UNIVERSIDAD POLITÉCNICA DE VALENCIA.

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

SISTEMA DE RECONOCIMIENTO GESTUAL PARA UNA UNIDAD DE MEDICIÓN INERCIAL

PROYECTO FINAL DE CARRERA

Presentado por:

Gustavo Vidal Moreno

Dirigido por:

D. Carlos Monserrat Aranda

Valencia, 2010

Sistema de Reconocimiento Gestual para una Unidad de Medición Inercial.

Autor: Gustavo Vidal Moreno

Director: Carlos Monserrat Aranda

TABLA DE CONTENIDO

TABLA DE ILUSTRACIONES

RESUMEN

El Reconocimiento Gestual es el proceso de registrar un gesto del mundo real, de analizar los parámetros capturados y de dar como resultado qué gesto humano se ha realizado.

El Sistema de Captura de Movimiento es la solución electrónica o mecánica que se encarga de transformar el movimiento puramente físico en información de tipo digital.

Hoy en día, los Sistemas de Reconocimiento Gestual se encuentran en el punto de mira de la mayoría de compañías importantes relacionadas con el software de alguna manera. Nos encontramos en una batalla comercial por ver qué sistema de reconocimiento o dispositivo de captura de movimiento es más usable, intuitivo, barato… o dejando de lado las características del propio dispositivo y del sistema, cuál tendrá más éxito a nivel comercial.

El primer paso lo dieron las empresas relacionadas con los videojuegos, buscando mejorar sus animaciones gráficas, añadieron los movimientos registrados directamente de un cuerpo humano a la animación. Luego se buscó dar un cambio radical a la interfaz humana de las videoconsolas y se añadieron mandos inalámbricos con sensores inerciales.

Actualmente, todos los fabricantes de videoconsolas se han subido al carro de las nuevas interfaces que relacionan al Ser Humano con las máquinas de una manera más natural. Día a día, se ven más alejados los antiguos sistemas de interacción: tarjetas perforadas, ratón, teclado, gamepads, botones, joystick… los cuales han perdido su papel dominante en ciertos contextos.

El proyecto se centra en el desarrollo de un Sistema de Reconocimiento de Gestos, utilizando una solución empresarial de Captura de Movimientos. En este proyecto no se buscará mejorar, investigar o analizar el funcionamiento de la electrónica del dispositivo sino simplemente se usará como medio para obtener los valores, nada más.

Cabe destacar que más allá que buscar un desarrollo dedicado puramente al entretenimiento, al software que nos ocupa se le buscan aplicaciones relacionadas con la medicina y el ser humano, en este caso su primera aplicación será en la rehabilitación de personas con problemas de movilidad.

Para su desarrollo se ha usado una plataforma tipo PC, una IMU de captura de movimiento; como software se han usado el Visual Studio 2008, el 3ds Max para las animaciones y el 3d GameStudio para el desarrollo de la interfaz gráfica.

El resultado es, pues, una librería con las capacidades de reconocer patrones predefinidos de movimiento y, además, con capacidades para la gestión de su memoria de gestos.

ESTRUCTURA DEL DOCUMENTO

El documento que se presenta se divide en diferentes capítulos, y como se puede observar, cada uno de ellos corresponde con una fase de desarrollo de software tradicional. A continuación se expone a que se ocupa cada parte de la memoria:

- El primer capítulo habla del ámbito del proyecto. Intentando dar una pincelada muy general del proyecto, marcando desde un primer momento el objetivo principal así como el contexto en que se realizó el proyecto. Para finalizar el apartado se incluye un pequeño estado del arte, que intenta exponer, de forma sencilla, cómo se encuentra el mundo empresarial en relación con los Sistema de Reconocimiento Gestual y los dispositivos de captura de movimiento.
- El segundo capítulo se centra en la especificación de requisitos, detectando los requisitos que debe tener el software. Por otro lado, se explican a fondo todos los objetivos del sistema, además de comentar todos los tipos de restricciones que ha tenido que respetar el desarrollo.
- El capítulo tercero se centra en explicar de forma genérica los aspectos principales del software, utilizando modelos UML para representarlos de forma gráfica y visual. También, se exponen aspectos relacionados con la funcionalidad, la estructura y el comportamiento interno del software.
- El cuarto capítulo muestra la parte perteneciente al diseño. Así pues, se mostrará cómo se ha puesto solución a los requisitos. Básicamente, se explica la estructura seleccionada para desarrollar el sistema, cómo se ha definido cada capa y qué componentes se han usado para poner solución.
- En el quinto capítulo en orden cronológico, se focaliza la parte de implementación, donde se expone tanto de qué se ha implementado como de los problemas que se tuvieron y las soluciones que se aplicaron.
- Finalmente se comentan las conclusiones y trabajos futuros del proyecto, intentando reflexionar sobre todo lo que ha conllevado el desarrollo del software.

Al final del documento se presentan algunos anexos con información relevante de ser añadida y mostrada en la memoria.

DEFINICIONES, ACRÓNIMOS Y ABREVIACIONES

PERTENECIENTES AL DOCUMENTO

ACRÓNIMOS

SRG: Sistema de Reconocimiento Gestual.

IMU: Unidad de Medición Inercial (Inertial Measurement Unit). Mide los 3 ejes del campo magnético, de las aceleraciones y del giro. (Technaid SL).

PFC: Es, en ciertos países europeos, como España, un proyecto exigido como condición para obtener finalmente la titulación en ciertas carreras académicas (también llamado trabajo fin de carrera).

ERS: Especificación de Requisitos Software.

RV: Realidad Virtual.

SDK: Kit de Desarrollo Sofware (en inglés Software Development Kit). Es generalmente un conjunto de herramientas de desarrollo que le permite a un programador crear aplicaciones para un sistema concreto.

DEFINICIONES

Acelerómetro: Se emplean para medir vibraciones y oscilaciones en muchas máquinas e instalaciones, así como para el desarrollo de productos, como componentes o herramientas. La medición proporciona los siguientes parámetros: aceleración de la vibración, velocidad de vibración y variación de vibración [IBER].

Giroscopio: Dispositivo mecánico formado esencialmente por un cuerpo con simetría de rotación que gira alrededor de su eje de simetría.

Magnetómetro: sirven para detectar las cargas de su entorno personal y profesional: transformadores, postes de transmisión, líneas de corriente, aparatos eléctricos… Se usan para la orientación [IBER].

PERTENECIENTES AL SOFTWARE

DEFINICIONES

Punto: Valores obtenidos de los sensores en cierto instante temporal. Está formado por los valores recibidos de los acelerómetros, de los giroscopios y del magnetómetro.

Movimiento: Secuencia de puntos registrados. Entre una aceleración y desaceleración que ha sobrepasado los umbrales.

Unidad de Movimiento: Movimiento analizado y reducido a los valores interesantes para el sistema.

Gesto: Conjunto de Unidades de movimiento, que puede poseer nombre y número si se encuentra registrado con anterioridad.

CAPÍTULO 1: ÁMBITO DEL PROYECTO

INTRODUCCIÓN

Nos encontramos en una época de grandes cambios, y la industria del software y del videojuego no se ha mantenido al margen. Hoy en día, las grandes compañías del sector del entretenimiento digital, están buscando nuevas maneras de relacionar el ser humano con la máquina más allá del teclado, del ratón o del común mando de videoconsola.

El proyecto se ha centrado en obtener un software capaz de recoger las gestualidades del ser humano, para luego ser utilizadas según sea la necesidad. Sin necesidad de teclas o botones, buscando la interacción con las persona de una forma más natural

En el primer capítulo se describe el objetivo que persigue el proyecto, el análisis de la problemática que da lugar a su desarrollo y, para finalizar, el estado del arte al inicio del desarrollo.

OBJETIVO

El objetivo de éste proyecto ha sido el desarrollo de una librería que es capaz de reconocer gestos realizados por un ser humano, normalmente realizados con la mano.

Los gestos son definidos a priori, es decir, el sistema tiene una colección de gestos hospedados en un sistema de persistencia. Adicionalmente, el programa facilita la gestión de su memoria persistente, con la idea de facilitar la inclusión y borrado de gestos, además de permitir la desconexión de ciertos gestos para que no sean comparados en ciertos momentos.

Para llevar a cabo la detección de la gestualidad del usuario es necesario disponer de un sistema de captura del movimiento (Motion Tracking System), con la idea de obtener toda la gestualidad del cliente de forma *parametrizada* para su posterior análisis. En este caso se ha optado por una IMU (Unidad de Medición Inercial), la especificación y el porqué de su uso se explicarán más adelante [Ver Capítulo 2, sección Restricciones de Diseño].

El Software está estructurado como una biblioteca para permitir su inclusión en otros sistemas software de manera fácil y rápida. De hecho, en este mismo proyecto la biblioteca generada ha sido aplicada a un programa de carácter gráfico para su testeo y exposición.

CONTEXTO

El desarrollo del proyecto se ha realizado en *Labhuman* (Laboratorio de Tecnologías Centradas en el Humano). Laboratorio público, científico y tecnológico, que es parte del Instituto Interuniversitario de Investigación en Bioingeniería y Tecnología Orientada al Ser Humano (*I3BH*) de la Universidad Politécnica de Valencia (UPV).

En Labhuman se intenta unir las nuevas tecnologías y la actividad humana. Con la idea de conseguir una mejor vida futura para las personas.

Actividades verticales:	**Actividades horizontales:**	**Actividades transversales:**
– *Rehabilitación Virtual* – *Tecnologías persuasivas y asistivas* – *Psicología Asistida por Ordenador* – *Presencia en entornos virtuales* – *Ing. Emocional/Diseño Centrado en el Usuario* – *Intervención Médicas Asistida por Ordenador* – *Procesamiento de imagen médica* – *Biomecánica/modelo deformables*	– *Gráficos 3D/ realidad virtual* – *Realidad Aumentada* – *TIC móviles* – *Programación Web* – *Modelos deformables* – *Interfaces gestuales y 3D* – *Modelado 3D/animación* – *Contenidos 2D dinámicos* – *Procesamiento de imagen*	– *Bienestar y calidad de vida* – *Gerontecnología* – *Cirugía* – *Salud Mental* – *Odontología* – *Diseño Centrado Usuario* – *Entrenamiento y Educación* – *Entretenimiento* – *Simulación*

El PFC forma parte del proyecto Zootevi, el cual pretende desarrollar sistemas de RV que se puedan utilizar como alternativa a las sesiones de terapia con animales, en concreto las que se realizan con leones marinos. Para ello se implementará una aplicación de realidad virtual cuyo objetivo será dar órdenes a un león marino con la mano, a las cuales él responderá con animaciones definidas a priori.

El sistema contendrá dos de las principales actividades del Laboratorio: como es la Realidad Virtual y su aplicación a un ámbito humano, cómo es el de la rehabilitación de personas con ciertos problemas de movilidad.

Una vez desarrollado, la biblioteca podrá ser usada en multitud de aplicaciones software. Siempre y cuando, el movimiento del ser humano sea el medio de comunicación con la máquina. Algunas de las posibles aplicaciones directas será en contextos como:

- Juegos: Al estilo de las consolas de última generación.
- Software de exposiciones: Para interactuar con el sistema de presentación de diapositivas,…
- Programas de rehabilitación física y mental: Como es su primera aplicación.
- Aplicado en dispositivos inalámbricos: Para añadir funcionalidades más complejas. Como podría ser en mandos a distancia de televisores, de reproductores de música,…

ESTADO DEL ARTE

INTRODUCCIÓN

En la actualidad el reconocimiento gestual es uno de los retos tecnológicos a los que se enfrenta la informática, su objetivo es interpretar la gestualidad humana vía algoritmos matemáticos. Los gestos pueden ser originados por algún movimiento o estado del cuerpo, comúnmente originados con la mano. La función básica de todo reconocimiento gestual es saber qué gesto se ha realizado.

Todo software de Reconocimiento Gestual dispone de un sistema encargado de suministrar y recoger información de movimiento generada por un usuario, éste tipo de sistemas suelen ser conocidos como Sistemas de Captura de Movimiento. Por captura de movimiento (Motion Tracking) entendemos el proceso de grabar el movimiento con algún dispositivo y trasladarlo a formato digital.

Esta sección se dividirá en tres bloques: Visión General, los Sistemas de Reconocimiento Gestual y los dispositivos de captura de movimiento.

VISIÓN GENERAL

Como se ha comentado anteriormente todo Sistema de Reconocimiento Gestual dispone de uno o varios dispositivos de captura de movimiento. Hay multitud de sistemas aplicados a este ámbito, algunos se centran en la movilidad de alguna parte del cuerpo (p.e. la movilidad facial o la movilidad de la mano), en cambio, otros optan por registrar todo el cuerpo (muy utilizados para las animaciones en películas y videojuegos).

Para llevar a cabo la captura de movimiento existen dos grandes tipos de tecnologías, las ópticas y las no ópticas.

Están las basadas en la visión u ópticas, centradas en la captura de imágenes. Estos sistemas, a través del análisis de la imagen, son capaces de detectar que gesto se ha realizado. Para llevar a cabo la captura del movimiento los sistemas ópticos usan dispositivos de entrada específicos, de entre los que cabe destacar:

- ZCam: Cámara que captura imágenes en 3 dimensiones. Utiliza los infrarrojos para capturar la profundidad.
- Stereo cameras: Cámaras que cuentan con dos objetivos sincronizados que capturan la misma imagen, pero desde distinto ángulo.
- Una única cámara. De tipo normal, como por ejemplo una webcam o una camera de video familiar.

Estos dispositivos de captura de imagen son aplicados en distintas tecnologías de análisis de imagen, como las que siguen a continuación:

- Marcas pasivas: Son marcas que se disponen en el objeto a capturar, por ejemplo el brazo. Las marcas reflejan la luz emitida sobre el cuerpo, con el propósito de ser capturada. Luego, a través de cálculos matemáticos y de las imágenes capturadas con las marcas es posible proyectar un cuerpo virtual sobre la imagen.
- Marcas activas: A diferencia de las anteriores, estás emiten luz. Por ejemplo, algunas suelen estar compuestas de LEDS.
- Markerless: Son sistemas de captura de movimiento sin marcas. A través de complejos algoritmos que son capaces de detectar formas humanas, los cuales parten el cuerpo de las imágenes para poder obtener las capturas de las partes de interés.

Dejando de lado los ópticos, el otro gran bloque se compone de dispositivos no ópticos. Respecto a estos, las tres tecnologías más utilizadas de tipo no visual son:

- Sistemas inerciales: Compuestos de dispositivos electrónicos, como acelerómetros y giroscopios.
- Sistemas mecánicos: Estos sistemas son dispuestos directamente sobre el cuerpo humano junto a los ángulos de torsión de las extremidades permitiendo registrar el movimiento. Suelen ser como trajes al estilo de un exoesqueleto, estos trajes captan el movimiento del cuerpo humano haciendo que los sistemas mecánicos se muevan, permitiendo la captura del movimiento relativo de la persona.
- Sistemas mecánicos: Compuestos de dispositivos que capturan el magnetismo. Calculan la posición y la orientación por el flujo relativo magnético entre las tres bobinas situadas de forma ortogonal.

SISTEMAS DE RECONOCIMIENTO GESTUAL.

Pocas empresas ofrecen soluciones comerciales consistentes en sistemas de reconocimiento gestual. Y más difícil aún es encontrar el software de reconocimiento por separado. Los productos a la venta suelen ser sistemas cerrados, con un software de reconocimiento específico al sistema de captura empleado. De hecho, la mayoría de software que se describe en esta sección lleva incluido su capturador de movimiento.

A continuación se mostrarán algunas de las plataformas más interesantes que se encuentran a la venta en ambos tipos de captura:

Inerciales

- Animazoo: Fabricante de sistemas de captura de movimiento inercial, con base en Brighton, Reino Unido. Todos los sistemas Animazoo son ensamblados y probados por ingenieros calificados en Reino Unido garantizando la calidad y rendimiento.
    - IGS-Mini: Los sistemas de IGS Mini se configuran para medir movimientos de las articulaciones o extremidades y son ideales para los investigadores cuando necesitan analizar áreas específicas del

movimiento del cuerpo sin la necesidad de un sistema de captura de cuerpo completo.
    - IGS-190-M: Es la novena versión del primer sistema mundial de captura de movimiento giroscópico-inercial. Según su fabricante es fácil de usar, excepcionalmente preciso y portátil. El IGS-190-M es el sistema de captura de movimiento de mayor precisión en tiempo real. Incluye diecinueve diminutos sensores inerciales conectados a un traje de lycra flexible. Los giroscopios son móviles para una óptima colocación que nos garantice la exactitud de la medición de la articulación y no del músculo.
- Xsens: Proveedor global de productos de motion tracking en 3D basados en sensores MEMS inerciales en miniatura. Desde su creación en 2000, han desplegado con éxito varios miles de sensores de movimiento y soluciones de captura de movimiento, en áreas como la animación de personajes 3D, la rehabilitación y las ciencias del deporte,además de en estabilizadores de robots y cámaras. Entre sus clientes se incluye a Electronic Arts, Sony Pictures Imageworks, INAIL Prótesis Centro, Daimler, Saab Underwater Systems, Kongsberg Defence & Aerospace y muchas otras empresas e institutos de todo el mundo. Su departamento de investigación de Xsens ha desarrollado los algoritmos de fusión de datos provenientes de múltiples sensores, la combinación de sensores inerciales con las tecnologías como el GPS, el posicionamiento de RF y con modelos biomecánicos. Esta compañía dispone del **Xsens MVN - Inertial Motion Capture**. Solución de captura de movimiento consistente en sensores inerciales adheridos al cuerpo por un traje de lycra (también disponible en las correas). Xsens MVN le da la libertad de movimiento porque no utiliza MVN cámaras. Es un flexible y portátil sistema de captura de movimiento que se puede utilizar en interiores y al aire libre, no sólo en el set en el estudio, sino también fuera. La facilidad de uso y el corto tiempo de calibración le permite configurar el sistema en menos de 15 minutos.
- AiLive: Es una empresa dedicada al reconocimiento gestual y es colaborador en la creación del Wii Mote Plus. Uno de sus productos estrella es el LiveMove Pro, herramienta especialmente creada para desarrollar paquetes de reconocimiento gestual para juegos. Este software es usado para la creación de juegos para la Wii.

Visuales

- Handvu: Es una colección de software que implementa una interfaz, basada en la visión, para los gestos de la mano. Detecta la mano en una postura estándar, entonces, hace un seguimiento y reconoce posturas clave - todo en tiempo real y sin necesidades de calibraciones de cámara o de usuario. La salida es accesible a través de llamadas a las bibliotecas, mediante una infraestructura de cliente-servidor. El paquete de software consiste en una biblioteca principal y varias aplicaciones que muestran la captura de vídeo con OpenCV de highgui, DirectShow, y pronto el ARtoolkit. Dispone de la funcionalidad de realizar capturas de pantalla de objeto rastreado, lo que permite la recogida de datos para la investigación u otros fines.
- IGesture: Es un framework de reconocimiento de gestos basados en Java. Se centra en la extensibilidad y la reutilización de aplicaciones cruzadas,

proporcionando una solución integrada que incluye el iGesture recognition framework, además del componente de iGesture Tool para la creación de conjuntos de gestos específicos. La herramienta iGesture, además, permite la evaluación y optimización de los algoritmos de reconocimiento de gestos nuevos o existentes. El framework iGesture no se limita a un dispositivo de entrada específico. Además de la pantalla y la interacción tradicional basado en el ratón, iGesture proporciona la funcionalidad para el manejo de lápiz digital.

- Sofkinetic –IISU: Es una completa plataforma para el desarrollo y despliegue de aplicaciones de reconocimiento gestual. Es compatiple con cámaras de tipo Depth Sensing y permite a los desarrolladores construir interfaces naturales, envolventes, transparentes e intuitivas para los videojuegos, aplicaciones de PC, home media control, aplicaciones de marketing interactivo,…
- Kinect – Microsoft: Es un Sistema de reconocimiento gestual, desarrollado por Microsoft. Aún no está a la venta, fue presentado el 13 de junio del 2010. Sera compuesto de una cámara RGB, un sensor de profundidad y un micrófono. De la parte de software poco se conoce por ser un sistema cerrado.

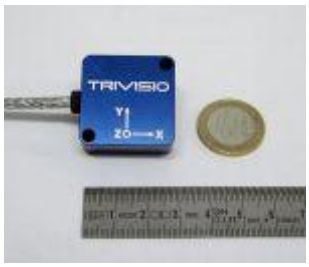
DISPOSITIVOS DE CAPTURA DE MOVIMIENTO

En ésta sección nos centramos en los dispositivos de captura de movimiento. Sólo mostramos los de tipo Inercial, ya que la utilización de una IMU fue una de las premisas principales del proyecto. No se entra en detalles, pues no es objetivo de este proyecto hacer un estado del arte del mundo de la IMUs, aunque no queda de más dejar constancia de la gran actividad del mundo empresarial en este aspecto.

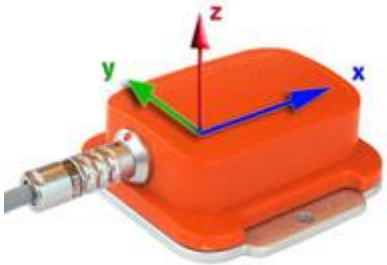
Algunos de los dispositivos que nos han parecido interesantes son:

- Ascencion Technology Corporation – TrackStar: Rastreador de movimiento magnético, con 16 sensores con 6 grados de libertad y capaz de rastrear a 240-420 actualizaciones por segundo. Apropiados para instrumentos de navegación 3d, no produce radiación de iones.

- TRIVISIO-Colibrí: Sistema con un bajo consumo. Lleva 3 ejes para medir aceleración, velocidad angular y campo magnético. Posee un sensor de temperatura que elimina la influencia de otros sensores. Se puede activar y desactivar desde cualquier sensor y cambiar la frecuencia de 10 a más de 100 Hz. Se suministra con API de Windows y Linux e implementa el filtro Kalman para la orientación del tracker. Su uso permite obtener datos de orientación en ángulos Euler ó quaternion. Colibrí está revestido de aluminio robusto de alta precisión.



- Xsens Technologies B.V.
  - MTi: IMU capaz de determinar la orientación, la aceleración, el giro y el campo magnético terrestre en 3 dimensiones.



  - Mti-G: Igual como la anterior pero incorpora un GPS.
- Intersense
  - Wireless InertiaCube 3: Es el sistema de referencia y orientación más pequeño del mundo (31.3.2 mm x 43.2 mm x 14.8 mm). Dispone de nueve sensores que aseguran una máxima precisión, sensibilidad y estabilidad, que abarca tracking los 360º del movimiento. Incluye el software MagCal para compensación de los campos magnéticos.
    Puede estar operando durante 8 horas con una sola batería +9 VDC

alcalina. Además, dispone de receptor Wireless que soporta hasta 4 sensores independientes InertiaCube3 con un alcance de hasta 30 metros. Éste receptor Wireless conecta directamente a un conector serial RS-232 ó USB port. El producto incorpora una librería para Windows Vista/XP/2000 y para Linux, además incluye la SDK para Windows y Linux. Las librerías permiten el control, la conectividad y configuración de los sensores, y la conectividad de red y emulación de joystick. También existe un versión no-wireless.

- InertiaCube 2: Otro sistema de tracking de Intersense. Abarca tracking en 360º a 180Hz ajustables a la necesidad del movimiento. Se pueden ajustar los filtros de salida y sensibilidad rotacional. A nivel de software incluye lo mismo que el anterior. Se puede encontrar con la interfaz: USB y Serial.

CAPÍTULO 2: ESPECIFICACIÓN DE REQUISITOS SOFTWARE

INTRODUCCIÓN

Todo desarrollo software empieza con el análisis y especificación de requisitos. En este capítulo se abordará este tema perteneciente al desarrollo del sistema.

Según el STD 610-1990 IEEE Standard Glossary of Software Engineering Terminology un requisito es:

1. Una condición o capacidad necesaria para que un usuario resuelva un problema o alcance un objetivo.
2. Una condición o capacidad que debe reunir o poseer un sistema o un componente de un sistema para satisfacer un contrato, un estándar, una especificación u otros documentos impuestos formalmente.
3. Una representación documentada de una condición o capacidad como en (1) o (2).

El propósito del presente capítulo es definir cuáles son los requerimientos que debe tener el Sistema de Reconocimiento Gestual para una IMU.
Esta especificación de requerimientos está destinada a ser leída por los usuarios o cualquier sujeto que tenga interés en saber cuáles son las funcionalidades que cumple el sistema.

La especificación se compone de 3 partes:

1. La Introducción, se dedica a la introducir qué es la ERS y en qué consiste.
2. La Descripción General, describe el producto sin entrar en detalles.
3. El apartado de Requisitos específicos es el más extenso de la ERS puesto que contiene todos los requisitos software, aparte de ser el más importante ya que es el propósito por el que se lleva a cabo el capítulo.

ÁMBITO DE LA ERS

El producto que se describe a continuación desempeña el papel de librería encargada de analizar los datos provenientes de los sensores de una IMU, y sacar conclusiones del tipo: Qué gesto ha hecho, Cuándo, Cómo,..También permite gestionar los gestos incluidos en el sistema de persistencia, además de gestionar el registro de nuevos gestos.

El análisis deber ser en tiempo real y lo suficientemente rápido para no ocasionar desfases entre el usuario y la aplicación usuaria de la librería.

Podrá ser incluido en multitud de aplicaciones que requieran análisis de gestos. Un uso directo será la inclusión de la librería en el proyecto terapéutico Zootevi, dirigido a la rehabilitación de personas con problemas mentales y físicos.

DESCRIPCIÓN GENERAL DEL PRODUCTO

PERSPECTIVA DEL PRODUCTO.

El producto final es un software en forma de librería, el cual siempre formará parte de una aplicación mayor y más compleja. Su fin es añadir nuevas funcionalidades a otros programas informáticos. Por ejemplo: Para la validación del software se ha diseñado e implementado una interfaz gráfica.

Además, el sistema incluye un conjunto de librerías que le han añadido mayor capacidad y funcionalidad. Algunas librerías usadas han sido la de gestión de la IMU, la librería de lectura y escritura de los XML y la librería de gestión de hilos.

FUNCIONES DEL PRODUCTO.

Se desea obtener un producto que cumpla con las siguientes funciones:

1. Análisis en tiempo real: El reconocimiento se realizará en tiempo real, siendo necesidad indispensable que el desfase temporal entre el movimiento del usuario y el resultado del reconocimiento gestual sea mínimo.
2. Dos modos de ejecución:
    - 2.1. Modo Reconocimiento Gestual: El cual es función capital de la librería. Este modo deberá llevar a cabo las siguientes tareas:
        - 2.1.1. Detección de movimientos: El sistema estará preparando esperando que el usuario gesticule. Pasando a registrar el gesto ante el mínimo atisbo de movimiento.
        - 2.1.2. Comparación de los patrones del Gesto registrado con los de los gestos guardados en el sistema de persistencia. Así el software es capaz de distinguir qué movimiento se ha hecho y devolver como resultado el tipo de movimiento. Si por el contrario el análisis es negativo lo notifica igualmente.
        - 2.1.3. Desactivación de Gestos para el análisis: Según qué aplicaciones puede ser de gran utilidad la desactivación de ciertos gestos en el análisis, ya puede ser por tema de niveles de dificultad en un juego o por facilitar el análisis al sistema, evitando confusiones y reduciendo, de éste modo, la complejidad en la detección del gesto.

2.2. Modo Gestión Gestos: Este modo es el encargado de gestionar la capa de persistencia, permitiéndole:
2.2.1. Insertar Gestos nuevos: Para llevar a cabo el registro de un nuevo Gesto el sistema se deberán satisfacer ciertas funciones como son
2.2.1.1. Capturar patrones de gestos: Permitiendo capturar cuantos se deseen, con la idea de minimizar el error humano en el gesto.
2.2.1.2. Interpolar varias capturas para obtener un Gesto como resultado.
2.2.1.3. Incluirlos en un sistema de persistencia: Todo gesto resultado de la interpolación podrá ser insertado en el sistema de persistencia, para luego ser comparado junto los otros gestos guardados en el modo de reconocimiento. El gesto será guardado junto a un nombre para poder distinguirlos de otros.
2.2.1.4. Borrado de gestos: Una de las premisas de más sentido común es dar la funcionalidad de borrado. Pues los gestos insertados tal vez sean erróneos, o ya no sean de interés en el sistema.
2.3. Los modos podrán ser detenidos e Iniciados.

3. Dar acceso a los valores de los sensores en tiempo real: La librería permite en todo momento, sin depender del modo en que se encuentre, acceder a los valores de los sensores. Así pues se permite el acceso a los valores de los acelerómetros, de los giroscopios y del magnetómetro. Los valores de la rotación también son devueltos en formato quaternion.
4. Fácil adaptación a otros dispositivos inerciales: El software es usado con una IMU específica pero podrá ser sustituida por otra.
5. Cambios de configuración durante la ejecución: El programa permite cambiar ciertos parámetros de los análisis internos que lleva a cabo el sistema. Con la idea de complicar más o menos las captaciones de movimientos.
6. Interfaz grafica amigable para mostrar la funcionalidad del producto. Para el desarrollo se usó un interfaz tipo consola para poder manejar con más comodidad el software, aunque el proyecto será presentado con una aplicación de realidad virtual amigable.

CARACTERÍSTICAS DEL USUARIO.

Desde el punto de vista de la interacción con el software se podrían hacer dos tipos de análisis de usuario. Interacción directa con las funcionalidades y la interacción con la IMU.

En un principio ningún usuario interactuará directamente con el software. Como se ha comentado el programa tendrá formato de biblioteca. Así pues, no existirá ningún usuario físico que ejecute directamente sus funcionalidades. Será un sistema global el que lo manejará a su antojo. Desde este punto de vista el usuario será otro software.

La segunda opción es la interacción con la IMU. Con esto se quiere reflejar y comentar que usuarios podrán usar el sistema para que se les reconozca los gestos. Así pues, el software estará dispuesto para reconocer cualquier gesto, ya sea para el brazo o cualquier otra parte del cuerpo. El usuario final será de tipo general, lo cual exige que posea una movilidad mínima de la extremidad a rastrear. Los únicos conocimientos que se deberán exigir son el conocimiento de los gestos que reconoce el sistema en ese instante.

RESTRICCIONES GENERALES.

El software tiene las siguientes restricciones:

1. En primer lugar, la ejecución ha de ser en tiempo real. Por tanto, el software debe trabajar en tiempo real y es necesario el análisis de los sensores en cada instante en particular, tratando de dar una respuesta lo más rápidamente posible y siempre en un tiempo prudencial. Con lo que se evita un excesivo desfase entre la ejecución del movimiento por parte de un usuario y el resultado del escaneo final.
2. En segundo lugar se ha optimizado el uso de los recursos del sistema por parte del producto. Es probable que el software deba ser ejecutado en distintos tipos de maquinas de muy diversa índole. La premisa es que el cálculo sea mínimo, evitando que se esté ejecutando partes del código que no son necesarias en cada instante temporal. Hay que tener en cuenta que el producto será parte de un todo, es decir, el Sistema de Reconocimiento siempre tendrá por encima alguna aplicación que haga uso de la librería. Será necesario tener libre de uso la CPU y no tener un uso de RAM desproporcionado ya que, seguramente las aplicaciones que usen la librería serán de carácter gráfico, las cuales requieren un uso alto de los recursos del sistema.

SUPUESTOS Y DEPENDENCIAS.

El software siempre será usado en sistemas Microsoft Windows. Se tiene que tener en cuenta que la mayoría de motores gráficos son para este sistema operativo. Esto se debe a razones de tipo económico, ya que es el sistema operativo más extendido en las computadoras personales, lo cual provoca no ser abordable económicamente un motor gráfico para otros sistemas operativos de menos alcance.

Siguiendo con lo anterior, la adaptabilidad de la librería a otros sistemas operativos no ha sido considerada como un requisito, aunque se puede llevar cabo modificando algunas directivas e instrucciones usadas en el código fuente.

Cómo se comentará en la sección de restricciones de este mismo capítulo, el desarrollo del software se ha realizado junto al dispositivo Colibrí, la IMU de Trivisio. Se debe tener en cuenta que cada dispositivo inercial posee su propia SDK, con toda la funcionalidad que ésta permite. A pesar de eso se ha tenido en cuenta el requisito de la adaptabilidad a otras IMUs. Para conseguirlo se aisló y localizó al máximo la interacción con la IMU.

En cualquier caso la IMU sustitutoria siempre ha de ser capaz de obtener tres tipos de valores, necesarios para el reconocimiento gestual: las aceraciones, los giros y el magnetismo en los tres ejes, y todo en cada instante temporal. Para cumplir con la premisa anterior debe poseer acelerómetro, giroscopio y magnetómetro. Esto se debe a que cada gesto se compone de estos tres parámetros en mayor o menor medida.

REQUISITOS ESPECÍFICOS.

En esta sección se procederá a describir todos los requisitos del software con mayor nivel de detalle (para la especificación de métodos y funciones ver en Anexos, Manual de Referencia).

REQUISITOS FUNCIONALES

Inicialización del sistema en Modo Reconocimiento Gestual

El sistema se iniciará en modo de reconocimiento gestual. De este modo sus funciones se enfocan a analizar los sensores y el reconocimiento de gestos. Además, esta funcionalidad se encarga de cargar el sistema, que incluye la carga en memoria de gestos predefinidos y su configuración.

Actualización del gesto actual

Actualiza el gesto actual del sistema para así, poder obtener todos sus valores. El gesto actual se obtendrá de una cola de gestos interna y siempre se obtendrá el que lleve más tiempo de espera. Si el tiempo de vejez sobrepasa el tiempo de espera éste es descartado, y se pasa a obtener el siguiente.

Obtener número del gesto actual

Acción de tomar el número del gesto ultimo realizado. Si se quiere obtener el último, ésta instrucción debe ser precedida del *Update* explicado anteriormente.

Obtener nombre del gesto actual

Acción encargada de tomar el nombre del gesto último realizado. Si se quiere obtener el último, ésta instrucción debe ser precedida del *Update* explicado anteriormente.

Obtener fuerza del gesto actual

Acción de tomar la fuerza media (media de las aceleraciones que lo componen) del gesto ultimo realizado. Si se quiere obtener el último, ésta instrucción debe ser precedida del *Update* explicado anteriormente.

Desactivación de un Gesto

Esta funcionalidad se encarga de no incluir en el reconocimiento gestual aquellos gestos guardados que no sean de interés.

Inicialización del sistema en Modo Gestión de Gestos

Inicia el sistema en modo Gestión de los gestos. De este modo sus funciones son enfocadas a gestionar los gestos, capturar nuevos gestos, interpolar los capturados, guardar los capturados o borrar algún gesto guardado.

Además se encarga de cargar el sistema, los gestos predefinidos y su configuración.

Captura de un Gesto

Cuando se activa esta funcionalidad el sistema se pone a la espera de recibir un gesto. El usuario debe proceder a hacer los movimientos que quiera que sean capturados como gesto. El gesto capturado es guardado en una lista de capturados. A la espera de ser interpolado, y guardado.

Interpolación de los gestos capturados.

Con su ejecución el sistema interpola todos los gestos que han sido capturados, obteniendo como resultado un gesto.

La interpolación consiste en construir un gesto medio entre todos los gestos capturados con anterioridad. Cada unidad de gesto debe aparecer, al menos, en un 80% de los gestos capturados. Es por eso que se deben capturar diversas veces el mismo gesto para que el posible error de gestualidad hecho por el usuario pueda ser obviado.

Una vez hecha la interpolación esta funcionalidad también se encarga de comparar el resultado interpolado con los gestos guardados en el sistema de persistencia, para no permitir gestos repetidos.

Guarda el Gesto capturado

El sistema guarda el gesto capturado con un nombre. La función llama a interpolar los gestos (la función anterior), si todo va bien deposita el gesto interpolado en el sistema de persistencia asignándole el nombre de entrada.

Reiniciar la captura de gestos

El sistema borra todos los gestos capturados con anterioridad, para proceder a iniciar la captura de gestos.

Borrado de Gesto

Borra del sistema de persistencia el Gesto dado.

Obtener la lista de gestos guardados

Se obtiene la lista de gestos guardados en el sistema de persistencia.

Restaurar el XML de gestos

Funcionalidad encargada de dejar el sistema con los gestos predefinidos en el sistema. Esta funcionalidad evita que debido a un error del usuario se borren gestos que luego se necesiten.

Actualiza el Punto actual

El sistema actualiza el punto actual, como se explicó en el apartado definiciones. Un punto está formado por los valores de todos los sensores en un instante concreto. Ésta instrucción siempre debe ser ejecutada al menos una vez antes de obtener todos los valores actuales. Es decir, si se desean obtener todos los valores de los sensores debe ser ejecutado, al menos, una vez.

Obtener la aceleración en el eje X del Punto actual

El sistema obtiene la aceleración en el eje X del punto actual.

Obtener la aceleración en el eje Y del Punto actual

El sistema obtiene la aceleración en el eje Y del punto actual.

Obtener la aceleración en el eje Z del Punto actual

El sistema obtiene la aceleración en el eje Z del punto actual.

Obtener el giro en el eje X del Punto actual

El sistema obtiene el giro en el eje X del punto actual.

Obtener el giro en el eje Y del Punto actual

El sistema obtiene el giro en el eje Y del punto actual.

Obtener el giro en el eje Z del Punto actual

El sistema obtiene el giro en el eje Z del punto actual.

Obtener el magnetismo en el eje X del Punto actual

El sistema obtiene el magnetismo en el eje X del punto actual.

Obtener el magnetismo en el eje Y del Punto actual

El sistema obtiene el magnetismo en el eje Y del punto actual.

Obtener el magnetismo en el eje Z del Punto actual

El sistema obtiene el magnetismo en el eje Z del punto actual.

Obtener el quaternion X del Punto actual

El sistema obtiene el quaternion X del punto actual.

Obtener el quaternion Y del Punto actual

El sistema obtiene el quaternion Y del punto actual.

Obtener el quaternion Z del Punto actual

El sistema obtiene el quaternion Z del punto actual.

Obtener el quaternion W del Punto actual

El sistema obtiene el quaternion W del punto actual.

Finalizar el sistema

Independientemente del modo en que se haya iniciado el sistema, esta función se encargará de detener el software.

REQUISITOS DE RENDIMIENTO Y EFICIENCIA

CONSUMO DE RECURSOS

Todo desarrollo de software debe poner cierta atención en el uso que hace el producto final de los recursos de la máquina. Éste hecho se convierte en tema capital si estamos hablando de una librería, que formará parte de un sistema más grande y complejo.

No se puede saber a priori los recursos de que se dispondrán. El SRG podrá ser utilizado en distintos tipos de aplicaciones, desde aplicaciones gráficas de rehabilitación hasta en aplicaciones de control de un dispositivo de un televisor. Esto posee cierta problemática pues, las aplicaciones que usarán este software pueden hacer tanto un uso intensivo de CPU como ocupar el 90% de la memoria, o directamente ser muy livianas. Además, para el uso al que está destinado, la librería deberá funcionar tanto en un PC de condiciones normales como en uno con recursos más limitados, al estilo de un Netbook o un Mini-PC

En el desarrollo del sistema se ha optado por la solución de más sentido común: Realizar todas las operaciones, escaneados y funcionalidades con el mínimo esfuerzo computacional sin perder eficiencia y rendimiento.

Así un requisito esencial será el mínimo uso de CPU en cualquier estado del programa. Requisito cumplido, viendo que en un PC Intel Core 2 Duo a 2 GHz el sistema llega a picos del 2% de uso de CPU.

La misma política de uso se ha aplicado al consumo de memoria virtual. Llegando a un picos de 3 MB de uso. Éste tipo de consumo puede ser incrementado no coordinando bien el sistema, con lo que ciertas colas de objetos del software podrían llegar a crecer en demasía si no hubieran algunos controles.

El producto final ese tipo de problemáticas han sido solucionadas como se explica en esta memoria [Velocidad de cálculo, capítulo 5].

RENDIMIENTO

Otro requisito ineludible en el Sistema debe ser el tiempo respuesta y, cómo no, el análisis en tiempo real.

Se debe evitar a toda costa el posible desfase entre la gestualidad del usuario, el análisis de gestos y el resultado del análisis. Éste desfase podría llevar a incongruencias en la respuesta del sistema. Pues, el sistema podría estar devolviendo un valor que se ha realizando con anterioridad, mientras el usuario ya está pensando que tal vez el gesto realizado no ha sido captado, y puede llevarle a la idea de cambiar de gesto o repetirlo otra vez.

OTROS REQUISITOS

ADAPTABILIDAD A CAMBIOS DE IMU

Es posible el cambio de dispositivo de captación de movimiento, para ellos se ha minimizado y localizado el uso de la SDK de gestión que toda unidad posee.

INTERFAZ DE USO SIMPLE

Ya que la inclusión del sistema como librería en otros sistemas puede venir con restricciones de *tipado* de datos, se ha optado por utilizar funciones que devuelvan tipos de datos estándares como enteros o cadenas de texto.

RESTRICCIONES DE DISEÑO

LENGUAJES

C++

El lenguaje especificado para el desarrollo ha sido el C++. Ha sido elegido por múltiples aspectos entre los que cabe destacar:

- La rapidez: Al no ser un lenguaje interpretado, necesita de una compilación que genera un código objeto específico para cada máquina, provocando que el resultado sea un programa totalmente adaptado a las peculiaridades del sistema donde se encuentre.
- Eficiencia: Característica unida a la anterior.
- Librerías: Casi en todos los sistemas gráficos, posibilidad de ser adaptado para ser usado como librería externa en cualquier proyecto de realidad virtual.

XML

El Extensible Markup Language (siglas de XML) será el lenguaje elegido para la capa de persistencia. Se optó por él, por ser muy eficiente y fácil de acceder. Además de tener en cuenta que las interacciones entre la capa de negocio y la capa de persistencia son mínimas, con lo que nos llevó a pensar que la implementación de un sistema de memoria al estilo base de datos relacional no tenía sentido debida al poco volúmen de datos.

LITE-C de 3D GameStudio A7

Para la implementación de la interfaz gráfica en la capa de presentación se usó el 3d GameStudio A7, sistema para el desarrollo de aplicaciones gráficas en 3d. Se optó por usarlo, por ser de fácil manejo, por tener una curva de aprendizaje relativamente fácil y por ser uno de los más usados en el instituto de investigación.

EL COLIBRÍ COMO IMU

Una de las restricciones de diseño más importantes que se tuvieron que tener en cuenta fue la IMU. En la proposición del proyecto se decidió el uso de IMU y en concreto el Colibrí de Trivisio.

Se llegó a esta decisión por razones de tipo de diseño e implementación del software, además de tener en cuenta las características del Colibrí y su bajo coste económico.

Se tuvieron diferentes razones, entre las más destacable fueron:

- El reconocimiento de gestos a través de un sistema inercial es de desarrollo más corto: Ya que exige menor grado de aplicación de teoremas y algoritmos matemáticos, provocando que las fases de pre-implementación sean más cortas.

- Sensibilidad y velocidad de la IMU: Este tipo de sensores dan muy buena precisión del movimiento a unas buenas velocidades de actualización de datos.
- Exploración de nuevos campos: El instituto tiene en sus repositorios implementaciones de éste estilo para los dispositivos de manejo de la Nintendo Wii.
- El Colibrí de Trivisio. La elección de este IMU se debió a las excepcionales características a nivel tecnológico, a destacar:
    - Captación de giros, aceleraciones y orientación en los tres ejes.
    - Sensores de temperatura.
    - Buena tasa de refresco de los valores de los sensores.

Además de temer un coste económico relativamente bajo, lo que provoca que esta IMU tenga un porcentaje calidad-precio alto en el mercado actual.

CAPÍTULO 3: ANÁLISIS DEL SISTEMA

INTRODUCCIÓN

El análisis constituye una de las actividades primordiales en el desarrollo de cualquier proyecto de sistemas. En esta actividad es donde se han analizado todas las consideraciones técnicas del proyecto, así como la comprensión y solución del problema que se plantea.

Todo desarrollo software debe poseer una etapa de análisis, antes de la codificación. Se debe desarrollar con profundidad el análisis para evitar errores grabes de diseño difíciles de solucionar. No entender la funcionalidad o los requisitos esenciales que el cliente pide al producto lleva a errores de muy difícil solución.

El Análisis desarrollado en el presente proyecto se ha distribuido en tres apartados: Análisis funcional, Análisis estructural y el análisis de funcionamiento.

ANÁLISIS FUNCIONAL

Como se ha comentado con anterioridad, para el modelado se usa UML, el cual propone el uso de los Casos de Uso para capturar las funcionalidades del sistema. Cada caso se compone de uno o más escenarios que representan cómo interactúan los usuarios con el sistema. Normalmente se evita el uso de tecnicismos, pues a veces la creación de los casos se realiza junto al cliente, el cual no tiene por qué tener conocimientos de Ingeniería Informática. Básicamente, un Caso de Uso es una interacción cliente-software.

Los usuarios del sistema son llamados actores. Los actores siempre son entidades externas al sistema, los cuales suelen desempeñar un rol, como podría ser usuario, administrador… Estos deberán realizar ciertas acciones para poder conseguir sus objetivos, cada uno de los objetivos quedan representado como un Caso de Uso.

En el diagrama también queda representado el sistema o software, normalmente como un cuadrado o rectángulo que engloba todos los casos detectados.

Cada caso o funcionalidad detectada se representa con una elipse, con el nombre de la función en su interior. Las elipses pueden estar relacionadas con otras mediante flechas de distinto tipo y distinto significado [ESS03]:

1. Relación (asociación): Ésta relación representa una comunicación del Actor con una funcionalidad y denota que ambos están relacionados y que el actor debe interactuar con el sistema para que la funcionalidad pueda ser ejecutada.
2. Inclusión: Representa una relación de dependencia entre casos de uso. Denota la inclusión del comportamiento de un caso en otro. Se representada como una flecha etiquetada con un nombre <<include>>. En el tema que nos ocupa hemos utilizado este tipo de relación en algunos casos. Un ejemplo es el Cargar Sistema, el cual se encuentra incluido en el inicio de cada modo de ejecución [Ilustración 1 y 2].

En nuestro Sistema de reconocimiento solo se dispone de un Actor, el cual es denominado como Cliente. Se ha de tener en cuenta que el SRG terminará teniendo

forma de librería con lo que se debe tener en cuenta que ningún usuario físico interactúa directamente con él.

Para mayor comodidad y simplicidad en este proyecto se ha optado por clasificar los casos de uso en:

1. Casos de Uso del sistema en Modo Reconocimiento de Gestos.
2. Casos de Uso del sistema en Modo Gestión Gestos.
3. Casos de Uso del Obtener valores del Punto Actual.

CASOS DE USO RECONOCIMIENTO GESTUAL

Para empezar se ha optado por analizar los casos de uso relacionados con el Modo Reconocimiento de Gestos.

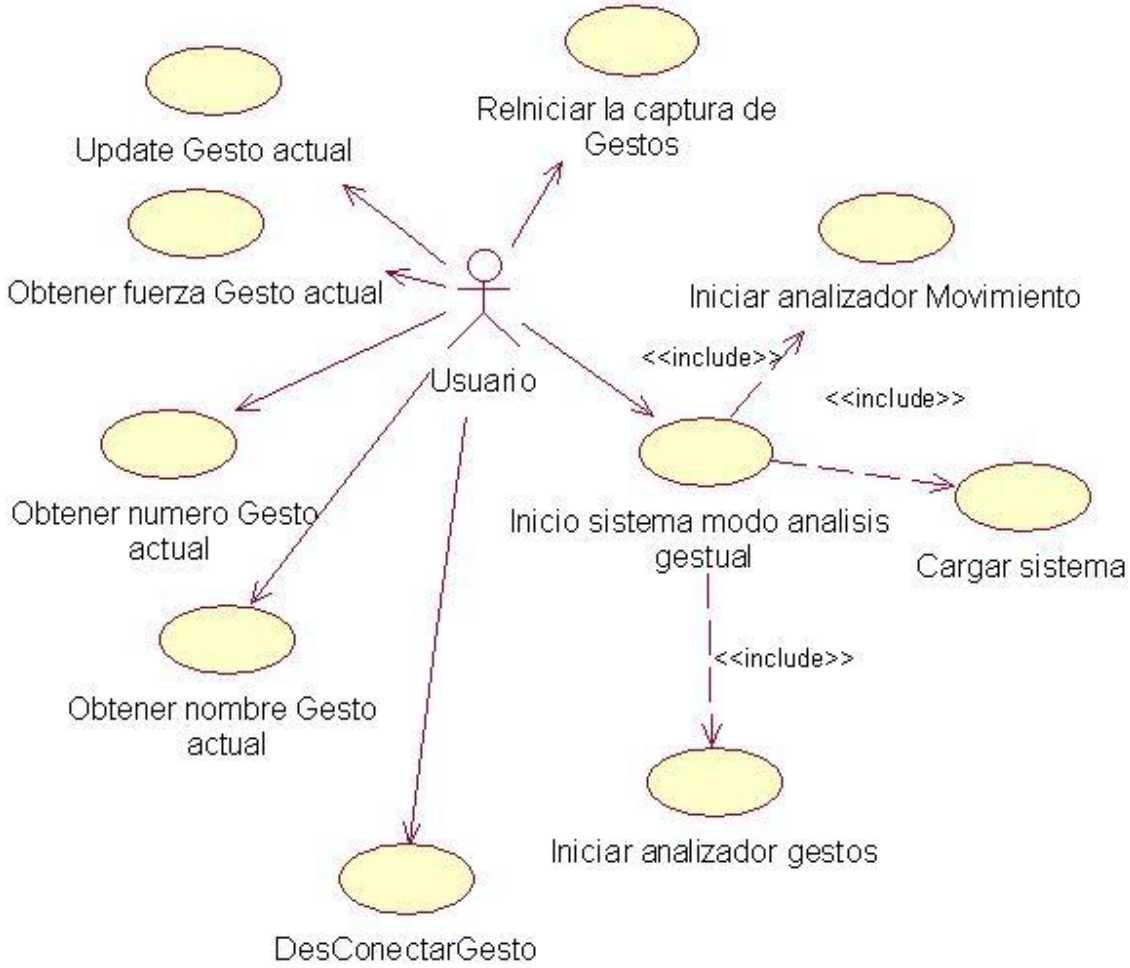


ILUSTRACIÓN 1: DIAGRAMA DE CASOS DE USO, CENTRADO EN EL MODO RECONOCIMIENTO GESTOS

En la imagen anterior [Ver *Ilustración 1*: Diagrama de Casos de Uso, centrado en el modo Reconocimiento Gestos] podemos examinar que este modo hace hincapié en el reconocimiento de Gestos. Básicamente consiste en iniciar el sistema y una vez iniciado, si los hay, obtener los valores del gesto actual. Finalmente se encuentra la funcionalidad de detener el sistema.

Se puede ver que se han añadido los casos de uso de inicio del Analizador de Movimiento y el inicio del Analizador de Gestos. Se decidió insertarlos como casos de uso pues, aunque el cliente no interactúe con ellos directamente, es esencial activar ambos para que el sistema funcione. Ésta división se podrá ver más fácilmente en los diagramas de clase [Ilustración 4]. Por la misma razón se reflejó la carga sistema de ese modo, ya que es una parte importante del sistema.

Finalmente la *Ilustración 1:* Diagrama de Casos de Uso, centrado en el modo Reconocimiento Gestosdispone de una funcionalidad llamada *Update Gesto Actual.* La cual se encarga de actualizar el Gesto actual del sistema. Siempre se debe ejecutar antes de proceder a ejecutar las funcionalidades de tipo Obtener.

CASOS DE USO DE GESTIÓN GESTOS

A continuación, siguiendo la idea de separación de los casos de uso, se presenta el Diagrama centrado en el modo Gestión Gestos [Ver Ilustración 2: Diagrama de Casos de Uso, centrado en la Gestión de Gestos.].

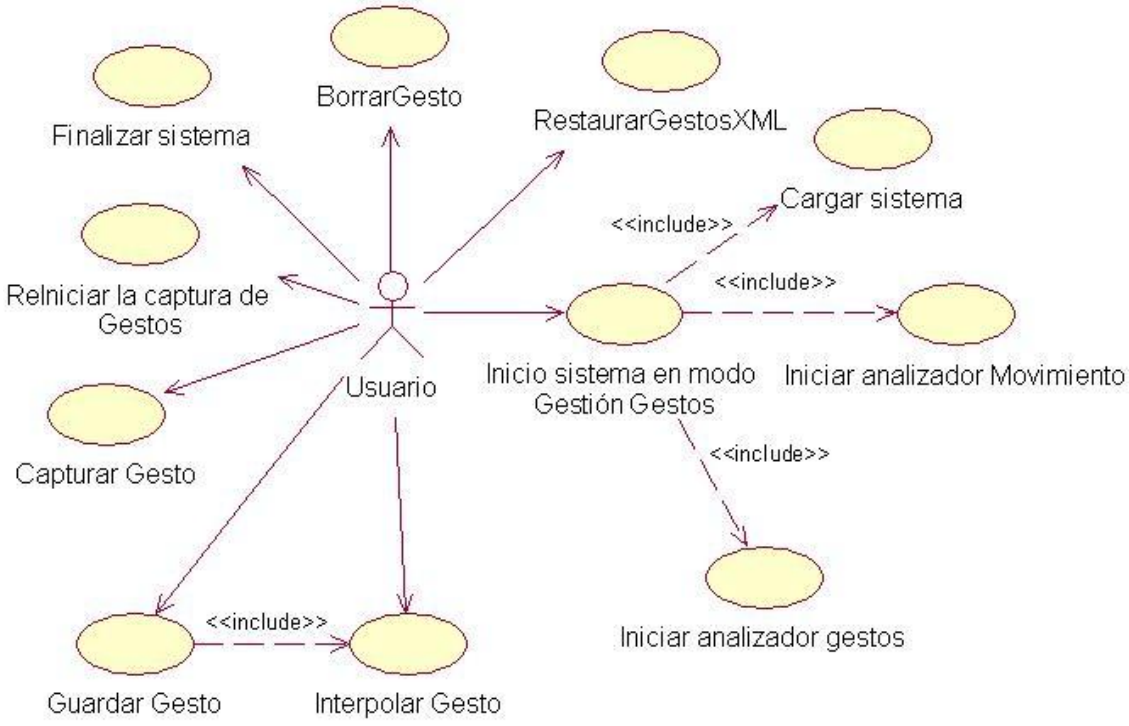


ILUSTRACIÓN 2: DIAGRAMA DE CASOS DE USO, CENTRADO EN LA GESTIÓN DE GESTOS.

Este modo, como indica su nombre, se dedica a la Gestión de los Gestos. Este modo de ejecución del sistema comparte con el anterior el inicio sistema, la carga sistema y la finalización de éste, pero varía en la funcionalidad que nos ofrece, pues permite: hacer capturas de gestos, interpolarlos para obtener un gesto medio (intentando huir de los errores provocados por el usuario final al hacer algún gesto), guardarlos en el sistema de persistencia, y Reiniciar la captura si se detecta que se ha provocado un error gestual demasiado grande. También se ha decidió reflejar el Inicio de ambos Analizadores y de la Carga del sistema.

CASOS DE USO DE OBTENER VALORES DEL PUNTO ACTUAL

Para finalizar con este apartado, nos centraremos en la funcionalidad *Obtener Valores del Punto Actual*.

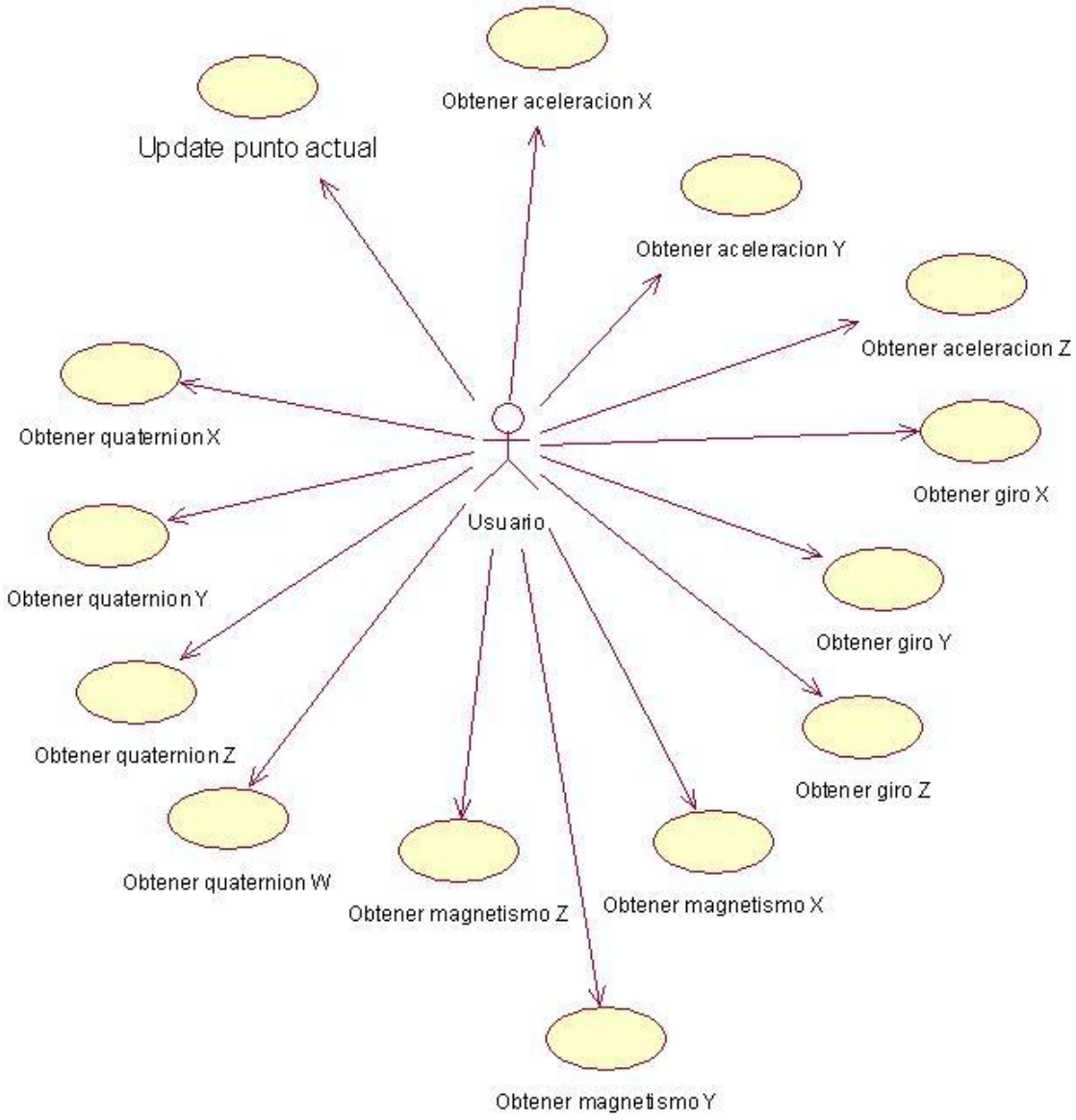


ILUSTRACIÓN 3: DIAGRAMA DE CASOS DE USO, CENTRADO EN OBTENER VALORES DEL PUNTO ACTUAL

En el diagrama anterior [Ver Ilustración 3: Diagrama de Casos de Uso, centrado en Obtener Valores del Punto Actual] expresa gráficamente las funcionalidades de que dispone el sistema para obtener los valores actuales de los sensores. Así se podría obtener la aceleración, el magnetismo o el giro de los tres ejes. Como con el diagrama de Análisis de gestos, se ha incluido una función Update Punto encargada de actualizar el punto actual del sistema. Si no se actualiza el punto antes de proceder a ejecutar todos los Obtener, el punto actual podría estar desfasado.

ANÁLISIS ESTRUCTURAL

En este apartado se procederá a analizar la estructura interna del sistema. Para ello se usa los diagramas de clases, estandarizados a través del UML. Según OMG, el consorcio que respalda el UML, se entiende por diagrama de clases aquel que muestra una colección elementos estáticos de modelado UML, como clases y tipos con su contenido y relaciones.

Se procederá a mostrar la estructura, dividida en cuatro bloques para simplificar, en cada caso, el objeto de análisis. Es decir, todos forman parte de la misma estructura y todo diagrama tendrá una clase central, llamada *Sistema*, que es la encargada de manejar el software. Así se mostraran los siguientes diagramas:

- Diagrama Completo.
- Sub-diagrama Analizador Movimiento.
- Sub-diagrama Analizador Gestos.
- Sub-diagrama carga sistema.

VISIÓN GENERAL - DIAGRAMA COMPLETO

Se procede a explicar el diagrama completo [Ver Ilustración 4: Diagrama de Clases completo]. En éste diagrama podemos ver la vista general del software. Ésta se puede dividir, básicamente, en 3 apartados que no coinciden exactamente con el paradigma de programación en 3 capas, pero nos darán una idea general del funcionamiento.

El software tiene una clase principal llamada *Sistema*. Ésta clase es la encargada de comunicar cualquier interfaz externa con la capa negocio. Por decirlo de alguna manera es la clase de manejo/entrada al sistema. Como se puede ver, ésta se asocia con las restantes clases, de manera directa o indirecta.

Sistema marca la funcionalidad del software. En ella se ve reflejada toda la funcionalidad explicada en apartados anteriores. Además, la clase se encarga de hacer como de "puente" o "canal" de información entre clases. Así, todo gesto nuevo y toda información de carga o descarga proveniente de las clases *Cargar Sistema* o *Guardar sistema,* circula por esta clase y es derivada a otras clases para que la gestionen.

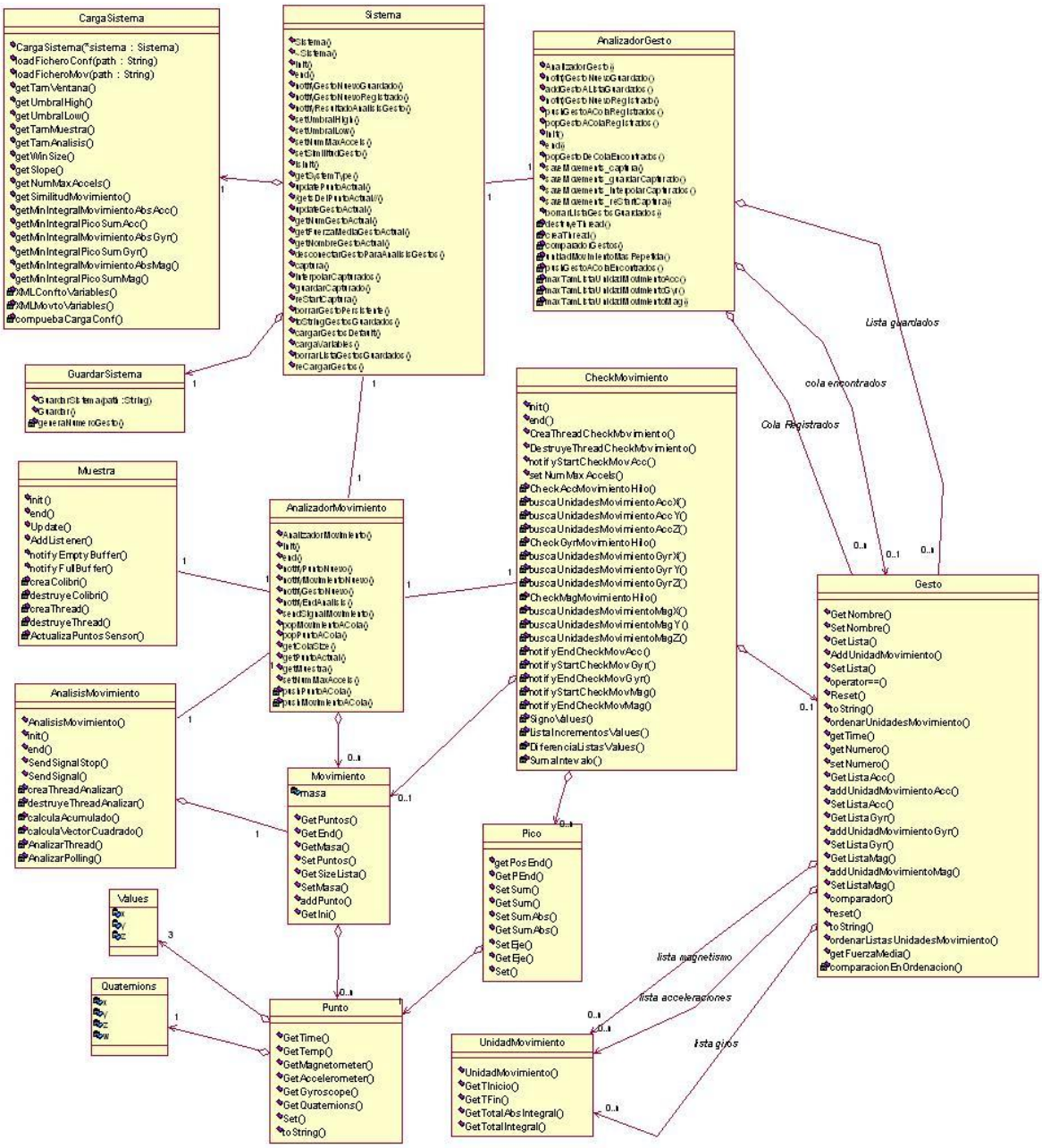


ILUSTRACIÓN 4: DIAGRAMA DE CLASES COMPLETO

Si se procede a un análisis más a fondo, se puede reflejar la gran importancia de dos clases: Analizador Gestos y Analizador Movimiento. Así pasamos a analizarlas por separado.

SUB-DIAGRAMA ANALIZADOR MOVIMIENTO

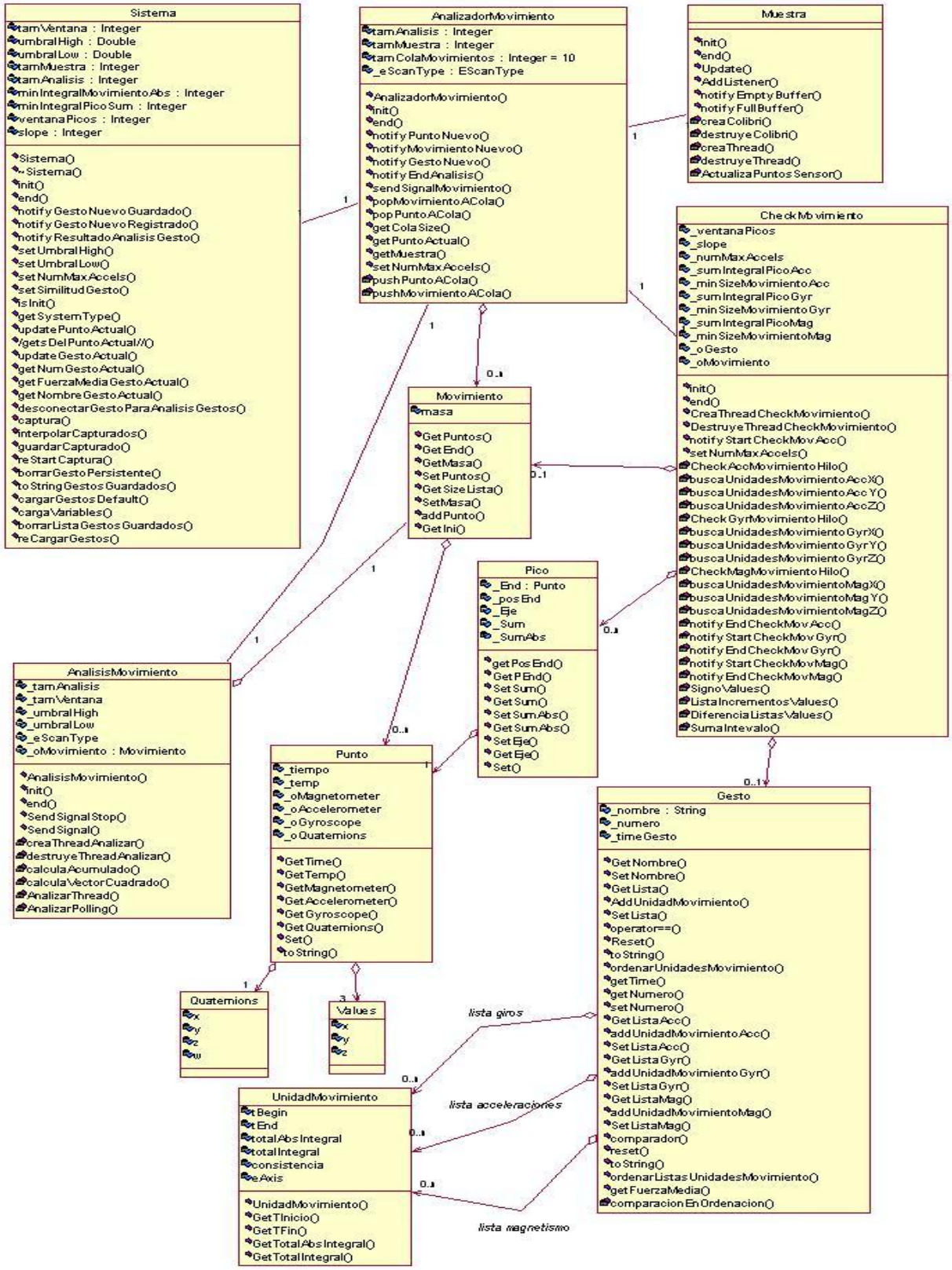


ILUSTRACIÓN 5: DIAGRAMA DE CLASES, CENTRADO EN LA CLASE *ANALIZADORMOVIMIENTO*

La clase *Analizador Movimiento* se encarga de la coordinación y el análisis del tráfico de valores generado por los sensores.

Su principal objetivo es obtener la información de la IMU. Después, mediante unos algoritmos de detección, se buscarán los posibles Movimientos. Una vez se ha obtenido un movimiento este será analizado (*Chequeado*), en busca del Gesto que éste pueda albergar. Tras descubrir un gesto lo suficientemente consistente, se avisará a la clase Sistema y se delegara en ella el gesto.

Para finalizar se describirán, a grosso modo, todas las clases:

- Values: Clase que contiene 3 valores (uno por cada eje de coordenadas) y toda la operatividad que requieren estos.
- Quaternion: Similar a Values pero con 4 valores.
- Punto: Valores de todos los sensores de la IMU en un instante temporal concreto. Contiene, aparte de otros valores, 3 Values (Aceleración, Giro y magnetismo) y un Quaternion.
- Muestra: la encargada de comunicarse con la IMU, detectar los valores nuevos del sistema y construir los puntos.
- Análisis Movimiento: Encargada de detectar y construir los Movimientos.
- Movimiento: Secuencia de puntos que cumplen ciertas condiciones y podrían albergar un gesto.
- CheckMovimiento: Encargada de analizar los Movimientos y buscar Gestos.
- Pico: Clase usada durante el chequeo. Expresa un cambio de signo en los valores de los sensores.
- Gesto: No aparece en el diagrama anterior por cuestiones de presentación. Básicamente un gesto será la entidad final de un análisis de movimiento positivo. Éste será pasado al Sistema para que determine si es algún gesto de los guardados.

SUB-DIAGRAMA ANALIZADOR GESTOS

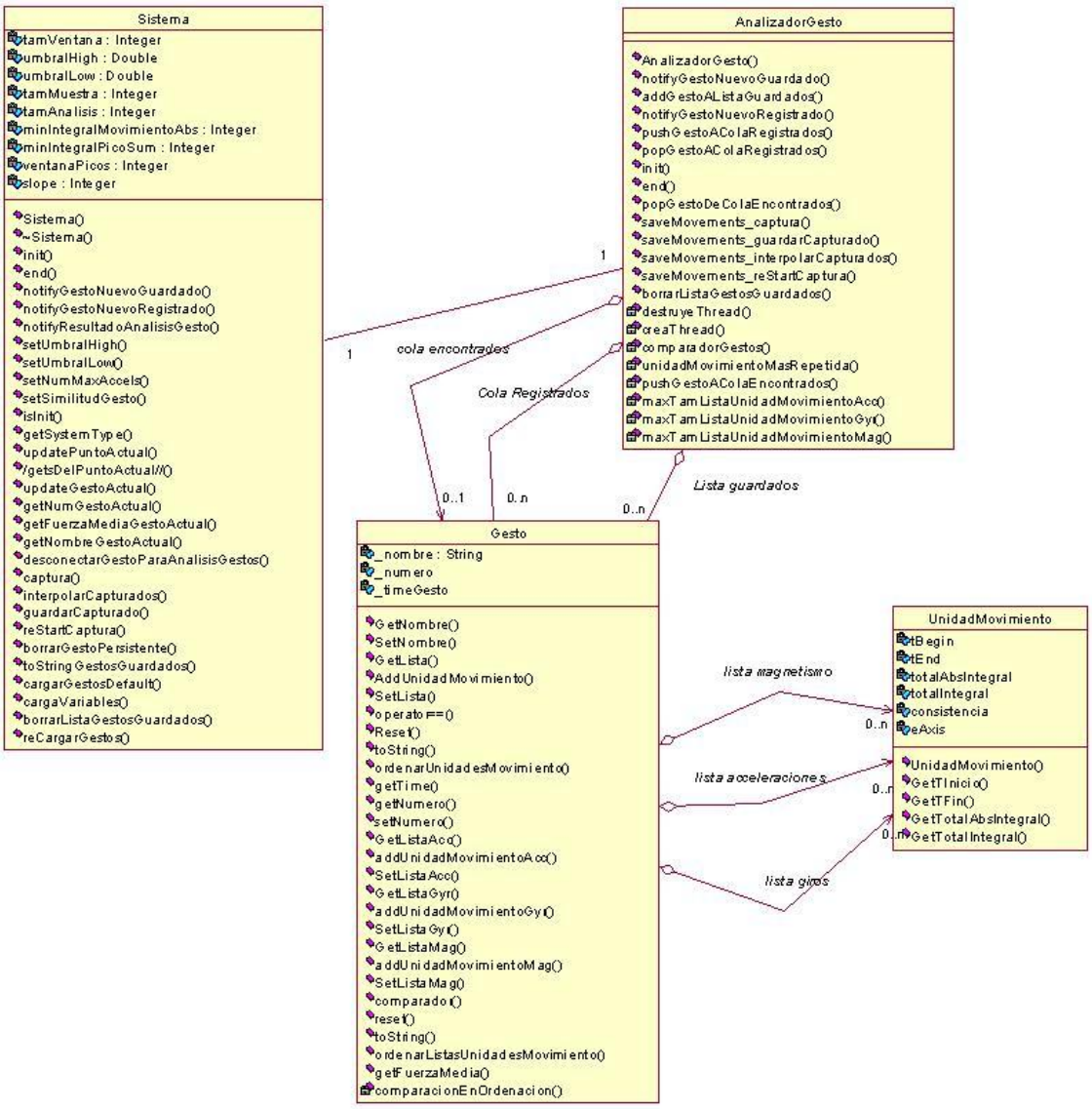


ILUSTRACIÓN 6: DIAGRAMA DE CLASES, CENTRADO EN LA CLASE ANALIZADORGESTO

En esta sección focalizamos el análisis en *Analizador Gesto* [*Ver* Ilustración 6: Diagrama de Clases, centrado en la clase AnalizadorGesto]. Esta clase que se encarga básicamente de determinar si un gesto se corresponde a algún gesto guardado en el sistema de persistencia.

Recibe el gesto registrado en el *analizadorMovimiento* y lo compara con los guardados, dando como resultado el veredicto de cual se asemeja más.

Para llevar a cabo la tarea el Sistema dispone de las clases siguientes:

- AnalizadorGesto: Encargada coordinar el análisis. Contiene toda la operatividad y todas las relaciones para realizar la tarea.
- Gesto: nombrada también en el diagrama anterior. Es el objeto más importante del sistema. Y por el que se diseña. El resultado final debe ser un gesto, el cual contendrá:

  - Nombre.
  - Número.
  - Fuerza (aceleración) media.
  - Unidades de movimiento.
- UnidadMovimiento: Todo gesto está compuesto de Unidades de movimiento. Las cuales representan las unidades esenciales y representativas de un *Movimiento*. Cada unidad está compuesta por:
  - Tipo: aceleración, giro, magnetismo.
  - Masa integral.
  - Masa integral absoluta.
  - Eje.
  - Punto inicio y fin.
  - Tiempo inicio y fin.

SUB-DIAGRAMA CARGA SISTEMA

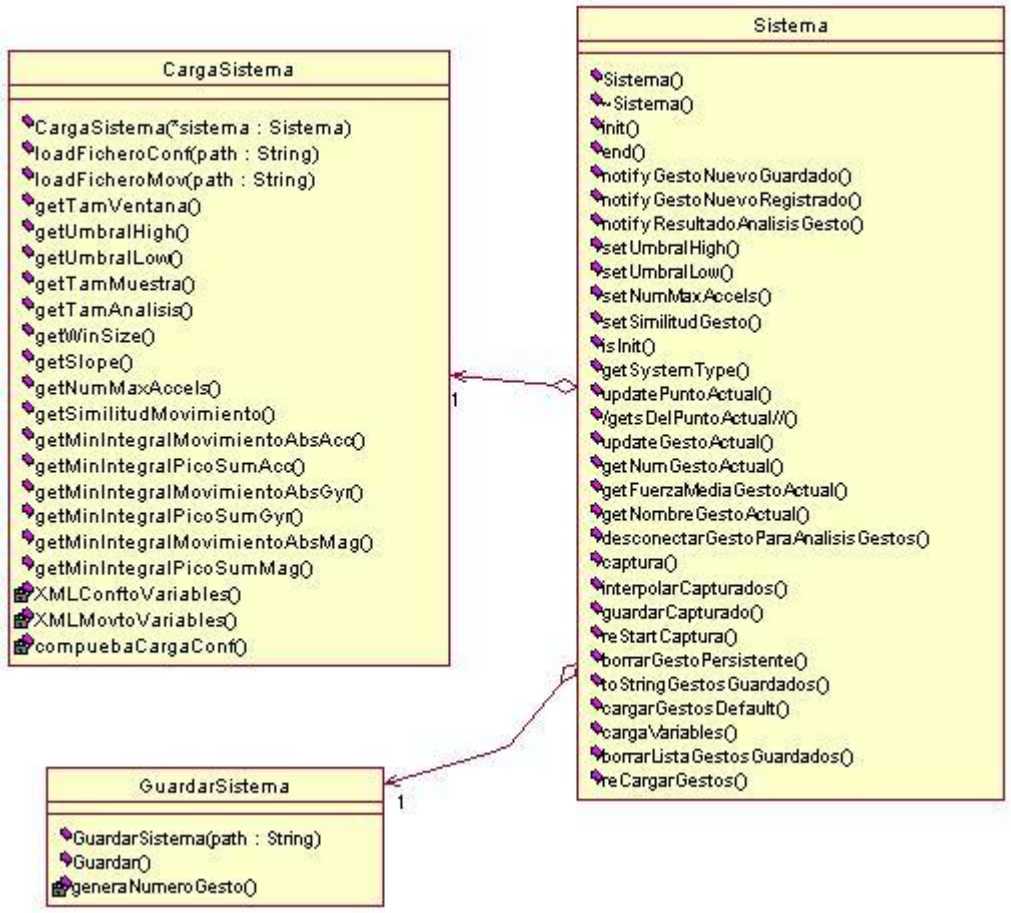


ILUSTRACIÓN 7: DIAGRAMA DE CLASES, CENTRADO EN LA CARGA DEL SISTEMA

En el diagrama anterior [Ver Ilustración 7: Diagrama de Clases, centrado en la Carga del Sistema] nos centramos en la carga del sistema, la cual está compuesta de dos clases que se encargan de cargar toda la información de los XML y de guardar los cambios que se realizan en la gestión de los gestos.

ANÁLISIS DEL COMPORTAMIENTO

En este apartado se va a analizar cómo se comporta el sistema. El software está formado de clases internamente, las cuales interactúan entre ellas para llevar a cabo todos los objetivos que el programa debe cumplir.

A continuación se procede a mostrar cómo se comporta el sistema en los casos:

- Inicio del Sistema.
- Carga del Sistema.
- Detención o Stop del Sistema.
- Proceso Reconocimiento.

INICIO DEL SISTEMA

El inicio del sistema es el proceso de activación del software. Se encarga de activar todos los análisis, según el modo en que se desee de ejecución.

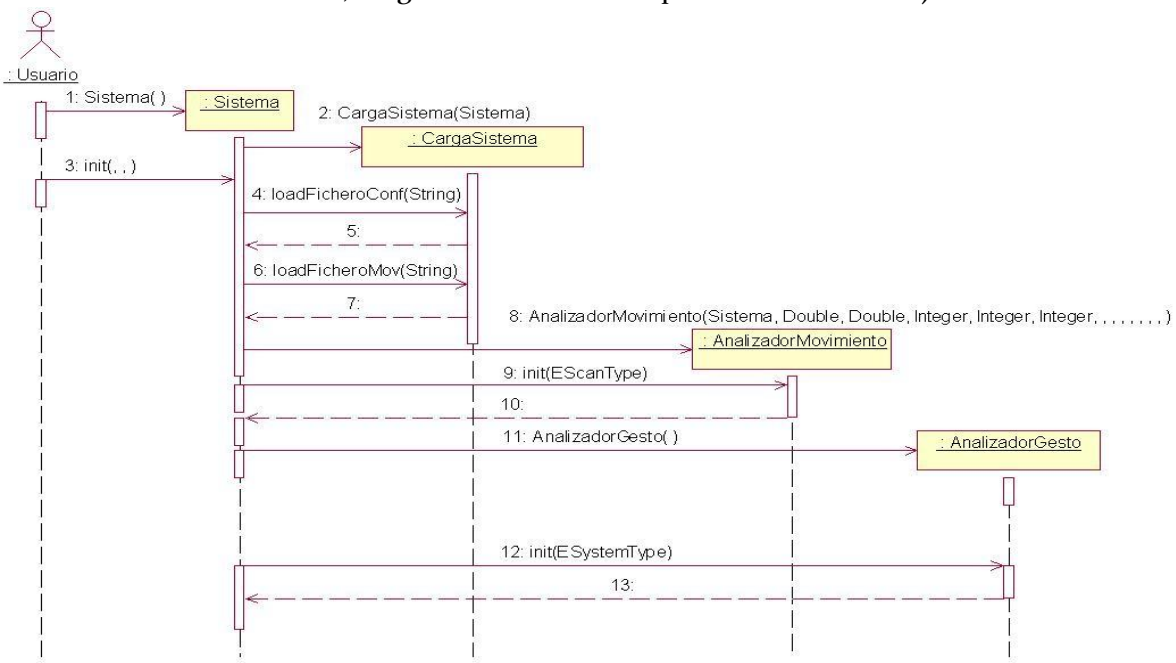


ILUSTRACIÓN 8: DIAGRAMA DE SECUENCIA, INICIO DEL SISTEMA

CARGA DEL SISTEMA

La Carga es el proceso encargado de traer todos los valores al sistema provenientes de la capa persistencia. El sistema carga dos tipos de XML: El archivo de configuración y el archivo de gestos. En las siguientes líneas pasaremos a mostrar como son ambas cargas, pues su proceso es distinto.

Primero se muestra cómo se carga la configuración. El sistema inicializa la clase *CargaSistema*, llama al proceso de carga (*LoadFicheroConf*) y después llama a todos los GETS para ir obteniendo todos sus valores.

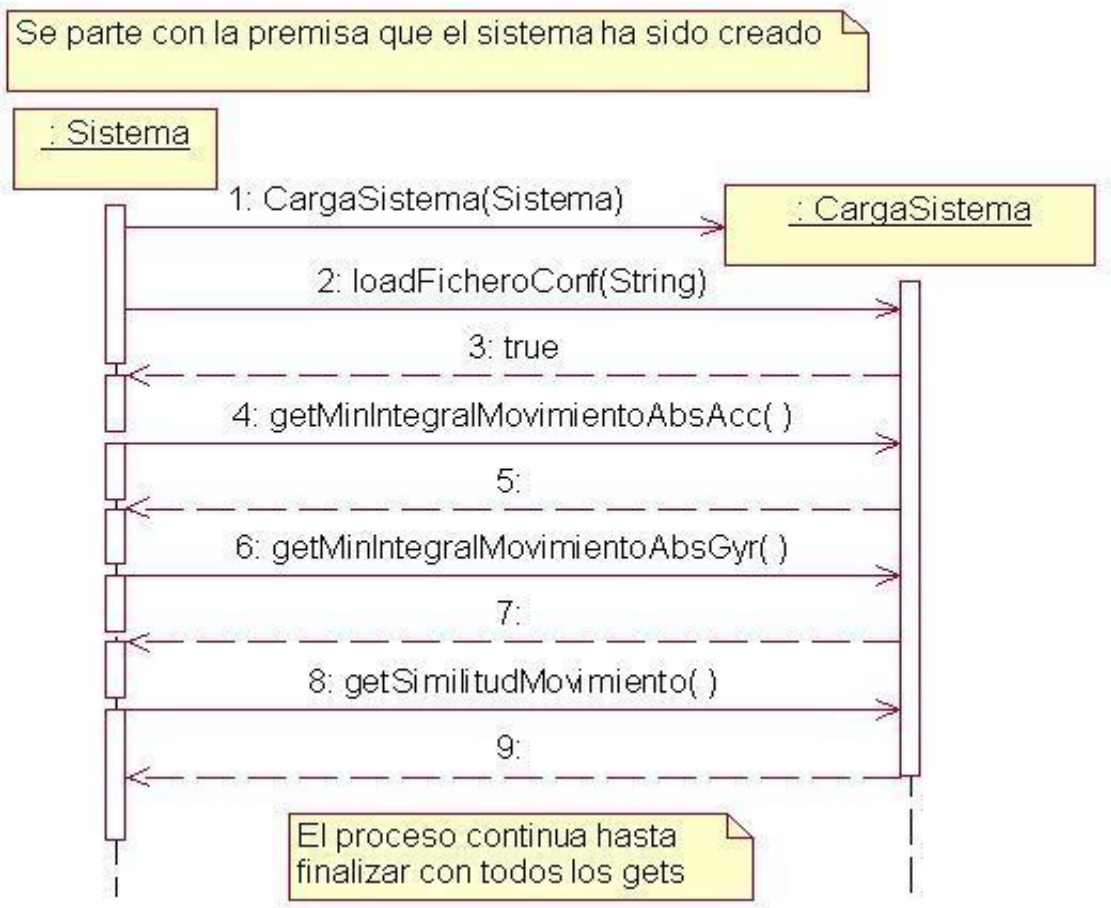


ILUSTRACIÓN 9: DIAGRAMA DE COLABORACIÓN, CARGA DE LA CONFIGURACIÓN

La Ilustración 10: Diagrama de Colaboración, Carga Gestos muestra cómo se realiza la carga de todos los gestos en el Sistema.

ILUSTRACIÓN 10: DIAGRAMA DE COLABORACIÓN, CARGA GESTOS

El proceso de carga de los gestos, como se ha comentado, es distinto. El Sistema llama a la función de *CargaGestos* igual que en la carga de la configuración, pero en este caso es la clase *CargaSistema* la que va comunicando cada Gesto que encuentra en el XML. Este proceso es muy distinto al anterior caso, como se vió en la Ilustración 9: Diagrama de Colaboración, Carga de la configuración, la clase sistema tenía un papel activo y se encargaba de traerse todos los valores de la configuración, mientras que la clase *CargaSistema* tenía un funcionamiento pasivo. Con la Carga de Gestos los papeles se intercambian y es la clase *CargaSistema* la que toma el relevo de actividad.

DETENCIÓN O STOP DEL SISTEMA

Como su nombre indica, este proceso es el encargado de detener el sistema. El paso que nos ocupa es muy importante pues el sistema trabaja con un hardware, la IMU, que se debe detener siempre. Así se evita incongruencias en el sistema, o problemas de detección de hardware en los inicios del sistema posteriores.

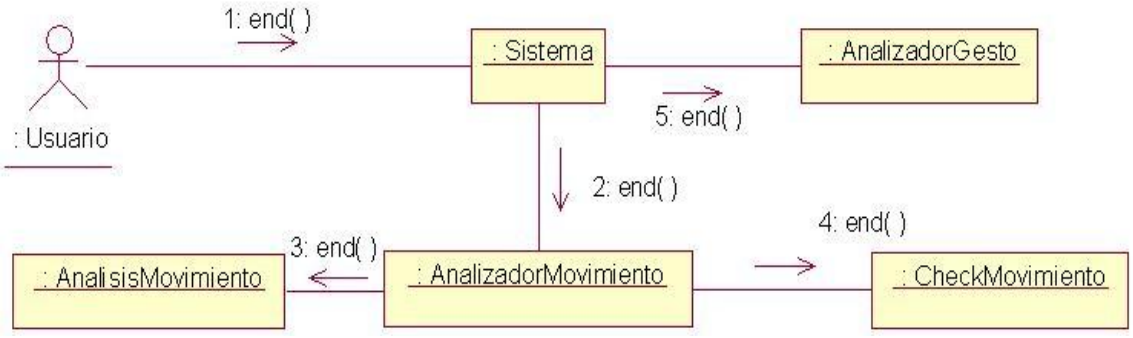


ILUSTRACIÓN 11: DIAGRAMA DE COLABORACIÓN, DETENCIÓN O STOP DEL SISTEMA

ANÁLISIS DE ESTADOS

En este sub-apartado analizaremos los estados lógicos que puede tener el Software. Para especificarlos usaremos los Diagramas de Estado que ayudan a representar las rutas o caminos que puede tomar la información internamente en el software.

El sistema tendrá dos tipos básicos de diagramas de estado. Como se analizó anteriormente, la librería tiene dos modos básicos, los cuales permiten ciertas funcionalidades y al mismo tiempo excluyen otras. Así, si se ejecuta en Modo Gestión de Gestos no se podrá obtener el gesto actual, de la misma manera si se está en el Modo de Reconocimiento Gestual no se podrá borrar un gesto. Como consecuencia de lo anterior se ha decidido especificar cada modo en un diagrama de estado.

DIAGRAMA DEL MODO GESTIÓN DE GESTOS

El primer diagrama que se presenta es el de Gestión de Gestos [Ilustración 12: Diagrama de Estados, Gestión de Gestos]. En él se puede ver que el sistema puede estar detenido o Iniciado.

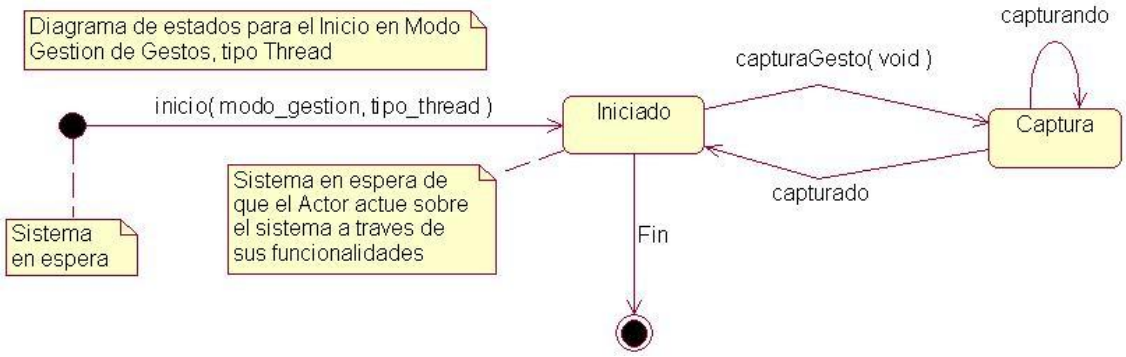


ILUSTRACIÓN 12: DIAGRAMA DE ESTADOS, GESTIÓN DE GESTOS

Una vez el sistema esta iniciado, según interactúe el Usuario, el sistema podrá pasar a estar en estado Captura, en ese estado el usuario podrá almacenar nuevos gestos en memoria persistente. Para ello el sistema se pondrá a analizar el tráfico, esperando que el cliente haga algún gesto.

DIAGRAMA DEL MODO DE RECONOCIMIENTO GESTUAL

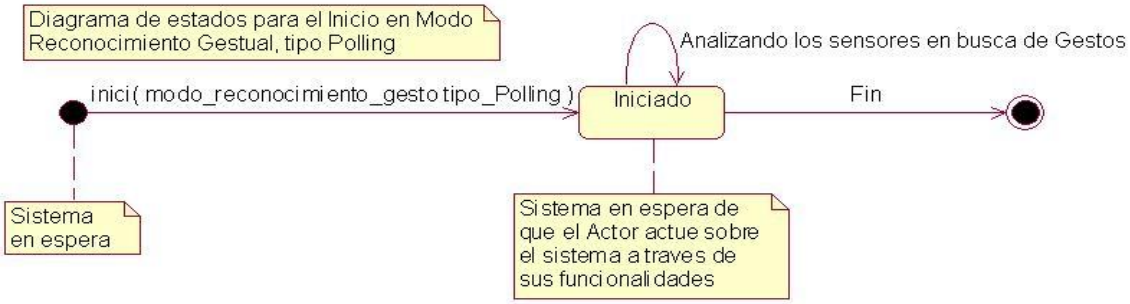


ILUSTRACIÓN 13: DIAGRAMA DE ESTADOS, RECONOCIMIENTO GESTUAL

El segundo diagrama [Ver Ilustración 13: Diagrama de Estados, Reconocimiento Gestual] se presenta la Gestión de Gestos. Viendo el diagrama de interés, se contempla como el sistema puede estar detenido o iniciado, cuando su estado es iniciado el sistema se encuentra en continuo análisis, buscando gestos. Una vez se registra un gesto de entrada, se guardan en una cola a la espera de que el usuario interactúe con él y obtenga gestos.

CAPÍTULO 4: DISEÑO DEL SISTEMA

INTRODUCCIÓN

Cómo se ha visto en el apartado anterior la parte de análisis se centra en presentar la problemática en el software que se está desarrollando. Definiendo sus relaciones lógicas o conceptuales entre los componentes del sistema, los datos necesarios, la información de salida, los procesos que deben llevarse a cabo para satisfacer los requisitos, que restricciones deben tenerse en cuenta, los usuarios y las necesidades de comunicación, independientemente de cómo debe realizarse físicamente. Sin embargo, en el diseño abordaremos cómo se van a satisfacer las funcionalidades descritas en la ERS. El diseño conlleva la visión física de los procesos diseñados, el almacenamiento de los datos, los formatos exactos, etc. [IPI03] Así, procederemos a explicar la arquitectura del sistema, el aspecto externo del sistema, las interfaces de usuario propuestas como ejemplo y por otra, la forma física en la cual se implementaran los procesos y los ficheros XML llegando a especificar alguna característica de implementación.

ARQUITECTURA POR CAPAS

Se ha elegido dividir el sistema en capas. La distribución por capas es un estilo de programación cuyo objetivo principal es separar en diferentes partes o capas el sistema con el objetivo de que el desarrollo final se componga de componentes no dependientes entre sí. Cada capa ofrece un conjunto de servicios, siempre relacionados con el contexto de la capa.

Para el sistema se quiso usar una distribución tradicional en 3 capas, aunque finalmente se optó por variar el esquema como respuesta a un requisito del sistema que se explica a continuación [Ver Ilustración 14: Diagrama de la arquitectura del sistema].

Un requisito importante de la ERS fue el posible cambio de dispositivo de captura de movimiento. A nivel arquitectónico la respuesta fue la definición de una nueva capa. Con la idea de aislar y localizar la interacción con la SDK del dispositivo. Con todo esto, no se soluciona del todo la adaptación con otros hardwares de Motion Tracking pero las modificaciones necesarias a nivel de código son mínimas y fáciles.

El resultado de respetar el requisito anteriormente comentado es un software compuesto de una capa de presentación, una capa de negocio y dos capas al mismo nivel: la capa de persistencia y la capa de Sensores.

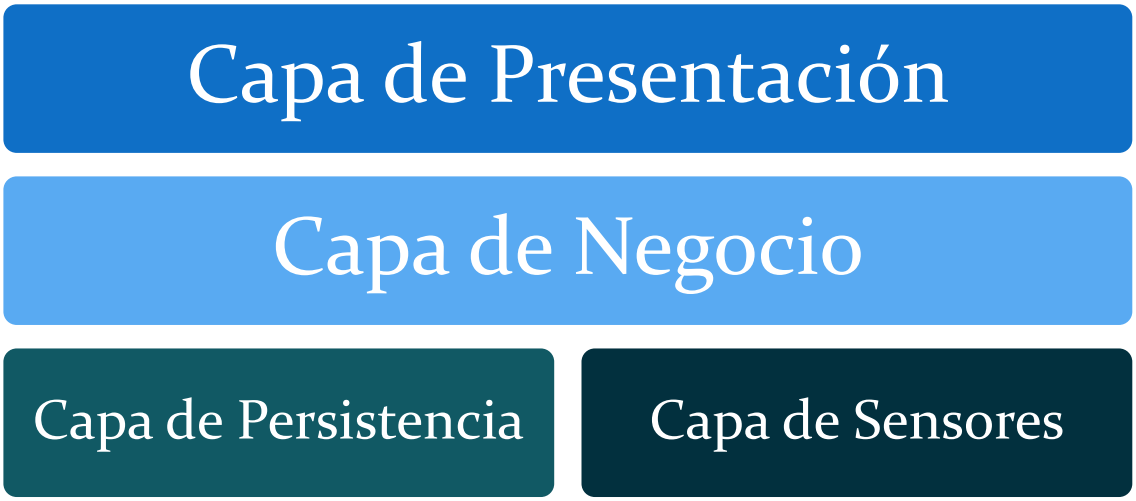


ILUSTRACIÓN 14: DIAGRAMA DE LA ARQUITECTURA DEL SISTEMA

CAPA DE PRESENTACIÓN

La capa de presentación es la capa que ve el Usuario, es por eso que también suele ser llamada capa de Usuario. Se encarga de la interacción con los actores, gestionando que deben y pueden hacer en cada instante.

En nuestro software se realizaron dos soluciones de presentación. Una interfaz de tipo consola, la cual fue usada para el testeado y pruebas durante las fases de implementación, y otra interfaz de tipo gráfico, como posible aplicación del software.

Para el diseño de cada interfaz se han realizados dos tareas claramente diferenciadas: el diseño externo y el diseño interno.

1. En el diseño externo se definen las interacciones del usuario con el programa con el objetivo de diseñar los elementes tangibles que el usuario ve.
2. En el diseño interno se analiza cómo la capa actual se comunica con la capa de nivel inferior.

En éste apartado se ha optado por dividirlo en dos grandes bloques uno perteneciente a la interfaz de consola y otro perteneciente a la interfaz gráfica. En cada bloque se aborda las dos tareas principales del diseño de la interfaz.

Interfaz de Consola

Durante la implementación se usó un desarrollo iterativo. En cada iteración se han ido añadiendo diferentes funcionalidades. Con cada iteración se realizaban pruebas de testeado y para ello se optó por el diseño de una interfaz de tipo texto, con la idea de facilitar las pruebas.

DISEÑO EXTERNO

Como la consola fue diseñada con la idea de ser usada para el testeado durante la implementación, se permite el acceso a todas las funcionalidades del sistema. Ésta interfaz siempre será usada por administradores o desarrolladores del sistema.

La consola tiene las funcionalidades típicas de cualquier aplicación modo texto.

- Permite la navegación entre menús.
- Las opciones se eligen mediante letras.
- Se usa mediante el teclado, el ratón no tiene ninguna función en las aplicaciones de éste estilo.
- La información se muestra de forma iterativa y secuencial.

Cuando se ejecuta la aplicación el primer menú que se muestra es el Menú Principal [Ver Ilustración 15: Consola, menu de Inicio].

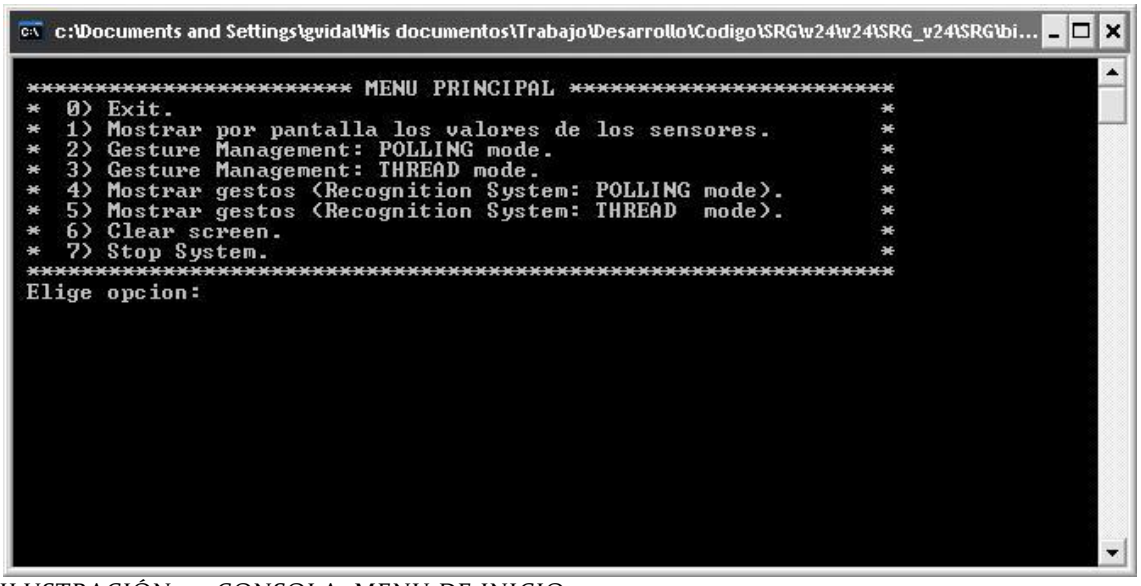


ILUSTRACIÓN 15: CONSOLA, MENU DE INICIO

El menú de inicio es el punto de partida para el uso de todas las funcionalidades del sistema. Así se permite:

- El escaneado de Puntos actuales: Como se vio en las ERS, uno de los requisitos era poder tener acceso al escaneo de los sensores. Esta opción permite ejecutar esa función [Ver Ilustración 16: Consola en escaneo de sensores].

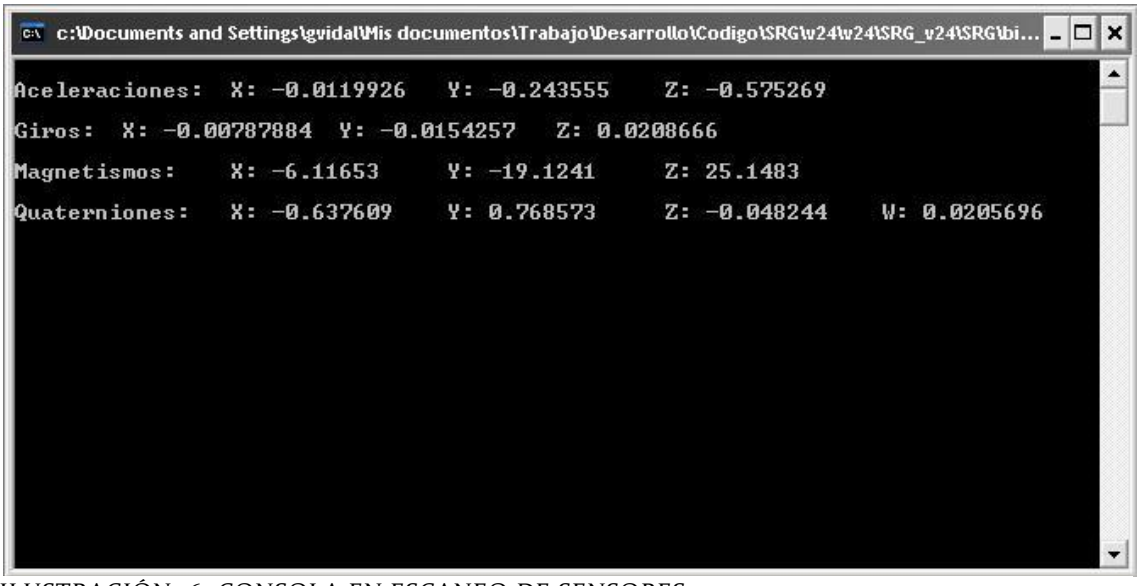


ILUSTRACIÓN 16: CONSOLA EN ESCANEO DE SENSORES

La opciones 2 y 3 permiten la ejecución del sistema en modo Gestión de gestos. Ésta modalidad de ejecución posee su propio menú, con distintas funciones encargadas de realizar tareas de gestión con los gestos. [Ver Ilustración 17: Consola en el menú Gestión de Gestos.]. Cada opción de inicio se encarga de ejecutar el programa de una manera en particular. En tipo *polling*, el sistema es el que pide los valores a los sensores, mientras que en el tipo *thread*, la lectura de los sensores se realiza siempre, avisando al sistema de que se tienen nuevos valores provenientes de la IMU.

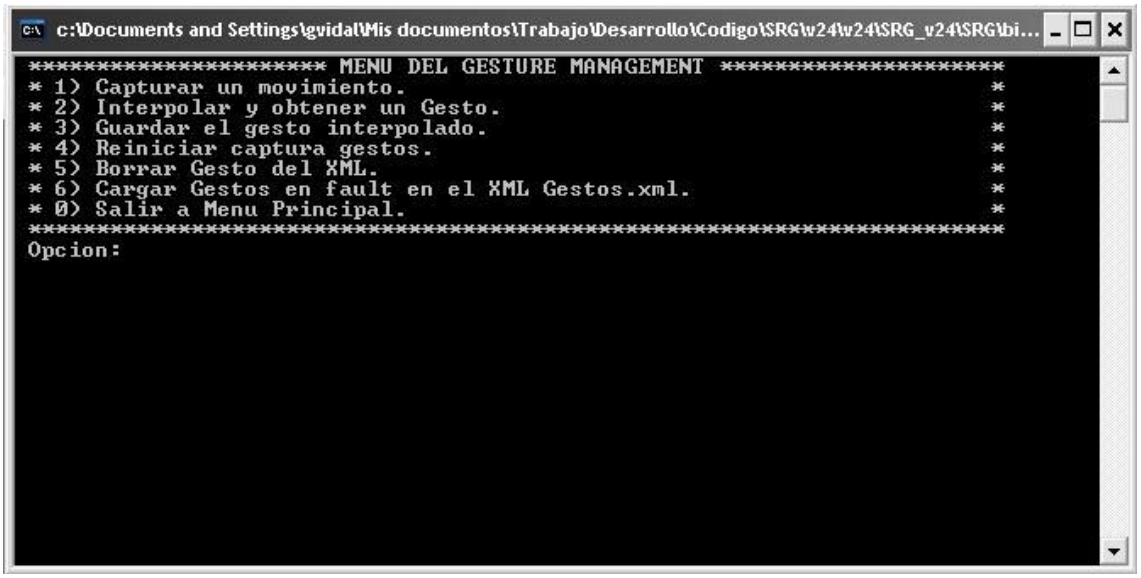


ILUSTRACIÓN 17: CONSOLA EN EL MENÚ GESTIÓN DE GESTOS.

Viendo la Ilustración 17: Consola en el menú Gestión de Gestos., se pueden ver una colección de opciones, las cuales representan todos los requisitos especificados para este modo. Así se puede:

- Capturar gestos, los cuales serán guardados en memoria.
- Interpolar y obtener un gesto nuevo, de los gestos capturados con anterioridad.
- Guardar el gesto resultado de interpolar.
- Reiniciar la captura de gestos.
- Borrar algún gesto del XML.
- Cargar los gestos default al XML de gestos. Esta opción sirve para dejar los gestos del sistema que vienen de forma definida desde fábrica.

- Por último, si volvemos la atención al menú principal [Ver Ilustración 15: Consola, menu de Inicio], las opciones 4 y5 son las encargadas de iniciar el sistema en el Modo de Reconocimiento Gestual. Esta opción nos permite ver en todo momento que gestos estamos realizando. También se permite la opción de desconectar gestos del análisis. En la ilustración 18 se muestra cómo es la interfaz de consola en modo Reconocimiento Gestual.

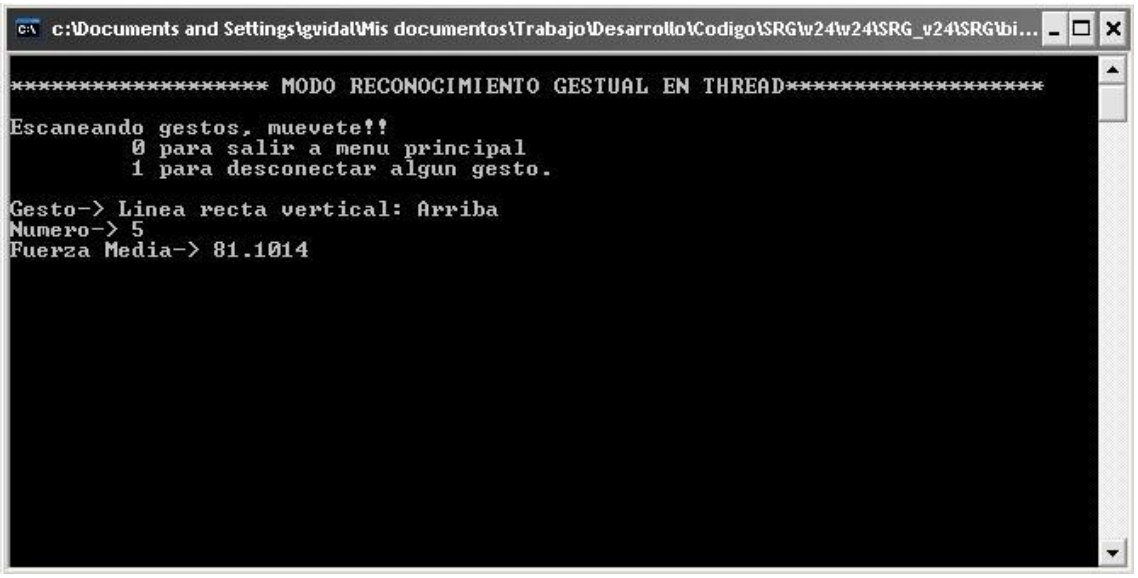


ILUSTRACIÓN 18: CONSOLA EN MODO RECONOCIMIENTO GESTUAL

DISENO INTERNO

No es objetivo de este apartado analizar cómo se une la interfaz de consola con el Sistema. Simplemente se explicará sin tecnicismos como se consigue y en qué punto se comunican ambas capas.

Se ha de tener en cuenta que la aplicación generada junto a la consola va completamente integrada en un ejecutable final, con un único EXE junto a los XML.

La consola, al ser ejecutada, contendrá un objeto de tipo Sistema y a través de éste se podrán realizar todas las funcionalidades del sistema, además de recibir las respuestas a cada ejecución.

En el apartado de implementación se entrará en más detalle como son llamadas las funciones y en qué orden para poder cumplir cada requisito del sistema.

Interfaz gráfica

DISEÑO EXTERNO

Para presentar el proyecto se diseñó una interfaz gráfica. En la cual aparece un perro virtual que obedece e imita los gestos del usuario en base a un conjunto de patrones de movimiento almacenados previamente.

ILUSTRACIÓN 19: INTERFAZ GRÁFICA, PERRO VIRTUAL

Se buscó en todo momento que la interfaz fuera de fácil manejo, intuitiva y amigable, intentado que se centrara en mostrar los requisitos más importantes del proyecto. Dado que no se quería dar la imagen de que el proyecto iba dirigido a realizar esta interfaz, se optó por un diseño y animación simples.

En lo referente al desarrollo del modelado y animación del objeto virtual se realizó con el 3DS MAX, software desarrollado por Autodesk para el modelado y animación virtuales.

En la aplicación gráfica se pueden ver y realizar las funcionalidades básicas del Sistema de Reconocimiento Gestual, como son el Reconocimiento Gestual y la Gestión de los Gestos.

Así se dispone de un menú [Ilustración 20] que da la posibilidad de ejecutar el sistema en cualquiera de los dos modos de que dispone la aplicación.

ILUSTRACIÓN 20: INTERFAZ GRÁFICA, MENU

DISEÑO INTERNO

Para la realización de esta interfaz gráfica se usó el 3d GameStudio A7.Este software dispone de su propio lenguaje, el Lite-C. Este lenguaje permite el uso de bibliotecas externas al propio programa, así se puede añadir más funcionalidades. Nosotros generamos nuestra propia librería llamada SRG.dll con todas las funcionalidades del sistema. Ésta librería fue incluida en el proyecto generado con el 3d GameStudio, además de un fichero con las cabeceras de las funciones de la librería.

CAPA DE NEGOCIO

Introducción

La capa de negocio es la capa intermedia. Es el núcleo del sistema, donde reside toda la lógica de la aplicación. La capa de negocio es la encargada de resolver las llamadas al sistema y dar los servicios demandados.

Ésta sección del sistema se comunica con la capa de presentación para obtener las peticiones del sistema. Al mismo tiempo también se comunica con las capas de persistencia, y con la capa de Sensores para obtener los valores del escaneo de la IMU.

En esta sección se abordarán todos aspectos referentes a esta capa. Primero pasaremos a comentar como se ha modulado la lógica del sistema, intentando separar los distintos análisis. Finalmente hablaremos de las bibliotecas externas que se han usado para gestionar ciertos aspectos del sistema.

Módulos de la Capa

En el apartado de análisis ya se dejó constancia que el sistema se podía dividir en dos grandes bloques. En los diagramas de clases, ya se optó por hacer una división para intentar ser más didácticos en el análisis. Cada uno de los dos grandes bloques se encargan de tareas distintas, siempre coordinados por la clase *Sistema*.

1. Módulo de análisis de Movimiento: Éste módulo es el encargado de recoger los datos de la Capa de Sensores para proseguir con el análisis con ellos. La clase *AnalizadorMovimiento* es la encargada de gobernar el análisis y obtener los Movimientos, si es que se cumplen las condiciones las condiciones que se explicarán en el capítulo de implementación.
   El módulo se divide en varias secciones lógicas, encargadas de diferentes temas pero que trabajando coordinadamente obtienen el resultado esperado.
   Así, si procedemos a un análisis más profundo del modulo podemos encontrar las distintas secciones:
   - Sección de recogida de valores: Su función básica es gestionar la IMU y obtener todos sus parámetros de la IMU. Construye Puntos y lo inserta en una cola. De esta sección lógica se encarga la clase Muestra.
   - Sección de análisis de valores: Se ocupa, básicamente, de escanear los puntos nuevos en busca de Movimientos. Está sección implementada en clase *AnalisisMovimiento*.
   - Sección de *Checkeado de Movimientos*: Esta sección se encuentra implementada en la clase *CheckMovimiento*. Se encarga de analizar los movimientos nuevos y de construir Gestos.

2. Módulo Análisis Gestos: Éste módulo se encarga del análisis de los gestos nuevos. Los analiza y compara con los gestos guardados, buscando patrones similares entre el gesto en análisis y todos los guardados en sistema de persistencia. Este se encuentra gobernado por la clase *AnalizadorGesto*, la cual se encarga de recibir las peticiones de la clase Sistema, y gestionar las comparaciones entre gestos. Prácticamente el modo Gestión Gestos se encuentra implementado en su conjunto en éste módulo.

Bibliotecas Externas

En esta capa se han usado distintas librerías que han ayudado y añadido funcionalidad en el sistema. A continuación se procederá a nombrarlas y explicar cuál ha sido su función y en qué han colaborado.

- Librería *MathUtils*: Librería que ha añadido la lógica de las matrices y los cálculos entre ellas. También ha sido usada en el sistema para la lógica de las operaciones con Quaternion. Esta Biblioteca es de Labhuman, creada por y para los proyectos del instituto.
- Librería *CCtrlThread*: Ésta librería se encarga de la gestión y construcción de hilos de ejecución. Biblioteca muy utilizada pues el rendimiento y las restricciones temporales son clave en el proyecto. Esta biblioteca es propiedad de Labhuman, creada por y para los proyectos del instituto.

CAPA DE PERSISTENCIA

Introducción

La capa de datos o de persistencia es la encargada de la gestión del almacenamiento de datos, procurando que sean guardados en la memoria para conseguir persistencia entre distintas ejecuciones en el tiempo.

Ésta capa contiene una distribución en modulos, parecida al apartado de la capa de presentación, aunque, además se le ha añadido un sub-apartado de definición de los XML usados.

Módulos de la capa

Esta capa se distribuye en dos módulos de pequeño tamaño. El módulo de Carga del Sistema y el módulo de Guardar el Sistema. Ambos módulos se encuentran bajo las órdenes de la clase principal *Sistema*, la cual los coordinará y gestionará según sea conveniente.

1. Módulo de Carga del Sistema: Es sección lógica de la aplicación que se encarga de la lectura de los XML. Éste modulo será siempre ejecutada en cada inicio del sistema para obtener todos los parámetros de configuración, además de cargar todos los movimientos guardados del sistema.
2. Módulo Guardar el Sistema: Sección que se encarga de guardar en el sistema los nuevos gestos o de borrar los gestos que el cliente encuentre de interés eliminar del sistema. En un futuro debería de gestionar también cambios en la configuración en busca de persistencia de los cambios de configuración. Éste requisito se aborda de distinta manera permitiendo los cambios de configuración durante la ejecución del sistema, pero no permitiendo la persistencia de ellos entre distintas ejecuciones del sistema.

XML

El sistema de persistencia se sostiene sobre dos pilares: El XML de configuración y el XML de Gestos[1].

En el XML de configuración (conf.xml) residirán todos los parámetros de configuración de la aplicación. Como se ha comentado no se ha implementado la gestión de cambios sobre él de manera automática.

El XML de Gestos (Gestos.xml) contiene todos los gestos guardados que interesa reconocer en la aplicación. Sobre éste XML sí que se permite la gestión automática. Así, se pueden añadir nuevos gestos o borrarlos según las preferencias del Usuario.

Bibliotecas externas

Para la gestión de escritura y lectura en los XML se ha usado la librería TinyXML[2]. Esta biblioteca es de fácil manejo y de rendimiento rápido, es muy liviana y permite un acceso a los valores en forma de árbol muy útil y usable.

[1] Los XML utilizados en el Sistema se encuentran en el Anexo para ver cómo han sido diseñados y que información contiene.
[2] Se añade un enlace a la página de TinyXML en referencias, por si se desea obtener más información.

CAPA DE SENSORES

Con el objeto de solucionar el requisito de adaptabilidad del código ante posibles cambios de IMU, se decidió la creación de una nueva capa al mismo nivel que la de persistencia con la idea de cumplir ciertos aspectos. Ésta nueva capa tendrá las siguientes características:

1. Localización de la interacción Sistema IMU: El aspecto más importante, evita que el código se enmarañare y sea difícil localizar donde se han utilizado funciones pertenecientes a la SDK del dispositivo.
2. Minimizar los cambios de código: Aspecto relacionado con el primero, se intento hacer el menor uso de la SDK, facilitando los cambios de código en el proceso de adaptación.
3. Aislar la SDK: Para evitar que al hacer los cambios de código se puedan hacer cambios o añadir funciones que no interesan en el sistema. La clase *Muestra*, marca la funcionalidad que necesita el sistema en relación con el dispositivo y no se necesita nada más.

El manejo de esta capa está implementado en clase *Muestra*. Ella es la que coordina y obtiene los valores de los sensores. Esta clase tiene dos formas de funcionar:

- Con un papel activo respecto la SDK del dispositivo: Mediante *polling* obtiene los puntos según el Sistema le pida la obtención de valores nuevos.
- Con un papel pasivo: la clase hace el escaneado todo el tiempo capturando toda la informacion de los sensores y los va comunicando a la clase *AnalizadorGestos* para que los gestione. Este tipo de operar se ha llamado *Threading*, porque el hilo es independiente y siempre está en funcionamiento, sin tener en cuenta si el sistema no está usando toda la información que genera.

CAPÍTULO 5: IMPLEMENTACIÓN DEL SISTEMA

INTRODUCCIÓN

Este capítulo se centra en documentar la etapa de implementación. Por implementación entendemos el proceso de convertir una especificación del sistema en un sistema ejecutable [ISW05].

En esta sección no se va a describir todo el proceso de implementación, se ha querido evitar la inclusión de temas superfluos o excesiva muestra de codificación. Esto ha sido así para evitar que la memoria sea dependiente del lenguaje de programación o complicar los razonamientos de las soluciones en demasía.

El capítulo se centrará en los problemas más significativos que se han abordado, donde explicando el análisis de la problemática, su solución y finalmente mostramos cómo han sido implementados en seudocódigo.

PROBLEMAS

ESPECIFICACIÓN DE ENTIDADES

Análisis de la Problemática

Al inicio del desarrollo se nos plantearon ciertos aspectos problemáticos en las entidades que, a nivel conceptual pueden ser simples de entender pero al transcribirlas a código ocasionan ciertas dificultades.

La RAE define un movimiento como un estado de los cuerpos mientras cambian de lugar o de posición. En cambio define Gesto como un movimiento del rostro, de las manos o de otras partes del cuerpo con que se expresan diversos afectos del ánimo. Para entender la diferencia se debe poner la atención en la palabra *expresar* de la definición de Gesto, ya que es ahí donde se encuentra el principal hecho distintivo. Un Gesto es un Movimiento que "expresa" alguna cosa, es un Movimiento con un significado agregado.

Siguiendo este hecho característico se intentó solucionar la especificación, como se verá a continuación.

Solución

A nivel de sistema ambas definiciones se tendrán en cuenta. De este modo, un Gesto será un Movimiento que ha pasado ciertos análisis y ciertas premisas, el cual posee unas particulares características.

Siguiendo con lo anterior, un Movimiento será una secuencia de instantes de todos los sensores (Puntos), es decir una secuencia de valores. Sin embargo, un Gesto será, a groso modo, un Movimiento analizado y transformado a la mínima expresión. Es más, un Gesto será compuesto de varias Unidades de Movimiento, las cuales representa un punto de interés en un Movimiento.

La palabra exacta para definir el proceso de paso de Movimiento a Gesto, tal vez, sería digestión. Ya que un *Movimiento* es un *Gesto* en bruto, aunque se tiene que tener en cuenta que no todo *Movimiento* llegará ser un *Gesto*.

En resumen el Sistema propone plantear ciertos aspectos en cada análisis para filtrar el tráfico generado por la IMU buscando Movimientos que pudieran llegar a ser gestos finalmente.

Cómo se detectan y cómo son los análisis se explicarán en este capítulo, pero en otros Problemas.

Implementación

Para mostrar cómo se ha implementado, se ha decidido poner imágenes de las clases con sus atributos, creyendo que de esta manera queda más legible y fácil de entender.

Primero mostramos la entidad Movimiento, cómo se ve en la *Ilustración 21*: Entidad Movimiento, compuesta de Puntos.

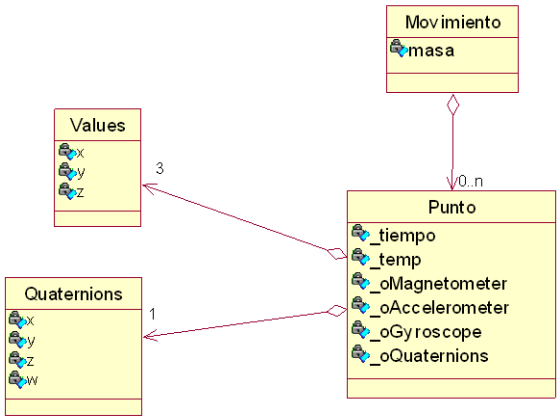


ILUSTRACIÓN 21: ENTIDAD MOVIMIENTO

Seguidamente se presenta la entidad *Gesto* [Ver Ilustración 22: Entidad Gesto], compuesta de *Unidades de Movimiento*.

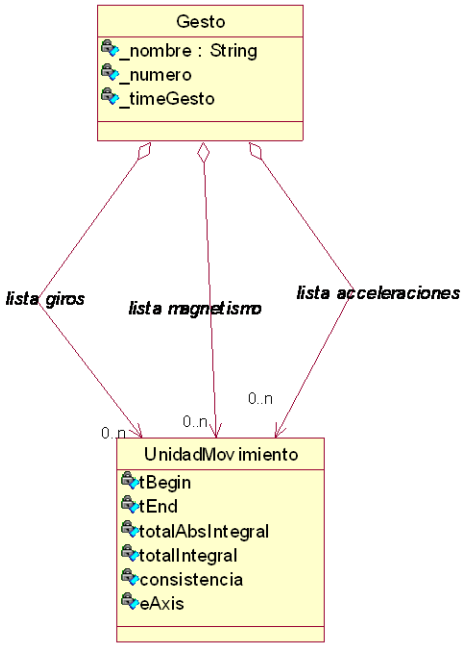


ILUSTRACIÓN 22: ENTIDAD GESTO

DETECCIÓN MOVIMIENTOS

Análisis de la Problemática

Uno de los problemas que nos asaltaron rápidamente durante el desarrollo del sistema es el hecho de buscar un algoritmo eficaz para la detección de los movimientos.

Este algoritmo es y será uno de los más importantes en nuestro software, ya que como se explicó, un gesto es la entidad de que parte todo el análisis.

Para el diseño de la solución se debió poner énfasis en una serie de factores que resaltan sobre las demás, como son:

- Detección de inicio y fin de un gesto.
- Umbrales de detección y descarte de un movimiento.

Solución

Para buscar la solución se recurrió a los estudios de Ari Yosef Bembasat [AYBoo], que incluyen un Sistema de Reconocimiento Gestual. En ellos se describe el análisis de la información provenientes de una IMU y la detección de secciones de valores interesantes para el análisis posterior.

El algoritmo se basa en una ventana de varianzas y usa el cálculo de la varianza para obtener el inicio y fin de cada sección de datos que son de interés como Movimientos.

En una distribución de valores, la varianza se define como el promedio de los cuadrados de las desviaciones a la media, y se denota por s^2:

$$\bar{x}$$

Si la Varianza es cero, todos los valores de la variable coinciden con la media, lo que significa que la dispersión es nula. Cuánto más alejadas estén las observaciones de la media, mayor será la varianza [EDI95].

En el proyecto disponemos de una distribución de valores provenientes de los acelerómetros. Nuestro propósito será analizar los valores buscando los instantes donde la varianza supere cierto intervalo.

Si el usuario no realiza ningún movimiento, los valores de los sensores estarán siempre rondando el 0, con lo que la media será 0, sin embargo, cuando la IMU sea agitada los valores se distribuirán alejándose de la media.

Siguiendo con lo anterior, esa característica de la varianza será la utilizada para saber que mientras un usuario esté ejerciendo fuerzas de aceleración sobre la IMU, la varianza se disparará a valores alejados de la media.

A continuación, se expondrán una gráfica, mostrando una sección de valores capturados donde se realizó un movimiento.

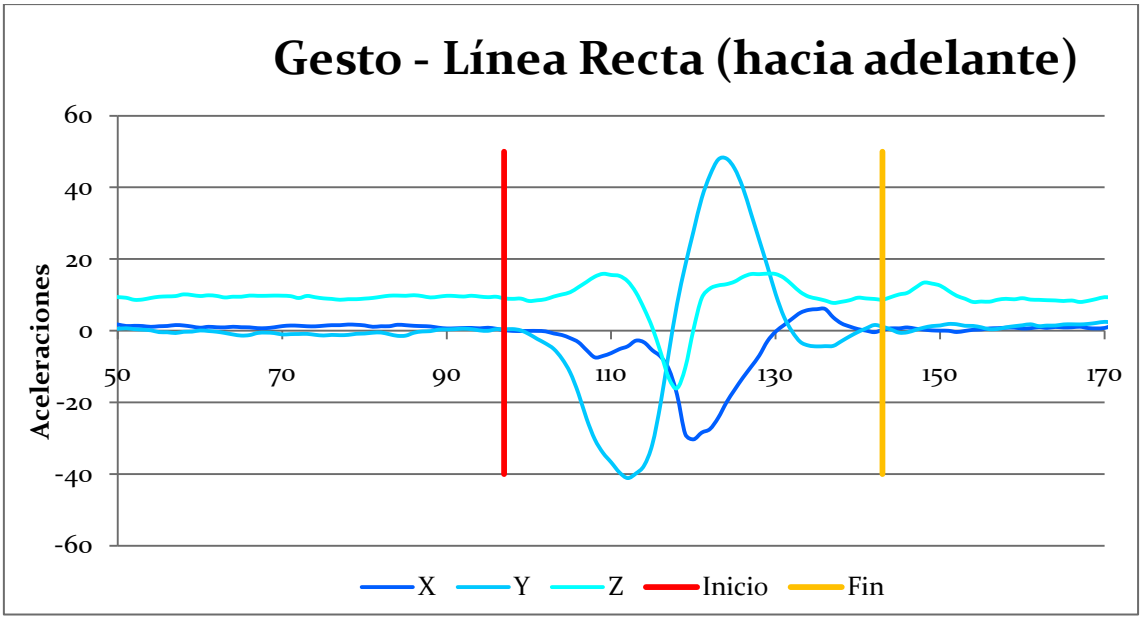


ILUSTRACIÓN 23: GRÁFICA DEL GESTO LINEA RECTA HACIA ADELANTE

Si nos fijamos en la Ilustración 23: Gráfica del Gesto Linea Recta hacia Adelante, se puede comprobar cómo los valores son casi constantes mientras no se realiza ningún Movimiento. Pero cuando el Usuario realiza alguna acción con el sensor, los valores varían muchísimo de la media dada. En la ilustración el movimiento queda comprendido entre las dos líneas verticales.

Antes de seguir con el análisis cabe destacar que los datos de la Z, tienen un incremento de 9,81 por la acción de la gravedad. En el siguiente apartado se explicará cómo se ha solucionado es te hecho.

Para realizar los cálculos de la varianza, como se ha comentado, se usó un algoritmo de Ventana. El algoritmo aplica la formula de la varianza en una sección de valores en concreto (ventana), comprobando si el resultado sobrepasa unos umbrales dados, el hecho de sobrepasarlos es el usado para la detección del inicio o en el fin del movimiento.

Para calcular la varianza se ha utilizado una fórmula equivalente, la cual nos beneficia en la implementación por su facilidad de cálculo:

El algoritmo es aplicado en cada eje de las aceleraciones, buscando los puntos de interés de cada Movimiento, sin importar la dirección del eje.

Finalmente cabe destacar que el algoritmo, conforme está planteado, exige que todo movimiento deba empezar y terminar con una aceleración o una velocidad angular nula, con lo que se deben normalizar todos los valores mediante la pendiente, con el fin de que la varianza empiece en cero y termine en o. Éste hecho puede llevar a errores en movimientos cíclicos o con rápidas transiciones.

Seudocódigo

```
Mientras (Sistema!=Stop)
{
   Obtención muestra de puntos con tamaño X;
   Cálculos para Obtener acumulado simple;
   Cálculos Acumulados de los cuadrados;

   Durante (int i = tamVentanaMedios;i<(Muestra - tamVentana) ; i++)
   {
      Cálculos varianza de la ventana;

      Si (no en un movimiento Y varianza>umbralAlto)
      {
         Empieza un movimiento;
         Guardar punto;
      }
      Sino
      {
         Si (en movimiento)
         {
            Guardo Punto;
            Si (varianza<umbralBajo)
            {
               Fin del Movimiento;
               Añadir Movimiento en cola;
            }
         }
      }
   }
}
```

ILUSTRACIÓN 24: SEUDOCÓDIGO DETECCIÓN MOVIMIENTO

ACELERACIÓN DEPENDIENTE DE LA GRAVEDAD

Análisis de la Problemática

En el problema de la detección de los movimientos, ya se introdujo una peculiaridad de los acelerómetros. Estos tipos de sensores dan unos valores dependientes de la gravedad. Esto quiere decir que estando la IMU en posición estática con el eje z en perpendicular a la horizontal de la tierra sus valores rondan el 9,81. En la siguiente gráfica [Ilustración 25: Gráfica del Gesto Linea Recta hacia adelante en las Aceleraciones del eje Z] se muestra una toma de valores en la realización de un gesto.

ILUSTRACIÓN 25: GRÁFICA DEL GESTO LINEA RECTA HACIA ADELANTE EN LAS ACELERACIONES DEL EJE Z

Sólo mostramos las aceleraciones en Z ya que, como se ha comentado, la IMU se encuentra en posición horizontal. El hecho de ser dependientes del la gravedad provocaba errores en la detección de movimientos, según estaba orientada la IMU era más fácil traspasar los valores del umbral.

Solución

Para normalizar las aceleraciones se usaron los quaternion, los cuales nos dan la orientación del dispositivo.

La solución se basa en que el dispositivo posee un sistema global y un sistema local. Para poder normalizar las aceleraciones simplemente había que encontrar la relación entre ambos.

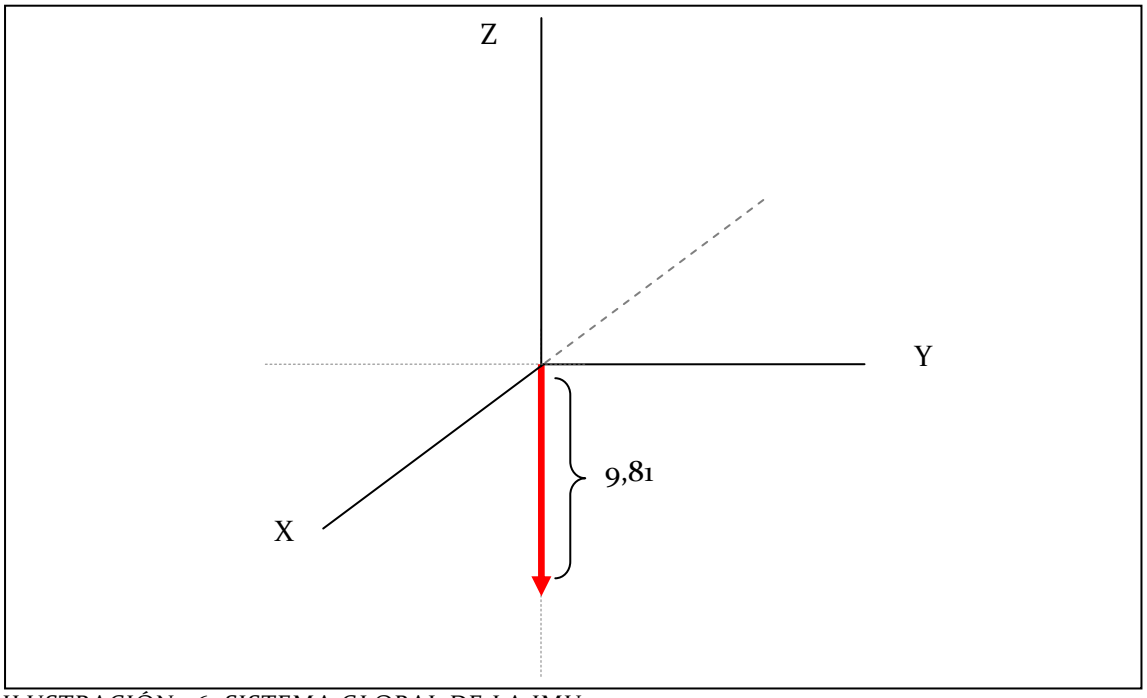


ILUSTRACIÓN 26: SISTEMA GLOBAL DE LA IMU

En la Ilustración 26: Sistema Global de la IMU se ve el sistema en posición estable, sin estar ejerciendo ninguna aceleración hacia ningún lado. Con lo que la aceleración en X e Y son 0, mientras que en Z es 9,81 por el efecto de la gravedad. Sobre el sistema global del dispositivo se sitúa el sistema local con las variaciones en la aceleración siempre dependiente del global. El factor gravedad siempre se distribuye entre los tres ejes según esté orientado el sistema local.

En el gráfico siguiente se sitúa encima del sistema global una captura en un instante dado [Ver Ilustración 27: Sistema Local de la IMU].

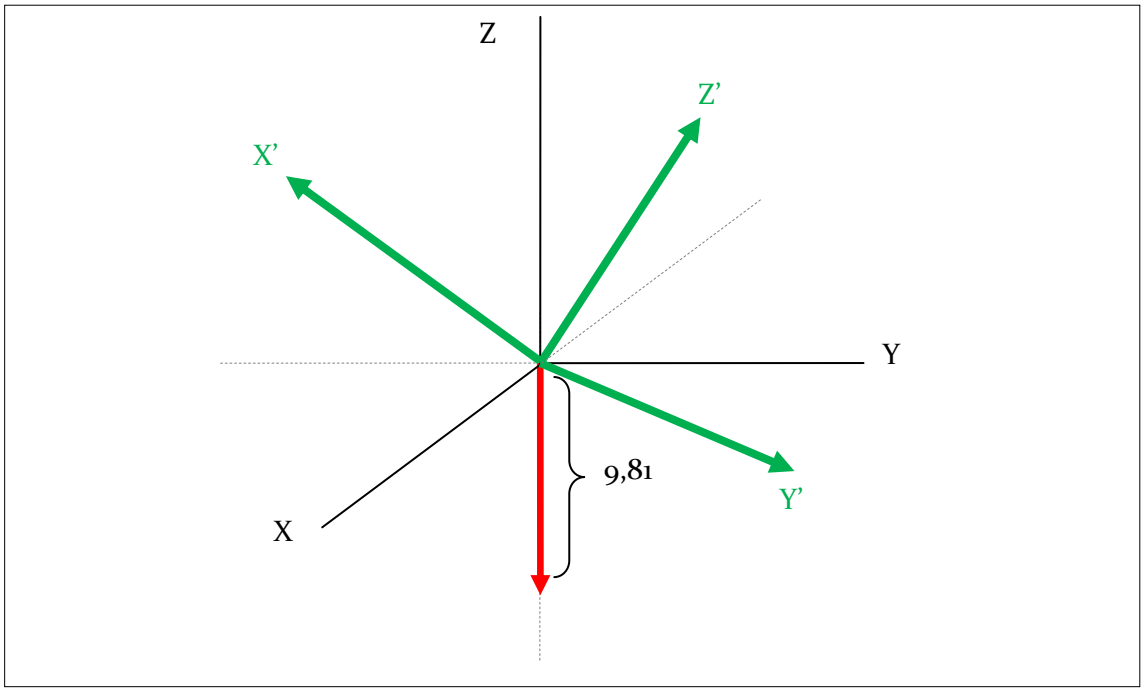


ILUSTRACIÓN 27: SISTEMA LOCAL DE LA IMU

La relación entre ambos sistemas la podemos obtener a través de su orientación. Es decir, si se sabe la orientación se puede restar a cada eje la parte de la aceleración de la gravedad que le per toca.

Para obtener la orientación utilizamos los quaternion, los cuales nos informan de la orientación del dispositivo. Para poder operar con ellos se tienen que transformar a una matriz de giro.

La relación entre las aceleraciones y la gravedad resulta sencilla y de fácil cálculo, a continuación mostramos la formula:

Donde, es el vector de aceleraciones que se dispone, son las aceleraciones que se desean obtener y es el vector de la gravedad. Para obtener la se debe tener la siguiente relación, ya que la gravedad es distribuida entre los tres ejes según la orientación:

Donde, es la gravedad que se tiene, es la gravedad que se desea obtener, libre del factor de la orientación, y es la matriz de rotación del sistema en ese instante.

La función final de la relación quedará definida como sigue:

Utilizando la formula anterior obtenemos el vector con las aceleraciones libres de la gravedad.

Seudocódigo

```
Mientras ( Sistema!=Stop )
{
        Data = Obtener Valores Sensores;
        oQuaternion=Obtener Objeto quaternion;

        Matriz_Rotacion = Obtener matriz de Rotación a partir del objeto oQuaternion;

        Aceleración_X = (Aceleracion_x_de_Data ) - ( Matriz_Rotacion[0,2]x(9.81));
        Aceleración_Y = (Aceleracion_y_de_Data ) - ( Matriz_Rotacion[1,2]x(9.81));
        Aceleración_Z = (Aceleracion_z_de_Data ) - ( Matriz_Rotacion[2,2]x(9.81));
}
```

ILUSTRACIÓN 28: SEUDOCÓDIGO DE LA NORMALIZACIÓN DE LAS ACELERACIONES

DETECCIÓN DE LOS GESTOS

Análisis de la problemática

En el primer problema se comentó cómo se detectaba un Movimiento y en la especificación de entidades se explicó que un Gesto no dejaba de ser un movimiento sin analizar y transformar. En este apartado se pasará a solucionar cómo detectar que un movimiento es un Gesto.

Una de las primeras dudas que nos deben asaltar es cómo se puede caracterizar un Gesto, buscar qué será aquello que nos permita identificar cada gesto y que más adelante nos permitirá distinguirlo de otros.

Analizando las características de un gesto podremos percibir que cuando un usuario realiza un movimiento la distribución de valores genera una curva similar a la siguiente:

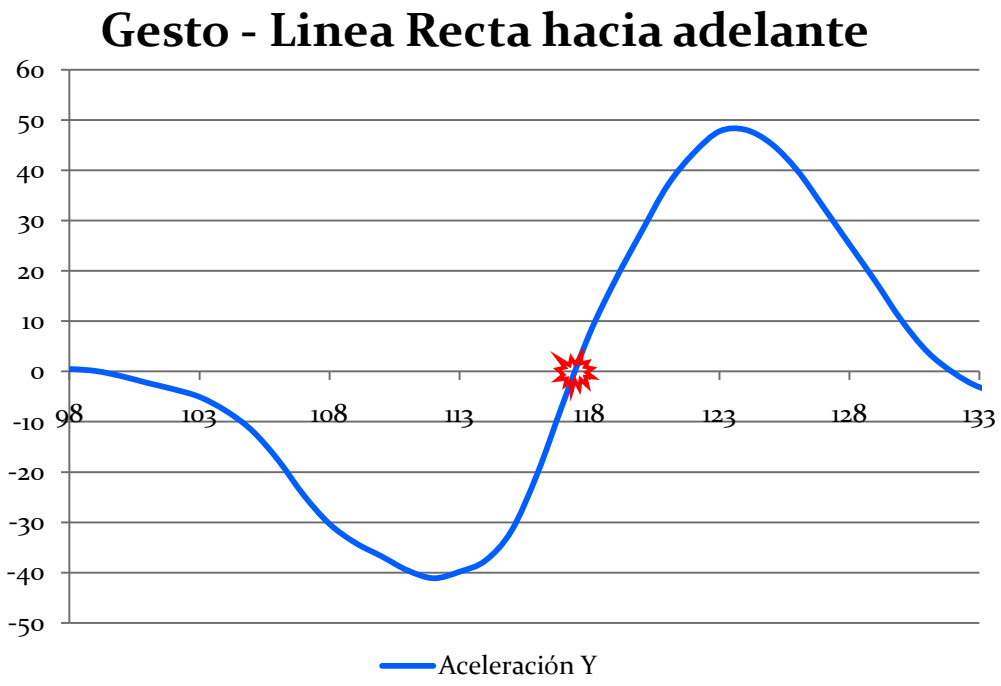


ILUSTRACIÓN 29: GRÁFICA DE GESTO LINEA RECTA HACIA ADELANTE EN LAS ACELERACIONES DEL EJE Y

Solución

El paso de un *Movimiento* a un *Gesto* tiene rasgos similares a un proceso de *digestión,* a partir de una secuencia de Puntos, más o menos larga, se obtienen una secuencia de *Unidades de Movimiento* de bastante menor tamaño y simplicidad. Así, por ejemplo de 300 instantes de los sensores se puede llegar a tener sólo dos o tres unidades, las cuales contiene la información que de verdad es de interés para los análisis posteriores.

Nuestra idea será buscar patrones de movimiento que sean similares a la distribución anterior. El algoritmo analiza los valores buscando instantes que cumplan, entre otras premisas, el cruce del eje de las ordenadas (Punto rojo en la gráfica). Con cada positivo del algoritmo el sistema generará un Pico, los cuales serán de utilidad en los análisis posteriores para obtener las Unidades de Movimiento.

Una Unidad de Movimiento es una entidad que comprende un cruce de eje en las ordenadas, la masa integral, la masa integral absoluta, el instante temporal, además del eje de la IMU de donde proviene y otros parámetros de interés. A continuación se muestra la clase que contiene la definición:

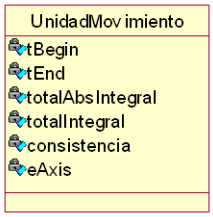


ILUSTRACIÓN 30: CLASE UNIDAD DE MOVIMIENTO

Al finalizar el análisis del Movimiento, si se han cumplido las condiciones, se devolverá un Gesto compuesto de Unidades.

Para finalizar, cabe destacar que en la creación de los Picos y las Unidades de Movimiento, se usan las masas de la Integral. El cálculo de la integral es provechoso para saber si un movimiento tiene la suficiente fuerza y la suficiente longitud como para serlo. Con estos umbrales se evita que posibles cruces en el eje a causa de la inestabilidad de los sensores o de los movimientos generados por usuario al mantener la IMU en la mano sean detectados como Gestos y provoquen un uso excesivo del sistema sin ningún sentido.

```
Durante (Sistema!=Stop){
  Obtengo Movimiento;
  Calculo_incremento_de_los_Valores;
  Normalizamos_los_valores; //Para que el movimiento empiece y termine por 0
  //Buscando Picos
  Durante (valores) {
    Sumatorio+=valor;
    sumatorio_abs+=absoluto (valor);
    Si (cambio_signo_valores Y ventana==0) {
      //Cambio de Signo
      Si (sumatorio_abs>totAbs){
        guardo_Nuevo_Pico;
        Ventana=valor_ventana_default;
        totSum = suma;
        totAbs = sumaAbs;
      }
      Sino {
        Si (lista_picos>0)
        {
          //Existe algun Pico ya.
          PicoAnterior+=valor;
          totSum = suma;
        totAbs = sumaAbs;
        }
      }
    }
    Sino {
      Si (ventana>0) {
        Ventana--;
      }
    }
    Si (final_lista_valores Y (sumaAbs-totAbs)>Umbral_sumatorioIntegralPico){
      guardo_Nuevo_Pico;
      totSum = suma;
      totAbs = sumaAbs;
    }

  }
  // Checkear los Picos
  Si (totAbs==0) {
    Consistencia=0;
  }
  Sino {
    Consistencia= valor_absoluto (suma/sumaAbs);
  }
  Si (consistencia>Umbral_Slope Y Umbral_sumatorioIntegralMovimientoEntero){
    Durante (Picos) {
      Si (cambio_signo_entre_picos) {
        Si (totAbs=0) {
          Consistencia=0
        }
        Sino {
          consitencia=sumatorio_pico/sumatorio_pico_abs;
        }
        Si (consistencia>umbral_slope) {
          //Nueva Unidad de movimiento
          Añado_UnidadMovimiento_a_gesto_nuevo;
        }
      }
    }
  }
  Si (gesto_nuevo_contiene_UnidadesMovimiento)
  {
    Devolver_gesto_nuevo;
  }
}
```

ILUSTRACIÓN 31: SEUDOCÓDIGO DE DETECCIÓN DE LOS GESTOS

COMPARACIÓN DE GESTOS

Análisis de la Problemática

El último escalón antes de dar el resultado del análisis del Gesto es la comparación entre el gesto nuevo y los guardados en el sistema.

El comparador analiza todos los gestos guardados, buscando las similitudes entre los guardados y el registrado, y devuelve un tanto por ciento de similitud.

El resultado de la similitud es de vital interés ya que el sistema debe poseer un umbral de similitud que puede ser cambiado según el usuario desee, para poder generar grados de dificultad.

Solución

Con cada registro de un nuevo Gesto, el sistema pasa a compararlos con los guardados en la capa de persistencia, las comparaciones se realizan entre pares de gestos dando como resultado un tanto por cien de similitud.

La idea del algoritmo es ir analizando y comparando Unidad por Unidad, buscando si son iguales las unidades. Con cada acierto se pasará a la siguiente unidad, sumando el valor de cada acierto al resultado final.

La Ilustración 33: Seudocódigo de la función de comparación de dos gestos muestra el algoritmo de comparación, el cual va integrado dentro del algoritmo principal y es el encargado de comparar dos gestos y devolver como resultado un porcentaje de similitud.

Finalmente, las unidades que se quedan detrás de alguno de los dos gestos analizados penalizan el resultado final.

El algoritmo de comparación mostrado es el encargado de analizar las aceleraciones. Para los giros y los magnetismos se utilizan versiones de éste mismo pero con algunas modificaciones. Cabe destacar que el algoritmo de comparaciones de los magnetismos no tiene en cuenta si las unidades son iguales simplemente si ambos gestos tienen el mismo número de unidades, esto es así porque utilizar los magnetismo lleva el problema de que siempre se tiene que hacer un proceso de calibración que en este software se ha descartado, para simplificar la puesta a punto.

Seudocódigo

```
Mientras (Sistema!=Stop)
{
  Mientras (Existe_gesto_registrado)
  {
    gesto_reg=Obtener_gesto_registrado;
    ordenar_por_tiempo_las_UnidadesMovimiento;
    while(gestos_guardados)
    {
      similitudAux=compara_gesto_registrado_con_gesto_guardado;
      if(similitudRes<similitudAux)
      {
        similitudRes=similitudAux;
        Nombre=nombre_gesto_guardado;
        Numero=numero_gesto_guardado;
      }
      gestos_guardados++;
    }
    if(similitudRes>=umbral_similitudGesto)
    {
      //Gesto buevo encontrado
      oGesto.SetNombre (Nombre);
      oGesto.SetNumero (Numero);
      devolver_gesto_nuevo;
    }
  }
}
```

ILUSTRACIÓN 32: SEUDOCÓDIGO DE LA COMPARACIÓN DE LOS GESTOS - ALGORITMO PRINCIPAL

```
//ACELERACIONES
Mientras (existe_unidadesMovimiento_guardado)
{
  obtengo_puntero_inicio_unidadesMovimiento_gestoRegistrado;
  avanzo_punteroUnidadesRegistradas_contRegistrado_posiciones;

  Mientras (existe_unidadesMovimiento_registrado)
  {
    Si ( unidad_guardada_igual_unidad_registrada)
    {
      // \brief La unidad es IGUAL en eje y signo. ACIERTO.
      SimilitudRes+=(100.0/numUMovimiento);
      punteroUnidadesRegistradas++;
      contRegistrado++;
      Salimos_del_bucle_secundario;
    }
    punteroUnidadesRegistrado++;
  }
  punteroUnidadesGuardado++;
  contGuardado++;
}
Si (hay_unidades_registradas_no_analizadas)
{
  // \brief Penaliza las unidades del gesto registrado que se han quedado detrás sin
    analizarse. Penalización.
  UnidadesRestantesDeRegistrado= numero_unidadesMovimiento_registrado -contRegistrado;
  SimilitudRes-=((100.0/numUMovimiento)/2)*unidadesRestantesDeRegistrado;
}
Si (hay_unidades_guardadas_no_analizadas)
{
  // \brief Penaliza las unidades del gesto guardado que se han quedado detrás sin
    analizarse. Penalización.
  UnidadesRestantesDeGuardados=numero_unidadesMovimiento_guardado-contGuardado;
  SimilitudRes-=((100.0/numUMovimiento)/2)*unidadesRestantesDeGuardados;
}
```

ILUSTRACIÓN 33: SEUDOCÓDIGO DE LA FUNCIÓN DE COMPARACIÓN DE DOS GESTOS

INTERPOLACIÓN DE LOS GESTOS

Análisis de la Problemática

En la especificación de requisitos se especifica que es necesaria la funcionalidad de *Captura* y *Guardado* de gestos nuevos. El hecho de capturar un gesto nuevo y de insertarlo en el sistema de persistencia es simple y no sería de importancia comentar en éste apartado, sin embargo su funcionalidad depende en gran medida del factor "error humano".

El usuario podría capturar un gesto y insertarlo, pero el problema viene en si el usuario a realizado el gesto exactamente como él deseaba que fuera, o si por el contrario el gesto guardado ha sido sensiblemente distinto a la idea inicial.

Ésta problemática llevo a la implementación de ciertas funcionalidades extras como son que el usuario puede capturar distintas repeticiones del gesto y sacar un gesto medio como resultado. El proceso de obtener el gesto medio ha sido llamado Interpolación.

Solución

Atendiendo al análisis anterior, el objetivo de la interpolación es obtener un gesto nuevo intentando evitar los posibles errores de gestualidad del usuario.

Para ello, el algoritmo de interpolado recorre todos los gestos capturados que residen en la memoria virtual del software, unidad por unidad., comprobando que en esa posición concreta el 70% de los gestos capturados tienen la misma unidad de movimiento. Si no es así, esa unidad será descartada para el nuevo Gesto. De esto se deduce que cuantas más capturas se realicen de un Gesto, más exacto será el gesto guardado en el sistema.

En la Ilustración 34: Seudocódigo Algoritmo Interpolar sólo se muestra la interpolación de las aceleraciones por ser más breves ya que las interpolaciones del resto de tipo de *UnidadMovimiento* el proceso son exactamente igual.

Seudocódigo

```
//  \brief  Obtener interpolación.
Durante (; unidadMov<maxTamListaUnidadMovimientoAcc (_lGestosCapturados); ++unidadMov)
{
  //Se obtiene el tamaño maximo de unidades que podra tener el nuevo gesto.
  //Para cada posicion se buscara lo más repetido.
  ItCapturados=_lGestosCapturados.begin ();
  Mientras (existen_gestos_capturados)
  {
    Si (tamaño_listaAceleraicones_capturado>unidadMov)
    {
      avanzamos_puntero_listaAceleraciones_unidadMov_posiciones
      guardamos_en_listaUM_la_unidad_de_esa_posición;
    }
    avanzamos_posicion_gesto_capturado;
  }
  //Buscar el que más se ha repetido.
  UnidadMovimientoMasRepetida (listaUM, eAxis, integral, cont);
  //VALOR CONSTANTE; TANTO POR CIEN DE SIMILITUD PARA PODER ACEPTAR
  Si ( (cont*100)/_lGestosCapturados.size() >=70 )
  {
    añado_unidad_en_el_nuevo_gesto;
  }
}
//Comprobaciones de Giros y Magnetismos
//Por último se busca que no haya un gesto igual en el sistema de persistencia
```

ILUSTRACIÓN 34: SEUDOCÓDIGO ALGORITMO INTERPOLAR

VELOCIDAD DE CÁLCULO

Análisis de la Problemática

En ésta memoria se ha hecho mucho hincapié en la problemática de la rapidez de respuesta del sistema. Ya en la especificación de requisitos se vio éste factor como una característica capital del software.

Como se comentó, un desfase excesivo en el reconocimiento de los gestos puede llevar a errores y confusiones de los usuarios de las aplicaciones finales. Aparte de quitar cierto atractivo al programa usuario de la librería.

Solución

Para conseguir velocidad de cálculo se realizaron ciertos cambios en la implementación, los cuales se pasarán a comentar:

1. El primer paso fue insertar los algoritmos en distintos hilos de ejecución, consiguiendo que los diversos cálculos pudieran ejecutarse al mismo tiempo, en paralelo. Éste cambio puede parecer simple a simple vista, pero no fue así, de hecho ocasionó tener que:
   1.1. Insertar semáforos de control en cada hilo: Para evitar que estuvieran sobreexplotando los recursos del sistema sin sentido.
   1.2. Insertar colas de objetos: Si ciertos algoritmos funcionan más rápidos que otros, con las colas se evita que la información se pierda o se tenga que detener la ejecución del hilo. Los algoritmos hacen su trabajo y almacenan la información en las colas para seguir con el siguiente cálculo.
2. Se dividieron los algoritmos por eje: Es decir, los algoritmos como los de detección de Gestos hacen cálculos con la información proveniente de los 3

ejes. Este hecho podría provocar tener que iterar 3 veces el mismo cálculo, una para cada eje, lo cual hubiera añadido retrasos considerables en el resultado final. Éste problema se ha resuelto aplicando cada algoritmo en cada eje, teniendo que diseñar un sistema de coordinación para cada algoritmo.

En la Ilustración 35: Seudocódigo Algoritmo de Coordinación de la detección de gestos mostraremos como el algoritmo de detección de Gestos como ejemplo de la aplicación de la solución descrita.

Seudocódigo

```
Mientras  (sistema!=Stop)
{
  Mientras (obtengo_movimiento)
  {
    despierto_hilo_CheckMovGyr;
    despierto_hilo_CheckMovMag;
    // \brief Variables Linea Base
    calculos_normalización_aceleraciones;
    /*//////////////////////////////////////
    *INICIO HILOS DE ACC X
    *Y ESPERO FIN.
    *//////////////////////////////////////
    despierto_hilo_CheckAccX;
    despierto_hilo_CheckAccY;
    despierto_hilo_CheckAccZ;
    Mientras (esperamos_señal_fin_CheckMovGyr
      && esperamos_señal_fin_CheckMovMag
      && esperamos_señal_fin_CheckAccX
      && esperamos_señal_fin_CheckAccY
      && esperamos_señal_fin_CheckAccZ)
    {}
    Si (hay_unidadesMovimiento_aceleraciones && unidadesMovimiento_aceleraciones<numMaxAccels)
    {
      devuelvo_gesto_nuevo;
    }
  }//Mientras
  pCheckMovimiento->notSi yEndCheckMovAcc();
}//Mientras
```

ILUSTRACIÓN 35: SEUDOCÓDIGO ALGORITMO DE COORDINACIÓN DE LA DETECCIÓN DE GESTOS

```
Mientras  (Sistema!=Stop)
{

  espero_a_que_me_despierten;

  //ALGORITMO MOSTRADO EN EL PROBLEMA - DETECCION GESTOS ACELERACIONES X
  calculos_algortmo_detección_gestos;

  enviar_señal_fin_calculo_a_algoritmo_coordinador;

}//Mientras
```

ILUSTRACIÓN 36: SEUDOCÓDIGO DE EJEMPLO DEL CHEQUEADO DE LAS ACCELERACIONES ENEL EJE X

CAPÍTULO 6: RESULTADOS, CONCLUSIONES Y TRABAJOS FUTUROS

INTRODUCCIÓN

Este capítulo expone los resultados obtenidos en diversas pruebas. Posteriormente en conclusiones se lleva a cabo una reflexión sobre los resultados y finalmente se abordan ciertos aspectos del sistema a mejorar o desarrollar en mayor profundidad en la sección de trabajos futuros.

RESULTADOS

La sección de resultados se compone de 4 pruebas realizadas a usuarios con distinto nivel. El primer usuario es un cliente experto, el cual domine los gestos definidos en el sistema. Los dos siguientes han utilizado el software en mayor o menor medida. Para finalizar se presentan los datos de las pruebas realizadas a un nuevo usuario.

Usuario 1 : Con conocimiento de uso						
GESTOS	RESULTADOS (1 - Acierto, 0 - Fallo)					TOTALES
Línea recta a la derecha	1	1	1	1	0	80
Línea recta a la izquierda	1	1	1	0	1	80
Línea recta a arriba	1	1	1	1	1	100
Línea recta a abajo	1	1	1	1	1	100
Loop en sentido horario	1	0	1	1	0	60
Loop en sentido anti horario	1	1	1	1	1	100
La ola hacia la derecha	1	1	1	1	1	100
La ola hacia la izquierda	1	1	1	1	1	100
Lanzar pescado	1	0	0	1	1	60
Letra Z	1	1	1	1	1	100
Loop hacia adelante	0	0	1	1	1	60
TOTALES	90,9	72,72	90,9	90,9	81,81	85,454545
						85,446

Usuario 2 : Con conocimiento de uso bajo.						
GESTOS	RESULTADOS (1 - Acierto, 0 - Fallo)					TOTALES
Línea recta a la derecha	1	1	1	0	1	80
Línea recta a la izquierda	1	1	1	0	1	80
Línea recta a arriba	1	0	1	1	1	80
Línea recta a abajo	0	0	0	1	1	40
Loop en sentido horario	1	0	0	1	1	60
Loop en sentido anti horario	1	0	1	1	1	80
La ola hacia la derecha	0	0	1	0	1	40
La ola hacia la izquierda	1	0	1	1	1	80
Lanzar pescado	0	0	1	1	1	60
Letra Z	0	0	1	0	1	40
Loop hacia adelante	0	0	1	1	0	40
TOTALES	54,54	18,18	81,81	63,63	90,90	61,82
						61,81

Usuario 3 : Con conocimiento de uso medio.						
GESTOS	RESULTADOS (1 - Acierto, 0 - Fallo)					TOTALES
Línea recta a la derecha	0	0	1	0	1	40
Línea recta a la izquierda	1	1	1	0	1	80
Línea recta a arriba	1	0	1	1	1	80
Línea recta a abajo	1	1	0	1	1	80
Loop en sentido horario	1	0	0	1	1	60
Loop en sentido anti horario	1	0	1	1	1	80
La ola hacia la derecha	0	1	0	1	1	60
La ola hacia la izquierda	1	0	1	1	1	80
Lanzar pescado	0	0	1	1	1	60
Letra Z	0	1	1	1	1	80
Loop hacia adelante	0	0	1	1	1	60
TOTALES	54,54	36,36	72,72	81,818	100	69,09
						69,0876

Usuario 4 : Sin conocimiento de uso.						
GESTOS	RESULTADOS (1 - Acierto, 0 - Fallo)					TOTALES
Línea recta a la derecha	0	0	1	0	1	40
Línea recta a la izquierda	1	1	0	1	0	60
Línea recta a arriba	1	0	1	0	1	60
Línea recta a abajo	0	0	0	1	1	40
Loop en sentido horario	1	0	1	1	1	80
Loop en sentido anti horario	0	1	1	0	1	60
La ola hacia la derecha	0	1	0	1	1	60
La ola hacia la izquierda	1	0	1	1	0	60
Lanzar pescado	0	1	1	0	1	60
Letra Z	0	1	1	0	0	40
Loop hacia adelante	0	0	1	1	1	60
TOTALES	36,36	45,45	72,72	54,54	72,72	56,36 56,358

Vamos a pasar a analizar los datos de las diferentes trazas realizadas por distintos usuarios. Se ha decidido hacer los test a 4 usuarios de diferente nivel, dispuestos de más a menos conocimiento. El primer usuario es un experto en el uso del software, los dos siguientes son usuarios casuales y finalmente el último es un usuario nuevo.

El primer hecho que salta a la vista es que cuanto más conocimiento de la gestualidad se tiene, más alto es el porcentaje de aciertos. Esto se debe a que se tiene que tener un tiempo de aprendizaje y adaptación previo para comprender como el sistema pide que se realice el gesto. Es decir, a un nuevo usuario se le dice como realizar el movimiento pero este debe practicar un poco para conseguir un porcentaje de aciertos razonablemente aceptable.

El segundo hecho de interés es relacionado con el anterior. Si se miran con atención los datos anteriores podemos ver una tendencia curiosa pero al mismo tiempo con una fácil explicación. El hecho es que cuando se empiezan a hacer las pruebas, se puede comprobar que en los primeros intentos la tasa de aciertos es bastante menor, en comparación por ejemplo a la columna número cinco de los resultados. Este hecho guarda cierta relación con el anterior y es que cuando se empieza a usar el software siempre se suele necesitar de un intento o dos para recordar cómo se realizaba el gesto, de todas formas también puede depender de la pericia de manejo del usuario.

También cabe destacar los porcentajes totales de acierto. El software y el hardware son bastantes estables y exactos, con lo que la curva de aprendizaje de los gestos es muy corta. Como resultado el software siempre mide los movimientos de igual manera, así con la práctica de 2 o 3 veces cada gesto se puede conseguir resultados por encima del 80 por ciento de aciertos.

CONCLUSIONES

Como resultado del análisis de los valores anteriores, se llega a la conclusión de ciertas ideas que subrayasen a ellos.

La primera de todas es la muy corta curva de aprendizaje que todo usuario ha tenido. Se ha conseguida un Sistema de Reconocimiento Gestual bastante preciso en la lectura de los movimientos humanos, con lo que es muy fácil adaptarse a cómo se debe realizar un movimiento para que el sistema lo detecte como un acierto. Las curvas de aprendizaje se pueden apreciar en los usuarios 2 y 3 pasando de tasas por debajo del 60 por ciento a tasas superiores al 70% en los intentos posteriores.

Siguiendo con la idea anterior, todo usuario con cierto conocimiento necesita, según su pericia, un par de intentos para *recordar* cómo se hacia el gesto. Tras, digamos, ese período de adaptabilidad la tasa de aciertos se suele colocar en niveles superiores al 80 por ciento en la mayoría de los casos. Esta curva de aprendizaje se ha denominado curva de recuerdo.

También cabe destacar los resultados del usuario 1, el cual puede ser ejemplo de cualquier usuario común y reiterativo del sistema. En las pruebas realizadas los resultados nos muestran la alta tasa de aciertos que un usuario puede alcanzar.

Finalmente comento que en general las tasas de aciertos suelen se un poco más bajas por estar mezclando gestos complejos como son dibujar una Z o un *loop*, con gestos más simples como son líneas rectas. Si nos fijamos son estos últimos los que suelen tener un porcentaje de aciertos mayor.

TRABAJOS FUTUROS

A lo largo del desarrollo del software se detectaron ciertos errores de concepto, ciertos planteamientos que no provocan un mal funcionamiento del sistema pero que permiten vislumbrar una mejora a largo plazo del producto.

Son detalles que permitirían mejorar el software pero que no se llevaron a cabo en su momento de detección por ser demasiados costosos de aplicar. Factores como la velocidad de detección o el aumento de aciertos son objetivos del producto que han sido cumplidos en los intervalos propuestos, aunque se sabe y se tiene la idea de que podrían ser mejorados.

Sobre todo el detalle de mejora de velocidad podría ser implementado, aunque supondría un cambio tan grande en distintos niveles del proceso de desarrollo realizado que debería plantearse si de verdad es interesante la profunda transformación, o no es de interés alcanzar una velocidad en la cual no se apreciara aparentemente el cambio entre versiones.

Se analizó como intentar hacer los análisis con mayor rapidez y soltura, como resultado se obtuvo una idea de concepción fácil si se conoce como es el sistema. La idea básica es no esperar a tener los *movimientos* y los *gestos* completos, intentar hacer el análisis final de comparación entre gestos con la primera detección de una *unidad de movimiento*. Es decir, el sistema estaría escaneando los valores en busca de movimientos pero en cuanto detectara algún cambio en los valores, el análisis de gestos y la comparación empezarían en ese mismo instante. Así, hasta ahora era el modo de actuación es un poco diferente pues el software esperaba a tener un *movimiento* completo para analizarlo y convertirlo a un gesto, para finalmente compararlo con los guardados en el sistema. Dicho así, sería hacer el sistema más ágil y más dinámico.

El otro factor a poder mejorar sería en la sección de comparación de gestos. El algoritmo de comparación se podría mejorar aunque tal vez sería más difícil llegar a una mejora, pues el algoritmo funciona bastante bien. Así, se debería de buscar un modelo estadístico que pueda, mediante operaciones de probabilidad llegar a una mejor respuesta.

Ambos campos de mejora son objetivos reales, que pueden ser conseguidos. Simplemente no han sido abordados en el desarrollo por suponer cambios a nivel muy bajo en el diseño. Por cuestiones de costos temporales no han sido aplicados.

VALORACIÓN PERSONAL

El desarrollo del Proyecto Final de Carrera ha sido una experiencia bastante enriquecedora e interesante, tanto a nivel académico como a nivel personal.

A nivel académico es donde más se ha dejado sentir el desarrollo del PFC. En referencia a la implementación de soluciones software he profundizado más en las técnicas de programación y he mejorado el conocimiento de los desarrollos en C++. También se necesitó el aprendizaje, a nivel usuario, del lenguaje LITE-C usado en la implementación de la interfaz gráfica.

Respecto al software comercial utilizado, se ha adquirido conocimientos respectivos a programas como: el 3ds Max y el 3D *GameStudio*. Con el 3DS Max se ha aprendido a modelar y animar objetos virtuales, siempre a un nivel principiante pues las posibilidades del software son muchísimo mejores que lo que se muestra en nuestra interfaz gráfica. Con el ·3D *GameStudio* se diseñó y programó la interfaz gráfica.

A nivel personal el proyecto ha supuesto un enfrentamiento directo con la realidad del mundo empresarial fuera de lo puramente académico pues, ésta ha sido la primera vez que me enfrentaba al desarrollo de una solución a un problema del mundo real con propósitos de investigación o de comercialización. Se ha aprendido a solucionar los problemas por cuenta propia, siempre intentando respetar las premisas estipuladas en el inicio. Y no rehuyendo a buscar soluciones en artículos y libros de alto nivel académico.

También cabe destacar a toda la gente que me ha ayudado, con mucha paciencia, en el desarrollo del sistema.

ANEXOS

MANUAL DE REFERENCIA

INTRODUCCIÓN

En la siguiente sección pasaremos a mostrar el manual de referencia, en el cual se describen todas las funcionalidades de que dispone la librería generada para la interfaz del perro virtual.

Este manual de referencia explica el software en su primera versión. No debería ser usado en posibles versiones siguientes, pues las funcionalidades podrían haber sido ampliadas o, tal vez, puede haber sido suprimida alguna.

La funcionalidades son mostradas a continuación, por cada una se explican su descripción, su sintaxis, los parámetros de entrada y salida, y finalmente su uso en el código.

FUNCIONALIDADES

NOMBRE initSRG

DESCRIPCION	Encargada de inicializar el sistema, siempre con el funcionamiento interno tipo thread. Los dos parámetros que recibe son para elegir el path de los ficheros XML y el modo del sistema, o Reconocimiento Gestual o de Gestión de los Gestos.
SINTAXIS	var initSRG (STRING* path, var mode)
ENTRADAS	El primer parámetro sirve para pasarle la ruta del directorio donde se encuentran los XML. En cambio el segundo es el usado para pasarle el modo de ejecución del sistema, el cero indica que se ejecuta en modo Reconocimiento y si se le pasa un 1 el sistema estará ejecutándose en modo Gestión.
SALIDAS	La salida de la ejecución de la función es un 1 si todo ha ido bién, mientras que un 0 indica un error.
USO	

```
function clickBotonAnalisisGestos(){
        ocultarPanel();
        if(!ini){
                if(initSRG(pathSRG,(var) 0)){
                        mostrarBotonDetener();
                        muevete_pan.flags=SHOW;
                        ini=1;
                        analisisGesto();
                        muevete_pan.flags=~SHOW;
                        if(var_F2==1){
                             mostrarVariables();
                        }
                }
                else{
                        error("Error al iniciar el sistema.");
                        ini=0;
                        endSRG ();
                }
        }
        else{
                ocultarBotones();
```

```
                ini=0;
                endSRG ();
        }
}
```

NOMBRE	endSRG
DESCRIPCION	Función utilizada para detener la ejecución del sistema. No es importante el modo de ejecución en que ha sido ejecutado, simplemente detiene la ejecución.
SINTAXIS	var endSRG ()
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	Devuelve un 0 si todo ha ido bien en la detención, si no ha sido así devolverá un 1.

USO	`function cerrar()`
	`{`
	`ini=0;`
	`endSRG ();`
	`sys_exit(NULL);`
	`}`

NOMBRE	updateGestoActualSRG
DESCRIPCION	Función usada para actualizar el gesto actual del sistema. Para obtener un gesto, primero se tiene que ejecutar este método, luego se deberán hacer los respectivos *gets*.
SINTAXIS	var updateGestoActualSRG()
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	Devuelve un 0 si todo ha ido bien en la actualización, si no ha sido así devolverá un 1.

USO	`while(ini){`
	`if(updateGestoActualSRG()!=0)`
	`{`
	`varNumGesto=getNumeroGestoActualSRG();`
	`str_cpy((Gesto_txt.pstring)[1],"");`
	`str_cpy(strNumGesto,"");`
	`str_cat_num(strNumGesto,"Numero gesto: %.0f",varNumGesto);`
	`str_cpy((Gesto_txt.pstring)[1],strNumGesto);`
	`str_cpy(strNomGesto,"");`
	`str_cpy((informacion_txt.pstring)[0],"");`
	`getNombreGestoActualSRG (strNomGesto);`
	`str_cpy((informacion_txt.pstring)[0],strNomGesto);`
	`varFuerza_media=getFuerzaMediaGestoActualSRG ();`
	`str_cpy(strFuerza_media,"");`
	`str_cpy((informacion_txt.pstring)[1],"");`
	`str_cat_num(strFuerza_media,"Fuerza:%.2f",varFuerza_media);`
	`str_cpy((informacion_txt.pstring)[1],strFuerza_media);`
	`movimientoGesto(varNumGesto,varFuerza_media,cuerpo_ent);`
	`}`
	`wait(1);`
	`}`

NOMBRE	getNombreGestoActualSRG
DESCRIPCION	Función encargada de obtener el nombre del gesto actual. Es de interés recordar que siempre debe ir precedida del *updateGestoActualSRG*.
SINTAXIS	void getNombreGestoActualSRG (STRING& nombre)
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	Se devuelve por parámetro un STRING, el cual contiene el nombre del último gesto realizado.

USO	`while(ini){`
	`  if(updateGestoActualSRG()!=0)`
	`  {`
	`  str_cpy((informacion_txt.pstring)[0],"");`
	`  getNombreGestoActualSRG (strNomGesto);`
	`  str_cpy((informacion_txt.pstring)[0],strNomGesto`
	`  (..)`
	`  movimientoGesto(varNumGesto,varFuerza_media,cuerpo_ent);`
	`  }`
	`  wait(1);`
	`}`

NOMBRE	getNumeroGestoActualSRG
DESCRIPCION	Función encargada de obtener el número del gesto actual. Es de interés recordar que siempre debe ir precedida del *updateGestoActualSRG*.
SINTAXIS	var getNumeroGestoActualSRG ()
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	Devuelve un tipo VAR, el cual contiene un entero que representa el último gesto realizado. Es cero si no hay ninguno nuevo.

USO	`while(ini){`
	`  if(updateGestoActualSRG()!=0){`
	`  varNumGesto=getNumeroGestoActualSRG();`
	`  str_cpy((Gesto_txt.pstring)[1],"");`
	`  str_cpy(strNumGesto,"");`
	`  str_cat_num(strNumGesto,"Numero`
	`  gesto:%.0f",varNumGesto);`
	`  str_cpy((Gesto_txt.pstring)[1],strNumGesto);`
	`  (…)`
	`  movimientoGesto(varNumGesto,varFuerza_media,cuerpo_ent);`
	`  }`
	`  wait(1);`
	`}`

NOMBRE	getFuerzaMediaGestoActualSRG
DESCRIPCION	Función encargada de obtener la fuerza media del gesto actual. La fuerza media es la media de las aceleraciones registradas en el gesto a lo largo de los tres ejes de aceleración. Es de interés recordar que siempre debe ir precedida del *updateGestoActualSRG*.
SINTAXIS	var getFuerzaMediaGestoActualSRG ()
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	Devuelve un tipo VAR, que contiene un entero.

USO	`while(ini){`
	`  if(updateGestoActualSRG()!=0){`
	`  (…)`
	`  varFuerza_media=getFuerzaMediaGestoActualSRG ();`
	`  str_cpy(strFuerza_media,"");`
	`  str_cpy((informacion_txt.pstring)[1],"");`
	`  str_cat_num(strFuerza_media,"Fuerza:%.2f",varFuerza_media);`
	`  str_cpy((informacion_txt.pstring)[1],strFuerza_media);`
	`  movimientoGesto(varNumGesto,varFuerza_media,cuerpo_ent);`
	`  }`
	`  wait(1);`
	`}`

NOMBRE	updatePuntoActualSRG
DESCRIPCION	Función usada para actualizar el punto actual del sistema. Para obtener un punto, primero se tiene que ejecutar este método, luego se deberán hacer los respectivos *gets*.
SINTAXIS	void updatePuntoActualSRG()
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	No devuelve ningún valor.

USO	`function mostrarVariables()`
	`{`
	`  STRING* strValue="";`
	`  var value=0;`
	
	`  Quaternions_txt.flags=SHOW;`
	`  Magnetometer_txt.flags=SHOW;`
	`  Gyroscope_txt.flags=SHOW;`
	`  Accelerometer_txt.flags=SHOW;`
	`  Gesto_txt.flags=SHOW;`
	`  informacion_txt.flags=SHOW;`
	
	`  while(ini && var_F2){`
	`  if(updatePuntoActualSRG()){`
	`  //Aceleracion`
	`  value=getAccelerometerXSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Accelerometer_txt.pstring)[1],strValue);`
	`  value=getAccelerometerYSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Accelerometer_txt.pstring)[2],strValue);`
	`  value=getAccelerometerZSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Accelerometer_txt.pstring)[3],strValue);`
	
	`  //Giro`
	`  value=getGyroscopeXSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Gyroscope_txt.pstring)[1],strValue);`
	`  value=getGyroscopeYSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Gyroscope_txt.pstring)[2],strValue);`
	`  value=getGyroscopeZSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Gyroscope_txt.pstring)[3],strValue);`

```
                        //Magnetometro
                        value=getMagnetometerXSRG();
                        str_for_num(strValue,value);
                        str_cpy((Magnetometer_txt.pstring)[1],strValue);
                        value=getMagnetometerYSRG();
                        str_for_num(strValue,value);
                        str_cpy((Magnetometer_txt.pstring)[2],strValue);
                        value=getMagnetometerZSRG();
                        str_for_num(strValue,value);
                        str_cpy((Magnetometer_txt.pstring)[3],strValue);

                        //Quaternion
                        value=getQuaternionXSRG();
                        str_for_num(strValue,value);
                        str_cpy((Quaternions_txt.pstring)[1],strValue);
                        value=getQuaternionYSRG();
                        str_for_num(strValue,value);
                        str_cpy((Quaternions_txt.pstring)[2],strValue);
                        value=getQuaternionZSRG();
                        str_for_num(strValue,value);
                        str_cpy((Quaternions_txt.pstring)[3],strValue);
                        value=getQuaternionWSRG();
                        str_for_num(strValue,value);
                        str_cpy((Quaternions_txt.pstring)[4],strValue);
                }
                wait(1);
        }
}
```

NOMBRE	getAccelerometerXSRG
DESCRIPCION	Función encargada de obtener el valor actual de la aceleración en el eje X. Cabe destacar que si se quiere obtener el último punto registrado por el dispositivo se debe ejecutar *updatePuntoActualSRG* con anterioridad.
SINTAXIS	var getAccelerometerXSRG ()
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	Devuelve una variable tipo VAR, la cual representa un número de coma flotante.

USO	`while(ini && var_F2){`
	`  if(updatePuntoActualSRG()){`
	`  //Aceleracion`
	`  value=getAccelerometerXSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Accelerometer_txt.pstring)[1],strValue);`
	`  value=getAccelerometerYSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Accelerometer_txt.pstring)[2],strValue);`
	`  value=getAccelerometerZSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Accelerometer_txt.pstring)[3],strValue)`
	`  }`
	`  wait(1);`
	`}`

NOMBRE	getAccelerometerYSRG
DESCRIPCION	Función encargada de obtener el valor actual de la aceleración en el eje Y. Cabe destacar que si se quiere obtener el último punto registrado por el dispositivo se debe ejecutar *updatePuntoActualSRG* con anterioridad.
SINTAXIS	var getAccelerometerYSRG ()
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	Devuelve una variable tipo VAR, la cual representa un número de

	coma flotante.

USO	`while(ini && var_F2){`
	`  if(updatePuntoActualSRG()){`
	`  //Aceleracion`
	`  value=getAccelerometerXSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Accelerometer_txt.pstring)[1],strValue);`
	`  value=getAccelerometerYSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Accelerometer_txt.pstring)[2],strValue);`
	`  value=getAccelerometerZSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Accelerometer_txt.pstring)[3],strValue)`
	`  }`
	`  wait(1);`
	`}`

NOMBRE	getAccelerometerZSRG
DESCRIPCION	Función encargada de obtener el valor actual de la aceleración en el eje Z. Cabe destacar que si se quiere obtener el último punto registrado por el dispositivo se debe ejecutar *updatePuntoActualSRG* con anterioridad.
SINTAXIS	var getAccelerometerZSRG ()
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	Devuelve una variable tipo VAR, la cual representa un número de coma flotante.

USO	`while(ini && var_F2){`
	`  if(updatePuntoActualSRG()){`
	`  //Aceleracion`
	`  value=getAccelerometerXSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Accelerometer_txt.pstring)[1],strValue);`
	`  value=getAccelerometerYSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Accelerometer_txt.pstring)[2],strValue);`
	`  value=getAccelerometerZSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Accelerometer_txt.pstring)[3],strValue)`
	`  }`
	`  wait(1);`
	`}`

NOMBRE	getGyroscopeXSRG
DESCRIPCION	Función encargada de obtener el valor actual del giro en el eje X. Cabe destacar que si se quiere obtener el último punto registrado por el dispositivo se debe ejecutar *updatePuntoActualSRG* con anterioridad.
SINTAXIS	var getGyroscopeXSRG ()
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	Devuelve una variable tipo VAR, la cual representa un número de coma flotante.

USO	`while(ini && var_F2){`
	`  if(updatePuntoActualSRG()){`
	`  //Giro`
	`  value=getGyroscopeXSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Gyroscope_txt.pstring)[1],strValue);`
	`  value=getGyroscopeYSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Gyroscope_txt.pstring)[2],strValue);`
	`  value=getGyroscopeZSRG();`

```
                str_for_num(strValue,value);
                str_cpy((Gyroscope_txt.pstring)[3],strValue)

        }
        wait(1);
}
```

NOMBRE	getGyroscopeYSRG
DESCRIPCION	Función encargada de obtener el valor actual del giro en el eje Y. Cabe destacar que si se quiere obtener el último punto registrado por el dispositivo se debe ejecutar *updatePuntoActualSRG* con anterioridad.
SINTAXIS	var getGyroscopeYSRG ()
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	Devuelve una variable tipo VAR, la cual representa un número de coma flotante.
USO	

```
while(ini && var_F2){
        if(updatePuntoActualSRG()){
                //Giro
                value=getGyroscopeXSRG();
                str_for_num(strValue,value);
                str_cpy((Gyroscope_txt.pstring)[1],strValue);
                value=getGyroscopeYSRG();
                str_for_num(strValue,value);
                str_cpy((Gyroscope_txt.pstring)[2],strValue);
                value=getGyroscopeZSRG();
                str_for_num(strValue,value);
                str_cpy((Gyroscope_txt.pstring)[3],strValue)

        }
        wait(1);
}
```

NOMBRE	getGyroscopeZSRG
DESCRIPCION	Función encargada de obtener el valor actual del giro en el eje Z. Cabe destacar que si se quiere obtener el último punto registrado por el dispositivo se debe ejecutar *updatePuntoActualSRG* con anterioridad.
SINTAXIS	var getGyroscopeZSRG ()
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	Devuelve una variable tipo VAR, la cual representa un número de coma flotante.
USO	

```
while(ini && var_F2){
        if(updatePuntoActualSRG()){
                //Giro
                value=getGyroscopeXSRG();
                str_for_num(strValue,value);
                str_cpy((Gyroscope_txt.pstring)[1],strValue);
                value=getGyroscopeYSRG();
                str_for_num(strValue,value);
                str_cpy((Gyroscope_txt.pstring)[2],strValue);
                value=getGyroscopeZSRG();
                str_for_num(strValue,value);
                str_cpy((Gyroscope_txt.pstring)[3],strValue)

        }
        wait(1);
}
```

NOMBRE	getMagnetometerXSRG
DESCRIPCION	Función encargada de obtener el valor actual del magnetismo en el eje X. Cabe destacar que si se quiere obtener el último punto registrado por el dispositivo se debe ejecutar *updatePuntoActualSRG* con anterioridad.
SINTAXIS	var getMagnetometerXSRG ()
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	Devuelve una variable tipo VAR, la cual representa un número de coma flotante.

USO	`while(ini && var_F2){`
	`  if(updatePuntoActualSRG()){`
	`  //Magnetometro`
	`  value=getMagnetometerXSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Magnetometer_txt.pstring)[1],strValue);`
	`  value=getMagnetometerYSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Magnetometer_txt.pstring)[2],strValue);`
	`  value=getMagnetometerZSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Magnetometer_txt.pstring)[3],strValue);`
	`  }`
	`  wait(1);`
	`}`

NOMBRE	getMagnetometerYSRG
DESCRIPCION	Función encargada de obtener el valor actual del magnetismo en el eje Y. Cabe destacar que si se quiere obtener el último punto registrado por el dispositivo se debe ejecutar *updatePuntoActualSRG* con anterioridad.
SINTAXIS	var getMagnetometerYSRG ()
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	Devuelve una variable tipo VAR, la cual representa un número de coma flotante.

USO	`while(ini && var_F2){`
	`  if(updatePuntoActualSRG()){`
	`  //Magnetometro`
	`  value=getMagnetometerXSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Magnetometer_txt.pstring)[1],strValue);`
	`  value=getMagnetometerYSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Magnetometer_txt.pstring)[2],strValue);`
	`  value=getMagnetometerZSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Magnetometer_txt.pstring)[3],strValue);`
	`  }`
	`  wait(1);`
	`}`

NOMBRE	getMagnetometerZSRG
DESCRIPCION	Función encargada de obtener el valor actual del magnetismo en el eje Z. Cabe destacar que si se quiere obtener el último punto registrado por el dispositivo se debe ejecutar *updatePuntoActualSRG* con anterioridad.
SINTAXIS	var getMagnetometerZSRG ()
ENTRADAS	No se le pasa ningún parámetro.

SALIDAS	Devuelve una variable tipo VAR, la cual representa un número de coma flotante.

USO	`while(ini && var_F2){`
	`  if(updatePuntoActualSRG()){`
	`  //Magnetometro`
	`  value=getMagnetometerXSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Magnetometer_txt.pstring)[1],strValue);`
	`  value=getMagnetometerYSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Magnetometer_txt.pstring)[2],strValue);`
	`  value=getMagnetometerZSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Magnetometer_txt.pstring)[3],strValue);`
	`  }`
	`  wait(1);`
	`}`

NOMBRE	getQuaternionXSRG
DESCRIPCION	Función encargada de obtener el valor actual del quaternion X. Cabe destacar que si se quiere obtener el último punto registrado por el dispositivo se debe ejecutar *updatePuntoActualSRG* con anterioridad.
SINTAXIS	var getQuaternionXSRG ()
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	Devuelve una variable tipo VAR, la cual representa un número de coma flotante.

USO	`while(ini && var_F2){`
	`  if(updatePuntoActualSRG()){`
	`  //Quaternion`
	`  value=getQuaternionXSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Quaternions_txt.pstring)[1],strValue);`
	`  value=getQuaternionYSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Quaternions_txt.pstring)[2],strValue);`
	`  value=getQuaternionZSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Quaternions_txt.pstring)[3],strValue);`
	`  value=getQuaternionWSRG();`
	`  str_for_num(strValue,value);`
	`  str_cpy((Quaternions_txt.pstring)[4],strValue);`
	`  }`
	`  wait(1);`
	`}`

NOMBRE	getQuaternionYSRG
DESCRIPCION	Función encargada de obtener el valor actual del quaternion Y. Cabe destacar que si se quiere obtener el último punto registrado por el dispositivo se debe ejecutar *updatePuntoActualSRG* con anterioridad.
SINTAXIS	var getQuaternionYSRG ()
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	Devuelve una variable tipo VAR, la cual representa un número de coma flotante.

USO	`while(ini && var_F2){`
	`  if(updatePuntoActualSRG()){`
	`  //Quaternion`
	`  value=getQuaternionXSRG();`
	`  str_for_num(strValue,value);`

```
                str_cpy((Quaternions_txt.pstring)[1],strValue);
                value=getQuaternionYSRG();
                str_for_num(strValue,value);
                str_cpy((Quaternions_txt.pstring)[2],strValue);
                value=getQuaternionZSRG();
                str_for_num(strValue,value);
                str_cpy((Quaternions_txt.pstring)[3],strValue);
                value=getQuaternionWSRG();
                str_for_num(strValue,value);
                str_cpy((Quaternions_txt.pstring)[4],strValue);
        }
        wait(1);
}
```

NOMBRE	getQuaternionZSRG
DESCRIPCION	Función encargada de obtener el valor actual del quaternion Z. Cabe destacar que si se quiere obtener el último punto registrado por el dispositivo se debe ejecutar *updatePuntoActualSRG* con anterioridad.
SINTAXIS	var getQuaternionZSRG ()
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	Devuelve una variable tipo VAR, la cual representa un número de coma flotante.
USO	

```
while(ini && var_F2){
        if(updatePuntoActualSRG()){
                //Quaternion
                value=getQuaternionXSRG();
                str_for_num(strValue,value);
                str_cpy((Quaternions_txt.pstring)[1],strValue);
                value=getQuaternionYSRG();
                str_for_num(strValue,value);
                str_cpy((Quaternions_txt.pstring)[2],strValue);
                value=getQuaternionZSRG();
                str_for_num(strValue,value);
                str_cpy((Quaternions_txt.pstring)[3],strValue);
                value=getQuaternionWSRG();
                str_for_num(strValue,value);
                str_cpy((Quaternions_txt.pstring)[4],strValue);
        }
        wait(1);
}
```

NOMBRE	getQuaternionWSRG
DESCRIPCION	Función encargada de obtener el valor actual del quaternion W. Cabe destacar que si se quiere obtener el último punto registrado por el dispositivo se debe ejecutar *updatePuntoActualSRG* con anterioridad.
SINTAXIS	var getQuaternionWSRG ()
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	Devuelve una variable tipo VAR, la cual representa un número de coma flotante.
USO	

```
while(ini && var_F2){
        if(updatePuntoActualSRG()){
                //Quaternion
                value=getQuaternionXSRG();
                str_for_num(strValue,value);
                str_cpy((Quaternions_txt.pstring)[1],strValue);
                value=getQuaternionYSRG();
                str_for_num(strValue,value);
                str_cpy((Quaternions_txt.pstring)[2],strValue);
                value=getQuaternionZSRG();
```

```
                str_for_num(strValue,value);
                str_cpy((Quaternions_txt.pstring)[3],strValue);
                value=getQuaternionWSRG();
                str_for_num(strValue,value);
                str_cpy((Quaternions_txt.pstring)[4],strValue);
        }
        wait(1);
}
```

NOMBRE	capturaGesto
DESCRIPCION	Función encargada de capturar un gesto. Con su ejecución, el sistema espera a la detección de un nuevo gesto, con el registro de uno nuevo, el sistema pasa a almacenarlo a la espera de la interpolación.
SINTAXIS	var capturaGesto ()
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	Devuelve un 0 si todo ha ido bien en la actualización, si no ha sido así devolverá un 1.

USO	`function clickBotonCaptura()`
	`{`
	`  printf("Se va a proceder a la captura.\nMuevete!");`
	`  wait(-1);`
	`  if(capturaGesto())`
	`  {`
	`  printf("Capturado!");`
	`  }`
	`}`

NOMBRE	guardarGesto
DESCRIPCION	Función encargada de guardar un gesto. Primero interpola todos los capturados, la cual sera insertada en el fichero de gestos.
SINTAXIS	var guardarGesto (STRING* nombre)
ENTRADAS	Se le pasa un puntero a una estructura de tipo STRING que contiene una secuencia de caracteres, los cuales representan el nombre del nuevo gesto a guardar. Esta función recoge todos los gestos capturados, los interpola y finalmente los guarda en el XML apropiado.
SALIDAS	Devuelve un Var, que representa un entero: True si todo bien, res devuelve: +1 correcto. False si error, res devuelve: -1 si la interpolación no da nada como resultado. -2 si la interpolación da como resultado un gesto igual a los guardados.

USO	`function clickBotonGuardar()`
	`{`
	`  str_cat(nombreNuevoGesto_str,"");`
	`  STRING* my_str=" ";`
	`  STRING* temp_str="";`
	`  while(key_lastpressed!=28){`
	`  while (key_any == 0) {`
	`  draw_text("Nombre del gesto:`
	`  ",100,300,vector(0,0,0));`
	`  draw_text(my_str,245,300,vector(0,0,0));`
	`  wait(1);`
	`  }`
	`  str_for_key(temp_str,key_lastpressed);`
	`  if(key_lastpressed==57)//si espacio{`

```
                        str_cat(my_str," ");
                }
                else
                {
                        if((key_lastpressed>=16 && key_lastpressed<=25)
                        || (key_lastpressed>=30 && key_lastpressed<=38)
                        || (key_lastpressed>=44 &&
                        key_lastpressed<=50)){
                                if(str_len(my_str)==30){
                                        printf("No puedes escribir
más.");
                                }
                                else
                                {
                                        str_cat(my_str,temp_str);
                                }
                        }
                        else
                        {

                                if(key_pressed(key_for_str("BkSp"))==1){
                                        str_trunc(my_str,1);
                                }
                        }
                }
                while (key_any == 1){
                        draw_text("Nombre del
                        gesto:",100,300,vector(0,0,0));
                        draw_text(my_str,245,300,vector(0,0,0));
                        wait(1);
                }
                wait(1);
        }
        str_cat(nombreNuevoGesto_str,my_str);
        var res=guardarGesto(nombreNuevoGesto_str);
        if(res==1)
        {
                printf("Guardado.");
        }
        else
        {
                printf("Error guardando. error: %i",(int)res);
        }
}
```

NOMBRE	reStartCapturaGesto
DESCRIPCION	Función encargada de borrar todas las capturas y volver a empezar el proceso de captura de un nuevo gesto.
SINTAXIS	void reStartCapturaGesto()
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	No devuelve nada, es tipo VOID.
USO	`reStartCapturaGesto ();`

NOMBRE	setUmbralHigh
DESCRIPCION	Función encargada de modificar el umbral High en ese instante. El umbral high es usado como el umbral de fuerza que debe sobrepasar un movimiento para considerar que un movimiento ha empezado.
SINTAXIS	void setUmbralHigh(float umbralHigh)
ENTRADAS	Se le pasa un integer que representa un entero.
SALIDAS	No devuelve nada, es tipo VOID.
USO	`setUmbralHigh(1.5);`

NOMBRE	setUmbralLow
DESCRIPCION	Función encargada de modificar el umbral Low en esa ejecución en particular. El umbral Low es usado como el umbral de fuerza minima que debe sobrepasar una movimiento para considerar que un movimiento ha terminado.
SINTAXIS	void setUmbralLow(float umbralLow)
ENTRADAS	Se le pasa un integer que representa un entero.
SALIDAS	No devuelve nada, es tipo VOID.
USO	`setUmbralLow(0.65);`

NOMBRE	setNumMaxAccels
DESCRIPCION	Función encargada de modificar el número máximo de unidades de aceleración que el sistema deberá recoger para pasar a realizar los análisis pertinentes.
SINTAXIS	void setNumMaxAccels(int numero)
ENTRADAS	Se le pasa un integer que representa un entero.
SALIDAS	No devuelve nada, es tipo VOID.
USO	`setNumMaxAccels(10);`

NOMBRE	setSimilitudGesto
DESCRIPCION	Función encargada de modificar el tanto por cien de similitud usado en las comparaciones de los gestos.
SINTAXIS	void setSimilitudGesto(int similitud)
ENTRADAS	Se le pasa un integer que representa un entero.
SALIDAS	No devuelve nada, es tipo VOID.
USO	`setSimilitudGesto(80);`

NOMBRE	desconectarGestoParaAnalisisGestos
DESCRIPCION	Función encargada de desconectar un gesto para que no sea usado en las comparaciones en ese ejecución en concreto.
SINTAXIS	bool desconectarGestoParaAnalisisGestos(int num)
ENTRADAS	Se le pasa un integer que representa un entero.
SALIDAS	No devuelve 1 si todo bien, o si se detecta algún error.
USO	`desconectarGestoParaAnalisisGestos(4);`

NOMBRE	borrarGestoPersistente
DESCRIPCION	Función encargada de borrar un gesto del fichero Gestos.xml.
SINTAXIS	Bool borrarGestoPersistente(int num)
ENTRADAS	Se le pasa un integer que representa un entero. El cual es el numero de gesto.
SALIDAS	No devuelve nada, es tipo VOID.
USO	`borrarGestoPersistente(1);`

NOMBRE	toStringGestosGuardados
DESCRIPCION	Función encargada de obtener un string que contiene todos los gestos guardados en el sistema.
SINTAXIS	std::string toStringGestosGuardados()
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	Devuelve un String.

USO	`function getListaString()`
	`{`
	`  Return toStringGestosGuardados ();`
	`}`

NOMBRE	cargarGestosDefault
DESCRIPCION	Función encargada de resetear el fichero Gestos.xml y dejarlo en su estado inicial.
SINTAXIS	bool cargarGestosDefault()
ENTRADAS	No se le pasa ningún parámetro.
SALIDAS	No devuelve 1 si todo bien, o si se detecta algún error.

USO	`function resetarSistema()`
	`{`
	`  Return cargarGestosDefault ();`
	`}`

XML

CONF.XML

Este fichero contiene todos los valores de configuraciones del software. La mayoría de estos parámetros son usados en los algoritmos de detección de gestos y de movimientos.

```
<CONFIGURACION>
        <!-- Tamanyo del buffer de valores -->
        <MUESTRA value='500' />
        <!-- Tamanyo ventana calculo Varianza -->
        <VENTANA value='5' />
        <!-- Tamanyo analisis calculo Varianza -->
        <ANALISIS value='60' />
        <!-- Umbrales Varianza en deteccion movimientos -->
        <UMBRALES>
        <UMBRAL tipo='High' value='2.6' />
        <UMBRAL tipo='Low' value='1.4' />
        </UMBRALES>
        <!-- Para el chequeo de movimientos -->
        <INTEGRALES>
                <INTEGRAL tipo='MovimientoAbsAcc' value="20" />
                <INTEGRAL tipo='PicoSumAcc' value='14' />
                <INTEGRAL tipo='MovimientoAbsGyr' value="10" />
                <INTEGRAL tipo='PicoSumGyr' value='4' />
                <INTEGRAL tipo='MovimientoAbsMag' value="290" />
                <INTEGRAL tipo='PicoSumMag' value='250' />
        </INTEGRALES>
        <!-- Slope para el calculo de integrales -->
        <SLOPE value='2'></SLOPE>
        <!-- Ventana minima para la busqueda picos -->
        <VENTANAPICOS value='3' />
        <!-- Umbral de similitud. -->
        <SIMILITUDGESTO value='85' />
        <!-- Numero maximo de aceleraciones que forman un gesto. -
->
        <NUMMAXACCELS value='7' />
</CONFIGURACION>
```

ILUSTRACIÓN 37: FICHERO CONF.XML

GESTOS.XML

En el siguiente código puede ver un ejemplo de un Gesto guardado en el Gestos.XML. Toda la estructura del Objeto Gesto queda representada en el XML para que perdure entre distintas ejecuciones. En la memoria se ha comentado que un Gesto está compuesto por *UnidadesMovimiento*.

```
<GESTOS>
  <GESTO numero="8" nombre="Lanzar Pescado">
      <UNIDADMOVIMIENTO tInicio="0" tEnd="0" totalAbsIntegral="0" totalIntegral="1"
        consistencia="0" eAxis="2" tipo="Aceleracion" />
      <UNIDADMOVIMIENTO tInicio="0" tEnd="0" totalAbsIntegral="0" totalIntegral="-1"
        consistencia="0" eAxis="1" tipo="Aceleracion"/>
      <UNIDADMOVIMIENTO tInicio="0" tEnd="0" totalAbsIntegral="0"totalIntegral="-1"
        consistencia="0" eAxis="0" tipo="Giro" />
      <UNIDADMOVIMIENTO tInicio="0" tEnd="0" totalAbsIntegral="0" totalIntegral="-1"
        consistencia="0" eAxis="2" tipo="Magnetismo" />
  </GESTO>
</GESTOS>
```

ILUSTRACIÓN 38: EJEMPLO GESTO GUARDADO EN GESTOS.XML

TABLA DE GESTOS DEL SISTEMA

	Aceleraciones							Giros					Magnetismos		
	1	2	3	4	5	6	7	1	2	3	4	5	1	2	3
Horizontal: Derecha	1														
Horizontal: Izquierda	-1														
Horizontal: Adelante	-0														
Horizontal: Atrás	0														
Vertical: Arriba	-2														
Vertical: Abajo	2														
Lanzar piedra	2												-2		
Letra Z	1	-1	1					2	0	-2					
Saludo	1	-2	2	-1				2	0				-1	1	
Plano Y-Z, sentido horario	2	-1	-2	1				2	-1	-2					
Plano Y-Z, sentido anti horario	2	1	-2	-1				-2	-1	2					
La ola hacia la derecha	-2	1	2	-1	-2	1	2	1	-1	1			-0		
La ola hacia la izquierda	-2	-1	2	1	-2	-1	2	1	-1	1			0		
Plano X-Y, sentido anti horario.	0	1	-0	-1				-2							

REFERENCIAS Y BIBLIOGRAFÍA

[DGSW90] IEEE STD 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.

[RPSW98] IEEE STD 830-1998, IEEE Recommended Practice for Software Requirements Specifications.

[IPI03] Ingeniería de proyectos informáticos: Actividades y Procedimientos. Ediciones Universitas.

[ISW05] Ingeniería del Software, Ian Sommerville.

[ESS03] Especificación de sistemas software en UML, UPC03

[AYB00] An Inertial Measurement Unit for User Interfaces, MIT 2000

[EDI95] Estadística Descriptiva e Inferencial, Antonio Vargas Sabadía.

[RAE10] RAE, diccionario de la real academia de lengua española.

[IBER] PCE-IBERICA , instrumentos de medida.

Technaid, compañía que se dedica a las tecnologías orientas al ser humano.

TinyXML, librería de gestión de XML.