓN DE GESTOS

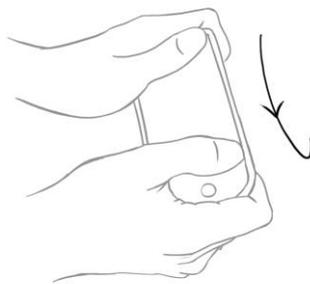
Tras un análisis de las curvas de cada gesto y las corroboraciones con lo captado por la cámara, se pudieron vislumbrar distintos gestos ganadores para cada acción. Para cada estudiante se anotó el gesto que realizó para cada acción. Luego, se consideró ganador al gesto que fue realizado por la mayoría. A continuación se muestran los gestos más populares para cada acción, junto con una representación visual del mismo. Los mismos fueron realizados con Photoshop CS6.

Sidescroller 2D



Mover en profundidad: *Tilt hacia adelante*

Juego de pelea 2D



Mover horizontalmente: *Tilt hacia la derecha*

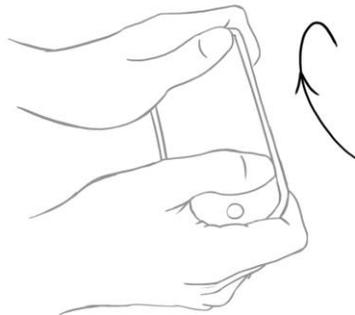


Saltar: *Movimiento hacia arriba*

Pinball



Lanzar la pelota: *Swing hacia adelante y arriba*

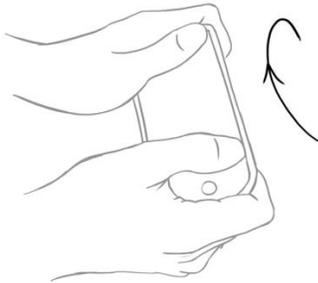


Usar paleta (paleta derecha): *Tilt hacia la izquierda*



Pegarle al tablero: *Shake de rotación*

Juego de plataformas

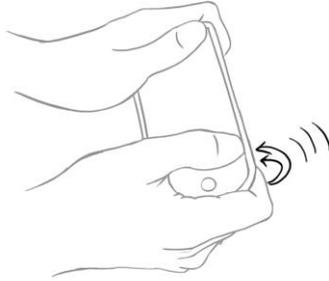


Frenar/acelerar personaje: *Tilt hacia la izquierda*

First-person Shooter

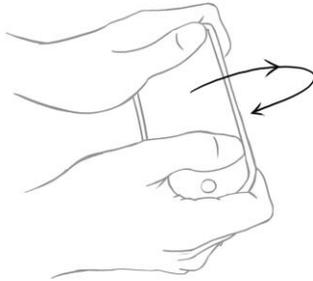


Recargar armar: *Tilt y movimiento hacia persona*



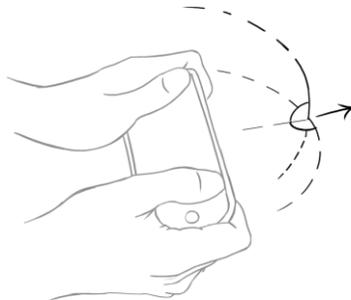
Cambiar de arma activa: *Doble tap atrás/derecha*

Shoot 'em up



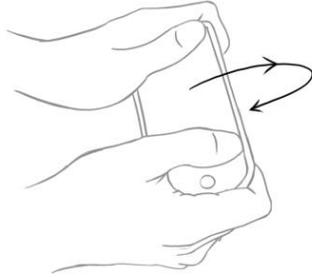
Disparar: *Tilt hacia adelante, puede ser rápido*

Survival Horror



Mover la cámara: *Orbitar*

Carreras de autos



Manejar caja de cambios: *Tilt hacia adelante, puede ser rápido*

Tenis

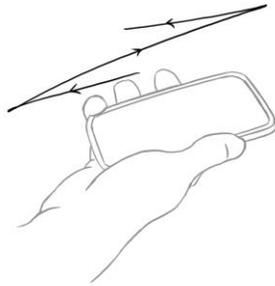


Pegarle a la pelota: *Swing arriba y adelante, con posible dirección*



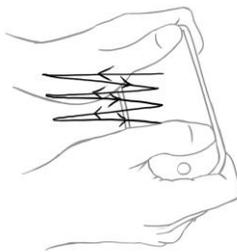
Lanzar y sacar: *Swing arriba y adelante, con posible remate*

Juego de billar



Disparar: *Movimiento hacia la persona y luego hacia adelante*

Juegos de dados



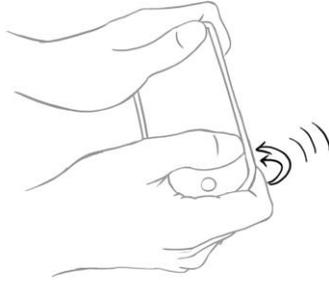
Lanzar dados: *Shake de movimiento hacia adelante y atrás, con posible remate hacia adelante.*

Juegos de cartas



Barajar cartas: *Shake de movimiento y rotación con una mano hacia la otra.*

Ajedrez



Pasar turno: *Tap atrás del celular en la parte superior derecha*

4. RECONOCIMIENTO DE GESTOS PROPUESTOS POR USUARIOS

4.1 OBJETIVO

Ya con la tabla de los gestos ejemplares para cada acción, es deseable investigar si dichos gestos son reconocibles, y si tienen conflictos entre sí, ya que muchas veces varios de estos gestos pueden convivir en el mismo contexto. Para esto, se diseñó otro experimento con usuarios, en el que, utilizando otra aplicación desarrollada en Android, se guarda información generada por cada usuario para verificar y validar cada gesto.

Este segundo experimento no requiere ser centrado en el usuario, ya que simplemente se le explica el gesto que el usuario debe ejecutar para luego registrarlo y reconocerlo en el dispositivo mediante los sensores ya discutidos. Este proceso de reconocimiento puede ser tanto en *real-time* como *offline*, como se verá luego.

El principal objetivo es investigar y desarrollar uno o varios algoritmos y métodos para reconocer los gestos guardados y clasificados. Para esto, primero estandarizamos el concepto de gesto. Luego, se buscó si existían conflictos en el reconocimiento de gestos (señales similares para diferentes gestos) con el objetivo de sugerir evitar usarlos simultáneamente dentro de un mismo juego (o actividad).

4.2 DISEÑO DEL EXPERIMENTO

Se diseñó el experimento para la segunda fase del proyecto. El objetivo principal de este experimento es validar y verificar que los gestos ejemplares, descubiertos por los usuarios del primer experimento, sean reconocibles y encontrar sus conflictos, si los hubiera.

Básicamente, el experimento consiste en utilizar otra aplicación desarrollada en Android que posee todos los gestos ejemplares precargados, con una interfaz que ofrece al usuario grabar y/o probar si un gesto es reconocido como uno de los gestos ejemplares. Consiste de una secuencia de pantallas que le piden al usuario ejecutar el gesto esperado, y al terminar cada

gesto, muestra una pantalla con el resultado (es decir, si fue reconocido o no)

Si bien este experimento no requiere una audiencia en particular debido a que solo está enfocado en validar gestos, se convocaron estudiantes hombres y mujeres de entre 20 y 24 años como sujetos para probar la aplicación móvil. En este caso, tampoco fue necesario el uso de una cámara digital para registrar las entrevistas, sino simplemente los resultados de cada reconocimiento gestual.

Sin embargo, antes de la ejecución del experimento se hicieron pruebas preliminares en las que surgieron problemas técnicos (detallados en futuras secciones), por lo que el experimento volvió a la fase de diseño. En este nuevo rediseño del experimento se quitó el componente de reconocimiento, y se convirtió a la aplicación en un simple grabador de señales, muy similar a la aplicación de la primera etapa del proyecto.

Luego, se utilizarían estas grabaciones para probar el reconocimiento de los gestos, pero de forma *offline*.

4.3 IMPLEMENTACIÓN

4.3.1 ESTANDARIZACIÓN DE GESTOS EJEMPLARES

Se considera un gesto como un conjunto de señales que representan los valores obtenidos en los dispositivos utilizados. Un gesto es entonces una señal de 6 dimensiones (ver Figura 4.4.1.3), donde las 3 primeras dimensiones son la aceleración, y las últimas 3 dimensiones representan el *cambio* de orientación.

Se utilizaron entonces dos sensores para grabar las señales del conjunto de entrenamiento: el sensor de Aceleración lineal y el sensor de Vector de rotación. Ver figuras 4.4.1.1 y 4.4.1.2, respectivamente.

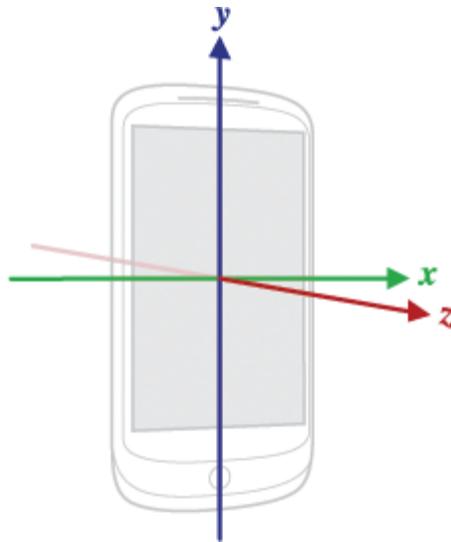


Figura 4.4.1.1 El espacio de coordenadas de la Aceleración Lineal, local a la orientación del dispositivo.

La particularidad de la aceleración lineal es que el efecto de la gravedad está anulado, es decir, no considera la aceleración generada por la gravedad del planeta.

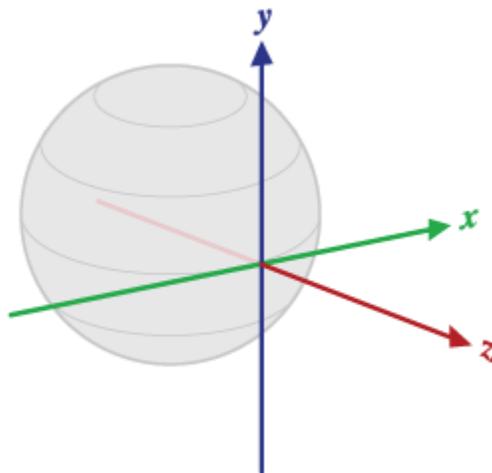


Figura 4.4.1.2 El espacio de coordenadas del Vector de Rotación.

Para la orientación se utilizó el vector de rotación, que está orientado localmente al planeta, y luego se calculó la diferencia del vector en grados Euler, ya que el vector original es un cuaternión, una estructura de datos comúnmente utilizada para manipular ángulos con precisión.

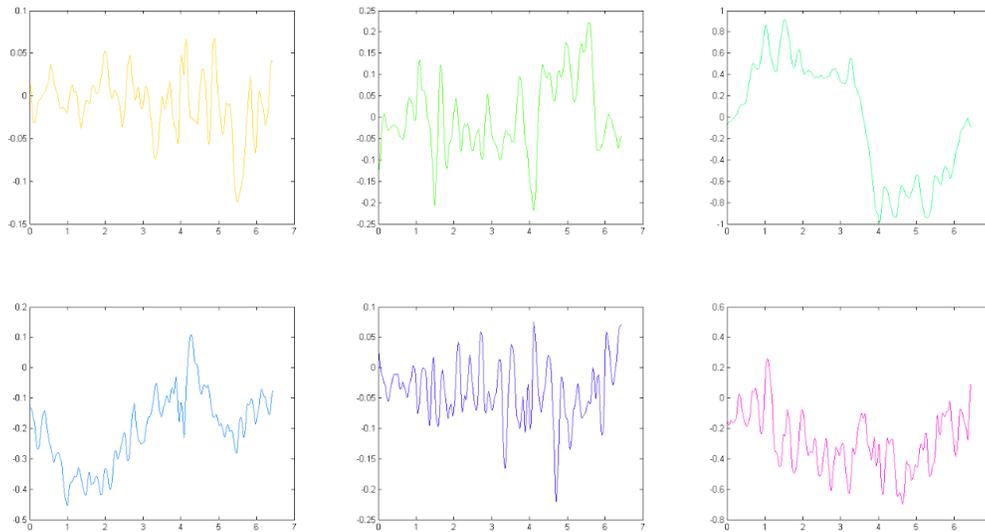


Figura 4.4.1.3 Ejemplo de gesto. Las señales del cuadrante superior representan la aceleración lineal, mientras que las inferiores representan el cambio de orientación.

Luego, para reconocer y diferenciar los gestos entre sí se decidió utilizar Dynamic Time Warping, un algoritmo que calcula una estimación de la distancia mínima o error entre dos señales, ajustando por las diferencias temporales que pueden poseer ambas señales.

4.3.2 LIMITACIONES DE LOS SENSORES

Los sensores utilizados, y la naturaleza del reconocimiento de gestos de movimiento en dispositivos móviles trae problemas y limitaciones que deben ser considerados. En el contexto del proyecto, y en base a las decisiones de implementación tomadas, existen dos limitaciones importantes que afectan los resultados: la velocidad del gesto y la dificultad de obtener la posición absoluta del dispositivo. Ambos problemas se deben al error no despreciable de los sensores (acelerómetro, giroscopio) que utilizan los dispositivos, y que tiende a acumularse.

Debido a que se decidió utilizar el delta del vector de rotación (o velocidad angular), los registros que la aplicación guarda son relativos al frame anterior, por lo que, generalmente, la magnitud de esta diferencia no supera los 5 grados, y suele estar entre 0 y 1. Esto, sumado al error del giroscopio, imposibilita grabar o reconocer gestos muy largos y suaves, ya que el cambio resulta ser muy chico y puede ser interpretado como error. Además, el

filtrado posterior de las señales puede llegar a ignorar estos cambios (por ejemplo, el umbral podría anular estos deltas). Sucede algo similar con la aceleración lineal; debido a que es esencialmente la segunda derivada de la posición, sus valores son muy chicos y pueden ser interpretados como error.

En principio, estos problemas podrían ser resueltos si se pudieran utilizar estos sensores para integrar la posición absoluta del dispositivo. Sin embargo, dado que es análogo a un sistema de navegación inercial, el error imposibilita la integración de la aceleración, ya que se acumula frame por frame.

Esto además se intensifica debido a la relación de la aceleración lineal con la orientación: el sistema de coordenadas de la aceleración lineal es local, por lo que es muy dependiente de la precisión de la orientación (y la velocidad angular). Si se considera además que este sensor virtual considera la gravedad en espacio local, debe sumarse otro error adicional.

Es por esto que se hace el reconocimiento de señales en espacio relativo, y con las consideraciones de error necesarias, como la utilización de filtros para reducir el ruido producido por los sensores.

4.3.3 DYNAMIC TIME WARPING

Para el reconocimiento de las señales que representan cada gesto se decidió utilizar Dynamic Time Warping, un algoritmo que calcula la distancia mínima ajustada temporalmente entre dos secuencias de datos. A grandes rasgos, el algoritmo calcula la similitud entre dos señales multidimensionales expresadas como una secuencia lineal. De esta forma es posible comparar una señal nueva contra un conjunto de señales ya conocidas, y obtener una estimación de la señal más próxima o similar.

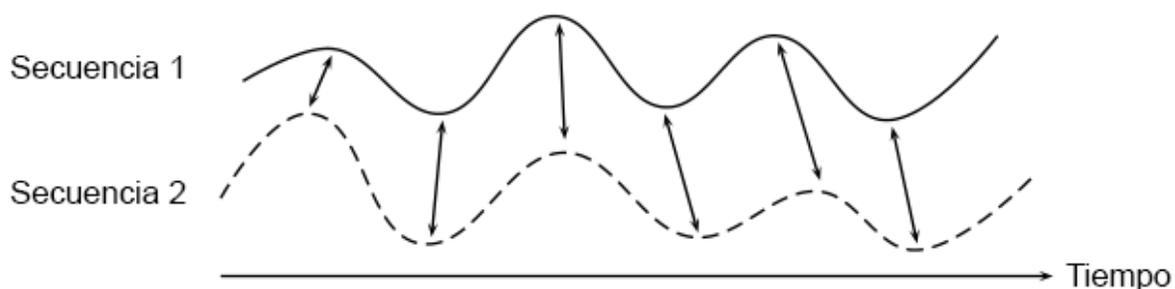


Figura 4.3.3.1. Ajuste temporal y no lineal entre dos secuencias de datos. [9]

El algoritmo recibe dos señales y ciertas restricciones que debe cumplir, y devuelve una matriz de costo acumulado, que representa el costo o distancia entre ambas secuencias en cualquier tiempo para poder alinearlas. Existe mucha literatura acerca del algoritmo, sus variaciones y optimizaciones. Para mayor detalle consultar la bibliografía sugerida [8,9].

De esta matriz de costo puede extraerse el último elemento calculado y utilizarlo como *score* o distancia mínima para la comparación de gestos. La dimensionalidad de esta distancia es de particular interés, y es analizada en la sección de Resultados. Un ejemplo de la matriz de costo puede verse en la Figura 4.3.3.2, y la matriz de costo acumulada en la Figura 4.3.3.3.

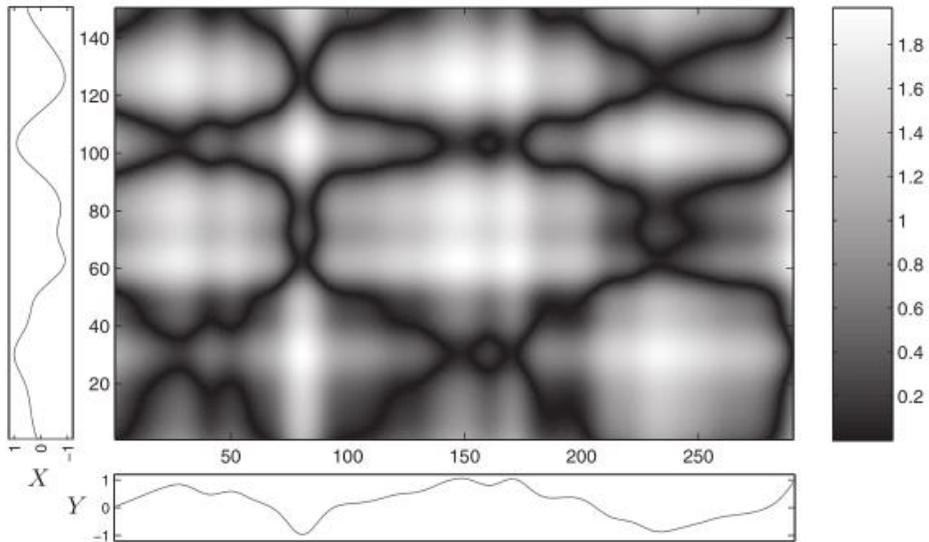


Figura 4.3.3.2. Matriz de costo generado por DTW. Cada eje representa una señal, y el valor de la matriz representa la distancia ajustada temporalmente en ese tiempo t. [9]

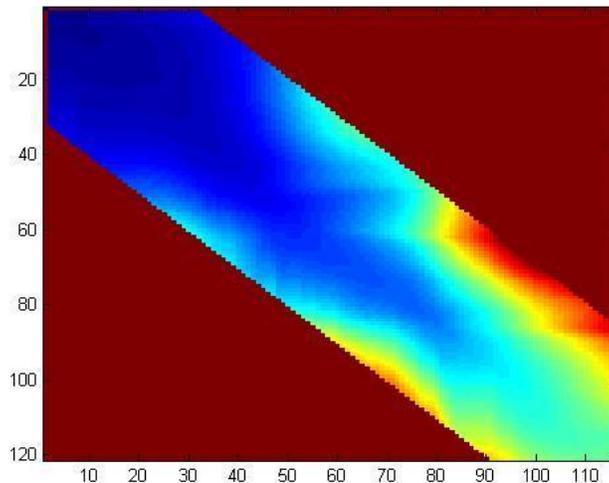


Figura 4.3.3.3. Matriz de costo acumulado generado por DTW, utilizando una ventana reducida para evitar desfases temporales muy grandes.

4.3.4 APLICACIÓN MOBILE

La aplicación GestureChecker fue desarrollada para utilizar en esta segunda etapa de pruebas con usuarios. El objetivo era evaluar el nivel de dificultad en la detección y reconocimiento de los gestos deducidos de la etapa previa.

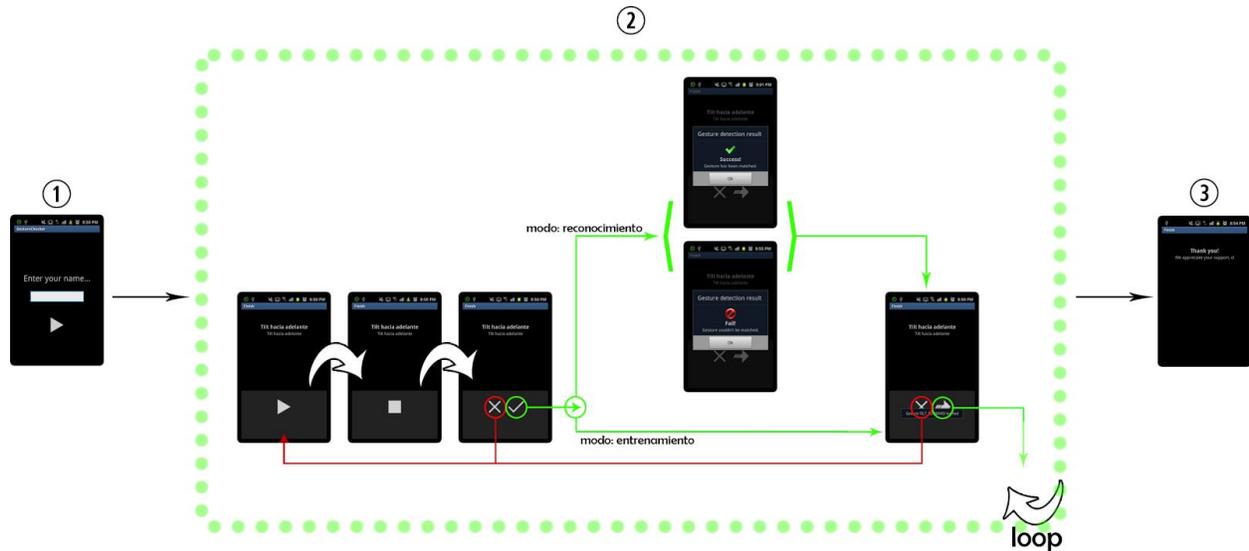
Similar a SensorLogger, cada usuario debería grabar sus movimientos para una serie de gestos que se le irían presentando secuencialmente. Sin embargo, la diferencia radica en que para SensorLogger, cada pantalla se correspondía con un escenario y se esperaba una resolución propuesta por el usuario. En GestureChecker, cada pantalla se corresponde con un gesto (deducido de la etapa previa) y el usuario debe simplemente realizarlo.

Por otro lado, esta aplicación posee dos modos de ejecución que se definen en tiempo de compilación: ENTRENAMIENTO y RECONOCIMIENTO. La única diferencia entre ambos modos es el procesamiento que se realiza para cada gesto antes de pasar al siguiente. En el modo ENTRENAMIENTO, el gesto introducido por el usuario es aprendido por la aplicación y se utilizará como base de comparación para el modo RECONOCIMIENTO.

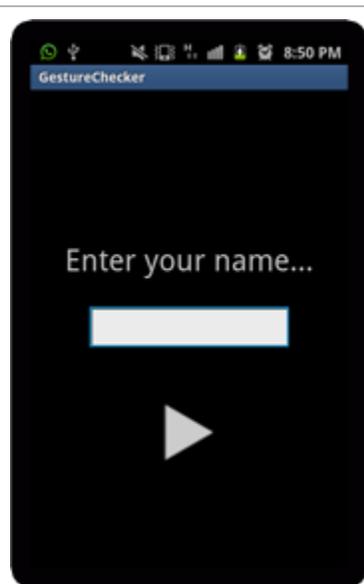
Por lo tanto, el proceso de las pruebas en esta etapa involucraría dos ejecuciones distintas de GestureChecker: una en fase de ENTRENAMIENTO y otra en fase de

RECONOCIMIENTO.

El flujo de GestureChecker es como se detalla a continuación:



Pantalla de bienvenida



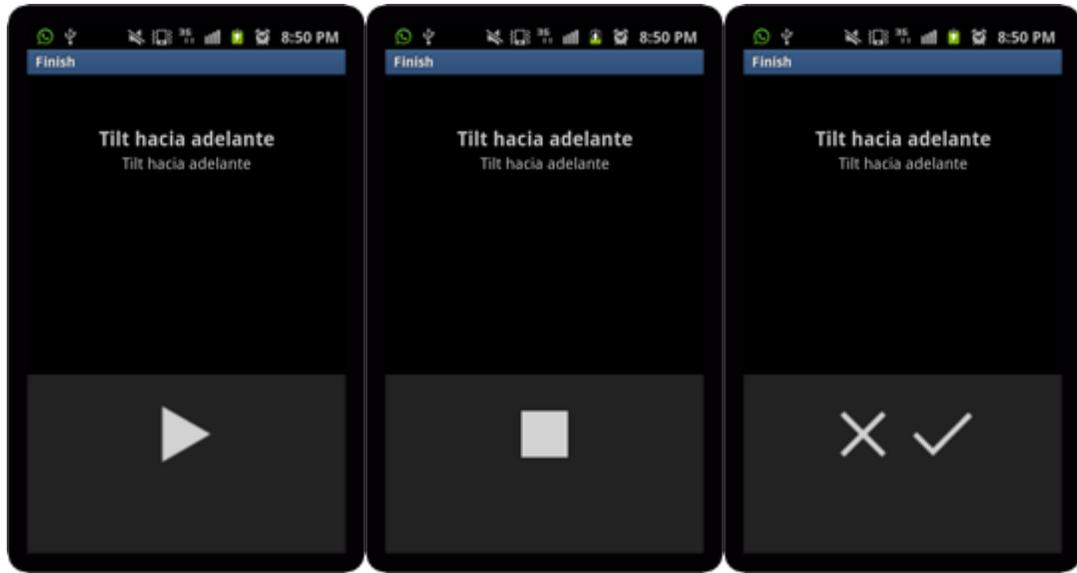
Lógica StartActivity.java

Vista activity_start.xml

Se le pide el nombre al usuario antes de iniciar el loop de aprendizaje/reconocimiento de gestos.

Cuando el usuario esté listo, lo único que debe hacer es presionar el único botón de esta pantalla para ir a la siguiente.

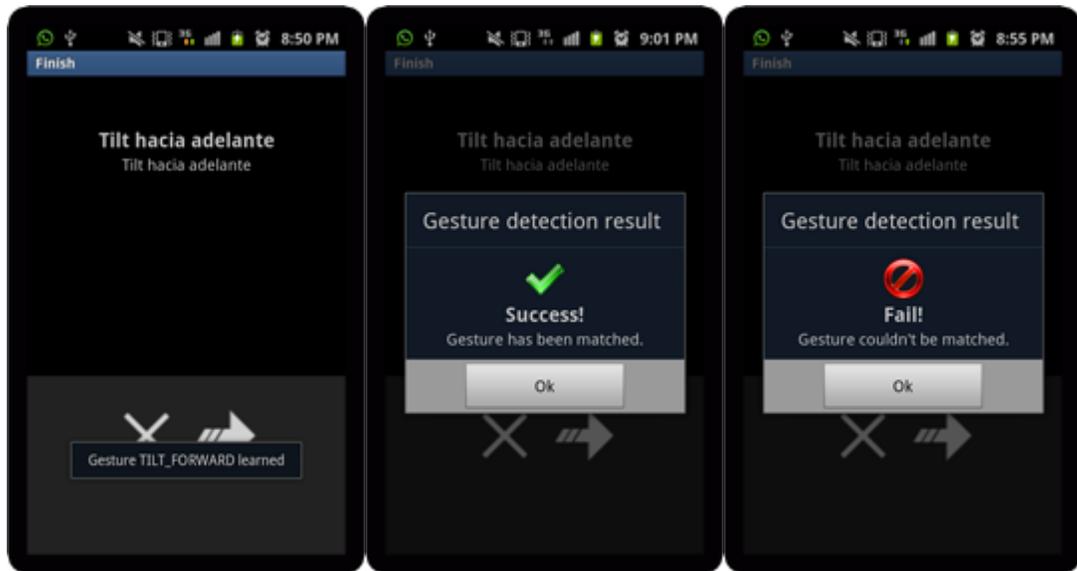
Pantalla de procesamiento de gesto



(1) Stand-by

(2) Grabando

(3) Grabado



(4) Procesado (aprendizaje)

(5) Procesado (reconocimiento)

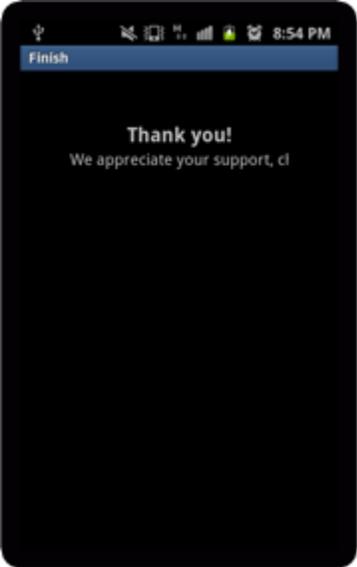
(6) Procesado (reconocimiento)

Lógica GestureActivity.java

Vista activity_gesture.xml

Pantalla de procesamiento de gesto. Cada imagen representa un estado de la misma pantalla.

Pantalla de despedida

	<p>Se le agradece al usuario su colaboración y finaliza la aplicación.</p>
<p>Lógica FinishActivity.java Vista activity_finish.xml</p>	

El único modo que devuelve output es el modo de ENTRENAMIENTO. Por cada gesto, graba los resultados en un archivo de extensión GST, que es básicamente un array de los valores de cada sensor en el tiempo. Estos datos serán útiles para realizar un análisis posterior.

4.3.5 RECONOCIMIENTO OFFLINE

En esta etapa del proyecto sucedieron varios cambios, ya que nos encontramos con problemas de implementación y se tuvieron que tomar soluciones de compromiso. Se había previsto hacer pruebas de usabilidad con una aplicación en *tiempo real* que reconozca dichos gestos. Sin embargo, en pruebas preliminares, los algoritmos utilizados no lograron reconocer bien los gestos ingresados por el usuario. Esto se debió principalmente a que el conjunto de gestos ejemplares del algoritmo era muy reducido.

Por esto, se optó por tener una librería más amplia de gestos ejemplares, pero el algoritmo perdió la cualidad de ser *tiempo real* debido a la cantidad de datos simultáneos a analizar, por lo que se decidió portar todos los algoritmos a **MATLAB** y hacer análisis offline de los gestos para verificar si son reconocibles y si tienen conflictos entre sí. Para esto, se

utilizaron grabaciones de los gestos ejemplares de múltiples personas para obtener un conjunto razonable de datos para el reconocimiento. Se comparó cada gesto de ese conjunto contra su complemento (todos los gestos excepto el que estaba siendo analizado) para ver si era reconocible.

Sin embargo, el hecho de que el reconocimiento sea offline no es detrimental al proyecto; ya que el objetivo es principalmente descubrir si los gestos son reconocibles y sus conflictos. Se asume que existen optimizaciones y métodos para que, utilizando los resultados, se puedan implementar soluciones en tiempo real. Además, si un videojuego quisiera implementar algún subconjunto de gestos, es poco probable que utilice todos, por lo que el volumen de datos será menor.

4.3.6 CONJUNTOS DE ENTRENAMIENTO

Inicialmente se hizo una grabación de cada gesto ejemplar, pero surgieron muchos problemas relacionados al reconocimiento que impidieron que sea una solución útil. Las razones de por qué no resultó útil una grabación por gesto ejemplar son varias.

Dado que los gestos son esencialmente una secuencia compleja de movimientos, las diferencias en aceleraciones, tiempos y distancias que cada individuo ejecuta son abismales. Si bien los algoritmos utilizados son capaces de reconocer y diferenciar gestos, hay un margen de error que no es despreciable, en particular cuando hay pocos gestos para analizar.

Por esto se decidió armar conjuntos de gestos ejemplares: se le pidió a un grupo de personas que graben todos los gestos, y luego se clasificaron por su tipo de gesto. Esto permitió que los algoritmos tengan mucha más flexibilidad al reconocer los gestos, ya que ahora cada gesto nuevo a reconocer es comparado con muchas grabaciones de personas distintas del mismo gesto, con distintos tiempos, aceleraciones y distancias. La información que genera cada usuario está organizada como describe la figura 4.3.6.1.

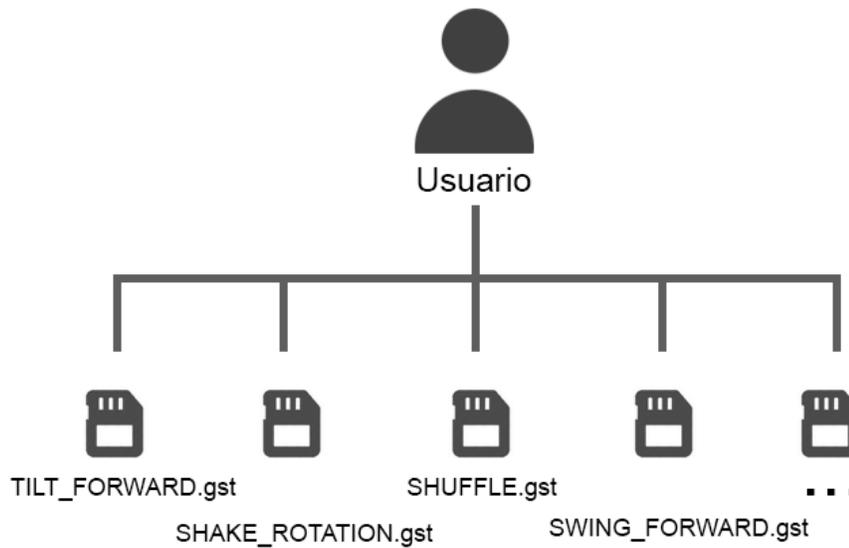


Figura 4.3.6.1. Archivos generados por una sesión con un usuario. Cada archivo generado se guarda bajo el directorio /matlab/recordings/usuario/archivo.gst

La diferencia en calidad de reconocimiento fue importante, ya que utilizando solo una grabación por gesto, el error de reconocimiento dependía mucho de quién estaba haciendo la prueba, y quién grabó los gestos ejemplares.

Utilizando un conjunto de grabaciones por cada gesto se logró aproximadamente un 90% de eficacia al reconocer gestos, un error aceptable considerando que todos los tipos de gestos están siendo comparados entre sí, un escenario extremo que en la práctica (una aplicación o un juego) es poco probable que sea necesario, ya que en general las aplicaciones no utilizan más de 3 o 4 gestos para su uso.

Sin embargo, como previamente se dijo, al utilizar tantos datos para el reconocimiento se perdió la capacidad de hacer el análisis en *tiempo real* en el dispositivo. Dado que nuestro interés está en analizar si los gestos conflictúan entre sí, no es crítico que este análisis se ejecute en tiempo real.

4.3.7 UTILIDADES

Se desarrollaron algunas utilidades para agilizar el proceso de análisis de las aplicaciones y algoritmos implementados.

Visor de gestos simple

Dado que las primeras pruebas de los usuarios generaron mucha información, se necesitó una forma práctica y ágil para revisar y visualizar las grabaciones que los usuarios generaron.

Para esto se desarrolló una aplicación en C# utilizando el motor Unity3D que rastrea todos los archivos de gestos, los descomprime, y permite al usuario visualizar las grabaciones según el sensor de cada señal. La aplicación fue desarrollada con una arquitectura desacoplada, permitiendo distintas visualizaciones de la misma información.

Sin embargo, sólo se implementaron dos visualizaciones: la aceleración y el giróscopo, y debido a los problemas inherentes a los sensores no se pudieron hacer aproximaciones en espacio absoluto del comportamiento del dispositivo en 3D, como muestra la Figura 4.3.7.1. Incluso se probó utilizando grabaciones "nulas" de los sensores para utilizarlas como calibración, pero la mejora no fue perceptible. De todas formas, sirvió para visualizar los gestos de una forma rápida y eficaz, algo de mucha utilidad para luego poder formalizarlas en las pruebas posteriores.

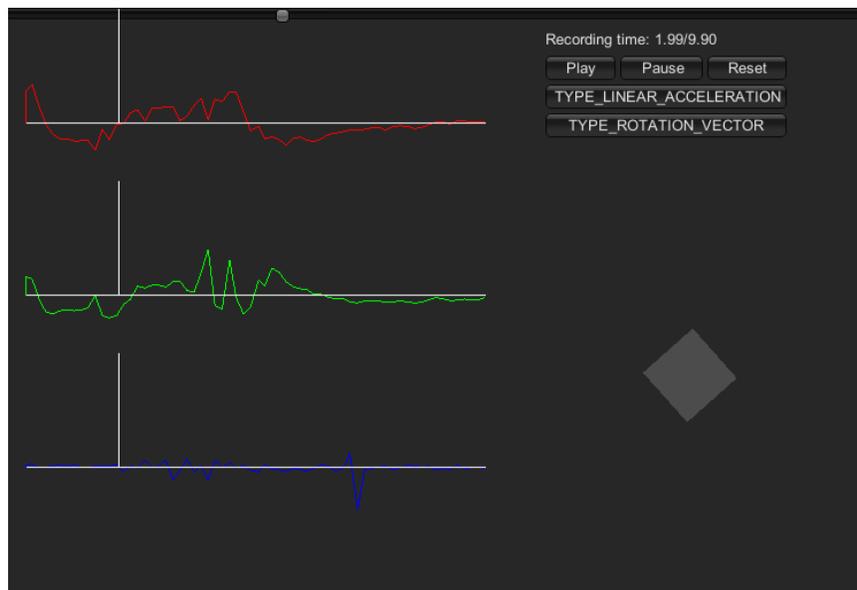


Figura 4.3.7.1. Aplicación de utilidad. El cubo rotado representa al dispositivo; si bien esto fue descartado, la utilidad sirvió para analizar las señales de forma ágil.

Visor de señales

Por último se desarrolló una utilidad sencilla pero necesaria: un visor de las señales del acelerómetro en el dispositivo, en tiempo real. Fue desarrollado para Android en C#, utilizando Unity, y si bien la API de Unity no ofrece la señal pura del acelerómetro (sino una combinación entre acelerómetro y giróscopo suavizada, que se reducía al *tilting* del dispositivo), sirvió como ayuda empírica del rango y curvas razonables que la mano podía ejercer con el dispositivo. La interfaz gráfica de la aplicación es sencilla, ya que simplemente muestra los valores obtenidos en un gráfico que se actualiza en tiempo real, como muestra la figura 4.3.7.2.

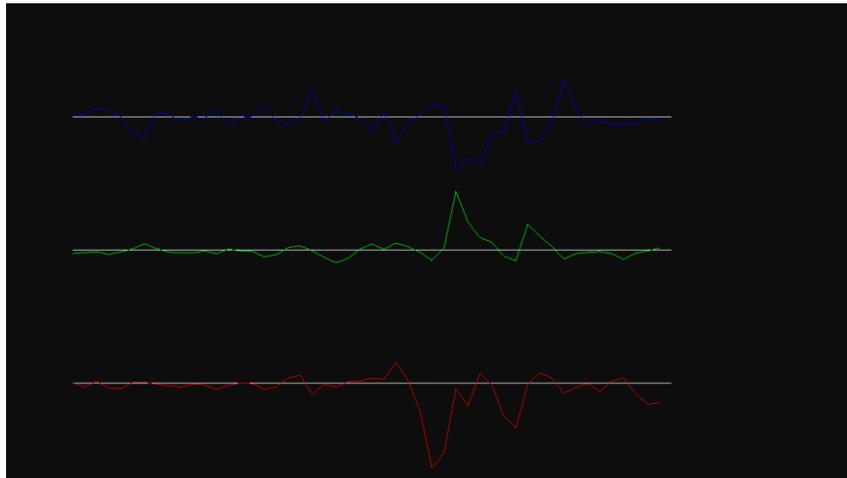


Figura 4.3.7.2. Captura de pantalla del visor de señales del acelerómetro en tiempo real, en el dispositivo.

4.4 RESULTADOS

4.4.1 MATRIZ DE CONFUSIÓN

Utilizando la siguiente nomenclatura para los gestos ejemplares, se ejecutó el algoritmo Dynamic Time Warping sobre todos los gestos obtenidos con su complemento dentro del conjunto de gestos. Como la sección 4.4.4 describe, los gestos Shake utilizan el mismo identificador, ya que colisionan entre sí. Para más detalles de las razones, ver la sección 4.4.4.

M1	Doble tap inferior
M2	Movimiento hacia arriba
M3	Disparo de billar
M4	Shake hacia persona
M4	Shake de rotacion
M4	Barajar cartas
M5	Swing hacia adelante y arriba
M6	Tilt hacia adelante
M7	Tilt hacia izquierda
M8	Tilt y movimiento hacia persona
M9	Tilt hacia derecha

*Tabla 4.4.1.1. Nomenclatura para el reconocimiento de gestos. Notar que los gestos **Shake** poseen el mismo identificador, ya que son reconocidos de la misma forma, y colisionan entre sí.*

Utilizando esta nomenclatura, y los algoritmos previamente mencionados, se obtuvo el

siguiente resultado:

	M1	M2	M3	M4	M5	M6	M7	M8	M9
M1	24	0	0	0	0	0	0	0	0
M2	0	23	0	0	1	0	0	0	0
M3	0	0	15	8	0	0	0	0	0
M4	0	1	0	68	0	0	0	0	0
M5	0	2	0	2	19	0	0	1	0
M6	0	0	0	0	0	23	0	0	1
M7	1	0	0	0	0	0	23	0	0
M8	0	0	0	0	2	0	0	22	0
M9	1	0	0	1	0	0	0	0	22

Tabla 4.4.1.2. Matriz de confusión resultante del análisis de reconocimiento de los gestos ejemplares. Cada entrada (i,j) representa la cantidad de gestos i que fueron interpretados como un gesto j.

Como puede verse en la tabla 4.4.1.2, la matriz de confusión posee una diagonal con valores muy altos, lo que representa que en general, al comparar un gesto nuevo contra el conjunto total de gestos, hay una probabilidad alta de ser reconocido como el gesto esperado.

Nuevamente, notar que en la fila/columna M4/M4 contiene el reconocimiento de los tres gestos Shake, por lo que posee valores más altos.

De 260 grabaciones de gestos utilizados, 238 son reconocidos correctamente, es decir, el algoritmo posee un error de 8.46%; un error razonable considerando la cantidad de gestos que deben reconocerse simultáneamente.

Este error puede ser minimizado si se utilizan conjuntos de gestos únicos más reducidos, es decir, menos definiciones de gestos. En la práctica, es poco probable que una aplicación o juego requiera reconocer múltiples gestos (en los escenarios propuestos hay hasta 3 gestos por juego), por lo que se considera que el error obtenido es aceptable para esta cantidad de

gestos.

4.4.2 DIMENSIONALIDAD DE LA DISTANCIA MÍNIMA

Como se describió previamente, Dynamic Time Warping calcula la distancia mínima entre dos señales, ajustada temporalmente. Para decidir la similitud entre dos señales se utiliza la distancia acumulativa, es decir, la suma de todas las distancias ajustadas por DTW. Esta distancia acumulativa se la considera como un puntaje que se asigna a cada par de señales, y luego se elige el par con menor puntaje.

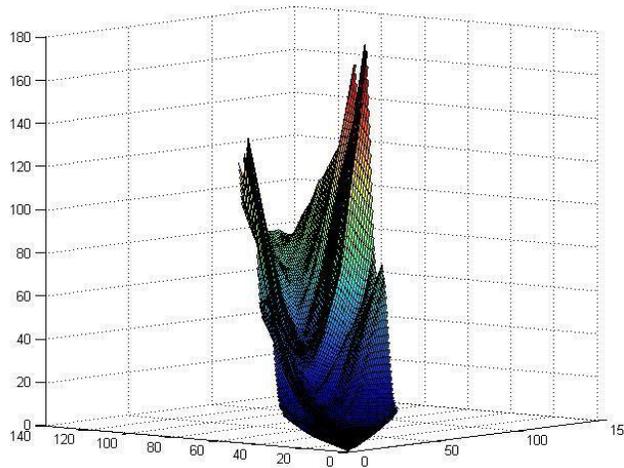


Figura 4.4.2.1 Valle de la distancia mínima ajustada temporalmente entre dos señales.

Sin embargo, el algoritmo de Dynamic Time Warping no explicita qué función se debe utilizar para calcular la distancia entre dos señales. Dado que nuestros gestos están compuestos por 6 señales distintas, surge el dilema de cuál es la mejor forma para calcular el puntaje.

Primer método: distancia acumulativa

El primer método utilizado consiste en calcular la distancia euclidiana entre dos gestos, considerando que una señal posee 6 dimensiones.

$$\|\mathbf{P}\| = \sqrt{p_1^2 + p_2^2 + \cdots + p_n^2} = \sqrt{\mathbf{P} \cdot \mathbf{P}}$$

Figura 4.4.2.2. Distancia euclidiana. En nuestro caso, $n=6$

Luego, la distancia acumulativa es unidimensional, y la búsqueda del puntaje consiste en encontrar el par de gestos con menor valor.

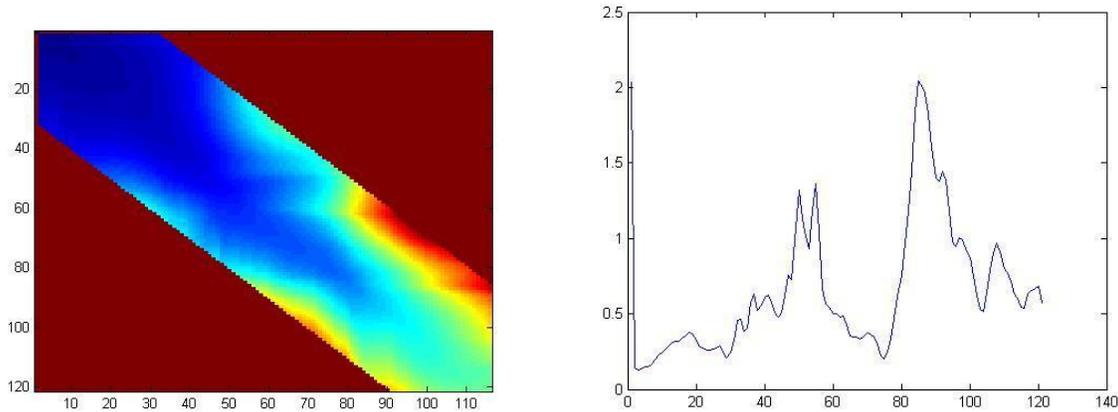


Figura 4.4.2.3 Matriz de costo acumulativo y mínima distancia entre dos señales, con una ventana restringiendo el máximo ajuste temporal.

Segundo método: distancia acumulativa 6-dimensional

El segundo método que se propuso consiste en calcular DTW sobre cada dimensión por separado, obtener 6 distancias acumulativas distintas, y formar un vector de distancias acumulativas. Luego, para calcular el puntaje entre dos señales, se consideran todos los vectores de distancias acumulativas como puntos en un espacio 6-dimensional, y se selecciona el vector con menor magnitud.

$$D = (\| \delta_1 \|, \| \delta_2 \|, \| \delta_3 \|, \| \delta_4 \|, \| \delta_5 \|, \| \delta_6 \|)$$

Figura 4.4.2.4 Vector de distancias D, en el que cada componente representa la distancia calculada en cada componente de la señal.

$$\| D \| = \sqrt{D \cdot D}$$

Figura 4.4.2.6. La magnitud del vector distancia entre dos gestos, utilizado como valor para encontrar el par de gestos más similar.

Luego, el par de gestos con menor magnitud $||D||$ será el seleccionado.

Comparación y análisis de los métodos de cálculo del puntaje

El primer método obtuvo mejores resultados, encontrando menos conflictos entre gestos. Si bien el segundo método propuesto posee la ventaja de ajustar individualmente cada una de las 6 señales de cada gesto, es probable que utilizar un solo ajuste temporal genere menos ruido en el resultado final.

El primer método reconoció 238 de 260 grabaciones de gestos en total, logrando un 91.54% de efectividad, mientras que el segundo método tuvo éxito con 223 grabaciones, es decir, un 85.77% de las veces reconoció el gesto adecuado. Además, el segundo método es varias veces menos eficiente que el primero, por lo que toma mucho más tiempo.

4.4.3 FILTRADO DE SEÑALES

Los sensores integrados en los dispositivos utilizados en celulares, tabletas y smartphones poseen mucho ruido en sus mediciones. Dada la naturaleza ruidosa de estos sensores, es deseable filtrar las señales antes de analizarlas, para tener datos más limpios y poder diferenciar mejor entre gestos.

Debido a que utilizamos la diferencia de orientación además del acelerómetro, pequeñas diferencias pueden ser muy importantes para la interpretación final del gesto; por ejemplo, el efecto del *drifting*, que hace imposible la doble derivación de la aceleración, es un resultado de dicho ruido en los sensores, que puede observarse en la Figura 4.4.3.1.

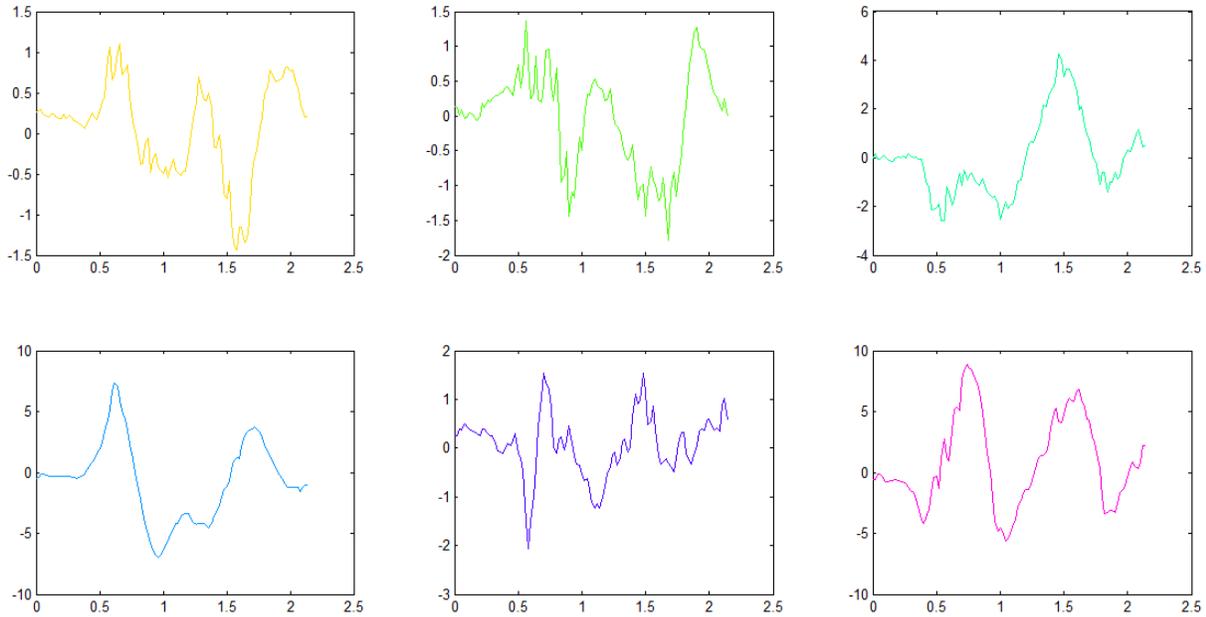


Figura 4.4.3.1 Señales generadas por un gesto, sin ningún tipo de filtro aplicado. Las pequeñas variaciones, consideradas como ruido, complican el reconocimiento de los gestos.

El filtrado de las señales puede ser variado, y existen muchos filtros distintos que pueden aplicarse. Sin embargo se decidió por utilizar dos filtros: un filtro Gaussiano y un umbral.

Umbral

Para evitar considerar valores demasiado chicos que pueden ser producto del ruido de los sensores, simplemente se ignoran esos valores y se les asigna valor nulo. La señal resultante entonces sólo contiene información que se considera relevante, es decir, cuando el sensor superó un umbral determinado.

En términos más formales, el umbral puede definirse como muestra la Figura 4.4.3.2, y el resultado de su aplicación a la señal está representado en la Figura 4.4.3.3.

$$umbral(x, \mu) = \begin{cases} 0, & \text{si } |x| < \mu \\ x, & \text{si } |x| \geq \mu \end{cases}$$

Figura 4.4.3.2 Definición de función umbral, donde μ es el valor del umbral utilizado.

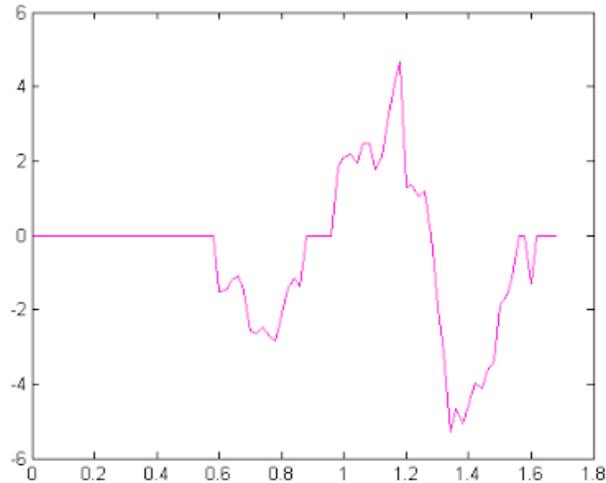


Figura 4.4.3.3. Ejemplo de un umbral sobre una de las seis señales generadas por un gesto. Cualquier valor absoluto menor a un umbral es anulado.

Filtro Gaussiano

Nuevamente, debido al ruido de los sensores es deseable suavizar las señales para promediar las pequeñas diferencias que no aporten al reconocimiento de los gestos. En general, las señales generadas por los gestos analizados poseen curvas muy definidas, con pocas variaciones de alta frecuencia. Es por esto que se buscó un filtro pasabajos que elimine ruido de alta frecuencia y permita vislumbrar la principal tendencia de la señal. Un ejemplo de dicho suavizado puede observarse en la Figura 4.4.3.6

Un filtro Gaussiano funciona perfectamente en este caso, ya que promedia valores cercanos en base a la función gaussiana. El filtro está definido como la Figura 4.4.3.4 indica.

$$g(x) = \sqrt{\frac{a}{\pi}} \cdot e^{-a \cdot x^2}$$

Figura 4.4.3.4. Filtro de Gauss en una dimensión.

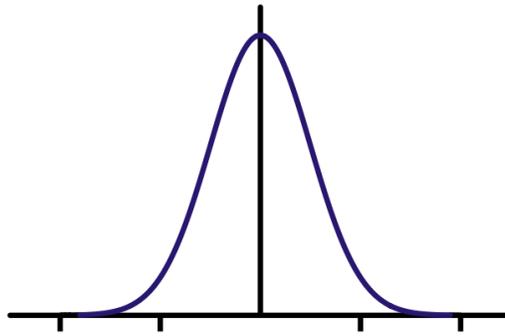


Figura 4.4.3.5. La importancia o contribución de los valores vecinos está definida por una curva de Gauss.

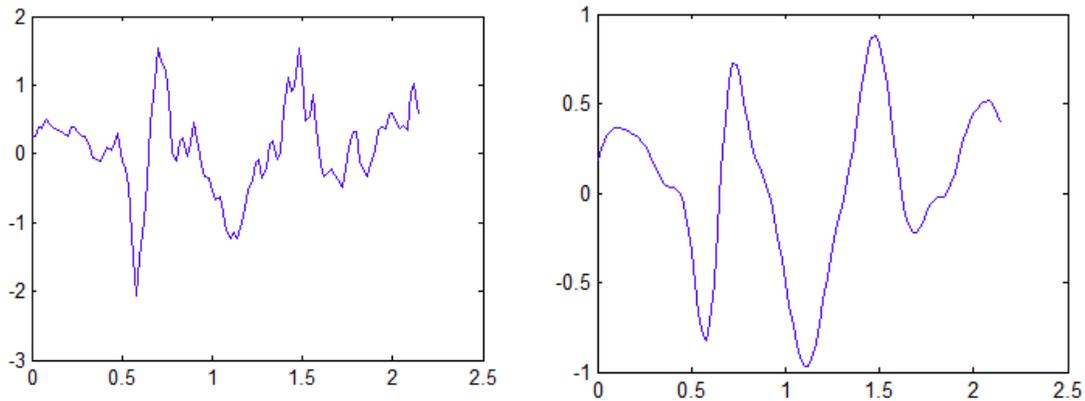


Figura 4.4.3.6. Filtro gaussiano sobre una de las señales generadas por un gesto. El resultado posee curvas mucho más definidas y suaves, lo que hace que el gesto sea más fácil de reconocer. Notar, sin embargo, que se pierde un pico alrededor $t=0.5$, lo cual no es deseable.

Composición de filtros

Si bien estos filtros son útiles al momento de mejorar el reconocimiento de las señales, surgieron problemas debido a la composición de estos filtros en la señal. Dado que los filtros pueden componerse, es importante analizar el orden en el que se filtra la señal, ya que hay efectos secundarios que pueden ser indeseables y complicar el reconocimiento.

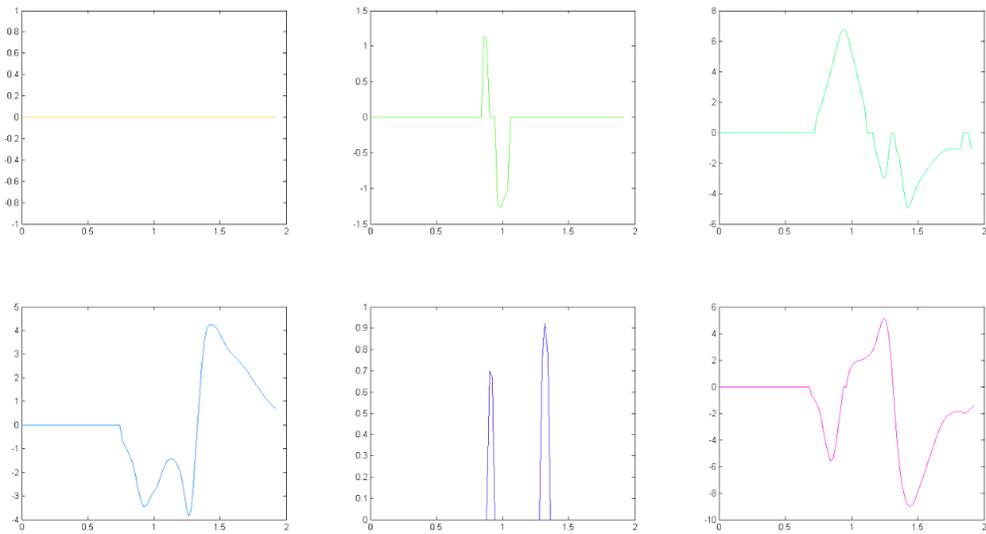


Figura 4.4.3.7. Composición de filtros: primero un filtro Gaussiano, luego un umbral. Notar cómo el umbral genera picos en ciertos casos. Además, se pierde el suavizado cerca del 0.

En este caso, si se filtra primero con Gauss y después se utiliza un umbral, puede suceder que el resultado final posea picos indeseables en los valles de las curvas suaves que genera Gauss (Ver Figura 4.4.3.7). Consideramos que esto es indeseable, ya que estas discontinuidades en la tendencia de la señal generan problemas con DTW.

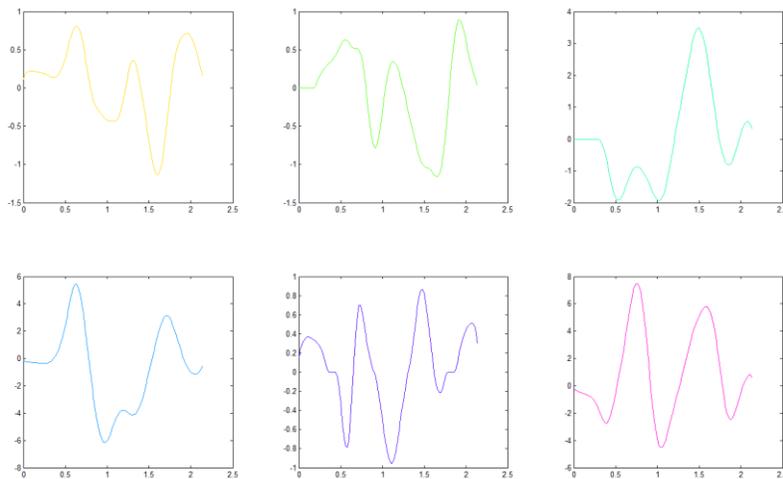


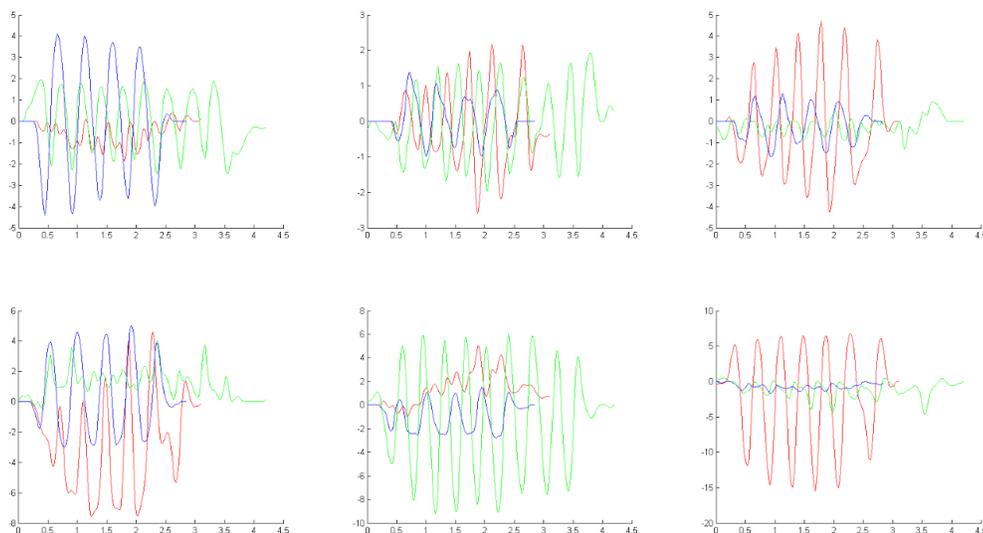
Figura 4.4.3.9. Composición final de filtros: Gauss con ventana grande, luego umbral y luego Gauss.

Se decidió entonces por ejecutar un filtro de Gauss con una ventana grande primero, luego un umbral para eliminar el ruido indeseable cercano al valor nulo, y por último un nuevo filtro de Gauss con una ventana más chica, para suavizar las pequeñas discontinuidades generadas por el umbral. Ver Figura 4.4.3.9 para un ejemplo de las señales resultantes.

Esta composición de los filtros fue la más beneficiosa para el reconocimiento de las señales, resultando en menores falsos positivos, y una mejor matriz de confusión.

4.4.4 GESTO SHAKE

De las pruebas de usabilidad surgieron 3 gestos que poseen la particularidad de tener periodicidad con una frecuencia alta en varios de los ejes de cada sensor. Estos gestos fueron denominados gestos de sacudir, o *Shakes*, y si bien poseen diferencias, su similitud complicó el reconocimiento de los mismos. La Figura 4.4.4.1 muestra un ejemplo de 3 Shakes.



4.4.4.1 Ejemplo de los tres gestos Shake propuestos por los usuarios, superpuestos. Notar como cada gesto posee un eje con mayor amplitud distinto, pero que en general hay mucha periodicidad en todos los ejes.

Conflicto con Dynamic Time Warping

Los primeros intentos de reconocer gestos *Shake* con Dynamic Time Warping fracasaron por la naturaleza de estos gestos, y su periodicidad particular.

Esto se debe a que Dynamic Time Warping calcula la distancia mínima entre dos señales que deben parecerse en forma. Sin embargo, dichos gestos pueden tener muchos picos, ya que los usuarios no definieron una cantidad exacta de sacudidas que deben ejercerse sobre el dispositivo. Esto resulta en que el algoritmo no puede ajustar gestos con diferentes periodos y frecuencias, por lo que era necesario utilizar otro método.

Algoritmo de reconocimiento de gestos *Shake*

Dada la naturaleza periódica de estos gestos, se podría utilizar la Transformada de Fourier para analizar los gestos en el dominio de frecuencia, pero se optó por un camino más práctico que obtuvo buenos resultados.

El algoritmo utilizado busca máximos locales con una ventana ajustada según observación de los gestos muestreados (Ver Figura 4.4.4.2). Si el gesto supera una cantidad determinada de máximos locales sobre un umbral particular, se lo considera un gesto *Shake*.

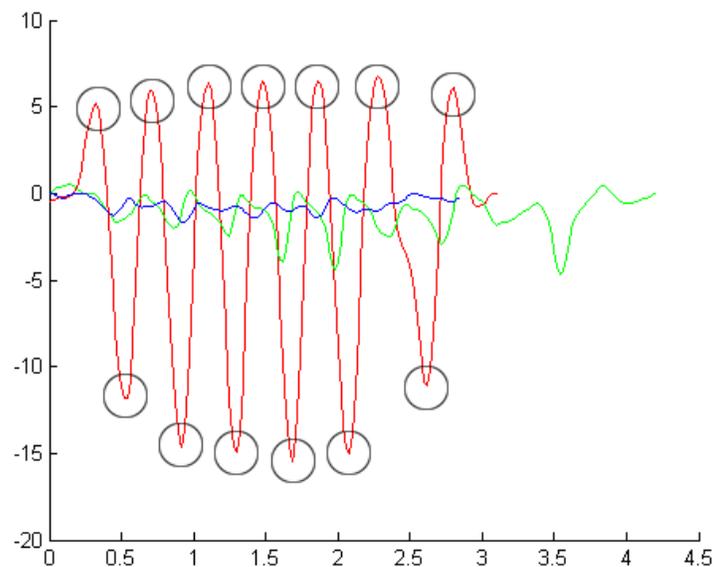


Figura 4.4.4.2 Máximos locales de un gesto *Shake* resaltados. Se requieren al menos 5 para considerar un gesto como *Shake*.

Conflictos entre Shakes

Utilizando este método es posible diferenciar los gestos no periódicos de los gestos *Shake*, pero no garantiza que éstos mismos sean diferenciables entre sí. Luego de analizar los gestos utilizando este método, se encontró que todos los gestos *Shake* conflictúan entre sí. Por esto, se decidió considerarlos por separado, y asumir que si en una aplicación es necesario algún tipo de *Shake*, solo puede convivir uno.

Esto se debe a que si bien existe una tendencia a que algún eje se destaque según su amplitud, las imperfecciones de la mano al ejecutar el gesto generan tanto ruido en los demás ejes que a veces es imposible diferenciarlos. Esto, sumado a que los gestos de sacudida seleccionados por los usuarios son relativamente complejos, y contienen cambios importantes en aceleración y orientación, hace que los gestos *Shake* deban ser reconocidos por separado.

4.4.5 GESTO ORBITAR

Un gesto interesante para analizar es el gesto para manipular la rotación de una cámara de 360 grados (Ver Figura 4.4.5.1). El escenario que se planteó en las primeras pruebas con los usuarios estaba principalmente orientado a juegos de exploración en primera persona, donde el jugador debe encontrar pistas, interactuar con objetos y observar su entorno.

Los resultados de este escenario fueron casi unánimes: la mayoría optó por orbitar con el celular alrededor del jugador, generando una traslación circular y manteniendo la orientación hacia el jugador.

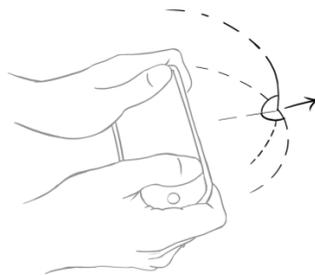


Figura 4.4.5.1 Gesto orbital. El usuario mueve el celular en 360 grados, como si fuese una cámara.

Este gesto se destacó por tener una diferencia particular con los demás gestos. La principal diferencia es que orbitar con el celular podría interpretarse como una mecánica para el ingreso

de input más que una secuencia fija de acciones, por lo que es difícil considerarlo como un gesto más clásico. Esto se debe a que no tiene ni principio ni fin, ya que puede empezar y terminar en cualquier punto de la circunferencia alrededor del jugador; es decir, podría considerarse como un gesto *continuo*.

Por esto se decidió evitar el gesto de órbita de una cámara en las pruebas de reconocimiento de gestos, ya que iba a tener conflictos con todos los gestos que obtuvimos debido a su naturaleza continua. Sin embargo, el reconocimiento de éste gesto continuo es relativamente sencillo; simplemente se debe tomar la orientación relativa y utilizarla como input de rotación para la cámara virtual dentro del juego.

Es interesante notar la unanimidad con la que los usuarios propusieron este gesto. Una posible razón de este resultado es interpretar el celular como una analogía a una cámara real y asumir que el jugador está inmerso en el mundo virtual, por lo que para observar su alrededor debe moverse y también orientarse.

5. CONJUNTOS DE GESTOS NO CONFLICTIVOS Y APLICACIÓN DEMOSTRATIVA

5.1 CONJUNTOS NO CONFLICTIVOS

Definimos un conjunto conflictivo como un conjunto de *tipos de gestos* que tienen problemas para ser reconocidos simultáneamente. En general, gestos que comparten movimientos muy similares, como mover el celular hacia adelante y mover el celular hacia adelante y arriba, resultaron ser conflictivos, mientras que gestos que son ortogonales en su eje de mayor amplitud son reconocibles.

Es de particular interés notar que el gesto de tocar el celular de con un dedo detrás de la pantalla (o M1 según la tabla 4.4.1.1) resultó ser muy conflictivo, y en general, cuando el algoritmo no encontraba una buena respuesta, devolvía este gesto. Asumimos que esto se debe a que este gesto tiene muy poca información y es muy sutil, por lo que es muy similar a un ruido estadístico, y DTW podría llegar a ajustarlo a cualquier curva con un margen razonable.

Sin embargo, como la matriz de confusión obtenida en los resultados indica, el error es razonable considerando el tamaño del conjunto de gestos. Es por esta razón que el único conjunto de gestos no conflictivos es el que no contiene ningún gesto Shake. De hecho, como previamente se discute en la sección del gesto Shake (ver Sección 4.4.4), todos los gestos de este estilo son conflictivos entre sí, por lo que se recomienda utilizar sólo un gesto Shake en una aplicación o videojuego.

En la implementación, cada vez que se reconoce un gesto, se evita compararlo con las señales de todos los gestos que pertenecen a su mismo conjunto de conflicto.

5.2 APLICACIÓN DEMOSTRATIVA

Una vez despejadas las incertidumbres acerca del reconocimiento de los gestos ejemplares, se utilizó la aplicación de Android desarrollada para la segunda parte del proyecto, solo que con un subconjunto de los gestos ejemplares. Esto se decidió para que el reconocimiento pueda

lograrse en *realtime*, ya que la cantidad de grabaciones por gesto es elevada para el dispositivo móvil. La aplicación es esencialmente *Gesture Checker*, pero que utiliza el conjunto de entrenamiento final.

Es decir, dado que los juegos estudiados en este trabajo no requieren más de 2 o 3 gestos simultáneamente, se analizó la factibilidad del reconocimiento online en el dispositivo móvil utilizando no más de 3 gestos que no sean conflictivos según el estudio realizado. Se aplicó la misma metodología descrita previamente, sólo que los gestos a tener en cuenta fueron pocos, pudiéndose obtener una respuesta en tiempo real.

6. CONCLUSIONES Y TRABAJO FUTURO

La principal motivación de este proyecto es encontrar y descubrir gestos *naturales* en el contexto de los videojuegos móviles utilizando la plataforma Android. Para esto, se desarrolló un experimento de diseño centrado en el usuario (UCD) para registrar y luego analizar propuestas de gestos por un conjunto de usuarios seleccionados.

Luego, utilizando la información obtenida en las entrevistas del experimento, se clasificaron todos los gestos propuestos y se formalizó un conjunto de *gestos ejemplares*, es decir, el conjunto de gestos populares para cada contexto seleccionado (en este caso, videojuegos). A partir de esta información, se decidió analizar de forma más concreta la posibilidad de reconocer estos gestos, por lo que se propuso un experimento para validar y verificar que dichos gestos puedan ser reconocidos simultáneamente. Y, si esto no es posible, encontrar cuáles gestos son conflictivos mediante la construcción de una matriz de confusión.

Para este segundo experimento se desarrolló una aplicación móvil para el reconocimiento *realtime* de los gestos, y se encontró que la cantidad de información requerida para poder reconocer correctamente las señales era voluminosa, por lo que se optó por el reconocimiento offline. Una vez re-implementado el algoritmo en MATLAB, filtros para las señales y ciertas aplicaciones de utilidades, se construyó una matriz de confusión para verificar los resultados.

Se encontró que el reconocimiento de los gestos propuestos por los usuarios no posee grandes conflictos, exceptuando el caso de los gestos que requieren sacudir el dispositivo (o Shake). En este último caso se decidió considerar que las aplicaciones solo pueden reconocer un solo gesto Shake simultáneamente; sin embargo, se considera que pocas aplicaciones (en particular videojuegos) requieran varios gestos Shake.

De esta forma, se validó que el reconocimiento de los gestos propuestos es posible, y se propone que los videojuegos pueden y deberían utilizar gestos similares para la interacción con el jugador. El área de interacción videojuego-jugador en dispositivos móviles mediante gestos de movimiento sigue sin ser explorada, por lo que es necesaria más investigación en este ámbito.

Diversas posibles extensiones de este trabajo pueden resultar interesantes. Por un lado, si bien los juegos en los dispositivos fueron usados por los usuarios en situaciones de aislamiento sería interesante extender el trabajo para ver en qué medida se dificulta el reconocimiento de los gestos cuando el usuario está, por ejemplo, en movimiento. Contextos como un vehículo de transporte público (ya sea colectivo, subte o auto) pueden ser de interés, ya que suelen ser los momentos de mayor ocio de los usuarios, y sería ideal verificar si el reconocimiento funciona en estos ambientes.

Por otro lado, el uso de los sensores en los dispositivos móviles se vuelve interesante para identificar las acciones que los usuarios realizan. Existen trabajos sobre cómo usar un dispositivo móvil guardado en un bolsillo para diferenciar si un usuario sentado en una silla está parándose o bien está ascendiendo una escalera. Asimismo, creemos que podría explorarse el uso de los sensores de movimiento del dispositivo para poder identificar en una sesión de gimnasia de distintos tipos de ejercicios que el usuario esté realizando, como ejercicios de abdominales o flexiones.

7. BIBLIOGRAFÍA

1. Edwin L. Hutchins, James D. Hollan, and Donald A. Norman. 1985. *Direct manipulation interfaces*. Hum.-Comput. Interact. 1, 4 (December 1985), 311-338.
2. Ruiz, J., Yang Li, Lank, E.. *User-defined Motion Gestures for Mobile Interaction*. 2011.
3. Matei Negulescu, Jaime Ruiz, Yang Li, and Edward Lank. 2012. *Tap, swipe, or move: attentional demands for distracted smartphone input*. In *Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI '12)*. ACM, New York, NY, USA, 173-180.
4. Carsten Schwesig, Ivan Poupyrev, and Eijiro Mori. 2003. *Gummi: user interface for deformable computers*. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems (CHI EA '03)*. ACM, New York, NY, USA, 954-955
5. Carsten Schwesig, Ivan Poupyrev, and Eijiro Mori. 2004. *Gummi: a bendable computer*. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*. ACM, New York, NY, USA, 263-270
6. Wobbrock, J. O., Morris, M. R., Wilson, A. D.. *User-defined Gestures for Surface Computing*. 2009.
7. Sang-Su Lee, Sohyun Kim, Bopil Jin, Eunji Choi, Boa Kim, Xu Jia, Daeop Kim, and Kun-pyo Lee. 2010. *How users manipulate deformable displays as input devices*. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA, 1647-1656.
8. Salvador, S., Chan, P.. *FastDTW: Toward Accurate Dynamic Time Warping in Linear Time And Space*.
9. Müller, M. 2007. *Information Retrieval for Music and Motion*, Springer.
10. GangOfFour, 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley Professional.

Arquitecturas

SensorLogger: aplicación de recolección de datos

SensorLogger es uno de los dos desarrollos principales de este trabajo. Es la aplicación que se encarga de presentarle al usuario tester los distintos escenarios y generar archivos de output con los valores obtenidos de los sensores, para luego post-procesar los datos y poder deducir información de utilidad. SensorLogger fue la app que se utilizó en la primera etapa de pruebas con usuario.

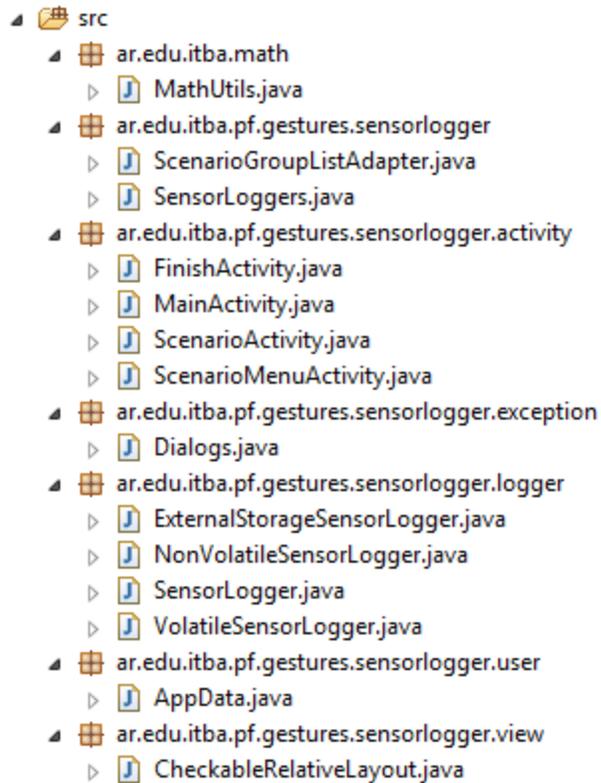
Esta aplicación fue hecha utilizando el SDK 17 de Android, objetivando dispositivos con SDKs entre 9 y 17, siendo este último el más recomendable. Los únicos permisos declarados por la aplicación son:

- `android.permission.INTERNET`
- `android.permission.READ_EXTERNAL_STORAGE`
- `android.permission.WRITE_EXTERNAL_STORAGE`

BACK-END

La arquitectura de la aplicación es similar a la de cualquier aplicación standard de Android. Cada *activity* representa una pantalla del flujo y la lógica de cada una de ellas se encuentra en su respectiva clase Java.

Así, la estructura de paquetes (y sus clases) del proyecto es como se muestra a continuación:



De todas las clases que se muestran en el árbol de la imagen, sólo las siguientes representan algún interés para este trabajo, organizadas de acuerdo al tipo de funciones que ofrecen:

Utilidades

- **SensorLoggers.java**

[ar.edu.itba.pf.gestures.sensorlogger]

Utility class estática con métodos para el manejo y mapeo de los tipos de sensores y la instanciación de cada uno de ellos.

El método más destacado de esta clase es:

```
public static SensorLogger createSensorLogger(...)
```

Algunas características de este método:

- Sigue el patrón de diseño *factory*.
- Es un método de clase global.
- Recibe:
 - **Sensor** sensor: el sensor que va a observar por input.

- **String** delay: el delay con el cual se va a registrar dicho sensor.
- **Scenario** scenario: el escenario en cuestión.
- **int** sampleSize: la cantidad de muestras que se van a tomar de ese sensor.
- **AppData** ad: instancia con información como nombre de usuario, tipo de persistencia elegido al comienzo, URL del archivo de escenarios, etc..
- Este método se encarga de obtener el tipo de persistencia elegida por el usuario e instanciar un nuevo *logger* de acuerdo con base a eso. Si el usuario decidió no persistir la información, el tipo de *logger* retornado será *VolatileSensorLogger*. En caso contrario, *ExternalStorageSensorLogger*. Más adelante se explica en detalle toda la jerarquía de la clase *SensorLogger*.

- **AppData.java**

[ar.edu.itba.pf.gestures.sensorlogger.user]

Se utilizó un Singleton, patrón de diseño propuesto por el Gang of Four [10] con el objetivo de manipular instancias únicas y globales de ciertos objetos, para facilitar información de la ejecución actual de la aplicación y el input del usuario:

- Timestamp de comienzo.
- Modelo del dispositivo.
- Usuario y URL del archivo de escenarios.
- Tipo de persistencia.

Lógica/controlador

- **MainActivity.java**

[ar.edu.itba.pf.gestures.sensorlogger.activity]

Se encarga de inicializar la aplicación. Cargar las preferencias y mantiene referencias a los distintos singletons que se usarán a lo largo del proceso (*SharedPreferences* y *AppData*).

- **ScenarioMenuActivity.java**

[ar.edu.itba.pf.gestures.sensorlogger.activity]

Se encarga de ir a buscar los escenarios a la URL introducida en el paso previo y mostrar los grupos en una lista seleccionable, para que el usuario elija qué

grupos desea ejecutar.

Durante el desarrollo de esta clase, encontramos problemas para cargar los escenarios y mostrarlos en una lista con un *checkbox* para cada grupo. Podíamos mostrar la lista de grupos de escenarios seleccionables, pero no podíamos hacer para que aparezca la cantidad de escenarios debajo de cada grupo; de alguna manera, se rompía la funcionalidad de lista chequeable. Esta restricción se debía a limitaciones en la disposición de los *layouts* de Android. De esta manera, terminamos escribiendo la clase **CheckableRelativeLayout.java**¹, que se encuentra en el paquete `ar.edu.itba.pf.gestures.sensorlogger.view`, para resolver esta cuestión.

Por otro lado, debido a que Android utiliza un modelo *single-threaded* para sus aplicaciones (cada aplicación corre en un único *thread*, llamado *UI-thread*, que ejecuta cada comando de manera serial), podíamos ver que mientras los escenarios estaban siendo cargados, la pantalla dejaba de responder. No consideramos esto muy correcto en criterios de usabilidad, ya que daba la impresión de que había surgido algún tipo de problema y la aplicación estaba a punto de dejar de responder. Por lo tanto, decidimos agregar un pequeño *loader* a modo de *feedback* mientras el usuario esperaba a que aparezca la lista de grupos de escenarios. Fue necesario entonces utilizar la clase `AsyncTask` que ofrece el mismo framework de desarrollo para poder disparar una consulta asincrónica en un *thread* distinto a *UI-thread*, que obtenga el XML de escenarios y le avise a *UI-thread* que ya puede mostrar el listado. La clase interna `ScenarioLoaderTask` hereda de `AsyncTask` y realiza este procesamiento.

Mantiene una variable global con una lista indizada con los escenarios seleccionados. Esta práctica no está muy bien vista por la comunidad de desarrolladores de Android ya que puede producir *memory leaks*, entre otras cosas. Sin embargo, por un tema de practicidad, y dada la envergadura de la aplicación que estamos desarrollando (no hay consecuencias realmente críticas), decidimos utilizar variables globales.

- **ScenarioActivity.java**

¹ Basada ampliamente en la clase `CheckableLinearLayout.java` del siguiente enlace:
<http://tokudu.com/2010/android-checkable-linear-layout/>

[ar.edu.itba.pf.gestures.sensorlogger.activity]

Se encarga de ejecutar cada uno de los escenarios incluidos en los grupos seleccionados en el paso previo.

Es importante destacar que cada ejecución de un escenario implica una nueva instancia de `ScenariActivity`, es decir, una *activity* distinta. Para mantener secuencialidad en las pruebas (que después de un escenario se avance al siguiente), cada `ScenariActivity` le pasa al siguiente el índice del escenario que le corresponde ejecutar, obteniéndolo efectivamente de la lista de escenarios global referenciada desde `ScenariMenuActivity`.

Una vez obtenido el escenario, se inicializan los sensores que van a estar involucrados invocando al método `initializeLoggers`. Para cada sensor se instancia un `SensorLogger` utilizando la *utility class* `SensorLoggers` mencionada en párrafos anteriores. Debido a que Android admite varios sensores de un mismo tipo en un dispositivo, se optó por utilizar el primer sensor devuelto en esa lista.

Para comenzar, pausar y parar las grabaciones basta con registrar/desregistrar los sensores del `SensorManager` de Android.

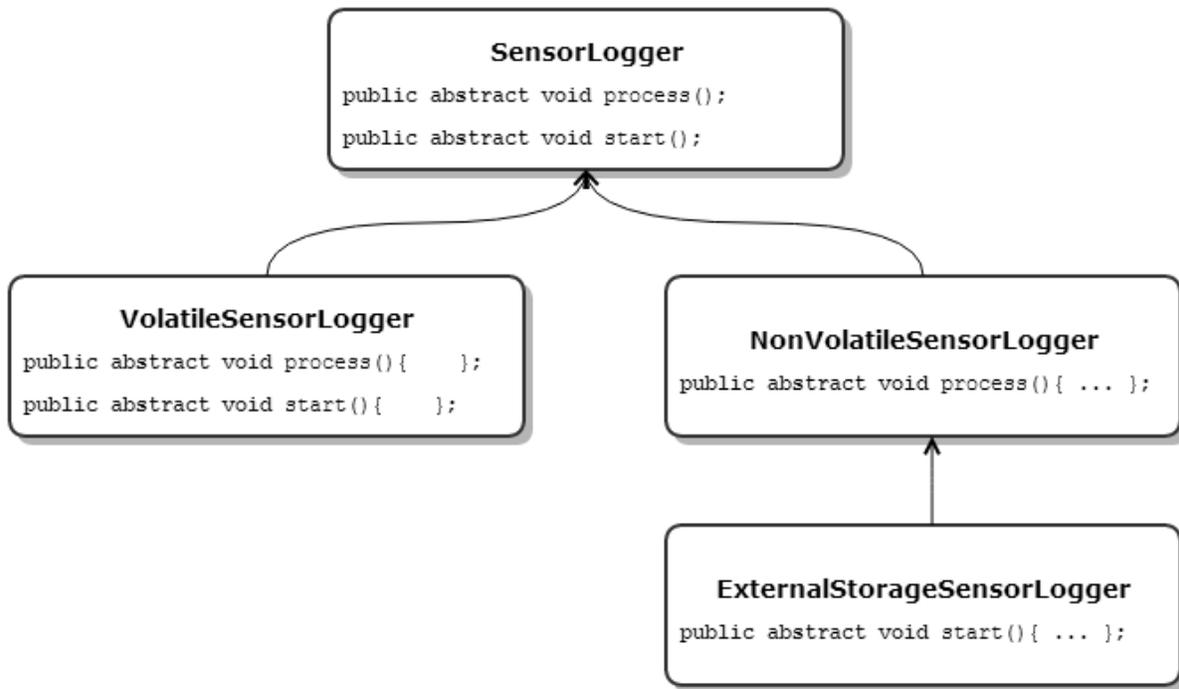
- **FinishActivity.java**

[ar.edu.itba.pf.gestures.sensorlogger.activity]

Se encarga de redirigir al usuario al menú principal o al menú de selección de escenarios. No hay ningún fragmento de código de interés.

Jerarquía de sensores

Para mantener un diseño centrado y respetar las buenas prácticas de programación orientada a objetos, se optó por desarrollar una completa jerarquía para los distintos tipos de sensores: aquellos que persisten los datos y aquellos que no. Cada `SensorLogger` es un wrapper de un sensor real del dispositivo (representado por la clase `Sensor` de la API). La jerarquía es como sigue:



- **SensorLogger.java**

[ar.edu.itba.pf.gestures.sensorlogger.logger]

Clase abstracta raíz de toda la jerarquía. Declara dos métodos **abstract**:

- **start**: No recibe parámetros ni retorna ningún valor. La idea de este método es que las clases que lo implementen realicen cierto tipo de operaciones de inicialización. Es invocado en el método `startRecording` de `ScenarioActivity`, en el loop que prepara cada `Sensor` del escenario y lo registra en el `SensorManager`.
- **process**: No recibe parámetros ni retorna ningún valor. Este método es invocado en el método `onSensorChanged` que implementa esta misma clase, cada vez que se reciben nuevos valores del sensor representado y se espera que las clases que lo implementen realicen operaciones en base a esos resultados.

El constructor en esta clase recibe únicamente los objetos necesarios para inicializar un `SensorLogger`: un `Sensor` a representar, una string con el delay con el cual se quiere registrar dicho sensor y la cantidad de muestras que van a ser tomadas.

Los valores posibles del delay se corresponden con los nombres de las constantes que representan los posibles valores del parámetro `rateUs` que reciben

los distintos erasures del métodos `registerListener` de la API²:

- `NORMAL` → `SENSOR_DELAY_NORMAL`
- `UI` → `SENSOR_DELAY_UI`
- `GAME` → `SENSOR_DELAY_GAME`
- `FASTEST` → `SENSOR_DELAY_FASTEST`

Finalmente, la clase `SensorLogger` implementa la interfaz de Android `SensorEventListener` y, en consecuencia, los métodos `onAccuracyChanged` y `onSensorChanged`. Es este último el que se invoca cada vez que se reciben valores nuevos del sensor en cuestión, y como se dijo en párrafos anteriores, el que a su vez llama a `process`.

Se intentó delegar la menor cantidad de comportamiento en las clases más abajo en la jerarquía. De esta manera, creemos haber logrado un buen aprovechamiento del código y una fácil extensión en caso de necesitarla.

- **`VolatileSensorLogger.java`**

[`ar.edu.itba.pf.gestures.sensorlogger.logger`]

Extiende de `SensorLogger`. Representa un logger de almacenamiento volátil, es decir, no persiste los datos. Implementa los métodos `start` y `process` dejándolos vacíos. No realiza ningún tipo de procesamiento ni al registrar un sensor ni al recibir nuevos valores.

- **`NonVolatileSensorLogger.java`**

[`ar.edu.itba.pf.gestures.sensorlogger.logger`]

Extiende de `SensorLogger`. Clase abstracta. Representa un logger de almacenamiento no volátil, es decir, ofrece persistencia de algún tipo. Contiene una referencia a una instancia de la clase `java.io.PrintWriter` en la variable **`protected`** `out` como stream de salida de texto. Implementa el método `process` invocando a `out.println(...)` con los últimos valores recibidos del sensor.

- **`ExternalStorageSensorLogger.java`**

[`ar.edu.itba.pf.gestures.sensorlogger.logger`]

Extiende de `NonVolatileSensorLogger`. Representa almacenamiento en la

² <http://developer.android.com/reference/android/hardware/SensorManager.html>

memoria externa del dispositivo. El constructor invoca el constructor super y además crea el archivo de output. Implementa el método `start`, que inicializa la variable `out` con un `PrintWriter` compuesto de un `OutputStreamWriter` compuesto de un `CompressedBlockOutputStream` compuesto de un `FileOutputStream`. De esta manera, se persisten los datos de manera comprimida por stream en el archivo de salida. Por otro lado, en el mismo método se escribe un header inicial para el archivo con la siguiente información acerca de la ejecución:

- Nombre del usuario.
- Modelo del dispositivo.
- Timestamp de comienzo.
- ID y nombre del escenario que se está persistiendo.
- Nombre y tipo del sensor.
- Número de grabación (cuántas veces fue regrabado el escenario).

GestureChecker: aplicación de comparación de gestos

`GestureChecker` es el otro desarrollo principal de este trabajo y aquél que se utilizó en la segunda etapa de pruebas con usuarios. Por cada gesto específico deducido de las pruebas previas, la aplicación presentará una pantalla en la cual el usuario realizará dicho gesto para que la aplicación lo aprenda (modo de operación de entrenamiento) para luego verificar si es capaz de volver a detectarlo (modo de operación de reconocimiento). `GestureChecker` permite chequear la complejidad y efectividad de detección de un gesto y en consecuencia, su viabilidad.

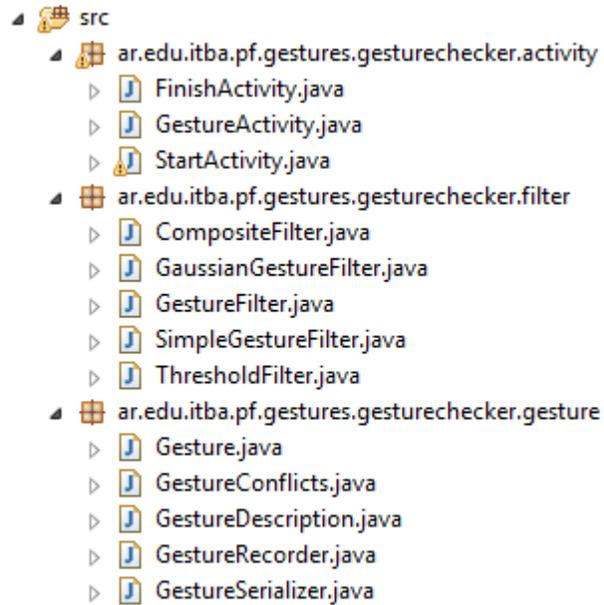
Esta aplicación fue hecha utilizando el SDK 17 de Android, objetivando dispositivos con SDKs entre 9 y 17, siendo este último el más recomendable. Los únicos permisos declarados por la aplicación son:

- `android.permission.WRITE_SETTINGS`
- `android.permission.ACCESS_FINE_LOCATION`
- `android.permission.WRITE_EXTERNAL_STORAGE`

BACK-END

La arquitectura de la aplicación es similar a la de cualquier aplicación standard de Android. Cada *activity* representa una pantalla del flujo y la lógica de cada una de ellas se encuentra en su respectiva clase Java.

Así, la estructura de paquetes (y sus clases) del proyecto es como se muestra a continuación:



De todas las clases que se muestran en el árbol de la imagen, sólo las siguientes representan algún interés para este trabajo, organizadas de acuerdo al tipo de funciones que ofrecen:

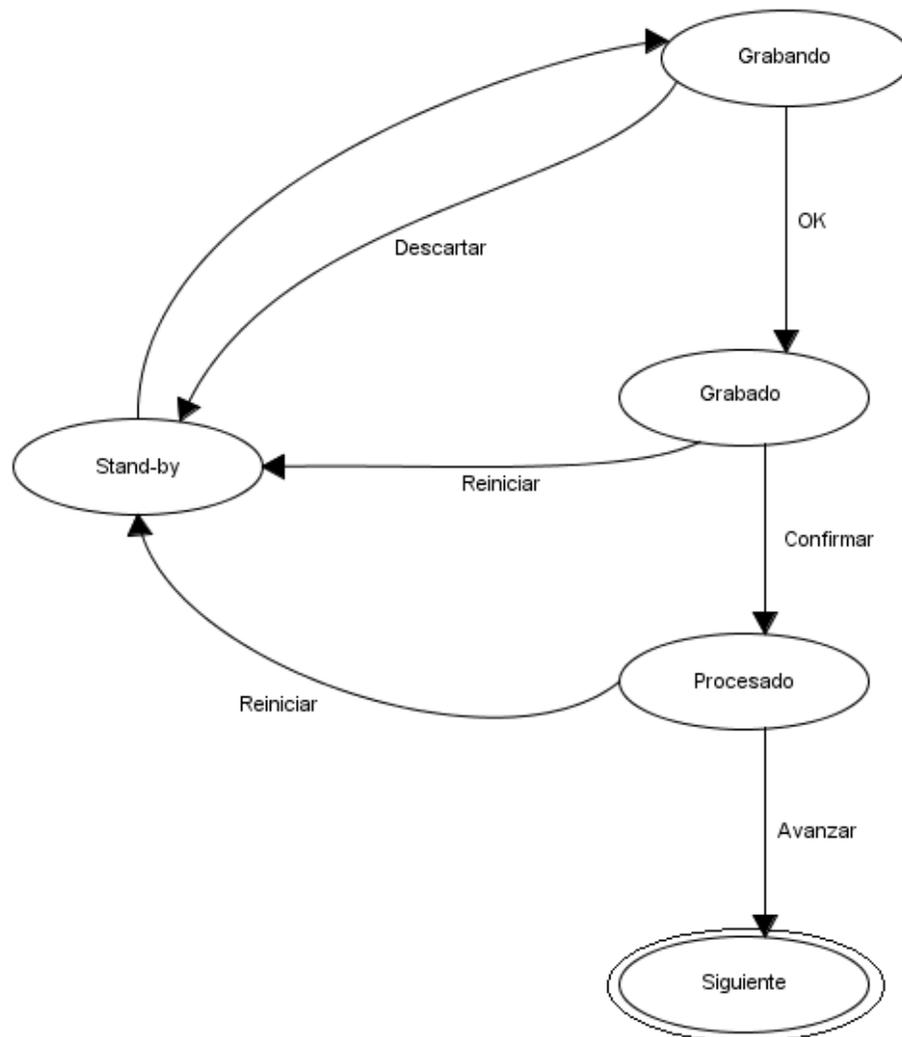
Lógica/controlador

- **GestureActivity.java**

[ar.edu.itba.pf.gestures.gesturechecker.activity]

Para cada gesto, se instancia una `GestureActivity` que representa todo el proceso que realizará el usuario antes de pasar al gesto siguiente. Muestra el título y la descripción del gesto actual, y una barra de acciones con estado que varía de acuerdo con la etapa en la cual se encuentre la pantalla.

Los estados por los que pasa la barra de acciones son como muestra la imagen a continuación:



- **Stand-by**: no se realizó ningún tipo de operación sobre el gesto. No hay datos de input grabados por el usuario. La única acción posible es iniciar la grabación.
- **Grabando**: la grabación ha sido iniciada. Una vez que ha terminado de ingresar input, el usuario puede finalizar la grabación y pasar a **Grabado** o descartarla y volver a **Stand-by**.
- **Grabado**: se aceptó la grabación. El usuario puede confirmarla y avanzar a **Procesado** para procesar los datos o rechazarla y volver a **Stand-by**.
- **Procesado**: los datos ingresados por el usuario fueron procesados.

Las operaciones que se realizan en este estado dependen del modo en el que esté ejecutándose la aplicación:

- Aprendizaje: se aprende el gesto en cuestión en base al input grabado por el usuario. Se muestra un mensaje para indicar que no hubo problemas en aprender el gesto.
- Reconocimiento: se compara el input del usuario con el input grabado para este gesto en la fase de aprendizaje. Se muestra un mensaje para indicar si la detección fue satisfactoria o no. El usuario puede reiniciar todo el proceso para el gesto en el cual se encuentre el loop y volver a **Stand-by** o avanzar al **Siguiente**.
- **Siguiente**: el procesamiento para el gesto actual fue terminado. La aplicación volverá a realizar todo este proceso para el siguiente gesto cargado.

Cada estado exceptuando **Siguiente** (es un estado efímero de terminación en el que la aplicación se encuentra previo a pasar al estado **Stand-by** del siguiente gesto) y **Stand-by** (no lo requiere) posee la opción de reiniciar el proceso, dándole al usuario la posibilidad de volver a empezar de nuevo en cualquier momento para cada gesto.

Los métodos que se encargan de realizar las transiciones entre los estados se marcan con rojo en la imagen y se describen a continuación:

- **void onRec(View)**
Origen: **Stand-by**
Destino: **Grabando**
Invoca a los métodos `clear` y `pause(false)` de la instancia de `GestureRecorder` que mantiene la actividad, descartando cualquier input grabado previamente e iniciando efectivamente la grabación.
-

8. APÉNDICE

8.1 Archivo de escenarios utilizado por SensorLogger

```
1. <scenarios xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="scenarios.xsd">
2.
3.     <group id="1" name="Juegos">
4.
5.         <scenario id="1" name="Sidescroller 2D de pelea" duration="10">
6.             <description>
7.                 Mover el personaje en el eje de profundidad. Ejemplo: Double
   Dragon.
8.             </description>
9.             <sensor>LINEAR_ACCELERATION</sensor>
10.            <sensor>PROXIMITY</sensor>
11.            <sensor>ACCELEROMETER</sensor>
12.            <sensor>GYROSCOPE</sensor>
13.        </scenario>
14.
15.        <scenario id="2" name="Sidescroller 2D de pelea" duration="10">
16.            <description>
17.                Mover el personaje en el eje de profundidad. Ejemplo: Double
   Dragon.
18.            </description>
19.            <sensor>LINEAR_ACCELERATION</sensor>
20.            <sensor>PROXIMITY</sensor>
21.            <sensor>ACCELEROMETER</sensor>
22.            <sensor>GYROSCOPE</sensor>
23.        </scenario>
24.
25.        <scenario id="3" name="Juego de pelea 2D" duration="10" >
26.            <description>
```

```

27.             Movearse horizontalmente hacia la derecha. Ejemplo: Street
Fighter.
28.             </description>
29. <sensor>LINEAR_ACCELERATION</sensor>
30.             <sensor>PROXIMITY</sensor>
31. <sensor>ACCELEROMETER</sensor>
32. <sensor>GYROSCOPE</sensor>
33. </scenario>
34.
35. <scenario id="4" name="Juego de pelea 2D" duration="10" >
36.     <description>
37.         Movearse horizontalmente hacia la derecha. Ejemplo: Street
Fighter.
38.     </description>
39.     <sensor>LINEAR_ACCELERATION</sensor>
40.         <sensor>PROXIMITY</sensor>
41.     <sensor>ACCELEROMETER</sensor>
42.     <sensor>GYROSCOPE</sensor>
43. </scenario>
44.
45. <scenario id="5" name="Juego de pelea 2D" duration="10" >
46.     <description>
47.         Saltar. Ejemplo: Street Fighter.
48.     </description>
49.     <sensor>LINEAR_ACCELERATION</sensor>
50.     <sensor>PROXIMITY</sensor>
51.     <sensor>ACCELEROMETER</sensor>
52.     <sensor>GYROSCOPE</sensor>
53. </scenario>
54.
55. <scenario id="6" name="Juego de pelea 2D" duration="10" >
56.     <description>
57.         Saltar. Ejemplo: Street Fighter.
58.     </description>

```

```

59.      <sensor>LINEAR_ACCELERATION</sensor>
60.      <sensor>PROXIMITY</sensor>
61.      <sensor>ACCELEROMETER</sensor>
62.      <sensor>GYROSCOPE</sensor>
63.      </scenario>
64.
65.      <scenario id="7" name="Pinball" duration="10" >
66.          <description>
67.              Lanzar la pelota.
68.          </description>
69.          <sensor>LINEAR_ACCELERATION</sensor>
70.              <sensor>PROXIMITY</sensor>
71.          <sensor>ACCELEROMETER</sensor>
72.          <sensor>GYROSCOPE</sensor>
73.      </scenario>
74.
75.      <scenario id="8" name="Pinball" duration="10" >
76.          <description>
77.              Lanzar la pelota.
78.          </description>
79.          <sensor>LINEAR_ACCELERATION</sensor>
80.              <sensor>PROXIMITY</sensor>
81.          <sensor>ACCELEROMETER</sensor>
82.          <sensor>GYROSCOPE</sensor>
83.      </scenario>
84.
85.      <scenario id="9" name="Pinball" duration="10" >
86.          <description>
87.              Usar paleta derecha.
88.          </description>
89.          <sensor>LINEAR_ACCELERATION</sensor>
90.          <sensor>PROXIMITY</sensor>
91.          <sensor>ACCELEROMETER</sensor>
92.          <sensor>GYROSCOPE</sensor>

```

```

93.         </scenario>
94.
95.         <scenario id="10" name="Pinball" duration="10" >
96.             <description>
97.                 Usar paleta derecha.
98.             </description>
99.         <sensor>LINEAR_ACCELERATION</sensor>
100.        <sensor>PROXIMITY</sensor>
101.        <sensor>ACCELEROMETER</sensor>
102.        <sensor>GYROSCOPE</sensor>
103.        </scenario>
104.
105.        <scenario id="11" name="Pinball" duration="10" >
106.            <description>
107.                Pegarle al tablero (tilt).
108.            </description>
109.        <sensor>LINEAR_ACCELERATION</sensor>
110.        <sensor>PROXIMITY</sensor>
111.        <sensor>ACCELEROMETER</sensor>
112.        <sensor>GYROSCOPE</sensor>
113.        </scenario>
114.
115.        <scenario id="12" name="Pinball" duration="10" >
116.            <description>
117.                Pegarle al tablero (tilt).
118.            </description>
119.        <sensor>LINEAR_ACCELERATION</sensor>
120.        <sensor>PROXIMITY</sensor>
121.        <sensor>ACCELEROMETER</sensor>
122.        <sensor>GYROSCOPE</sensor>
123.        </scenario>
124.
125.        <scenario id="13" name="Juegos de plataformas" duration="10" >
126.            <description>

```

```

127.         Frenar personaje cuando usa skate. Ejemplo: Wonder Boy.
128.     </description>
129.     <sensor>LINEAR_ACCELERATION</sensor>
130.     <sensor>PROXIMITY</sensor>
131.     <sensor>ACCELEROMETER</sensor>
132.     <sensor>GYROSCOPE</sensor>
133. </scenario>
134.
135. <scenario id="14" name="Juegos de plataformas" duration="10" >
136.     <description>
137.         Frenar personaje cuando usa skate. Ejemplo: Wonder Boy.
138.     </description>
139.     <sensor>LINEAR_ACCELERATION</sensor>
140.     <sensor>PROXIMITY</sensor>
141.     <sensor>ACCELEROMETER</sensor>
142.     <sensor>GYROSCOPE</sensor>
143. </scenario>
144.
145. <scenario id="15" name="First-person Shooter" duration="10" >
146.     <description>
147.         Recargar arma. Ejemplo: Dead Trigger.
148.     </description>
149.     <sensor>LINEAR_ACCELERATION</sensor>
150.     <sensor>PROXIMITY</sensor>
151.     <sensor>ACCELEROMETER</sensor>
152.     <sensor>GYROSCOPE</sensor>
153. </scenario>
154.
155. <scenario id="16" name="First-person Shooter" duration="10" >
156.     <description>
157.         Recargar arma. Ejemplo: Dead Trigger.
158.     </description>
159.     <sensor>LINEAR_ACCELERATION</sensor>
160.     <sensor>PROXIMITY</sensor>

```

```

161. <sensor>ACCELEROMETER</sensor>
162. <sensor>GYROSCOPE</sensor>
163. </scenario>
164.
165. <scenario id="17" name="First-person shooter" duration="10" >
166.   <description>
167.     Cambiar de arma activa. Ejemplo: Dead Trigger.
168.   </description>
169. <sensor>LINEAR_ACCELERATION</sensor>
170. <sensor>PROXIMITY</sensor>
171. <sensor>ACCELEROMETER</sensor>
172. <sensor>GYROSCOPE</sensor>
173. </scenario>
174.
175. <scenario id="18" name="First-person shooter" duration="10" >
176.   <description>
177.     Cambiar de arma activa. Ejemplo: Dead Trigger.
178.   </description>
179. <sensor>LINEAR_ACCELERATION</sensor>
180. <sensor>PROXIMITY</sensor>
181. <sensor>ACCELEROMETER</sensor>
182. <sensor>GYROSCOPE</sensor>
183. </scenario>
184.
185. <scenario id="19" name="Shoot 'em up" duration="10" >
186.   <description>
187.     Disparar. Ejemplos: Space Invaders, 1942.
188.   </description>
189. <sensor>LINEAR_ACCELERATION</sensor>
190. <sensor>PROXIMITY</sensor>
191. <sensor>ACCELEROMETER</sensor>
192. <sensor>GYROSCOPE</sensor>
193. </scenario>
194.

```

```

195. <scenario id="20" name="Shoot 'em up" duration="10" >
196.   <description>
197.     Disparar. Ejemplos: Space Invaders, 1942.
198.   </description>
199.   <sensor>LINEAR_ACCELERATION</sensor>
200.   <sensor>PROXIMITY</sensor>
201.   <sensor>ACCELEROMETER</sensor>
202.   <sensor>GYROSCOPE</sensor>
203. </scenario>
204.
205. <scenario id="21" name="Survival Horror" duration="10" >
206.   <description>
207.     Mover la cámara hacia la derecha. Ejemplos: Amnesia.
208.   </description>
209.   <sensor>LINEAR_ACCELERATION</sensor>
210.   <sensor>PROXIMITY</sensor>
211.   <sensor>ACCELEROMETER</sensor>
212.   <sensor>GYROSCOPE</sensor>
213. </scenario>
214.
215. <scenario id="22" name="Survival Horror" duration="10" >
216.   <description>
217.     Mover la cámara hacia la derecha. Ejemplos: Amnesia.
218.   </description>
219.   <sensor>LINEAR_ACCELERATION</sensor>
220.   <sensor>PROXIMITY</sensor>
221.   <sensor>ACCELEROMETER</sensor>
222.   <sensor>GYROSCOPE</sensor>
223. </scenario>
224.
225. <scenario id="23" name="Carreras de autos" duration="10" >
226.   <description>
227.     Manejar la caja de cambios. Subir un cambio. Ejemplos: Asphalt

```

8.

```

228.         </description>
229.     <sensor>LINEAR_ACCELERATION</sensor>
230.     <sensor>PROXIMITY</sensor>
231.     <sensor>ACCELEROMETER</sensor>
232.     <sensor>GYROSCOPE</sensor>
233. </scenario>
234.
235. <scenario id="24" name="Carreras de autos" duration="10" >
236.     <description>
237.         Manejar la caja de cambios. Subir un cambio. Ejemplos: Asphalt
8.
238.         </description>
239.     <sensor>LINEAR_ACCELERATION</sensor>
240.     <sensor>PROXIMITY</sensor>
241.     <sensor>ACCELEROMETER</sensor>
242.     <sensor>GYROSCOPE</sensor>
243. </scenario>
244.
245. <scenario id="25" name="Tennis" duration="10" >
246.     <description>
247.         Mover la raqueta/Pegarle a la pelota. Ejemplos: Wii Sports.
248.     </description>
249.     <sensor>LINEAR_ACCELERATION</sensor>
250.     <sensor>PROXIMITY</sensor>
251.     <sensor>ACCELEROMETER</sensor>
252.     <sensor>GYROSCOPE</sensor>
253. </scenario>
254.
255. <scenario id="26" name="Tennis" duration="10" >
256.     <description>
257.         Mover la raqueta/Pegarle a la pelota. Ejemplos: Wii Sports.
258.     </description>
259.     <sensor>LINEAR_ACCELERATION</sensor>
260.     <sensor>PROXIMITY</sensor>

```

```
261. <sensor>ACCELEROMETER</sensor>
262. <sensor>GYROSCOPE</sensor>
263. </scenario>
264.
265. <scenario id="27" name="Tennis" duration="10" >
266.   <description>
267.     Lanzar pelota. Ejemplos: Wii Sports.
268.   </description>
269. <sensor>LINEAR_ACCELERATION</sensor>
270. <sensor>PROXIMITY</sensor>
271. <sensor>ACCELEROMETER</sensor>
272. <sensor>GYROSCOPE</sensor>
273. </scenario>
274.
275. <scenario id="28" name="Tennis" duration="10" >
276.   <description>
277.     Lanzar pelota. Ejemplos: Wii Sports.
278.   </description>
279. <sensor>LINEAR_ACCELERATION</sensor>
280. <sensor>PROXIMITY</sensor>
281. <sensor>ACCELEROMETER</sensor>
282. <sensor>GYROSCOPE</sensor>
283. </scenario>
284.
285. <scenario id="29" name="Juegos de dados" duration="10" >
286.   <description>
287.     Lanzar dados. Ejemplos: Generala.
288.   </description>
289. <sensor>LINEAR_ACCELERATION</sensor>
290. <sensor>PROXIMITY</sensor>
291. <sensor>ACCELEROMETER</sensor>
292. <sensor>GYROSCOPE</sensor>
293. </scenario>
294.
```

```
295. <scenario id="30" name="Juegos de dados" duration="10" >
296.   <description>
297.     Lanzar dados. Ejemplos: Generala.
298.   </description>
299.   <sensor>LINEAR_ACCELERATION</sensor>
300.   <sensor>PROXIMITY</sensor>
301.   <sensor>ACCELEROMETER</sensor>
302.   <sensor>GYROSCOPE</sensor>
303. </scenario>
304.
305. <scenario id="31" name="Juegos de cartas" duration="10" >
306.   <description>
307.     Barajar cartas. Ejemplos: Magic The Gathering, Truco.
308.   </description>
309.   <sensor>LINEAR_ACCELERATION</sensor>
310.   <sensor>PROXIMITY</sensor>
311.   <sensor>ACCELEROMETER</sensor>
312.   <sensor>GYROSCOPE</sensor>
313. </scenario>
314.
315. <scenario id="32" name="Juegos de cartas" duration="10" >
316.   <description>
317.     Barajar cartas. Ejemplos: Magic The Gathering, Truco.
318.   </description>
319.   <sensor>LINEAR_ACCELERATION</sensor>
320.   <sensor>PROXIMITY</sensor>
321.   <sensor>ACCELEROMETER</sensor>
322.   <sensor>GYROSCOPE</sensor>
323. </scenario>
324.
325. <scenario id="33" name="Ajedrez" duration="10" >
326.   <description>
327.     Pasar (terminar) turno.
328.   </description>
```

```

329. <sensor>LINEAR_ACCELERATION</sensor>
330. <sensor>PROXIMITY</sensor>
331. <sensor>ACCELEROMETER</sensor>
332. <sensor>GYROSCOPE</sensor>
333. </scenario>
334.
335. <scenario id="34" name="Ajedrez" duration="10" >
336.   <description>
337.     Pasar (terminar) turno.
338.   </description>
339. <sensor>LINEAR_ACCELERATION</sensor>
340. <sensor>PROXIMITY</sensor>
341. <sensor>ACCELEROMETER</sensor>
342. <sensor>GYROSCOPE</sensor>
343. </scenario>
344.
345. <scenario id="35" name="Billar" duration="10" >
346.   <description>
347.     Disparar.
348.   </description>
349. <sensor>LINEAR_ACCELERATION</sensor>
350. <sensor>PROXIMITY</sensor>
351. <sensor>ACCELEROMETER</sensor>
352. <sensor>GYROSCOPE</sensor>
353. </scenario>
354.
355. <scenario id="36" name="Billar" duration="10" >
356.   <description>
357.     Disparar.
358.   </description>
359. <sensor>LINEAR_ACCELERATION</sensor>
360. <sensor>PROXIMITY</sensor>
361. <sensor>ACCELEROMETER</sensor>
362. <sensor>GYROSCOPE</sensor>

```

```
363.         </scenario>
364.
365.     </group>
366.
367.     <group id="2" name="Workout Routines">
368.
369.         <scenario id="37" name="Abdominales" duration="10" >
370.             <description>
371.
372.                 </description>
373.             <sensor>LINEAR_ACCELERATION</sensor>
374.             <sensor>PROXIMITY</sensor>
375.             <sensor>ACCELEROMETER</sensor>
376.             <sensor>GYROSCOPE</sensor>
377.         </scenario>
378.
379.         <scenario id="38" name="Abdominales" duration="10" >
380.             <description>
381.
382.                 </description>
383.             <sensor>LINEAR_ACCELERATION</sensor>
384.             <sensor>PROXIMITY</sensor>
385.             <sensor>ACCELEROMETER</sensor>
386.             <sensor>GYROSCOPE</sensor>
387.         </scenario>
388.
389.         <scenario id="39" name="Pushups" duration="10" >
390.             <description>
391.
392.                 </description>
393.             <sensor>LINEAR_ACCELERATION</sensor>
394.             <sensor>PROXIMITY</sensor>
395.             <sensor>ACCELEROMETER</sensor>
```

```
396. <sensor>GYROSCOPE</sensor>
397. </scenario>
398.
399. <scenario id="40" name="Pushups" duration="10" >
400.     <description>
401.
402.     </description>
403. <sensor>LINEAR_ACCELERATION</sensor>
404. <sensor>PROXIMITY</sensor>
405. <sensor>ACCELEROMETER</sensor>
406. <sensor>GYROSCOPE</sensor>
407. </scenario>
408.
409. <scenario id="41" name="Mariposa" duration="10" >
410.     <description>
411.
412.     </description>
413. <sensor>LINEAR_ACCELERATION</sensor>
414. <sensor>PROXIMITY</sensor>
415. <sensor>ACCELEROMETER</sensor>
416. <sensor>GYROSCOPE</sensor>
417. </scenario>
418.
419. <scenario id="42" name="Mariposa" duration="10" >
420.     <description>
421.
422.     </description>
423. <sensor>LINEAR_ACCELERATION</sensor>
424. <sensor>PROXIMITY</sensor>
425. <sensor>ACCELEROMETER</sensor>
426. <sensor>GYROSCOPE</sensor>
427. </scenario>
428.
429. <scenario id="43" name="Pesas (subir y bajar)" duration="10" >
```

```
430.         <description>
431.
432.         </description>
433.         <sensor>LINEAR_ACCELERATION</sensor>
434.         <sensor>PROXIMITY</sensor>
435.         <sensor>ACCELEROMETER</sensor>
436.         <sensor>GYROSCOPE</sensor>
437.     </scenario>
438.
439.     <scenario id="44" name="Pesas (subir y bajar)" duration="10" >
440.         <description>
441.
442.         </description>
443.         <sensor>LINEAR_ACCELERATION</sensor>
444.         <sensor>PROXIMITY</sensor>
445.         <sensor>ACCELEROMETER</sensor>
446.         <sensor>GYROSCOPE</sensor>
447.     </scenario>
448.
449.     <scenario id="45" name="Pesas frontales" duration="10" >
450.         <description>
451.
452.         </description>
453.         <sensor>LINEAR_ACCELERATION</sensor>
454.         <sensor>PROXIMITY</sensor>
455.         <sensor>ACCELEROMETER</sensor>
456.         <sensor>GYROSCOPE</sensor>
457.     </scenario>
458.
459.     <scenario id="46" name="Pesas frontales" duration="10" >
460.         <description>
461.
462.         </description>
463.         <sensor>LINEAR_ACCELERATION</sensor>
```

```
464.         <sensor>PROXIMITY</sensor>
465.         <sensor>ACCELEROMETER</sensor>
466.         <sensor>GYROSCOPE</sensor>
467.     </scenario>
468.
469.     <scenario id="47" name="Pesas laterales" duration="10" >
470.         <description>
471.
472.             </description>
473.         <sensor>LINEAR_ACCELERATION</sensor>
474.         <sensor>PROXIMITY</sensor>
475.         <sensor>ACCELEROMETER</sensor>
476.         <sensor>GYROSCOPE</sensor>
477.     </scenario>
478.
479.     <scenario id="48" name="Pesas laterales" duration="10" >
480.         <description>
481.
482.             </description>
483.         <sensor>LINEAR_ACCELERATION</sensor>
484.         <sensor>PROXIMITY</sensor>
485.         <sensor>ACCELEROMETER</sensor>
486.         <sensor>GYROSCOPE</sensor>
487.     </scenario>
488.
489. </group>
490.
491. </scenarios>
```

8.2 Formato de archivo de salida de SensorLogger

8.2.1 Ubicación

Los archivos se almacenan dentro del directorio SensorLogger en la memoria externa del dispositivo, en un subdirectorío cuyo nombre respeta el siguiente formato:

```
name@model [dd-MM-yyyy hh-mm-ss.SS]
```

donde:

- ✓ name: nombre del usuario.
- ✓ model: modelo del dispositivo.
- ✓ dd-MM-yyyy: fecha del día en el cual se iniciaron las pruebas.
- ✓ hh_mm_ss.SS: hora, minutos, segundos y milisegundos del comienzo de las pruebas.

Es decir que, suponiendo que el usuario Ian Curtis utilizó un Samsung S Advance (modelo GT-I9070) para realizar las pruebas el 7 de febrero de 2014 a las 13:22:29.549, la ruta absoluta desde la raíz del almacenamiento externo a la carpeta con los archivos de salida será:

```
/SensorLogger/Ian Curtis@GT-I9070 [07-02-2014 13-22-29.549]
```

Finalmente, es dentro de este directorio donde se encuentran efectivamente los archivos comprimidos con los datos de los sensores. Se genera un archivo por cada sensor interviniente en un escenario, cuyo nombre sigue una convención que permite identificar el escenario al que pertenece, el sensor que representa y el tipo del mismo:

```
id-scenario-name.deflate
```

donde:

- ✓ id: id del escenario.
- ✓ scenario: nombre del escenario.
- ✓ name: nombre del sensor.

De esa manera, para el usuario Ian Curtis del ejemplo anterior, dado un escenario "Next Image" de identificador 666, el path al archivo correspondiente a las muestras del sensor de luz GP2A Light sensor será:

.../Ian Curtis@GT-I9070.../666-Next Image-GP2A Light sensor.deflate

8.2.2 Contenido

Los archivos con los datos crudos de cada sensor en cada escenario se guardan comprimidos en formato DEFLATE³. Es necesario descomprimirlos utilizando el ejecutable **cbs** para poder visualizar la información deseada. Descomprimidos, estos archivos siguen el siguiente formato:

1	[nombre del usuario]@[modelo del dispositivo] [timestamp de comienzo]
2	[id del escenario] - [nombre del escenario]
3	[nombre del sensor] - [tipo del sensor]
4	Recording number: [número de grabación]
5	$t_1: v_1$
6	$t_2: v_2$
...	...
N	$t_{N-4}: v_{N-4}$

donde:

- ✓ t_k es el timestamp de v_k en formato dd-MM-yyyy HH-mm-ss.SS.
- ✓ v_k es el k -ésimo valor registrado por el sensor desde el comienzo de la grabación.
- ✓ número de grabación es la cantidad de veces que el usuario regrabó el escenario hasta que quedar conforme con los resultados.

A modo de ejemplo, el archivo de valores recolectados por un acelerómetro BMA222 en un Samsung S Advance (GT-I9070) actuando en el escenario 666, "Next Image", habiendo no quedado conforme con la primera toma, se vería:

```
Ian Curtis@GT-I9070 [07-02-2014 13-22-29.549]
666 - Next Image
BMA222 3-axis Accelerometer - TYPE_ACCELEROMETER
Recording number: 1
```

³ <http://en.wikipedia.org/wiki/DEFLATE>

```
07-02-2014 13-24-04.671: -3.37,9.19,7.35
07-02-2014 13-24-04.846: -2.60,8.27,6.43
...
07-02-2014 13-24-14.448: 3.21,5.21,8.89
```

8.3 Repositorio

FileUtils

Descripción

Proyecto que contiene una única clase, **FileUtils**, con un único método **static**, **createNewFileRecursive**, que recibe el path de un archivo a crearse para construir toda la jerarquía de directorios necesaria, incluyendo el archivo mismo.

Aplicación

Se incluye como dependencia en el proyecto **cbs** para crear archivos de output de manera *out-of-the-box*.

cbs

Descripción

CompressedBlockStreams, o **cbs**, es un proyecto con una única clase homónima que puede ser ejecutada desde línea de comandos y sirve para comprimir/descomprimir streams en formato DEFLATE⁴. Utiliza las clases *custom reader/writer* de streams comprimidos creadas por Philip Isenhour⁵.

Modo de uso

```
cbs.jar [ -d ] <source> <dest>
```

none: comprime datos en DEFLATE.

-d, --decompress: descomprime datos en DEFLATE.

Aplicación

Se incluye en el trabajo como herramienta de compresión/descompresión para los datos brutos obtenidos de las pruebas con los escenarios.

Además, se incluye como dependencia en el proyecto **SensorLogger** para comprimir

⁴ <http://en.wikipedia.org/wiki/DEFLATE>

⁵ Copyright 2005 - Philip Isenhour - <http://javatechniques.com>

el output de los sensores, utilizando directamente las clases de Isenhour.

scenarioloader

Descripción

Modelo de datos de escenarios de pruebas utilizado para la de-serialización del archivo XML de escenarios. Utiliza el framework Simple⁶ para ese propósito. Además, se incluye en el proyecto un archivo XML con escenarios de prueba y una clase

ScenarioLoader con un método **loadSampleScenarios** que levanta los escenarios en dicho archivo, para fines de *testing*.

Aplicación

Se incluye como dependencia en el proyecto **SensorLogger** como modelo de datos.

⁶ <http://simple.sourceforge.net/>