Universidad Zaragoza
1542

Trabajo Fin de Grado

Tecnologías extendidas: integración de sensores corporales e información cinestésica

Extended Technologies: integration of embodied and kinesthetic information

Autor

Juan Carlos Chamorro Aranda

Director

Manuel González Bedia

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2017

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

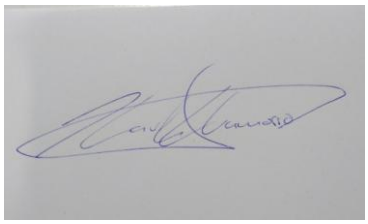
D./Dª. Juan Carlos Chamorro Aranda,

con nº de DNI 18458191R en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Tecnologías extendidas: Integración de Sensores corporales e información cinestésica,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 22 de septiembre de 2017



Fdo: Juan Carlos Chamorro Aranda

Tecnologías extendidas: integración de sensores corporales e información cinestésica

RESUMEN

La tecnología no es algo ajeno a nuestra naturaleza, nos ha acompañado desde el descubrimiento del fuego y la invención de la rueda hasta nuestros días y aunque su propósito principal es facilitarrnos las tareas cotidianas, también podemos usarla para nuestra diversión y entretenimiento. Los artefactos técnicos pueden ser utilizados como medios para la experimentación en el arte y la danza.

El objetivo de este proyecto es el desarrollo de la pieza de baile 'Pulse', presentada en el festival de Danza Trayectos y desarrollada en el Laboratorio de Danza y Nuevos Medios de Etopía en colaboración con el ISAAC[1] Lab. Para esta creación coreográfica, el equipo de ingenieros ha integrado algunos sofisticados dispositivos que han sido acoplados a los bailarines con el objetivo de permitirles bailar con ellos para desarrollar creaciones artísticas asociadas al movimiento y que así puedan mostrarnos desde una perspectiva distinta el arte de bailar.

Los sensores electrónicos han sido ajustados a través de complejos algoritmos que facilitan la comunicación entre los distintos dispositivos y ordenadores que componen este entorno con el propósito de que los bailarines puedan representar sus movimientos de una forma distinta, una forma más plástica, más visual, una forma diferente.

En cada una de las paredes que rodean la escena se proyectan en tiempo real imágenes que llevan el pulso de su ritmo en forma de bits y la velocidad de sus movimientos a modo de señales electrónicas.

La tarea no ha sido sencilla pues aún no se ha inventado un dispositivo que sienta la música como nosotros. Es por esto que el trabajo en equipo ha sido de suma importancia para integrar los conocimientos técnicos de ingenieros, quienes hemos programado los diversos algoritmos, con las habilidades artísticas de nuestro equipo de bailarines, quienes se han encargado de orientar las imágenes proyectadas al conjunto de emociones que quieren representar.

El trabajo de bailarines e ingenieros ha permitido integrar dos visiones: una más racional del espacio basado en coordenadas, con una noción del tiempo definida por el reloj de un ordenador con otra centrada en los cuerpos al danzar, en la que los tiempos se basan en el ritmo marcado de un compás.

El esfuerzo del equipo ha tenido buenos resultados con lo que el "cuerpo en movimiento" se puede ver como una herramienta de creación plástica y que permitirá llegar a nuestros sentidos.

[1] Grupo de investigación del Instituto de Investigación en Ingeniería de Aragón de la Universidad de Zaragoza.

Agradecimientos

En primer lugar quiero agradecer a todas aquellas personas que tanto directa como indirectamente me han apoyado durante el transcurso de la carrera. También quiero agradecer a mis tutores del proyecto, Tomás y Manuel, su ayuda y dedicación, ya que he aprendido muchísimo de ellos. No puedo dejar de agradecer a mi compañero Cesar, el cual me ha enseñado tanto y ha sido una pieza fundamental en el desarrollo del trabajo.

Especialmente quiero agradecer a mis padres Juan Carlos y Nieves, por apoyarme en todo momento, ya que en definitiva son artífices de lo que soy hoy en día y por enseñarme tanto. A mi hermana Marina, la más pequeña de la casa y a veces la más madura. Al resto de mi familia que me ayudo tanto al llegar a una ciudad nueva y siempre han estado para todo. En especial a mis tatitos que siempre han encontrado esas palabras de ánimo cuando más las necesitaba y han sido y son, uno de mis principales apoyos.

Me gustaría agradecer también a mis amigos del Carmelo donde pase dos años increíbles, a los amigos de BBQ que hicisteis que me sintiera como en casa desde el primer momento y a mis fratellos de Antonio Bordoni por hacerme ver la vida de otra manera.

Una etapa que concluye y otra que empieza, pero siempre sin pausa.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Objetivo y alcance del proyecto

El **objeto** del presente proyecto es el de unir tecnología, arte y movimiento. Esto se tratará de conseguir mediante la introducción de artefactos técnicos que permitan mostrar una perspectiva distinta del arte de la danza.

La **idea** fundamental consiste en la introducción, comprensión y optimización de diferentes sensores corporales en un entorno artístico, con la intención de acercarnos lo más posible a una hibridación entre arte y tecnología, que nos permita avanzar en el camino del diseño cyborg y, a su vez, nos permita también explorar y experimentar nuevas posibilidades cognitivas y sensoriales.

La **línea de trabajo** a seguir ha sido la del estudio en primer lugar de las distintas tecnologías que se tienen a disposición, para luego poder analizar los pros y las contras, así como plantear cuales podrían ser sus posibles utilidades en el entorno que nos concierne, tratando de intervenir lo menos posible en el desarrollo artístico por parte de los bailarines. Una vez decidido que sensores se van a utilizar, se ha procedido a realizar todos los pasos necesarios para su correcta incorporación, como pueden ser estudios estadísticos de los datos que con ellos se obtienen, desarrollo de procesos de lectura de esos datos para que nos sean de utilidad o implementación de algoritmos para normalizar, filtrar o interpolar esos mismos datos para que se ajusten a nuestras necesidades.

1.2. Contexto en el que se realiza el proyecto

El proyecto se desarrolla en un entorno colaborativo entre la Universidad de Zaragoza, más concretamente entre el ISAAC Lab[2] (Interdisciplinary Studies in Adaptivity, Autonomy and Cognition), y el Laboratorio de Danza y Nuevos Medios[3] de ETOPÍA.

El arte por definición siempre busca nuevas formas de sorprender o captar la atención, formas con las que conseguir despertar diferentes emociones en el espectador que hasta el momento no había experimentado. Ante esta naturaleza innovadora, los grandes avances tecnológicos que han tenido lugar en las últimas décadas han hecho plantearse a los artistas nuevas vías de comunicación valiéndose de las herramientas que esta nueva era pone a su disposición. En este punto se sitúa el porqué de este proyecto.

[2] http://isaaclab.unizar.es/
[3] http://www.danzatrayectos.com/laboratorio-de-danza-y-nuevos-medios/

Todo lo anterior se enfoca en el marco de la creación de la pieza de baile contemporáneo "**Pulse"**, que fue presentada el pasado 22 de Junio en el festival Trayectos (ETOPÍA). En esta pieza se unen los esfuerzos de ingenieros y bailarines para conseguir esa nueva vía de expresión artística por medio del uso de las nuevas tecnologías.

1.3. Trabajo a realizar

El trabajo a realizar consiste en primer lugar en la familiarización con los diferentes sensores que se tiene a disposición, manejo básico del software controlador y conocimiento de las diferentes magnitudes que se pueden conseguir de cada uno. Una vez se han decidido que sensores se van a utilizar en el proyecto, se elabora una planificación en la que se sitúan las diferentes etapas del mismo, que acciones se han de realizar en cada una de ellas y cuál va a ser la estrategia a seguir para la consecución de los objetivos.

En paralelo con el trabajo de ingeniería los bailarines comentan con el equipo técnico sus ideas sobre cómo les gustaría que fuera la estética y los posibles efectos a incluir, que respondan a determinados movimientos que puedan quedar registrados por los sensores. Y el equipo de ingenieros intenta acoplarse lo máximo posible a estos requerimientos proponiendo diferentes opciones y alternativas al equipo de baile, teniendo en cuenta las limitaciones existentes. Este proceso de avance con retroalimentación ha tenido lugar durante todo el proyecto, lo cual ha sido clave para la consecución del resultado final

Por lo tanto se debe desarrollar un algoritmo que sea capaz de incluir los sensores, seleccionando los datos que se necesiten de cada uno. Tratando adecuadamente estos mediante posibles filtrados, para que representen fielmente los movimientos de los bailarines y proporcionen una buena interacción con el entorno. Este código tambien se encargará de las proyecciones pertinentes, organizando el orden en que aparecen, cuál es el efecto que en cada una se produce y encargándose de todas las tareas que sean necesarias.

1.4. Herramientas utilizadas

Las herramientas que se han utilizado quedan definidas más extensamente en el apartado 3. Estas son básicamente herramientas informáticas y sensores.

Con respecto a las herramientas informáticas se ha trabajado con diferentes compiladores que utilizan distintos lenguajes de programación, además se ha trabajado en un entorno en el que funciona un sistema empotrado o embedido. En este campo también se ha trabajado con proyectores de imagen, pieza simple y a la vez fundamental.

En cuanto a tecnología sensorial, se utilizan diferentes tipos de sensores capaces de medir diversas magnitudes. En común tienen que todos deben ir acoplados al bailarín para su correcto funcionamiento.

1.5. Estructura del documento

La estructura de esta memoria está dividida en siete capítulos, incluyendo este capítulo introductorio. En el capítulo 2 se desarrolla el contexto en el que se enmarca la creación del proyecto y se exponen ejemplos de la influencia de las nuevas tecnologías en el mundo de la danza. En el capítulo 3 se describe la tecnología empleada para la creación de la aplicación, distinguiendo entre las distintas etapas del ciclo de vida del proyecto en el que se utilizó. El capítulo 4 trata sobre los Xsens, su estudio y su implementación. En el capítulo 5 se describe la estructura de la aplicación, parando a explicar cada una de las partes del código por separado. Por otro lado en el capítulo 6 se muestran los resultados obtenidos a con la aplicación, usando imágenes. En el capítulo 7 se recogen las conclusiones obtenidas durante el desarrollo del proyecto así como posibles líneas de trabajo futuro.

Capítulo 2

Estado del arte

En este capítulo se expone la situación en la que se enmarca el desarrollo del proyecto. Se comentarán las nuevas tendencias en el entorno de la danza y como la progresiva modernización del entorno artístico facilita nuevas formas de expresión, así como la aparición de instrumentos especializados.

En el caso concreto de nuestro proyecto se habla de danza contemporánea, por lo que se explicarán sus orígenes e influencias fundamentales, así como el desarrollo experimentado en los últimos años.

2.1. Danza contemporánea

La danza contemporánea surge como alternativa a la estricta técnica del ballet clásico, a finales del siglo XIX, en un comienzo del cuestionamiento de valores, y de la búsqueda de nuevas formas. En un primer momento se la llamó danza moderna pero a mediados del siglo XX paso a denominarse como se conoce hoy en día.

En la danza contemporánea, el bailarín se expresa a través de las técnicas del ballet clásico pero incorporando otros movimientos corporales más modernos. La mezcla de múltiples influencias es una de las principales características de este tipo de danza, que puede incluir formas de narración que no resultan lineales y hasta puede apostar por las herramientas multimedia para complementar las coreografías. Se convierte, por tanto, en un nuevo medio para que el hombre pueda hablar utilizando su cuerpo, para poder expresar sentimientos, ideas e historias por medio del lenguaje del movimiento.

La danza clásica se basa en pasos estructurados y ya codificados, poseyendo una dramaturgia[4] con principio, clímax y desenlace. Mientras que la danza contemporánea puede seguir esta estructura o bien dejar paso a la innovación, permitiendo que tanto bailarín como coreógrafo exploten su creatividad. Otra diferencia es que en la danza contemporánea no siempre es necesario contar una historia, simplemente se puede transmitir un concepto o proponer un ambiente con una estética particular. La danza contemporánea busca la conexión con lo terrenal, con lo humano y sus pasiones, la no estructura, la transgresión.

La música y la indumentaria de los bailarines se elige con especial interés, viendo cual es su valor estético y musical, que sensaciones puede transmitir y como puede influir en los movimientos de los bailarines y en la composición general de la obra.

Se distinguen dos vertientes de la danza contemporánea, la escuela europea y la escuela americana. Con relación a esta última, se encuentra la que según especialistas está considerada como la precursora de este género, Isadora Duncan (1877 - 1927).

[4] Arte y técnica de componer o poner en escena dramas teatrales u obras.

2.2. Danza y tecnología

La danza al igual que el resto de las artes ha tenido grandes cambios estéticos durante toda su historia. En los últimos cincuenta años la investigación y la búsqueda continua de las nuevas formas de expresión ha conducido a la danza por diferentes caminos hacía la perfección técnica y artística. Influida por los cambios de la vida social y política y también de otras artes, la danza ha cambiado su forma de ser, su estética y su percepción.

La tecnología se ha ido introduciendo en el mundo de la danza como la posibilidad de experimentar nuevas formas escénicas o técnicas de expresión. Llegados a este punto, el público ya no es considerado como simple espectador pasivo. En este sentido, las nuevas tecnologías se centran en los procesos de comunicación, en la creación de nuevos entornos comunicativos y expresivos que facilitan a los receptores la posibilidad de desarrollar nuevas experiencias formativas, expresivas y educativas.

En los últimos años se han ido incorporando diferentes técnicas al proceso creativo de la danza contemporánea, en la mayor parte se trata de técnicas audiovisuales, lo que se conoce como **Videodanza** o danza para la cámara. En este tipo de representaciones las coreografías están diseñadas desde un punto de vista audiovisual, en las que el movimiento de los bailarines es recogido y procesado por los diferentes medios de producción y postproducción. Según Douglas Rosemberg, "la cámara y el método de grabación debe ser entendido como un espacio, del mismo modo en que nos referimos al teatro como el lugar para un espectáculo de danza". Esto ofrece la posibilidad de establecer otra visión, una visión imposible de representar en la escena e imposible de captar por el ojo humano si no es a través de una pantalla.

También el avance que ha tenido lugar en las últimas décadas en el campo tecnológico, ha propiciado la aparición de instrumentos especializados del mundo de la danza. En este campo nos encontramos con diferentes medios que permiten captar movimientos concretos del bailarín, para su posterior evaluación, lo que puede ayudarle a mejorar su técnica, escenarios diseñados específicamente con proyecciones en 3D que interactúan con los artistas o vestimenta que puede emitir diferentes tonos en función de la música.

Con respecto a los instrumentos que permiten la captura de movimientos cabe destacar los **E - Trace** creados por la diseñadora y bailarina amateur Lesia Trubat. Este invento consiste en acoplar un pequeño mecanismo digital a la suela y los laterales del zapato de los bailarines, también conocidos como puntas. Las zapatillas electrónicas funcionan a través de la tecnología Arduino Lilypad, que se activa al contacto con el suelo, registrando la presión y el movimiento de los pies, mediante una aplicación que registra los datos exactos de forma gráfica. Con esto el bailarín puede observar en formato video cuales han sido sus movimientos, y puede corregir errores e imperfecciones.

Fig 2.1 Zapatillas E - Trace

En relación con lo expuesto anteriormente se encuentra el entorno en el que se ha desarrollado el trabajo. El proyecto ha aunado instrumentación tecnológica y medios audiovisuales para crear una intersección entre danza e imagen en movimiento. El punto de confluencia es muy amplio, ya que son tan diversas las formas de entender la danza como los estilos, tecnologías y géneros audiovisuales desde los que plantear la aproximación.

En el caso de los instrumentos utilizados, a diferencia de los E-Trace, no se trata de material creado específicamente con fines artísticos, sino que se ha trabajado con tecnología de sensores con diversas aplicaciones, que se han implementado para poder trabajar en el marco artístico.

Desde el punto de vista audiovisual no se ha procedido a grabar a los bailarines y trabajar con esas imágenes como en el campo de la Videodanza. Pero si mediante proyecciones interactivas, se ha tratado de buscar ese punto de comunicación que no sería posible compartir con el espectador si no es a través de los medios audiovisuales.

Capítulo 3

Tecnologías utilizadas

3.1. Sensores

Como se ha indicado con anterioridad el objetivo de este proyecto es el de la hibridación entre arte y tecnología. Con este motivo, en el transcurso del proyecto se ha trabajado con diferentes sensores que se especializan en distintos campos de percepción: En el campo del movimiento en tres dimensiones tenemos los Xsens, en el de la percepción visual se ha trabajado con Eye tracking, para el movimiento de abrir y cerrar la mano se ha usado los Data Glove y por último con referencia a la percepción de las ondas cerebrales el electroencefalograma (EEG), llamado Emotiv pro. Después de una primera toma de contacto con los citados sensores, viendo las diferentes posibilidades que ofrecían cada uno de ellos, se decidió que los sensores con mayores posibilidades de interacción con la danza y con los que, por tanto, se iba a trabajar eran los XSENS y los Data Glove.

Posteriormente se debatió con los bailarines acerca de la incorporación de ambos sensores exponiendo ventajas y desventajas, limitaciones, carencias y posibilidades de los mismos, obteniendo la conclusión de que el Data Glove interfería en exceso en el transcurso y estética del baile, llegando a incomodar a los bailarines e impidiendo ciertos movimientos técnicos de la danza como agarres o portés. Por todo esto se decidió trabajar únicamente con Xsens.

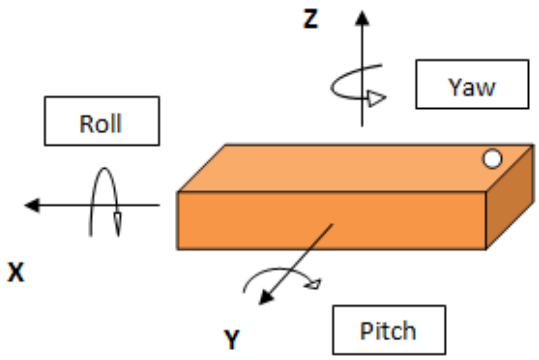
Además a los Xsens hay que añadir los UbiSense, que son unos sensores de localización en el espacio, que ya estaban previamente implementados en ETOPÍA (empresa colaboradora) en la zona correspondiente donde se ha realizado la función.

A continuación se va a exponer una explicación básica de funcionamiento y posibilidades de los Xsens y de los Data Glove, ya que son los sensores que han tenido más relevancia en el proyecto.

3.1.1. Xsens

3.1.1.1. Descripción

Los sensores propiamente dichos de Xsens son los Motion Traker (MTw), estos son unidades de medición inercial en miniatura que contienen acelerómetros lineales 3D, giroscopios de velocidad 3D, magnetómetros 3D y un barómetro. En lo que viene a continuación nos centraremos en las funciones que permiten recoger tanto el movimiento o aceleración en las tres direcciones del espacio como la rotación en torno a estos, es decir, los ángulos de alabeo, cabeceo y guiñada(siguiendo la terminología aérea) o también llamados pitch, roll y yaw. En la siguiente figura se pueden observar los ángulos apenas citados:



- X positivo cuando se señala al Norte magnético local.
- Y de acuerdo a la regla de la mano derecha (Oeste).
- Z positivo al apuntar hacia arriba.

Fig. 3.1 Orientación Motion Tracker

3.1.1.2. Componentes

Se tiene disposición de:

1. Seis sensores, llamados **Motion Tracker (MTw)** , que ya hemos comentado.

2. Una central, **Awinda Station** que va conectada al ordenador mediante USB. Esta estación controla la recepción de datos inalámbricos sincronizados de todos los MTw conectados de forma inalámbrica y carga hasta 6 MTW simultáneamente. Puede recibir datos inalámbricos de hasta 32.

3. Un sistema inalámbrico **Awinda USB Dongle** que consiste en un usb únicamente, por lo que es muy fácil de transportar y montar. Tiene las mismas capacidades inalámbricas que la Awinda Station (Controla la recepción de datos inalámbricos sincronizados de todos los MTw conectados de forma inalámbrica. Puede recibir datos de hasta 32 MTw).

4. Un juego de correas **MTw Click-in**. Estas correas tienen un mecanismo especialmente diseñado que permite que el MTw sea acoplado rápidamente y fácilmente mediante un click y quitado otra vez para cargar. Las correas están hechas de material elástico fuerte, respaldado con caucho de silicona para asegurar una fijación cómoda y ajustada a la piel. A su vez se sujetan y permiten el ajuste usando Velcro.

a) b) c)

Fig. 3.2 a) Motion Tracker; b) Awinda Station; c) Awinda USB Dongle y MTw Click- in

3.1.1.3. Software

De la parte del software de los Xsens se hablará más adelante en el capítulo 4.

3.1.2. 5DT Data Glove

3.1.2.1. Descripción

Los Data Glove son unos guantes que permiten medir la flexión de cada uno de los 5 dedos por separado. Están equipados con unas galgas extensiométricas , entonces lo que sucede al flexionar los dedos es que varía la resistencia interna de las galgas por lo que si se mantiene fijo el voltaje mediante la medición de la variación de la intensidad es posible obtener ese cambio en la resistencia[5] , sabiendo esto, la aplicación ya es capaz de conocer y poder mostrar el punto de flexión de cada dedo individualmente.

Fig. 3.3 Data Glove

[5] Ley de Ohm: $Voltaje = Resistencia \times Intensidad$

3.1.2.2. Componentes

Contamos con un sensor para la mano derecha y otro para la izquierda, así como con dos equipos:

1. Equipo inalámbrico

2. El segundo equipo está constituido únicamente por un cable con toma Rj12 (clavija internet) que va conectado al guante y por el otro lado con salida USB (A) que se conecta directamente al ordenador.

Fig. 3.4 Equipo Data Golves

3.1.2.3. Software

En el CD de los sensores se suministran dos aplicaciones:

1. **Glove Demo** que permite realizar un test rápido para comporbar el correcto funcionamiento de los data glove.

2. **Glove Manager** es un programa que te permite probar los guantes y acceder a funciones de guantes avanzadas que pueden no estar disponibles en los plug-ins.

El manejo de ambas aplicaciones es muy intuitivo, como ya hemos comentado primero por medio de la demo podemos calibrar el aparato realizando simplemente un chequeo con la mano abierta y otro con la mano cerrada. Luego ya dentro de la aplicación propiamente dicha (Glove Manager) no es complicado lanzar la visualización en tiempo real de la posición de cada dedo por separado así como de la toma de datos.

3.1.3. Ubisense

Los Ubisense son dispositivos de medición, de precisión de banda ultra-ancha (ultra-wideband, UWB), que contienen una serie de antenas y receptores de radio de banda ultra ancha. Los sensores detectan pulsos UWB a partir de las medallas Ubisense (llamadas Tags) que llevan colgadas los bailarines, lo que permite al sistema de localización Ubisense encontrar las posiciones de estos con una imprecisión de 15 cm * en 3D. En el entorno en el que se ha trabajado no se ha trabajado con la medición de la posición en el eje Z, pero si la situación en los ejes X e Y del escenario.

Ubisense utiliza una arquitectura celular para acoplarse desde a instalaciones pequeñas hasta otras muy grandes. Miles de sensores pueden integrarse en un único sistema de toda la empresa para monitorear un área ilimitada y administrar miles de Tags. Los sensores pueden conectarse entre sí en una gran variedad de formas, en función de los requisitos de la aplicación para que se usen.

Los sensores funcionan dentro de un entorno Ethernet [6] o wifi, utilizando una infraestructura de red estándar, como conmutadores Ethernet, puntos de acceso Wi-Fi y cables de red Cat5e para la comunicación entre sensores y servidores.

Fig. 3.6 Tags

Fig. 3.5 Sensor Ubisense

3.2.Arquitectura laboratorio de Danza y Nuevos Medios

A continuación se va a proceder a describir en líneas generales el entorno en el que ha sido realizado el proyecto a nivel informático y de telecomunicaciones. En el laboratorio de Danza y Nuevos Medios nos encontramos con un **sistema empotrado**[7] o embebido, cuya pieza clave es el broadcaster y por el que pasan todas las ordenes y mensajes.

[6] Ethernet es el protocolo por el cual se comunican ordenadores en un entorno de red local, es decir, es el sistema que normalmente se utiliza para comunicar ordenadores entre sí dentro de una industria. Este protocolo permite compartir la información y manejar completamente un ordenador desde otro.

[7] Es un sistema de computación diseñado para realizar una o algunas pocas funciones dedicadas, frecuentemente en un sistema de computación en tiempo real. Se diseñan para cubrir necesidades específicas.

En primer lugar hay que exponer el orden de jerarquía del sistema y luego se pasará a comentar las partes por separado (de más importante a menos): hardware, sensores y actuadores.

Los sensores sirven para detectar lo que se produce en el entorno y los actuadores para que el entorno intervenga. Los sensores que se han utilizado en el proyecto son los ubisense y los xsens pero también se podría trabajar con cualquier otro sensor que sirva para captar otro tipo de información. Todos esos sensores, lo que es el dispositivo, van asociados a un proceso, a un algoritmo, que se ejecuta dentro de un ordenador o dentro de un microcontrolador. La mayoría de los sensores, por ejemplo los ubisense, van conectados a un ordenador y ahí funciona un programa (programado en el lenguaje de programación pertinente).

Esos datos de los sensores en bruto se convierten en valores numéricos y posteriormente se normalizan. Por ejemplo si se dispone de un micro, cuando no detecta audio es un cero y cuando detecta audio es un uno. Cualquier sensor se normaliza entre cero y uno. Por otro lado hay sensores que solo detectan si sí o si no, por ejemplo un botón. Otros sensores detectan un rango de valores, por ejemplo el volumen de un micro (entre 0 y 1) eso es lo que se llama sensor 1D, porque tiene una sola dimensión de medida. Por otro lado los Ubisense, detectan valores de un vector bi o tridimensional mientras que Xsens lee aceleración y rotación en torno a tres ejes, por lo que sería un sensor 6D, con seis dimensiones. Otra cosa es que se pueda dividir y decidir trabajar solamente con el vector de las rotaciones, esta decisión de representar un vector de 6D o dos de 3D es decisión del usuario.

Esos valores los recibe el proceso asociado y crea una conexión osc [8] con el broadcaster, por ejemplo si se dispone de 20 sensores, se producirán 20 conexiones que se hacen con el broadcaster.

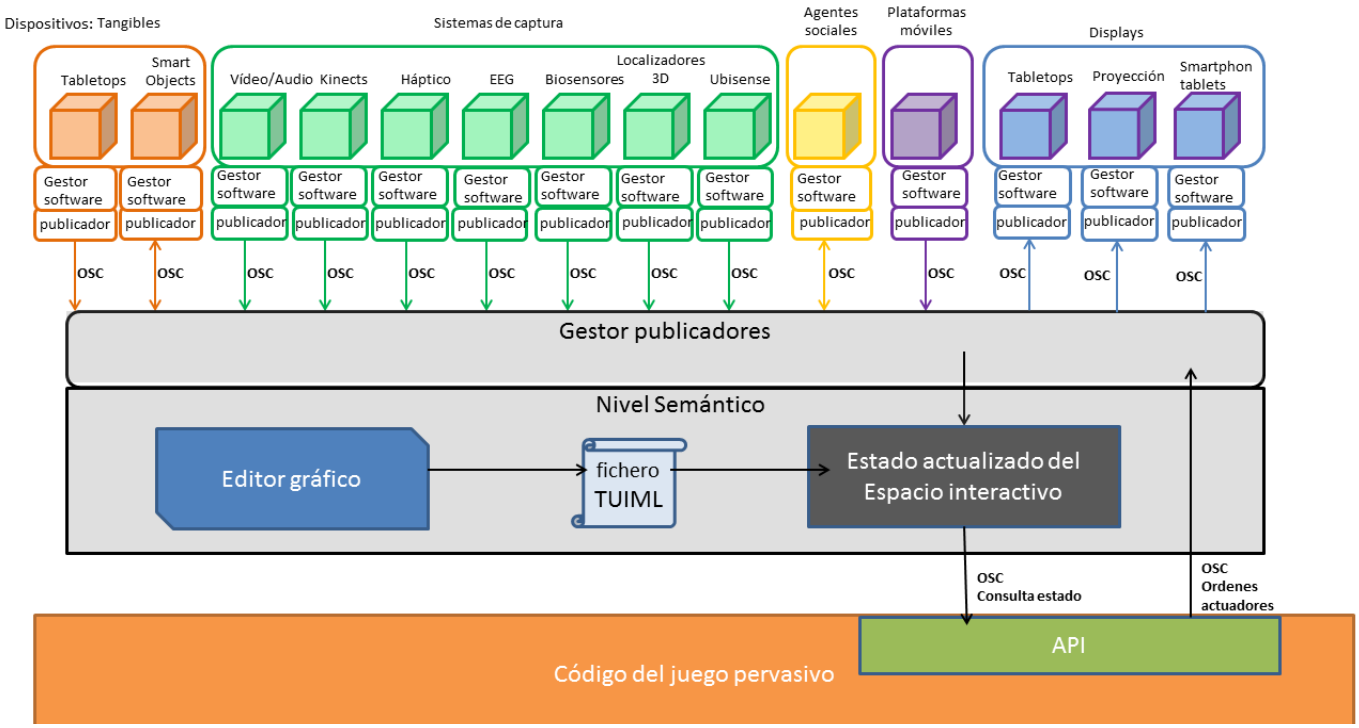


Fig. 3.7 Arquitectura Laboratorio Danza y Nuevos Medios

[8] Open Sound Control (OSC) es un protocolo para la comunicación entre ordenadores, sintetizadores musicales y otros dispositivos multimedia inspirado en la moderna tecnología de las redes. El protocolo tiene algunas ventajas como por ejemplo la independencia del medio de transmisión y la flexibilidad para transportar cualquier tipo de datos.

3.2.1. Broadcaster

El broadcaster sirve para la difusión de información o paquetes de datos a través de redes informáticas, desde un nodo emisor a una multitud de nodos receptores. El broadcaster, también llamado gestor de publicadores, mantiene conexión OSC individualizada con cada proceso publicador. Cada conexión OSC con el gestor de publicadores está diferenciada del resto de conexiones a través del número IP del ordenador o microcontrolador en el que corre el proceso publicador, y de un número de puerto.

El broadcaster acepta conexiones de tres tipos de procesos, cada uno de esos procesos puede estar ejecutándose quizás en esa misma máquina donde está funcionando el broadcaster o quizás en otras máquinas. Es por tanto posible que un mismo ordenador o microcontrolador gestione varios dispositivos, corriendo varios gestores software y publicadores, usando puertos de red distintos. En el entorno de trabajo hay una red wifi creada, a través de esa red wifi todos los procesos de todos los sensores se conectan a un número IP y a un determinado puerto, que es el broadcaster, quedando cuando se conectan listados como se muestra en la siguiente figura.

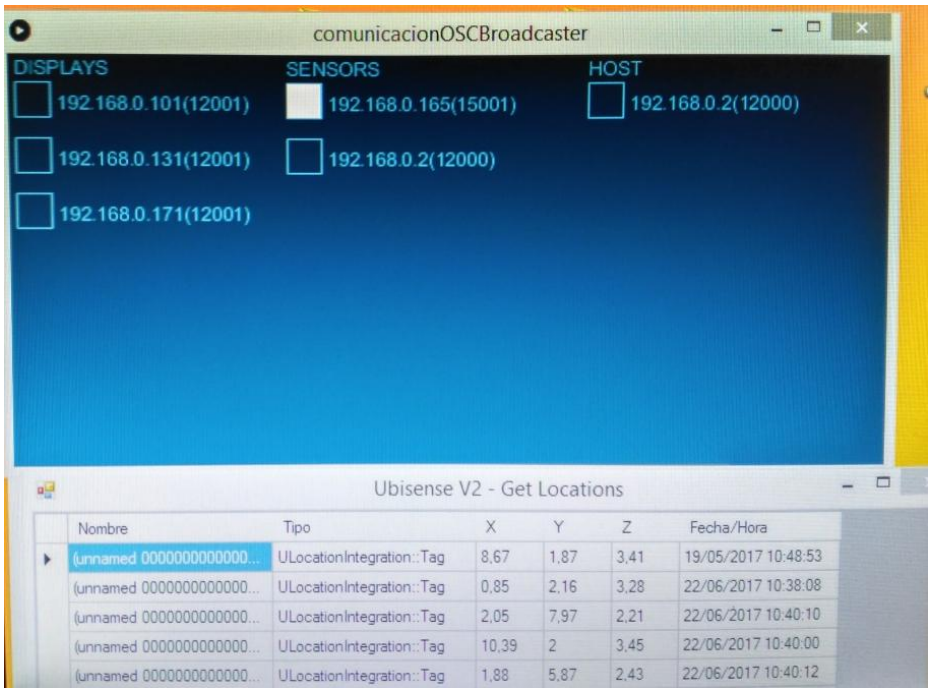


Fig. 3.8 Interfaz Broadcaster

Los tres tipos de dispositivos que se pueden conectar son:

1. **Sensores y actuadores:** En el caso de los sensores volver a los ya citados y como ejemplo de actuador tendríamos un foco[9], al cual se le pueden mandar órdenes para que se encienda y se apague (el foco también tiene asociado un proceso, no es un proceso que capture datos del propio actuador porque el foco no captura datos, sino que recibe datos que le manda el broadcaster, los interpreta y actúa según las ordenes obtenidas:

[9] Actuador que se encuentra en el laboratorio de Danza y Nuevos Medios pero que no se utiliza para el proyecto.

que se ponga en verde, en rojo, etc.). Hay que explicar que donde pone "sensors" en la pantalla del broadcaster en realidad se refiere a sensores y actuadores.

2. **Displays:** Son los dispositivos que permiten mostrar información visual y auditiva. Son otro tipo de dispositivos que están conectados a un instrumento que es capaz de mostrar imagen o de reproducir audio, como pueden ser los proyectores de las paredes, las mesas interactivas... Cada uno de esos proyectores está conectado a un ordenador y en ese ordenador corre un proceso que es llamado el pintor. El pintor es un proceso que lo único que se encarga de hacer es dibujar en cualquiera de esos displays y siempre es el mismo. A su vez ese proceso está conectado con el broadcaster para esperar órdenes: pinta un círculo, pinta una partícula.... Se producen entonces N procesos para dibujar porque hay n pantallas en las que dibujar, es decir un número indefinido de displays y un número indefinido de sensores, actuadores.

3. **Host:** En el laboratorio por norma general siempre se trabaja con un Host, ya que el algoritmo que desarrolla el usuario suele ser uno. El host es donde se ejecuta el programa, donde se ejecuta su lógica. Recibe datos de los sensores, los interpreta y según lo que ocurra ordena a los displays que pinte las cosas de una determinada manera, reaccionando a lo que dicen los sensores o por otro lado diciéndole a un actuador que realice una determinada acción, por ejemplo que el foco se encienda de una determinada manera.

   Por otro lado hay situaciones donde interesaría disponer de un host distribuido, por ejemplo si se tiene una excesiva saturación de tráfico de datos en los displays. En los pintores, cuando se quiere pintar cosas con muchos detalles o muy intensivas, se depende mucho de la capacidad para pintar sin que aparezcan retrasos o delays, que creen el efecto de que la proyección va como a saltos. Entonces si por ejemplo es necesario pintar 1000 partículas, pues implicaría 1000 mensajes que están enviándose, por lo que el pintor además de pintar las partículas debería pararse a escuchar cada uno de los mensajes.

   Una solución que se toma a menudo para evitar lo anterior es disponer de varios hosts, que estén ejecutándose en alguna máquina que pinta y el host directamente ya pinte, sin la necesidad de crear mensajes osc. Por ejemplo se podrían tener tres hosts, uno en cada pantalla y que sea el host el que ya se dedicara a pintar, pero pintar cosas muy intensivas, con esto se evitaría usar los displays para no tener mucho tráfico de información y combatir posibles delays o desfases temporales.

3.2.2. TeamViewer

El problema del sistema con el que se trabaja en el laboratorio es que existe un ordenador por cada display, que están repartidos y ocultos por el espacio para que no se vean. A veces se puede aprovechar y en un mismo ordenador acoger a un sensor, a un display y a un host, pero esto no evita que haya múltiples máquinas. Si cada vez que se debe cambiar algo, que instalar un programa nuevo, que ejecutar un programa o que hacer una prueba se debe ir físicamente a ese ordenador, conllevaría un gasto innecesario de tiempo y además tendrían que tener cada uno su ratón su teclado y su monitor. Esto se soluciona con la utilización del programa TeamViewer, que permite la utilización de un escritorio remoto, es decir habilita al usuario para poder manejar el ordenador desde cualquier otro lugar. En ETOPÍA concretamente existen 5 máquinas distintas y se manejan por medio del TeamViewer desde la máquina que recibe los datos de los sensores. Se podría decir que el TeamViewer es una ayuda, una comodidad y no algo imprescindible, pero evita tener que trabajar con cada una de las máquinas por separado.

3.3. Processing

Processing es un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java, de fácil utilización, y que sirve como medio para la enseñanza y producción de proyectos multimedia e interactivos de diseño digital. Fue iniciado por Ben Fry y Casey Reas, ambos miembros de Aesthetics and Computation Group del MIT Media Lab dirigido por John Maeda. Concretamente se ha utilizado la versión de Processing 3.3.3.

3.4. Microsoft Visual Studio

Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas operativos Windows. Soporta múltiples lenguajes de programación, tales como C++, C#, Visual Basic .NET, F#, Java, Python, Ruby y PHP, al igual que entornos de desarrollo web, como ASP.NET MVC, Django, etc. Permite a los desarrolladores crear sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión .NET 2002). Así, se pueden crear aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, dispositivos embebidos y consolas, entre otros.

Cabe destacar que la versión que ha sido utilizada para el proyecto es la 2015 y se ha usado el lenguaje de programación C++.

3.4.1. C++ (lenguaje de programación)

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al lenguaje de programación C mecanismos que permiten la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido. Una particularidad del C++ es la posibilidad de redefinir los operadores, y de poder crear nuevos tipos que se comporten como tipos fundamentales.

3.5. Matlab

MATLAB (abreviatura de MATrix LABoratory) es una herramienta de software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). Está disponible para las plataformas Unix, Windows, Mac OS X y GNU/Linux .

Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware.

3.6. Encuadre de las tecnologías durante el ciclo de vida del proyecto

Empezando por explicar la utilización que han tenido los diferentes sensores, hay que destacar dos periodos de tiempo que tuvieron lugar durante la realización del proyecto. Un primer periodo que transcurrió en el ISAAC Lab, en el que se tuvo una primera toma de contacto con los sensores y en el que se decidió seguir adelante con la implementación de Xsens y Data Glove. Y un segundo periodo donde se paso a trabajar en el entorno de Etopía, con el equipo de danza, en el que se descarto la opción del trabajo con Data Glove y se incluyo en el proyecto la utilización de los Ubisense.

A nivel de herramientas informáticas no hay unos periodos tan claros, porque se trabajo con los distintos programas y lenguajes tanto en un laboratorio como en el otro. Al igual que al comienzo del proyecto, en su versión final se trabajo con Processing, también en este periodo final se incluyó el programa Matlab como mecanismo de lectura de los datos de los Xsens. En el tiempo que transcurrió al final de la etapa en el ISAAC Lab y el comienzo en Etopía, se utilizo el entorno en C++ de Visual Studio. Las tecnologías que se han ido utilizando en el transcurso del proyecto, están definidas de manera más detallada en el siguiente capítulo (punto 4.2 y 4.2.1). Es necesario destacar que de cada etapa del trabajo y de cada una de las herramientas empleadas, se han ido extrayendo conceptos e ideas que finalmente se han puesto en conjunto en la obra final.

Capítulo 4

Xsens: Lectura, análisis y utilización.

Como ya hemos introducido anteriormente, los Xsens nos permiten medir diferentes movimientos de manera inercial, siendo la aceleración y la rotación en torno a los tres ejes del espacio tridimensional lo que ocupará nuestro estudio de ahora en adelante.

4.1. Software

A continuación se va a hablar de la parte del software de los Xsens. El kit de desarrollo MTw se suministra con un software suite que consta de MT Manager y un kit de desarrollo de software. MT Manager se usa para visualizar y registrar datos, lo que facilita el uso rápido y fácil de los MTw y la Awinda Master. En la aplicación **MT Manager** podemos destacar dos utilidades:

- Impresión en pantalla de la posición del sensor, con respecto a sistema de ejes de referencia que nosotros habremos establecido, y muestra de la aceleración en tiempo real

- Impresión de los ángulos que hemos mencionado con anterioridad en un rango entre 180 y -180.

Con respecto a la salida de datos en bruto, nos permite obtener gran cantidad de datos lo que será nuestro objeto de estudio seleccionar los necesarios para el desarrollo de nuestra aplicación.

Por otro lado, el software también proporciona el kit de desarrollo de software de **MT (SDK)**, con códigos de ejemplo en C, C ++, MATLAB y Linux. MTK SDK tiene la intención de hacer el desarrollo de aplicaciones de software para el MTw fácilmente accesible.

Es precisamente este kit de desarrollo el que se implementará finalmente para obtener los datos de los sensores en tiempo real y así poder trabajar con ellos para el desarrollo de la pieza de baile.

4.2. Lectura de datos

El primer problema con el que nos encontramos al utilizar los Xsens y quizás uno de los que ha creado más complicaciones a lo largo del proyecto es el de la obtención de los datos procedentes de los sensores en tiempo real, ya que con el software de los sensores, el MT Manager, no los podemos obtener. El MT Manager nos permite grabar los movimientos en

periodos de tiempo, de duración a elección del usuario, y posteriormente una vez finalizada la grabación obtener una lista de los diferentes datos que se han obtenido durante la misma como pueden ser la aceleración en los diferentes ejes, el pitch, el roll ... que queda también a elección del usuario.

Para afrontar esta necesidad de la lectura y obtención de datos en bruto, en tiempo real, se siguieron diferentes caminos en el transcurso del proyecto. En un primer lugar se empezó utilizando el programa Processing, que utiliza un lenguaje de programación en Java. Después de un periodo de aprendizaje en el entorno de processing se comenzó a profundizar en la búsqueda de la solución al problema apenas expuesto. Dado la inexperiencia en este campo y ante las complicaciones que se encontraron se decidió dejar apartada esta vía de desarrollo. Entonces se optó por avanzar desde una versión beta que se había realizado tiempo atrás en el propio ISAAC Lab. Esto supuso un nuevo cambio de entorno de programación ya que esta alternativa se desarrollaba usando el programa Visual Studio, el cual usa un lenguaje de programación en C++, a lo que hay que añadir la utilización de la extensión de openFrameworks[10] del propio Visual Studio, por lo que fue necesario otro periodo de aprendizaje en este nuevo marco.

Los periodos de aprendizaje fueron largos y tediosos ya que la única experiencia previa que tenía con la informática era la obtenida al cursar la asignatura que se imparte en primero del Grado de Ingeniería de Tecnologías Industriales (basada en el lenguaje de programación Pascal). Supuso comenzar desde cero, aprendiendo los comandos básicos, estructuras típicas de la escritura informática, utilización de bubles y arrays... hasta llegar a la creación, modificación y manejo de librerias, clases y demás estructuras complejas que se han utilizado para la realización del código final.

Utilizando esta nueva herramienta (Visual Studio) se consiguió una primera lectura e interacción de los Xsens, pero posteriormente se descartaría esta propuesta dada su elevada complicación y que gran parte del entorno tanto del laboratorio de Danza y Tecnología de Etopía, como parte del código que ya se había desarrollado por parte del equipo de ingenieros estaba realizado con Processing.

Finalmente se decidió aprovechar la arquitectura del laboratorio, utilizando la existencia del broadcaster y los demás medios de los que se tenía disposición. En este nuevo escenario se desarrolló un método que permite la lectura de datos de los Xsens en tiempo real y su emisión mediante un mensaje osc al broadcaster.

4.2.1. Mecanismo de lectura de datos

Se partió de un código de ejemplo que era proporcionado por Xsens para ayudar a realizar desarrollos con los sensores. En estos ejemplos se incluían demostraciones y librerías para C++ y para Matlab. Se eligió el uso de Matlab por ser más sencillo de implementar.

En el ejemplo se realizaba la conexión con los sensores, se configuraba la información que se deseaba extraer y se seleccionaba el canal de comunicación y la frecuencia de envío de datos. Se modificó el código de ejemplo para extraer los datos necesarios para el objetivo del proyecto. Estos eran la aceleración de los sensores y la orientación o rotación en torno a los tres ejes. El código modificado se incluye en el Anexo I.

[10] OpenFrameworks es un conjunto de herramientas C ++ de código abierto diseñado para ayudar al proceso creativo proporcionando un marco simple e intuitivo para la experimentación.

Debido a la falta de librerías para hacer uso de la tecnología OSC en el lenguaje de programación Matlab, se decidió enviar los datos a un servidor en Python que se encargaría de recibir la información de Matlab, limpiarla y crear los mensajes de manera apropiada para su uso con el protocolo OSC.

Para la comunicación entre Matlab y Python se hizo uso de la tecnología más sencilla de comunicación por red, los sockets[11] (lineas 286 - 292). En Python se declaró un socket que escuchaba en un determinado puerto. Desde Matlab se enviaban las medidas tomadas por los sensores a ese servidor.

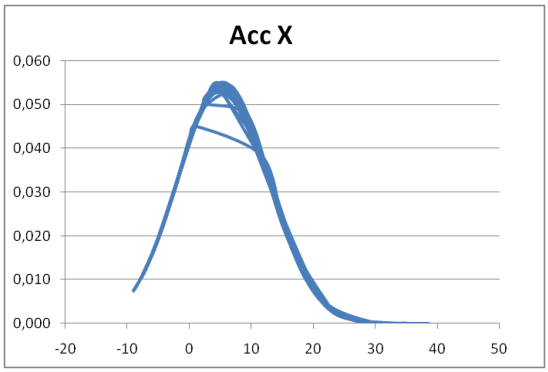
Por último estos datos se envían al servidor central (broadcaster), desde el servidor Python bajo el protocolo OSC. Los datos llegarán al broadcaster como mensaje entrante de sensores y ya se podrán mandar a cualquier máquina que esté conectada a él y precise de estos.

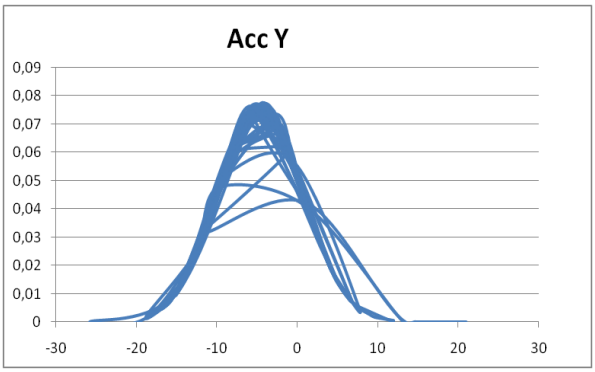
4.3. Estudio estadístico de los datos recibidos

Una vez se es capaz de manejar con soltura los mensajes que envían los sensores es necesario evaluar cuales son los datos que vamos a utilizar en nuestra aplicación y cuáles no. En primer lugar se ha estudiado cuales son las mediciones con las que mejor se puede representar el movimiento de los bailarines y con cuales las interacción con el entorno se hará más visual.

Después de esta primera experiencia se ha decidido que los valores que se utilizarán serán, como se viene diciendo, la aceleración y las rotaciones en torno a los tres ejes del espacio. Sin embargo llegados a este punto aparece la posibilidad de obtener lecturas de la aceleración libre, es decir, aceleración sin contar los efectos de la gravedad. Por lo que se tendrán que estudiar ambas aceleraciones para escoger la que más convenga.

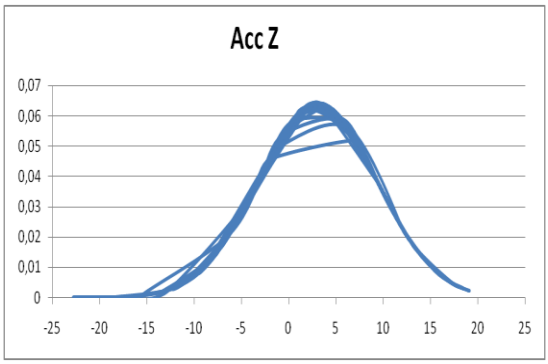
Una vez se han decidido las variables de interés se ha procedido a su estudio estadístico[12] para obtener, desde el punto de vista funcional, cuales son las variables más estables y cuales pueden presentar menos problemas a la hora de incorporar los sensores al espectáculo. A continuación se muestran las distribuciones de cada una de las variables que se han estudiado:

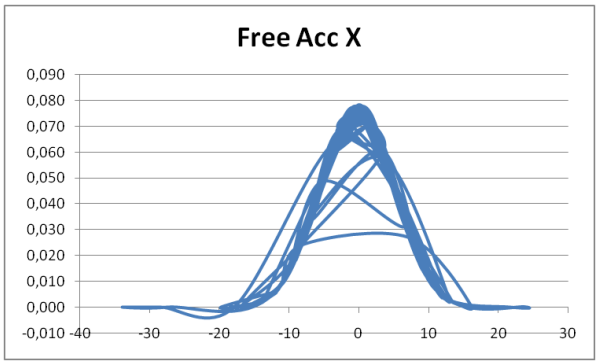


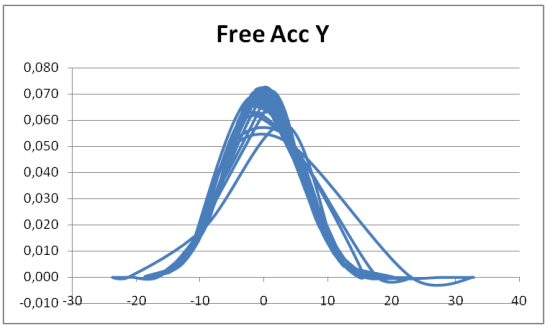


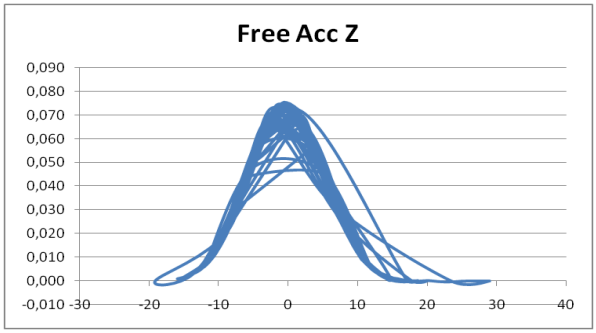
[11] Los sockets son un mecanismo que permite establecer un enlace entre dos programas que se ejecutan independientes el uno del otro (generalmente un programa cliente y un programa servidor). Cabe resaltar que tanto el cliente como el servidor solo deben conocer sus direcciones IP y el puerto por el cual se comunicarán.

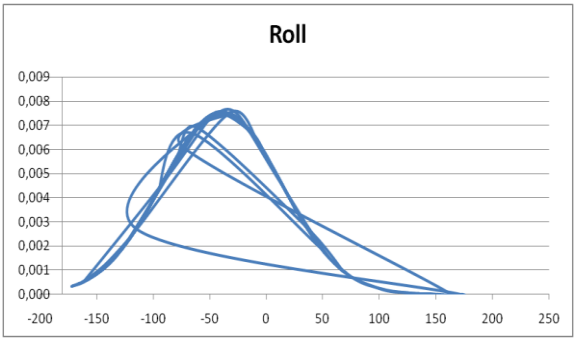
[12] Anexo IV -- Estudio realizado a partir de los datos grabados mediante MT Manager durante un ensayo de los bailarines

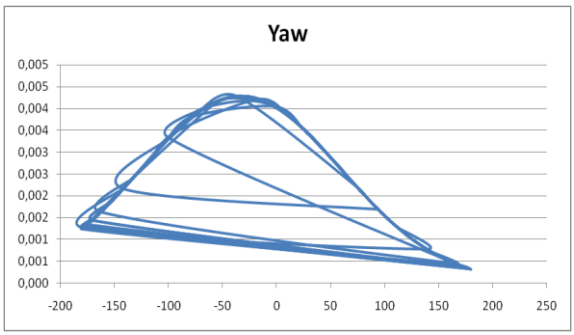

Acc Z
0,07
0,06
0,05
0,04
0,03
0,02
0,01
0
-25 -20 -15 -10 -5 0 5 10 15 20 25

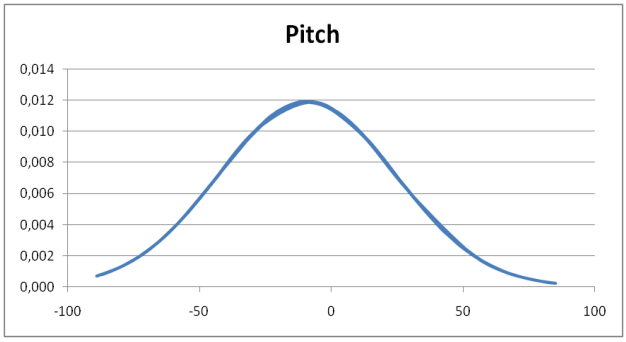

Free Acc X
0,090
0,080
0,070
0,060
0,050
0,040
0,030
0,020
0,010
0,000
-0,010 -40 -30 -20 -10 0 10 20 30


Free Acc Y
0,080
0,070
0,060
0,050
0,040
0,030
0,020
0,010
0,000
-0,010 -30 -20 -10 0 10 20 30 40


Free Acc Z
0,090
0,080
0,070
0,060
0,050
0,040
0,030
0,020
0,010
0,000
-0,010 -30 -20 -10 0 10 20 30 40


Roll
0,009
0,008
0,007
0,006
0,005
0,004
0,003
0,002
0,001
0,000
-200 -150 -100 -50 0 50 100 150 200 250


Yaw
0,005
0,005
0,004
0,004
0,003
0,003
0,002
0,002
0,001
0,001
0,000
-200 -150 -100 -50 0 50 100 150 200 250


Pitch
0,014
0,012
0,010
0,008
0,006
0,004
0,002
0,000
-100 -50 0 50 100

4.3.1. Conclusiones del estudio de datos

4.3.1.1. Aceleración

Como se puede observar, las distribuciones de las aceleraciones libres aparecen centradas en 0 m/s^2 mientras que las de las aceleraciones reales están desplazadas hacia valores positivos o negativos. Por lo tanto las aceleraciones reales pueden mandar datos erróneos cuando los bailarines están en reposo sin embargo las aceleraciones libres al no tener en cuenta la gravedad representan fielmente el movimiento de los artistas y se mantienen en 0 m/s^2 si estos no se mueven. Por lo tanto se va a trabajar con la **aceleración libre**, ya que nos permite una medición más fiel y consecuentemente una interacción con el entorno más real y con menos errores.

4.3.1.2. Ángulos de rotación

Como ya se ha comentado el rango en el que se representan los ángulos es desde 180 a -180 grados. En este apartado tenemos un problema con los ángulos que superan tanto superior como inferiormente ese rango, estos son el roll y el yaw. Una vez que cualquiera de estos dos ángulos supera superior o inferiormente el rango de medición establecido pasa inmediatamente al límite opuesto. Por ejemplo si el roll se va incrementando llegará un punto en el que pasará de valer 180 grados a valer - 180 y continuará creciendo hacia 0 grados, creando un salto que nos dará lugar a error. Este suceso podríamos corregirlo realizando lo que se llama un unwrapping.
Observando las distribuciones de roll y yaw vemos esos saltos representados con bastante claridad, en cambio el pitch describe una curva casi perfecta. Esto se debe a que el pitch varía en el rango de -90 a 90 grados y por lo tanto no excede los límites de medición.

En una primera toma de contacto ya se había destacado el pitch como el ángulo más representativo ya que si los bailarines llevan el sensor a la altura de su muñeca señaliza si el brazo está arriba (pitch positivo) o si esta abajo (pitch negativo), lo que podría ser de mucha utilidad a la hora de la interacción con el entorno. Los resultados del estudio corroboran esta hipótesis previa, siendo claramente el **pitch** el ángulo más estable y el que menos lugar a error deja y por lo tanto el ángulo con el que se ha decidido trabajar.

Capítulo 5

Descripción de la aplicación

Como ya se ha explicado el objetivo de este proyecto es el de la unión de arte y tecnología. Con este fin se ha creado un entorno que permite a los bailarines interactuar con él. Esto es posible gracias a la incorporación de unos sensores (Xsens y Ubisense) que hacen posible recoger de forma numérica las distintas variaciones (posición, aceleración...) que experimentan los bailarines en el transcurso de la obra. Para la creación de este entorno interactivo se ha usado el programa Processing v 3.3.3. En este capítulo se va a explicar la estructura del código o algoritmo que se ha creado, así como destacar los puntos que se consideran de mayor importancia.

5.1. Estructura del código

En el punto 3.2 se ha expuesto cual es la arquitectura del sistema empotrado que funciona en el Laboratorio de Danza y Nuevos Medios. Para el desarrollo del proyecto se ha seguido esta misma estructura, es decir, se ha desarrollado un Host y unos Displays que son los códigos sobre los que se va a hablar a continuación, además del código con el que leemos los datos procedentes de los Xsens (punto 4.2.1) y pasando, a su vez, por el broadcaster todos los mensajes tanto de sensores, actuadores, host.... A diferencia del esquema original del Laborarorio de danza y nuevos medios no se cuenta con un nivel semántico, ya que se buscaba una respuesta en tiempo real. Al no disponer de nivel semántico los mensajes desde la Api a los displays se realizan de forma directa dentro del código sin hacer uso de los mensajes osc, reduciendo el tiempo de respuesta de esta manera.

Antes de pasar a comentar cada una de las partes por separado y con el motivo de hacer más compresible su función, decir que las tres clases que se encargan de coordinar el funcionamiento de la aplicación son Dance Host, que recibe todos los mensajes y decide ¿quién? debe recibirlos, Pintar Dance, que estipula ¿qué? y ¿cómo? se ha de actuar y Displays Host que hace las funciones de mensajero.

5.2. Host

En el Host se encuentra la parte más lógica del código. El Host se encarga del correcto funcionamiento de todos los procesos que se están realizando al mismo tiempo, coordinando cuando debe terminar uno y comenzar el siguiente, recibiendo los datos de los sensores, filtrándolos e interpretándolos, junto con otras tareas. En definitiva sería como el cerebro del conjunto. El Host está dividido en tres funciones DanceHost, Displays Host y procesSignals.

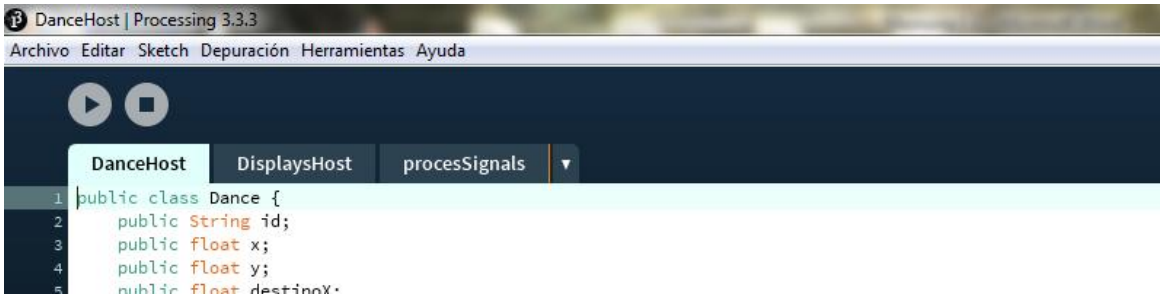


Fig. 5.1 Clases del Host

5.2.1. DanceHost

En primer lugar en el DanceHost tiene lugar la inicialización de variables que van a ser utilizadas, como por ejemplo el frame rate que se establece en 24 fotogramas por segundo (como en el cine). Dentro del DanceHost se encuentran también diferentes funciones de test, que se han ido utilizando conforme se iba avanzando en el proyecto para comprobar su correcto funcionamiento. Además se establece un sistema para avanzar al punto que se desee de la proyección en función de la tecla del teclado que se presione (líneas 307 - 348).

El Dance Host se encarga de recibir los datos procedentes de los distintos sensores y procesos, y redistribuirlos hacia las diferentes estructuras que precisen de estos, organizando así en primera instancia el tráfico de mensajes. Haciendo analogía con la figura x el Dance Host sería el corazón de la Api.

Una de las funciones más importantes que se realizan en este apartado es la de asignar los valores que se reciben de los Xsens a unas variables (líneas 139 - 305). Se puede observar partes del código comentadas, ya que en un principio antes de la creación de procesSignals, se hicieron pruebas filtrando los datos que se recogían directamente en DanceHost, para realizar los primeros ajustes.

5.2.2. DisplaysHost

Como ya se ha comentado anteriormente Displays Host hace las funciones de mensajero dentro del código. En un principio se trabajo para que con una sola proyección general esta parte del código supiera a que pantalla debía de mandar cada una de las partes de la simulación por separado, permitiendo así una interacción global de los bailarines con el entorno, pero aparecieron una serie de problemas (punto 5.3) por lo que esta parte no se llego a implementar.

Otra de las utilidades del Displays Host es la selección del número IP y del puerto del broadcaster que se va a escuchar (líneas 19 - 21).

5.2.3. procesSignals

Esta parte del Host se encarga del procesado de las diferentes señales que a él llegan. Concretamente se encarga de adecuar los datos recibidos de los Ubisense y de los Xsens para su correcta utilización. En primer lugar, a partir del estudio estadístico realizado, se establecen unos valores máximos y mínimos de las medidas que se admitirán validas (líneas 56- 63), es decir, si se recibe un dato fuera de ese rango se estimará que ha sido un error de medición y se descartará (líneas 67- 72; 93- 94). Así se evitará posibles cambios bruscos e imprecisiones que no reflejen fielmente el movimiento de los bailarines. Posteriormente se realiza un mapeo para convertir los datos obtenidos a un rango que haga más fácil su interpretación y manejo, así pues se mapean los datos (líneas 74-79; 95- 96) desde un rango de -12 a 12 m/s^2 , en el caso de la aceleración, a un rango de -50 a 50. Con esto conseguimos que se observen mejor las variaciones ya que los datos no se encontrarán tan próximos unos de otros.

En el caso de los Xsens se reciben datos según la frecuencia que el usuario elija, en nuestro caso se elige una frecuencia de 24 Hz ya que coincide con la velocidad con la que se pinta la pantalla, 24 fotogramas por segundo. Con esto se consigue que en todo momento se tengan datos actualizados de estos sensores. Sin embargo Ubisense manda sus mediciones una vez por segundo, lo cual introduce un desfase de la posición de los bailarines de un segundo con respecto a su posición real. Este problema no se ha podido afrontar desde el camino de intentar que los Ubisense enviaran más datos por segundo, ya que estos estaban ya implementados en el entorno del laboratorio por lo que se desarrollo una interpolación lineal tomando como datos las dos últimas posiciones del bailarín e intentando predecir cuál sería la próxima.

5.3. Display

El display es la parte que se encarga de la representación o proyección en las pantallas, por medio de los proyectores. Cada proyector dispone de un gestor de software (uno por pantalla) y de un publicador, que en el caso de nuestro proyecto se engloba en la misma clase. Está dividido en tantas funciones como presentaciones o efectos diferentes aparecen durante la obra. Antes de seguir comentando el display es necesario explicar la superficie y los medios que se disponían para la proyección.

El entorno que se disponía para la proyección está constituido por tres pantallas cada una con un proyector independiente. Las pantallas poseen dimensiones irregulares y están situadas en el espacio formando una especie de espacio tridimensional que rodea el lugar donde tiene lugar el baile. Algunas de estas pantallas son simplemente paredes pintadas de blanco, mientras que otras son estructuras expresamente diseñadas para el fin que nos atañe.

Fig. 5.2 Laboratorio de Danza y Nuevos Medios

Inicialmente se trabajó en un único display, que funcionara en la misma máquina donde corría el host y que unificara las tres pantallas en una sola a efectos de código. Sin embargo se encontraron dos problemas: la aparición de un delay o retraso en la proyección, debido al gran tráfico de información que tenía lugar y los problemas al ajustar el tamaño de proyección dadas las dimensiones irregulares de las pantallas. Para corregir esto se optó por utilizar un código display individual por cada pantalla, lo que evitaba el retraso debido al intercambio de datos ya que el display se encuentra en la misma máquina que va a controlar, y a su vez permite ajustar el tamaño de proyección de cada pantalla por separado. En definitiva a nivel de código es un único display, salvo pequeñas variaciones, pero que se encuentra funcionando independientemente en cada una de las tres máquinas (proyectores).

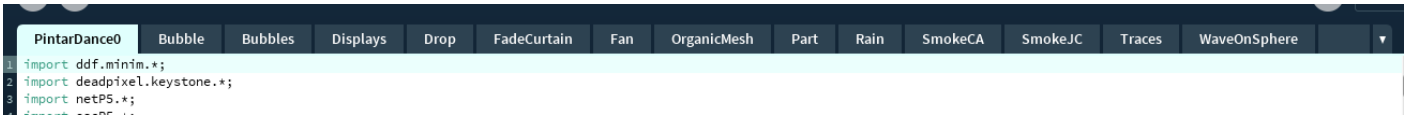


Fig. 5.3 Clases del Display

La obra se divide en un conjunto de proyecciones que se van sucediendo unas a otras, algunas de mayor duración y se podría decir más representativas, y otras con una función de transición entre estas. Todas estas partes están coordinadas por la clase principal **PintarDance**. Las simulaciones se separaron en distintas clases para facilitar el trabajo con el código.

Pintar Dance tiene un papel fundamental en la coordinación de la aplicación. Se encarga de coordinar que clase tiene que funcionar en un momento determinado y cuando debe dejar paso a otra, actuando como un reloj con los tiempos de las simulaciones (líneas 36 - 46). Además en el Pintar Dance asociado a cada pantalla se encuentran estipuladas las dimensiones de la misma. Por lo tanto, Pintar Dance sería ese gestor de software de cada pantalla, que antes hemos citado, y se encargaría tanto de gestionar la salida de audio como su proyección asociada.

Todas las proyecciones principales interaccionan en tiempo real con los bailarines, por medio de los sensores, siendo estas interacciones diferentes según el punto del espectáculo en el que nos encontremos. Es importante destacar que no en las tres pantallas se produce este efecto, únicamente la pantalla central reacciona con el movimiento de los bailarines. Esto se ha decido en consenso con el equipo de baile para que no se desviara mucho la atención del público hacia los laterales y hubiera un foco de atención centrado en ellos mismos y la pantalla central. Algunas de estas proyecciones se han desarrollado a partir de ejemplos de código abierto que se encuentran en la propia página de Processing y en otras librerías que se han incluido en la bibliografía.

A continuación se va a exponer el esquema principal de las proyecciones que se suceden y posteriormente se comentarán sus características y cuál es el tipo de interacción de cada una de ellas.

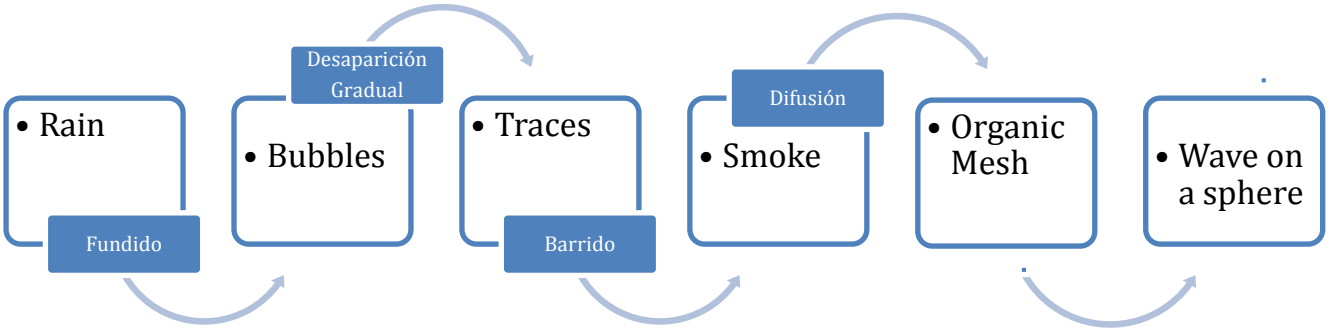


Fig. 5.4 Esquema de las presentaciones

1. **Rain:** Simula una lluvia sobre los bailarines. Cuando estos se encuentran en reposo no ocurre nada, pero conforme van acelerándose (Xsens) producen una especie de paraguas que despeja la lluvia, más grande cuanto mayor sea la velocidad de sus movimientos. La posición de este paraguas tiene dos componentes. Su posición en el eje x de la pantalla depende de la posición de cada bailarín, que es recogida por los Ubisense, y su posición en el eje y es función de si los brazos del bailarín se encuentran hacia arriba o hacia abajo, esto se puede saber por medio del pitch obtenido por los Xsens (positivo si los brazos se encuentran hacia arriba, negativo en el caso contrario). Si los brazos del bailarín se encuentran completamente a 90º el paraguas estará en el punto más alto de la pantalla, mientras que si se encuentran a -90º en el más bajo. Cabe destacar que el punto de 0º se da cuando los brazos se sitúan perpendiculares al pecho.

    1.1 **Fundido:** Cae una cortina blanca que transforma el fondo negro progresivamente en un fondo blanco

2. **Bubbles:** Intenta crear la sensación de estar inmersos en un espacio acuático tridimensional. Las burbujas tienen un movimiento aleatorio propio y además dos movimientos diferentes. Uno de ellos tiene que ver con la velocidad de movimiento de las burbujas, a medida que los bailarines se mueven más deprisa (Xsens) también lo hacen ellas. El otro va en función del pitch (Xsens), es decir, si los brazos del bailarín se encuentran inclinados hacia arriba las burbujas cambian su dirección hacia la parte exterior de la pantalla, como si intentaran salir de esta hacia los bailarines. Mientras que por el contrario si sus brazos se encuentran hacia abajo las burbujas se moverán en el sentido contrario como hacia el interior de la pantalla.

    2.1 **Desaparición gradual**: Las burbujas van desapareciendo poco a poco hasta quedar un fondo blanco.

3. **Traces:** En esta animación se ha tratado de crear una interacción muy sutil que no desvíe mucho la atención del público de los bailarines, ya que es una parte de la obra con movimientos muy suaves y con bastantes detalles. Sobre un fondo blanco aparecen tres trazos independientes, uno correspondiente a cada bailarín, con un movimiento

semialeatorio. La interacción consiste en que dentro del movimiento random de los trazos, cada x tiempo (aleatorio no muy grande) los trazos buscan a su correspondiente bailarín, se dirigen hacia su posición, que es transmitida por los Ubisense.

3.1 **Barrido**: La pantalla cambia gradualmente de blanco a negro, de izquierda a derecha, borrando a su paso los restos de los trazos de la simulación previa. Para entenderlo mejor se podría imaginar como si una brocha negra empezara a pintar las pantallas de izquierda a derecha una detrás de otra.

4. **Smoke:** Trata de conseguir el efecto visual de que los bailarines son focos emisores de humo. Estos focos se desplazarán en el espacio siguiendo los movimientos del bailarín por medio de su posición (Ubisense).

4.1 Difusión: Conforme se acerca el final de la presentación anterior se deja de producir humo, dejando que el que ya se había emitido ascienda y salga por la parte superior de la pantalla quedando la pantalla completamente negra.

5. **Organic Mesh:** Partiendo de un fondo negro van apareciendo segmentos de diferente orientación y longitud, desde diferentes posiciones de la pantalla y se dirigen a lugares concretos con la intención de ir creando una malla que acabe por cubrir todo el espacio de proyección. La interacción con esta proyección consiste en que los bailarines por medio de su movimiento vayan rompiendo la malla en lugares concretos. Es una situación parecida a la de Rain, se crea una zona que destruye la malla que depende tanto de la posición de los bailarines en el espacio (Ubisense) como de si sus brazos se encuentran hacia arriba o hacia abajo.

5.1 Aunque no aparezca en el esquema ya que no se ha realizado ningún efecto especial, la transición consiste simplemente en que desaparece la malla que cubría la pantalla y aparece instantáneamente la siguiente con forma de esfera.

6. **Wave on a sphere:** Se trata de una malla con forma de esfera, que se ha desplazado hacia una esquina de la pantalla para que solo se vea un fragmento de esa esfera, ya que se ha tratado de huir de formas geométricas a petición del equipo de baile, para adecuar la estética a un estilo de danza contemporánea. La malla conforme pasa el tiempo va haciéndose más compleja, dejando menos espacios sin rellenar. Esta esfera tiene un movimiento de rotación propio e inalterable pero cada cierto tiempo se producen unos pulsos que deforman esta estructura. Los pulsos aparecen en intervalos de tiempo mayores conforme nos acercamos al final de la obra y presentan una deformación mayor cuanto más se aleja el bailarín de la esfera (un solo bailarín en este caso), tomando como referencia su posición (Ubisense). Como final simplemente cuando termina la música desaparece la malla y se queda un fondo negro.

Capítulo 6

Resultados

La finalidad del Trabajo de Fin de Grado ha sido la de la creación de la pieza de danza contemporánea "Pulse". Como se ha ido explicando a lo largo del proyecto, se ha trabajado con diferentes tecnologías sensoriales y entornos gráficos y de programación, para conseguir transmitir una nueva idea de baile.

En el presente capítulo se incluyen capturas tomadas durante el ensayo final[13] de la coreografía, para poder apreciar los resultados obtenidos.

6.1."Pulse"

Se ha intentado plasmar lo mejor posible los diferentes detalles que tienen lugar durante la obra. Una tarea que no es sencilla dado que es difícil poder representar una acción de movimiento y reacción mediante capturas estáticas. Debido a que los detalles de cada presentación ya han sido explicados en el capítulo anterior, se van a hacer referencias de las diferentes funciones del display y comentando los detalles que se consideran más relevantes.

En el inicio de la obra los bailarines se encuentran sentados en los bordes del escenario. En el momento pactado, se acciona todo el proceso, la música comienza a sonar y los bailarines se acercan a la pantalla central, mientras esta va cubriéndose con lluvia poco a poco (Rain), hasta quedar como en la figura 6.1

Fig. 6.1 Rain a

[13] No fue posible grabar la actuación debido a la gran afluencia de público.

En la siguiente imagen se puede apreciar como varia la posición del "paraguas" en función de hacia dónde apunten los brazos del bailarín.

Fig. 6.2 Rain b

Al final de esta presentación tiene lugar la transición Fundido de la que se van a exponer capturas graduales. El fondo poco a poco se tornará blanco y dará paso a la siguiente parte.

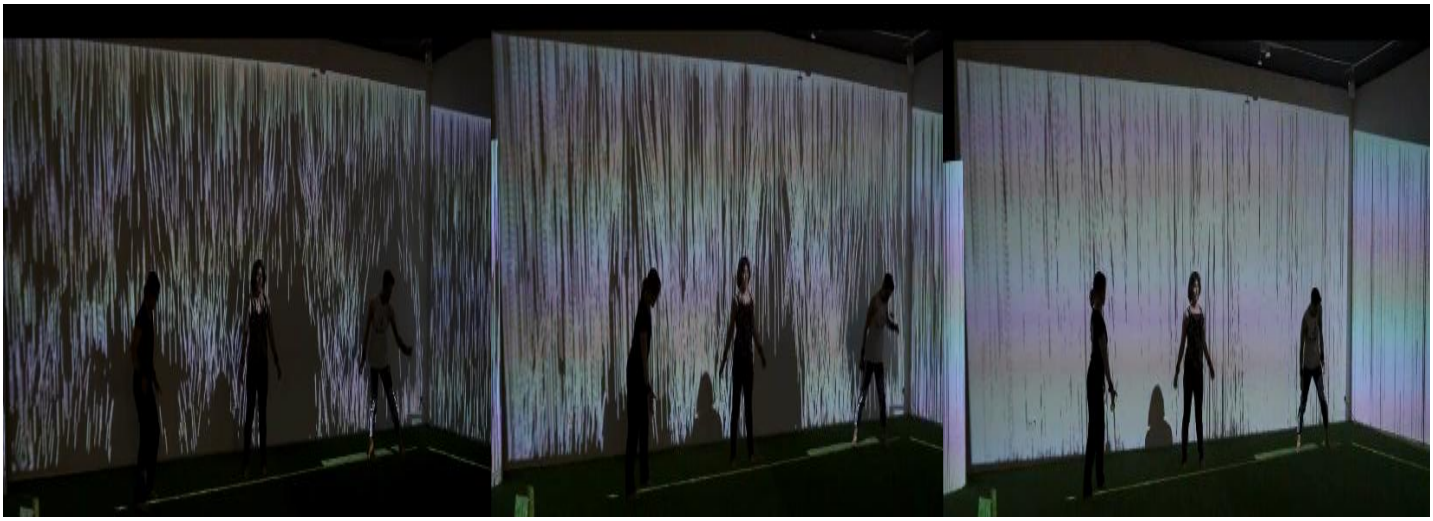


Fig. 6.3 Fundido

Para Bubbles no es posible escoger una instantánea que represente lo que sucede, ya que no se puede apreciar la dirección ni velocidad de las burbujas. Por esto, se ha intentado por medio de una sucesión de fotogramas captar el movimiento.

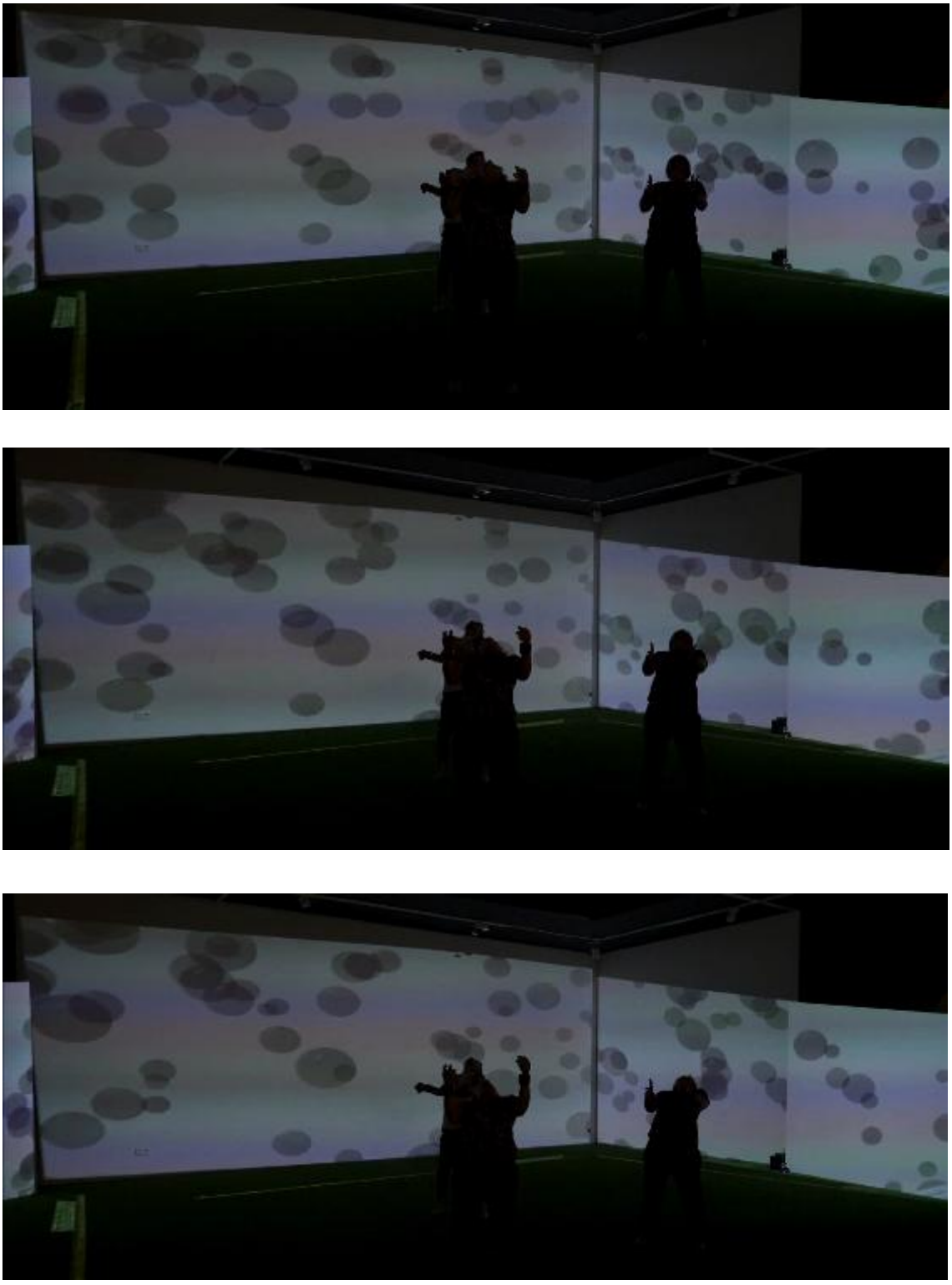


Fig. 6.4 Bubbles

Los bailarines se encuentran de frente a la cámara con sus brazos inclinados hacia arriba por los que las burbujas tienden a desplazarse hacia el exterior de las pantallas, aumentando de tamaño y desapareciendo de las pantallas. Y su velocidad en ese tramo concreto es lenta ya que los bailarines se encuentran prácticamente estáticos.

Con respecto a la transición de Desaparición Gradual, no se van a incluir imágenes, ya que únicamente las burbujas van desapareciendo poco a poco de la pantalla.

La siguiente parte de la obra es la correspondiente a Traces. En la siguiente imagen 6.5 se puede observar que cada trazo sigue un movimiento independiente, desplazado más a la izquierda o a la derecha según la posición del bailarín. Hasta que llega un momento en el que todos los trazos buscan a su bailarín.

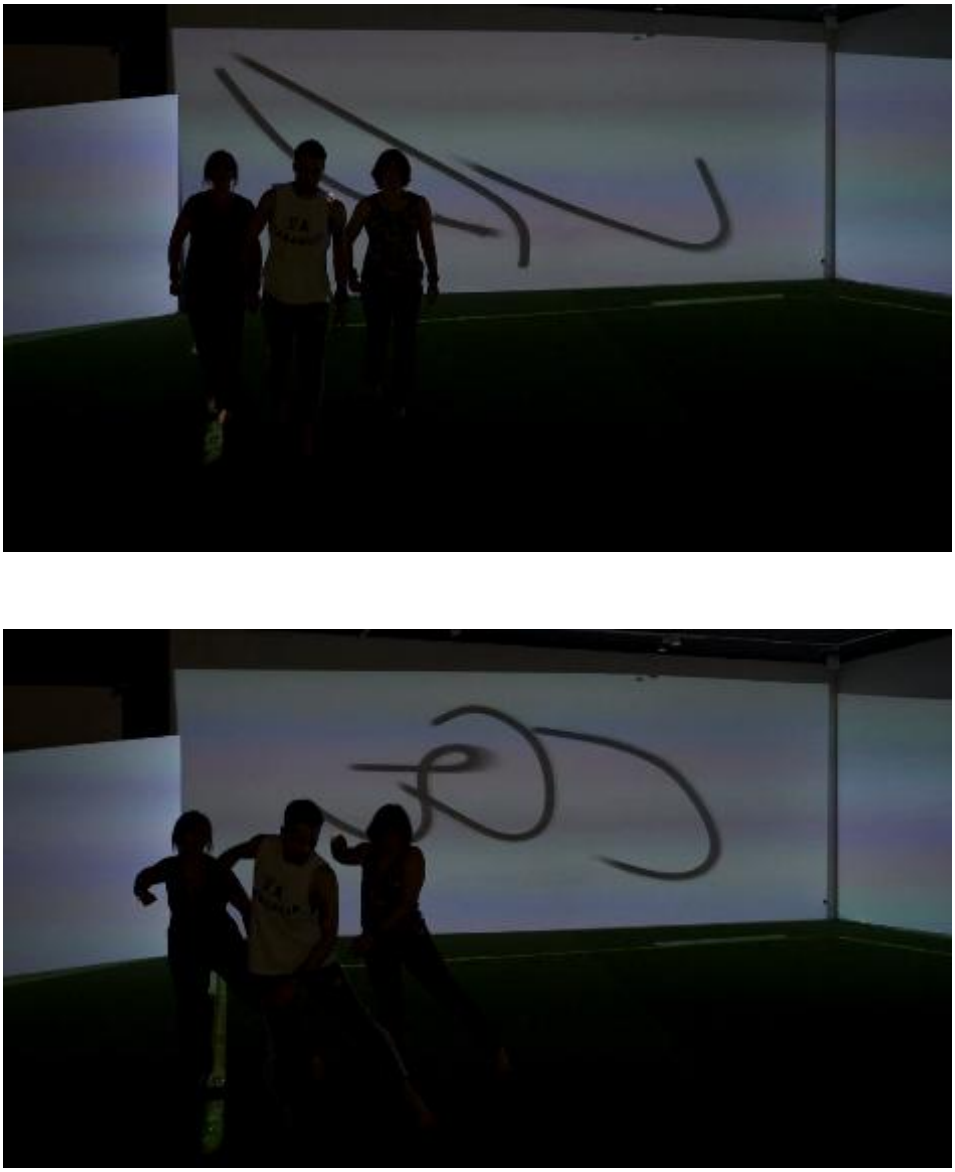


Fig. 6.5 Traces

Una vez se ha llegado al final de Traces, comienza la transición de Barrido. Se puede apreciar cómo va avanzando, pasando de una pantalla a otra, el barrido. No se pasa directamente de blanco a negro, sino que sucede una coloración gradual pasando por tonos grises que se van oscureciendo. Hasta quedar las tres pantallas completamente en negro.

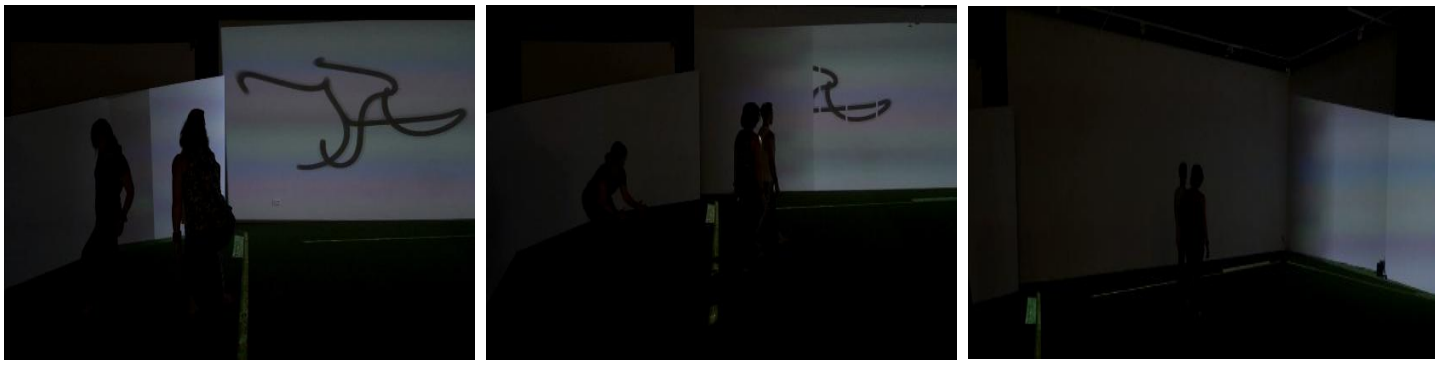


Fig. 6.6 Barrido

Partiendo de ese fondo negro comienza a funcionar Smoke. En la imagen que se expone a continuación se puede observar como dos focos de humo se localizan sobre los dos bailarines que se encuentran juntos en el centro, y otro se sitúa más a la derecha correspondiendo con la situación del bailarín restante. De la transición posterior Difusión no se va a incluir imagen, ya que únicamente deja de producir humo, mientras que el ya se ha emitido sigue ascendiendo en la pantalla hasta desaparecer.

Fig. 6.7 Smoke

Una vez se ha difuminado por completo el humo, comienza la creación de Organic Mesh. La malla va formándose poco a poco hasta cubrir la totalidad de las pantallas.

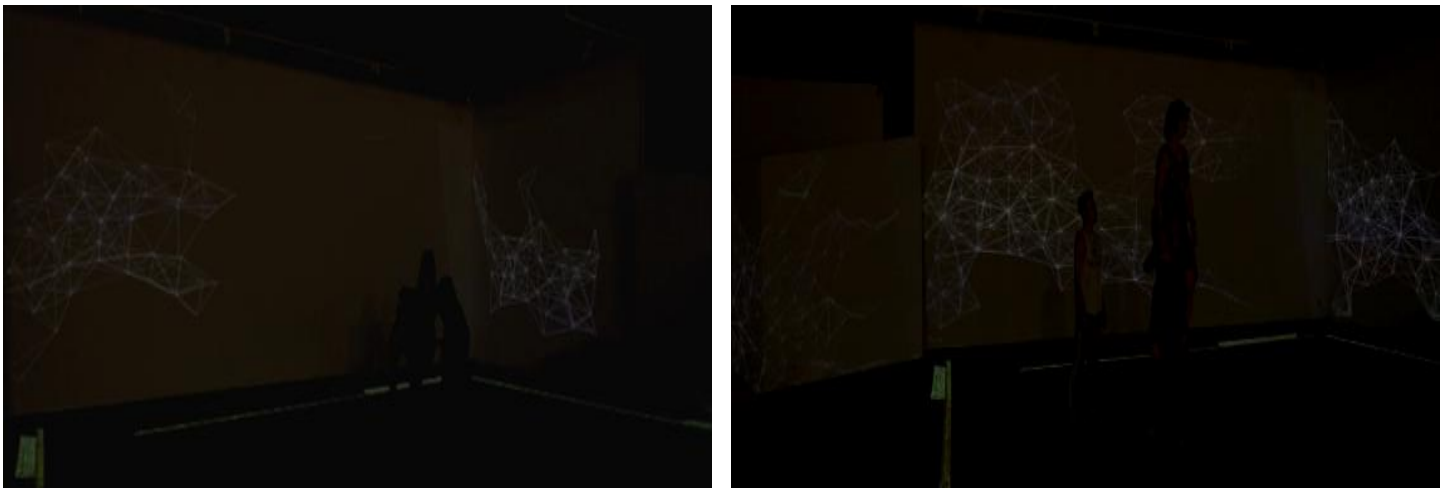


Fig. 6.8 Organic Mesh a)

En la siguiente imagen se puede apreciar el efecto que producen los bailarines en la malla según sea su localización y la posición de sus brazos. Se pueden observar dos grandes zonas en las que la malla se ha roto (esquina superior izquierda de la pantalla central), una encima de la otra. Esto se debe a que la posición de ambos bailarines es prácticamente idéntica, mientras que los brazos del bailarín de más a la izquierda apunta hacia arriba y los del otro bailarín más hacia abajo en una posición intermedia.

Fig. 6.9 Organic Mesh b)

Finalmente Organic Mesh deja paso a Wave On A Sphere. Se exponen a continuación dos imágenes en las que se puede apreciar el estado normal de la esfera, mientras esta se encuentra girando por su movimiento propio, y también el momento en el que se produce un pulso. Este pulso buscará en este caso concreto al bailarín que en ese momento se encuentra más alejado de la esfera.

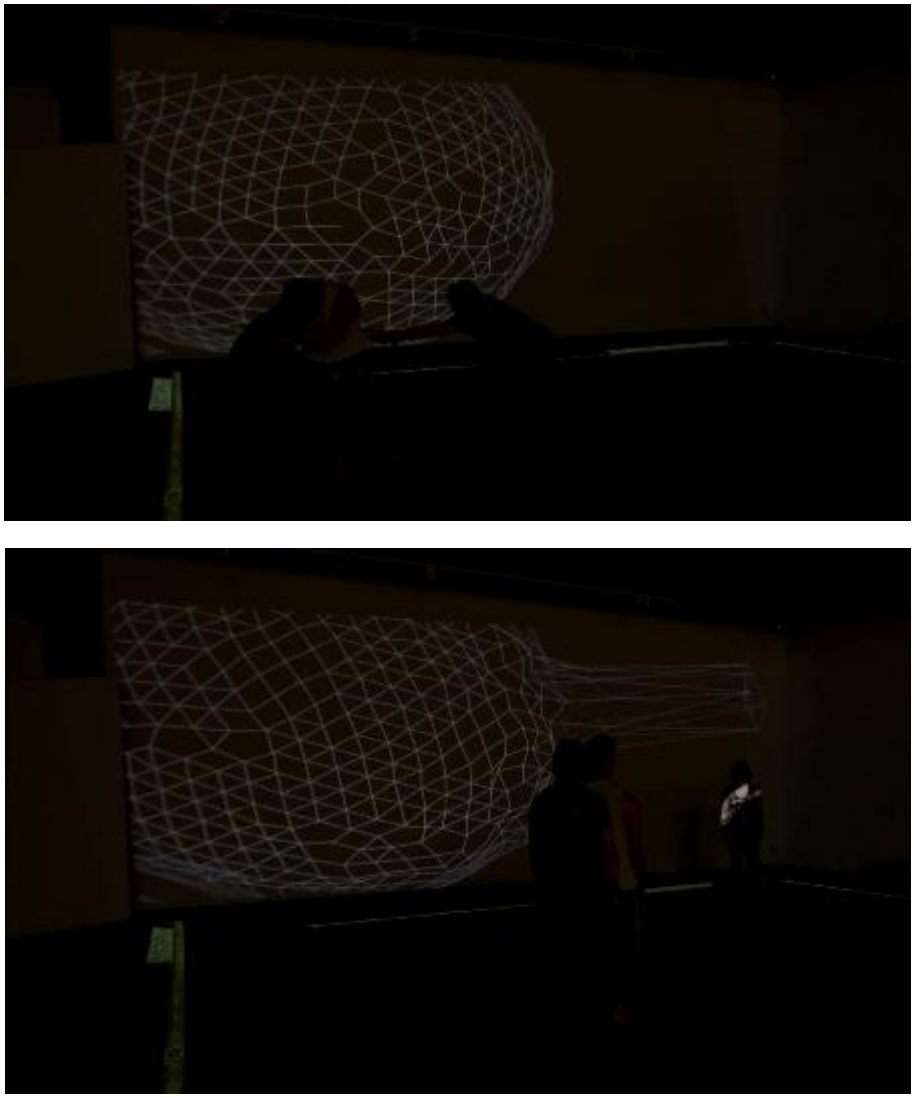


Fig. 6.10 Wave on a sphere

Capítulo 7

Conclusiones

7.1. Objetivos alcanzados

Con respecto a los objetivos alcanzados se podría decir que el resultado que se ha obtenido es satisfactorio, ya que se ha demostrado que es posible la integración de las nuevas tecnologías en el entorno del arte y que no es una tarea tan ardua como podría pensarse, si pensamos en la gran cantidad de oportunidades que esto ofrece. Se ha conseguido llegar un punto de hibridación en el que el bailarín, por medio de sus movimientos, es capaz de modificar, en tiempo real, el desarrollo de las diferentes proyecciones que tienen lugar durante la obra, permitiéndole experimentar nuevas vías de expresión que hasta ahora no se habían considerado.

Se ha trabajado con diferentes tipos de tecnologías, algunas ya conocidas y usadas en el entorno de la danza como proyectores, y otras de carácter muy innovador como los sensores de diferente tipo que se han empleado. En relación con estos últimos, los Ubisense son los que han limitado un poco más el desarrollo del proyecto debido al gran tiempo que requieren entre mediciones (1 segundo), ya que en un entorno dinámico como en el que nos encontramos es fundamental que los movimientos se reflejen lo más fielmente posible. Además estos sensores sufren un alto grado de imprecisión, que se ha intentado solucionar en la parte del código correspondiente (procesSignals). Por su parte los Xsens no tienen el problema de la frecuencia de las mediciones ya que el usuario puede seleccionar este intervalo a su gusto según el fin para el que los use. Cuando se consiguió incluir los Xsens en el proyecto supuso un gran salto de calidad, ya que no solo abría un gran abanico de posibilidades a nivel de interacciones bailarín-entorno, sino que hacía mucho más tangible cual era el efecto que el bailarín estaba produciendo con su danza.

Ha sido fundamental el desarrollo en paralelo de la parte artística y de la parte de ingeniería, así como el trabajo en equipo de bailarines e ingenieros. Se ha trabajado desde ambas partes para conseguir adecuarse a necesidades, deseos y limitaciones de la otra. Por parte del equipo de ingeniería se ha tratado de entorpecer lo menos posible el desarrollo artístico de la obra, procurando que los sensores estuvieran colocados en puntos que no molestaran a los bailarines pero que también captaran fielmente sus movimientos, al final se ha decidido que los Xsens estén situados en la parte interior de las muñecas y los Ubisense colgados del cuello como una medalla. También se ha intentado desarrollar presentaciones que cubrieran las necesidades artísticas pero también adecuándose a las limitaciones de los medios y calendarios establecidos. En cuanto al equipo de baile, se ha tratado de adecuar ciertas partes de la coreografía para maximizar el respuesta que se podía obtener del entorno.

7.2. Trabajo futuro

La línea de trabajo que se podría seguir es muy prometedora. Personalmente sólo he aprendido un poco del complejo mundo de las nuevas tecnologías sensoriales y estas ya representan una cantidad de posibilidades muy grande, no solo en el campo de la danza, sino para otros entornos, como por ejemplo el de juegos educativos para niños o el estudio y seguimiento a distancia de personas con movilidad reducida o diferentes enfermedades. La gran diversidad de tecnología sensorial de la que se dispone hoy en día está haciendo que desde hace unos años se empiecen a dar casos de ingeniería cyborg[14], en los que personas con carencias sensoriales acoplan a su cuerpo diversos aparatos que les permiten suplirlas.

Desde el entorno del proyecto, el entorno artístico, las posibilidades que aportan las nuevas tecnologías se podría decir que son casi ilimitadas. Solo hay que pensar que en el proyecto se comenzó estudiando diferentes sensores y que únicamente se han utilizado los Xsens, más los Ubisens. Y que además, no se han explotado todas las magnitudes que los Xsens pueden medir.

Por esto, en un próximo trabajo en este campo, se podría comenzar por aprovechar al máximo los Xsens. También se podrían incluir más sensores, como los DataGlove, u otros sensores que no fueran acoplados físicamente al bailarín, sino que fueran capaces de tomar las medidas pertinentes a distancia, como los Kinect[15] o unos sensores de temperatura. Con esto, se conseguiría aumentar tremendamente las maneras en las que el bailarín podría transmitir lo que quiere expresar, buscando otros puntos de vista y consiguiendo una interacción con el entorno rica y cargada de detalles.

[14] Neil Harbisson (Londres, Inglaterra, 27 de julio de 1984) es un artista vanguardista y activista cíborg británico e irlandés residente en Nueva York. Es la primera persona en el mundo reconocida como cíborg por un gobierno (2004) y la primera persona con una antena implantada en la cabeza. La antena le permite escuchar los colores y percibir colores invisibles como infrarrojos y ultravioletas así como recibir imágenes, videos, música o llamadas telefónicas directamente a su cabeza desde aparatos externos como móviles o satélites.

[15] Kinect es un dispositivo, inicialmente pensado como un simple controlador de juego, que gracias a los componentes que lo integran: sensor de profundidad, cámara RGB, array de micrófonos y sensor de infrarrojos (emisor y receptor), es capaz de capturar el esqueleto humano, reconocerlo y posicionarlo en el plano.

Anexo I

Código de lectura de los Xsens

A continuación se expone el código que se comenta en el apartado 4.2, que se ha desarrollado a partir de librerías que incluían los sensores para el entorno de Matlab.

```
  1 %   Copyright (c) 2003-2016 Xsens Technologies B.V. or
  subsidiaries worldwide.
  2 %   All
  rights
  reserved.
  3
  4 %   Redistribution and use in source and binary forms,
with or without modification,
  5 %   are permitted provided that the following
  conditions are met:
  6
  7 %   1.  Redistributions of source code must retain the above
  copyright notice,
  8 %       this list of conditions and the
  following disclaimer.
  9
 10 %   2.  Redistributions in binary form must reproduce the above
 copyright notice,
 11 %       this list of conditions and the following disclaimer in
    the documentation
 12 %       and/or other materials provided with
 the distribution.
 13
 14 %   3.  Neither the names of the copyright holders nor
the the names of their contributors
 15 %       may be used to endorse or promote products derived
from this software without
 16 %       specific prior
 written permission.
 17
 18 %   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS" AND ANY
 19 %   EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF
 20 %   MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL
 21 %   THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL,
 22 %   SPECIAL, EXEMPLARY OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT
 23 %   OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION)
```

```
 24 %   HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY OR
 25 %   TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS
 26 %   SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
 OF SUCH DAMAGE.
 27
 28
 29 function mainMTwRTdataViewer
 30 %%-------
 HELP
 31 %
 32 % This script allows the user to understand the step-wise
procedure to get data from devices connected to
 33 % the Awinda station in wireless mode and collect data. It is also
    possible
 34 % to use this example with a wired
 connected MTw device.
 35 %
 36 % The code is divided
 into two parts:
 37 %
 38 % 1) The first part regards the situation in which the MTw are
 docked to
 39 % the Awinda
 station. In this part:
 40 %
 41 %           a) information about the MTw connected are provided
 42 %           b) a communication channel is opened making the Awinda
    station
 43 %           enabled to receive MTw connections (the user is asked
    to choose
 44 %           the channel number)
 45 %           c) at this point the user is asked to undock the MTw
    devices from the
 46 %           Awinda station and wait for them to be
 wireless connected
 47 %
 48 % 2) The second part regards the situation of using the MTw in
 wireless
```

```
 49 % mode, soon after the end of the part 1.
 50 %
 51 %           a) operational mode is activated
 52 %           b) the user is asked to choose a specific update rate (this might
depend on the number of MTw used. See
 53 %           datasheet for this information)
 54 %           c) measurement mode is activated
 55 %           d) data are extracted from the devices and displayed live in
 56 %           graphs
 57 %           e) Awinda station is then disabled
 58 %           f) recorded data are saved in a log file
 59 %
 60 %%-------- IMPORTANT NOTES
 61 %
 62 % - For the code to work properly, make sure the code folder is your current
directory in Matlab.
 63 %
 64 % - This code supports multiple MTw devices connected at a time to one Awinda
station (although the suggested max number of connected devices is 4).
 65 %
 66 % - This code supports both 32 and 64 bits Matlab version.
 67 %
 68 % - The code requires xsensdeviceapi_com32.dll or xsensdeviceapi_com64.dll to be
registered in the Windows
 69 %   register (this is done automatically during the Xsens MT SDK installation)
 70 %
 71
 72 %% Launching activex server
 73     switch computer
 74         case 'PCWIN'
 75             serverName = 'xsensdeviceapi_com32.IXsensDeviceApi';
 76         case 'PCWIN64'
 77             serverName = 'xsensdeviceapi_com64.IXsensDeviceApi';
 78     end
 79     h = actxserver(serverName);
 80     fprintf( '\n ActiveXsens server - activated \n');
 81
 82     version = h.XsControl_version;
 83     fprintf(' XDA version: %.0f.%.0f.%.0f\n',version{1:3})
 84     if length(version)>3
 85         fprintf(' XDA build: %.0f %s\n',version{4:5});
 86     end
 87
 88 %% Scanning connection ports
 89     % ports rescanned must be reopened
 90     p_br = h.XsScanner_scanPorts(0, 100, true, true);
 91     fprintf( '\n Connection ports - scanned \n');
 92
 93     % check using device id's what kind of devices are connected.
 94     isMtw = cellfun(@(x) h.XsDeviceId_isMtw(x),p_br(:,1));
 95     isDongle = cellfun(@(x) h.XsDeviceId_isAwindaDongle(x),p_br(:,1));
 96     isStation = cellfun(@(x) h.XsDeviceId_isAwindaStation(x),p_br(:,1));
 97
 98     if any(isDongle|isStation)
 99         fprintf('\n Example dongle or station\n')
100         dev = find(isDongle|isStation);
101         isMtw = false; % if a station or a dongle is connected give priority to
```

```
it.
102     elseif any(isMtw)
103         fprintf('\n Example MTw\n')
104         dev = find(isMtw);
105     else
106         fprintf('\n No device found. \n')
107         h.XsControl_close();
108         delete(h);
109         return
110     end
111 
112     % port scan gives back information about the device, use first device found.
113     deviceID = p_br{dev(1),1};
114     portS = p_br{dev(1),3};
115     baudRate = p_br{dev(1),4};
116 
117     devTypeStr = '';
118     if any(isMtw)
119         devTypeStr = 'MTw';
120     elseif any(isDongle)
121         devTypeStr = 'dongle';
122     else
123         assert(any(isStation))
124         devTypeStr = 'station';
125     end
126     fprintf('\n Found %s on port %s, with ID: %s and baudRate: %.0f \n',
devTypeStr, portS, dec2hex(deviceID), baudRate);
127 
128     % open port
129     if ~h.XsControl_openPort(portS, baudRate, 0 ,true)
130         fprintf('\n Unable to open port %s. \n', portS);
131         h.XsControl_close();
132         delete(h);
133         return;
134     end
135 
136 %% Initialize Master Device
137     % get device handle.
138     device = h.XsControl_device(deviceID);
139 
140     % To be able to get orientation data from a MTw, the filter in the
141     % software needs to be turned on:
142     h.XsDevice_setOptions(device, h.XsOption_XSO_All, 0);
143     h.XsDevice_gotoConfig(device);
144 
145     % Get the list of supported update rates and let the user choose the
146     % one to set
147     supportUpdateRates = h.XsDevice_supportedUpdateRates(device, h.
XsDataIdentifier_XDI_None);
148     upRateIndex = [];
149     while(isempty(upRateIndex))
150         fprintf('\n The supported update rates are: ');
151         fprintf('%i, ',supportUpdateRates{:});
152         fprintf('\n');
153         selectedUpdateRate = input( Which update rate do you want to use ? );
154         if (isempty(selectedUpdateRate))
155             continue;
```

```
156         end
157         upRateIndex = find([supportUpdateRates{:}] == selectedUpdateRate);
158     end
159
160     % set the choosen update rate
161     h.XsDevice_setUpdateRate(device, supportUpdateRates{upRateIndex});
162
163     if(any(isDongle|isStation))
164         % Let the user choose the desired radio channel
165         availableRadioChannels = [11 12 13 14 15 16 17 18 19 20 21 22 23 24 25];
166         upRadioChIndex = [];
167         while(isempty(upRadioChIndex))
168             fprintf('\n The available radio channels are: ');
169             fprintf('%i, ',availableRadioChannels);
170             fprintf('\n');
171             selectedRadioCh = input( Which radio channel do you want to use ? );
172             if (isempty(selectedRadioCh))
173                 continue;
174             end
175             upRadioChIndex = find(availableRadioChannels == selectedRadioCh);
176         end
177
178         try
179             % enable radio
180             h.XsDevice_enableRadio(device, availableRadioChannels↙
(upRadioChIndex));
181         catch
182             fprintf(' Radio is still turned on, remove device from pc and try
again')
183         end % if radio is still on, this call will give an error
184
185         input('\n Undock the MTw devices from the Awinda station and wait until
the devices are connected (synced leds), then press enter... \n');
186
187         % check which devices are found
188         children = h.XsDevice_children(device);
189
190         % make sure at least one sensor is connected.
191         devIdAll = cellfun(@(x) dec2hex(h.XsDevice_deviceId(x))↙
children,'uniformOutput',false);
192         % check connected sensors, see which are accepted and which are
193         % rejected.
194         [devicesUsed, devIdUsed, nDevs] = checkConnectedSensors(devIdAll);
195         fprintf(' Used device: %s \n',devIdUsed{:});
196     else
197         assert(any(isMtw))
198         nDevs = 1; % only one device available
199         devIdUsed = {dec2hex(deviceID)};
200         devicesUsed = {device};
201     end
202
203 %% Entering measurement mode
204     fprintf('\n Activate measurement mode \n');
205     % goto measurement mode
206     output = h.XsDevice_gotoMeasurement(device);
207
208     % display radio connection information
```

```
209     if(any(isDongle|isStation))
210         fprintf('\n Connection has been established on channel %i with an update
rate of %i Hz\n', h.XsDevice_radioChannel(device), h.XsDevice_updateRate(device));
211     else
212         assert(any(isMtw))
213         fprintf('\n Connection has been established with an update rate of %i
Hz\n', h.XsDevice_updateRate(device));
214     end
215
216     % create figure for showing data
217     [t, dataPlot, linePlot, packetCounter] =
createFigForDisplay(nDevs, devIdUsed);
218
219     % check filter profiles
220     if ~isempty(devicesUsed)
221         availableProfiles = h.XsDevice_availableXdaFilterProfiles(devicesUsed{1});
222         usedProfile = h.XsDevice_xdaFilterProfile(devicesUsed{1});
223         number = usedProfile{1};
224         version = usedProfile{2};
225         name = usedProfile{3};
226         fprintf('\n Used profile: %s(%.0f), version %.0f.\n',name,number,version)
227         if any([availableProfiles{:,1}] ~= number)
228             fprintf('\n Other available profiles are: \n')
229             for iP=1:size(availableProfiles,1)
230                 fprintf(' Profile: %s(%.0f), version %.0f.\n',availableProfiles
{iP,3},availableProfiles{iP,1},availableProfiles{iP,2})
231             end
232         end
233    end
234
235     if output
236         % create log file
237         h.XsDevice_createLogFile(device,'exampleLogfile.mtb');
238         fprintf('\n Logfile: %s
created\n',fullfile(cd,'exampleLogfile.mtb')); 239
240         % start recording
241         h.XsDevice_startRecording(device);
242         % register onLiveDataAvailable event
243         h.registerevent({'onLiveDataAvailable',@handleData});
244         h.setCallbackOption(h.XsComCallbackOptions_XSC_LivePacket,
h. XsComCallbackOptions_XSC_None);
245         % event handler will call stopAll when limit is reached
            input('\n Press enter to stop measurement.\n');
246
 247
248     else
249         fprintf('\n Problems with going to measurement\n)
250     end
251     stopAl;
252
253 %% Event handler
254     function handleData(varargin)
255         % callback function for event: onLiveDataAvailable
256         dataPacket = varargin{3}{2};
257         deviceFound =
varargin{3}{1};
258
```

```
259         iDev = find(cellfun(@(x) x==deviceFound, devicesUsed));
260         if isempty(t{iDev})
261             t{iDev} = 1;
262         else
263             t{iDev} = [t{iDev} t{iDev}(end)+1];
264         end
265         if dataPacket
266             if h.XsDataPacket_containsOrientation(dataPacket)
267                 oriC = cell2mat(h.XsDataPacket_orientationEuler_1(dataPacket));
268                 packetCounter(iDev) = packetCounter(iDev)+1;
269                 dataPlot{iDev} = [dataPlot{iDev} oriC];
270             end
271
272             tst = cell2mat(h.XsDevice_children(device));
273             sup = cell2mat(cellfun(@(x)
dec2hex(h.XsDevice_deviceId(x)), children,'uniformOutput',false));
274             sup =str2num(sup);
275
276             id =sup(find(tst==deviceFound));
277
278             if h.XsDataPacket_containsOrientation(dataPacket)
279                 %acc = cell2mat(h.XsDataPacket_calibratedData(dataPacket));
280                 freeAcc = cell2mat(h.XsDataPacket_freeAcceleration(dataPacket));
281                 %acc3 = cell2mat(h.XsDataPacket_rawAcceleration(dataPacket));
282                 %acc4 = cell2mat(h.XsDataPacket_rawData(dataPacket));
283                 socket = tcpip('localhost', 32000, 'NetworkRole', 'client');
284                 fopen(socket);
285                 sensorId = mod(id,100);
286                 message =
[num2str(sensorId);';',num2str(oriC(1)),';',num2str(oriC
(2)),';',num2str(oriC(3)),';',num2str(freeAcc(1)),';',num2str(freeAcc(2)),';',num2str
(freeAcc(3))];
287                 fwrite(socket, message,'char');
288                 fclose(socket);
289                 delete(socket);
290                 clear socket
291                 freeAcc'
292                 fprintf('------');
293             end
294
295             h.liveDataPacketHandled(deviceFound,
dataPacket);
296
297             % draw
298             if packetCounter(iDev)>10
299                 if length(t) > 1000
300                     t{iDev}(1:end-990) = [];
301                     dataPlot{iDev}(:,1:end-990) = [];
302                     set(get(linePlot{iDev}(1),'parent'),'xlim',[t{iDev}(1)
t{iDev} (end)+10]);
303                 end
304                 for i=1:3
305                     set(linePlot{iDev}(i),'xData',t{iDev},'ydata',dataPlot{iDev}
(i,:));
306                 end
307                 packetCounter(iDev) = 0;
308             end
```

```
309         end
310   end
311
312     function stopAll
313         % close everything in the right way
314         if ~isempty(h.eventlisteners)
315             h.unregisterevent({'onLiveDataAvailable',@handleData});
316         h.setCallbackOption(h.XsComCallbackOptions_XSC_None,
h. XsComCallbackOptions_XSC_LivePacket);
317         end
318         % stop recording, showing data
319         fprintf('\n Stop recording, go to config mode \n);
320         h.XsDevice_stopRecording(device);
321         h.XsDevice_gotoConfig(device);
322         % disable radio for station or dongle
323         if any(isStation|isDongle)
324             h.XsDevice_disableRadio(device);
325         end
326         % close log file
327         fprintf('\n Close log file \n');
328         h.XsDevice_closeLogFile(device);
329         % on close, devices go to config mode.
330         fprintf('\n Close port \n');
331         % close port
332         h.XsControl_closePort(portS);
333         % close handle
334         h.XsControl_close();
335         % delete handle
336         delete(h);
337     end
338
339     function [devicesUsed, devIdUsed, nDevs] = checkConnectedSensors(devIdAll)
340         childUsed = false(size(children));
341         if isempty(children)
342             fprintf('\n No devices found \n')
343             stopAll
344             error('MTw:example:devicdes','No devices found')
345         else
346             % check which sensors are connected
347             for ic=1:length(children)
348                 if h.XsDevice_connectivityState(children{ic}) == h.
XsConnectivityState_XCS_Wireless
349                     childUsed(ic) = true;
350                 end
351             end
352             % show wich sensors are connected
353             fprintf('\n Devices rejected:\n')
354             rejects = devIdAll(~childUsed);
355             I=0;
356             for i=1:length(rejects)
357                 I = find(strcmp(devIdAll, rejects{i}));
358                 fprintf(' %d - %s\n', I,rejects{i})
359             end
360             fprintf('\n Devices accepted:\n')
361             accepted = devIdAll(childUsed);
362             for i=1:length(accepted)
363                 I = find(strcmp(devIdAll, accepted{i}));
```

```
364                 fprintf(' %d - %s\n', I,accepted{i})
365             end
366             str = input('\n Keep current status?(y/n) \n','s');
367             change = [];
368             if strcmp(str,'n')
369                 str = input('\n Type the numbers of the sensors (csv list, e.g.↙
"1,2,3") from which status should be changed \n (if accepted than reject or the other↙
way around):\n','s');
370                 change = str2double(regexp(str,',', 'split'));
371                 for iR=1:length(change)
372                     if childUsed(change(iR))
373                         % reject sensors
374                         h.XsDevice_rejectConnection(children{change(iR)});
375                         childUsed(change(iR)) = false;
376                     else
377                         % accept sensors
378                         h.XsDevice_acceptConnection(children{change(iR)});
379                         childUsed(change(iR)) = true;
380                     end
381                 end
382             end
383             % if no device is connected, give error
384             if sum(childUsed) == 0
385                 stopAll
386                 error('MTw:example:devicdes','No devices connected')
387             end
388             % if sensors are rejected or accepted check blinking leds again
389             if ~isempty(change)
390                 input('\n When sensors are connected (synced leds), press enter..↙
\n');
391             end
392         end
393         devicesUsed = children(childUsed);
394         devIdUsed = devIdAll(childUsed);
395         nDevs = sum(childUsed);
396     end
397  end
398
399 %% Helper function to create figure for display
400 function [t, dataPlot, linePlot, packetCounter] = createFigForDisplay(nDevs,↙
deviceIds)
401
402         [dataPlot{1:nDevs}] = deal([]);
403         [linePlot{1:nDevs}] = deal([]);
404         [t{1:nDevs}] =
deal([]); 405
406         %% not more than 6 devices per plot
407         nFigs = ceil(nDevs/6);
408         devPerFig = ceil(nDevs/nFigs);
409         m = ceil(sqrt(devPerFig));
410         n = ceil(devPerFig/m);
411         lDev = 0;
412         for iFig=1:nFigs
413             figure('name',['Example MTw_' num2str(iFig)])
414             iPlot = 0;
415             for iDev = lDev+1:min(iFig*devPerFig, nDevs)
416                 iPlot = iPlot+1;
```

```
417                 ax = subplot(m,n,iPlot);
418                 linePlot{iDev} = plot(ax, 0,[NaN NaN NaN]);
419                 title(['Orientation data ' deviceIds{iDev}]),
xlabel('sample'), ylabel('euler (deg)')
420                 legend(ax,  'roll','pitch','yaw');
421             end
422             lDev = iDev;
423         end
424         packetCounter = zeros(nDevs,1);
425 end
```

Anexo II

Host

II.1. Dance Host

```
1  public class Dance {
2      public String id;
3      public float x;
4      public float y;
5      public float destinoX;
6      public float destinoY;
7  }
8
9  DisplaysHost displaysClient;
10
11 float RATE = 24.0f;
12 int w = 1280;
13 int h = 768;
14 int l1 = 0;
15 int l2 = 0;
16 int l3 = 0;
17 int l4 = 0;
18 int fot = 0;
19 int display;
20 float tx=0;
21 float ty=0;
22
23 float rep = 1;
24 float movx;
25 float movy;
26
27 // tags de ubisense
28 PVector tag089=new PVector(0,0);
29 PVector tag143=new PVector(0,0);
30 PVector tag248=new PVector(0,0);
31
32 PVector d_tag089=new PVector(0,0,0); //x pared1, y pared2, z pared3 distancia tag
33 PVector d_tag143=new PVector(0,0,0);
34 PVector d_tag248=new PVector(0,0,0);
35 int pulso_tag089=30;
36 int pulso_tag143=60;
37 int pulso_tag248=60;
```

```
boolean p_tag089=true;
boolean p_tag143=true;
boolean p_tag248=true;


//tags de xSense
PVector xSens20=new PVector(0,0);
PVector xSens21=new PVector(0,0);
PVector xSens22=new PVector(0,0);
PVector xSens23=new PVector(0,0);
PVector xSens24=new PVector(0,0);
PVector xSens25=new PVector(0,0);

PVector d_xSens20=new PVector(0,0,0); //x pared1, y pared2, z pared3 distancia tag
PVector d_xSens21=new PVector(0,0,0);
PVector d_xSens22=new PVector(0,0,0);
PVector d_xSens23=new PVector(0,0,0);
PVector d_xSens24=new PVector(0,0,0);
PVector d_xSens25=new PVector(0,0,0);
int pulso_xSens20=30;
int pulso_xSens21=60;
int pulso_xSens22=90;
 boolean p_xSens20=true;
 boolean p_xSens21=true;
 boolean
p_xSens22=true; 63

 //float fltMaxX1=0.0f;
 //float fltMinX1=1.0f;
 //float fltMaxY1=0.0f;
 //float
fltMinY1=1.0f; 69
 int intCont = 0;
 float fltSum = 0;
 float fltMaxAcc=0.0f;
 float fltMinAcc=1.0f;
 float fltMaxRot=0.0f;
 float fltMinRot=1.0f;
 float fltMean =
0; 77
 void setup() {
   // Keystone will only work with P3D or OPENGL renderers,
   // since it relies on texture mapping to
deform 81  size(640, 380, P3D);
   frameRate(RATE);
   background(100)
; 84
   // virtual screens
   displaysClient = new
DisplaysHost("host"); 87
   movx = random(-6,6);
   movy = random(-
6,6); 90
   noStroke();
   fill(0);
   ellipseMode(RADIUS);
   blendMode(ADD);
```

```
 95 }
 96
 97 void draw() {
 98   //Pintar dibujable
 99   //setTestData();
100   //Ubisense
101   if ( displaysClient.isDrawing ){
102     displaysClient.dibujar("tag089",d_tag089.x,d_tag089.y,displaysClient.w,768,0,"
center");//Dancer 1
103     displaysClient.dibujar("tag143",d_tag143.x,d_tag143.y,displaysClient.w,768,0,"
center");//Dancer 2
104     displaysClient.dibujar("tag248",d_tag248.x,d_tag248.y,displaysClient.w,768,0,"
center");//Dancer 3
105     println("X1="+d_tag089.x+"\tY1="+d_tag089.y+"\tZ1="+d_tag089.z);
106     println("X2="+d_tag143.x+"\tY2="+d_tag143.y+"\tZ2="+d_tag143.z);
107     println("X3="+d_tag248.x+"\tY3="+d_tag248.y+"\tZ3="+d_tag248.z);
108     //display = 2;
109     //Xsense
110     //acceleration, rotation
111     displaysClient.dibujar("xSens20",d_xSens20.x,d_xSens20.y,displaysClient.w,
768,0,"center");//Dancer 1 hand A
112     displaysClient.dibujar("xSens21",d_xSens21.x,d_xSens21.y,displaysClient.w,
768,0,"center");//Dancer 1 hand B
113     displaysClient.dibujar("xSens22",d_xSens22.x,d_xSens22.y,displaysClient.w,
768,0,"center");//Dancer 2 hand A
114     displaysClient.dibujar("xSens23",d_xSens23.x,d_xSens20.y,displaysClient.w,
768,0,"center");//Dancer 2 hand B
115     displaysClient.dibujar("xSens24",d_xSens24.x,d_xSens21.y,displaysClient.w,
768,0,"center");//Dancer 3 hand A
116     displaysClient.dibujar("xSens25",d_xSens25.x,d_xSens22.y,displaysClient.w,
768,0,"center");//Dancer 3 hand B
117     println("A1a="+d_xSens20.x+"\tR1a="+d_xSens20.y);
118     println("A1b="+d_xSens21.x+"\tR1b="+d_xSens21.y);
119     println("A2a="+d_xSens22.x+"\tR2a="+d_xSens22.y);
120     println("A2b="+d_xSens23.x+"\tR2b="+d_xSens23.y);
121     println("A3a="+d_xSens24.x+"\tR3a="+d_xSens24.y);
122     println("A3b="+d_xSens25.x+"\tR3b="+d_xSens25.y);
123   }
124 }125
126 void setTestData(){
127   d_tag089 = new PVector(mouseX*2, mouseY*2);
128   d_tag143 = new PVector((mouseX*2)+200, mouseY*2);
129   d_tag248 = new PVector((mouseX*2)+400, mouseY*2); 130
131   d_xSens20 = new PVector(random(1,100), random(1,100));
132   d_xSens21 = new PVector(random(1,100), random(1,100));
133   d_xSens22 = new PVector(random(80,100), random(80,100));
134   d_xSens23 = new PVector(random(1,80), random(1,80));
135   d_xSens24 = new PVector(random(1,100), random(1,100));
136   d_xSens25 = new PVector(random(1,100), random(1,100));
137 }138
139 void mensajeRecibido(OscMessage theOscMessage){
140   //println("THE_OSC_MESSAGE=", theOscMessage.addrPattern());
141   if (theOscMessage.addrPattern().equals("/sensor/1D/xsens")) {
142     int id=theOscMessage.get(1).intValue();
143     //println("id=", id); 144
      if (id==20.0) {
145
```

```
146       //if (fltMinRot > theOscMessage.get(6).floatValue()){fltMinRot =
theOscMessage.get(3).floatValue();}
147       //if (fltMaxRot < theOscMessage.get(6).floatValue()){fltMaxRot =
theOscMessage.get(3).floatValue();}
148       //if (fltMinRot > theOscMessage.get(7).floatValue()){fltMinRot =
theOscMessage.get(4).floatValue();}
149       //if (fltMaxRot < theOscMessage.get(7).floatValue()){fltMaxRot =
theOscMessage.get(4).floatValue();}
150       //if (fltMinRot > theOscMessage.get(8).floatValue()){fltMinRot =
theOscMessage.get(5).floatValue();}
151       //if (fltMaxRot < theOscMessage.get(8).floatValue()){fltMaxRot =
theOscMessage.get(5).floatValue();}
152       //if (fltMinAcc > theOscMessage.get(6).floatValue()){fltMinAcc =
theOscMessage.get(6).floatValue();}
153       //if (fltMaxAcc < theOscMessage.get(6).floatValue()){fltMaxAcc =
theOscMessage.get(6).floatValue();}
154       //if (fltMinAcc > theOscMessage.get(7).floatValue()){fltMinAcc =
theOscMessage.get(7).floatValue();}
155       //if (fltMaxAcc < theOscMessage.get(7).floatValue()){fltMaxAcc =
theOscMessage.get(7).floatValue();}
156       //if (fltMinAcc > theOscMessage.get(8).floatValue()){fltMinAcc =
theOscMessage.get(8).floatValue();}
157       //if (fltMaxAcc < theOscMessage.get(8).floatValue()){fltMaxAcc =
theOscMessage.get(8).floatValue();}
158       //intCont += 3;
159       //fltSum += theOscMessage.get(6).floatValue();
160       //fltSum += theOscMessage.get(7).floatValue();
161       //fltSum += theOscMessage.get(8).floatValue();
162       //fltMean = fltSum/intCont;
163       //println("id="+id+"\tminAcc: "+fltMinAcc+"\tmaxAcc: "+fltMaxAcc);
164       //println("id="+id+"\tminRot: "+fltMinRot+"\tmaxRot: "+fltMaxRot);
165       //println("id="+id+"\tMean: "+fltMean);
166       //id=20   minAcc: -32.2936   maxAcc:
49.515
167       //id=20   minRot: -134.4615   maxRot: 73.4169
168       //id=20   Mean: -0.03960795
169
170       //float rotx=theOscMessage.get(3).floatValue();
171       //float roty=theOscMessage.get(4).floatValue();
172       //float
rotz=theOscMessage.get(5).floatValue(); 173
174       d_xSens20.x = accXsens
(theOscMessage.get(6).floatValue(),theOscMessage.get
(7).floatValue(),theOscMessage.get(8).floatValue());
175
176       //float accx=map(theOscMessage.get(6).floatValue(),-13,14,0,100);
177       //float accy=map(theOscMessage.get(7).floatValue(),-13,14,0,100);
178       //float accz=map(theOscMessage.get(8).floatValue(),-
13,14,0,100); 179
180       //println("id="+id+"\taccx="+accx+"\taccy="+accy+"\taccz="+accz);
181       //println("id="+id+"\trotx="+rotx+"\troty="+roty+"\trotz="+rotz);
182
183       //d_xSens20.x=(accx+accy+accz)/3;
184       //d_xSens20.x=clase.funcion(theOscMessage.get(6).floatValue(),theOscMessage.
get(7).floatValue(),theOscMessage.get(8).floatValue());
185
186       //logaritmo de la raiz cuadrada de la suma de los cuadrados de
```

```
las aceleraciones
187       //float sum=pow(rotx,2)+pow(roty,2)+pow(rotz,2);
188       //float raz=sqrt(sum);
189       //raz=log(raz);
190       d_xSens20.y=rotXsens(theOscMessage.get(4).floatValue());
191       //d_xSens20.y=clase.funcion2(theOscMessage.get(3).floatValue(),
theOscMessage.get(4).floatValue(),theOscMessage.get(5).floatValue());
192     }
193     if (id==21.0) {
194       //float rotx=theOscMessage.get(3).floatValue();
195       //float roty=theOscMessage.get(4).floatValue();
196       //float rotz=theOscMessage.get(5).floatValue();
197       //float accx=map(theOscMessage.get(6).floatValue(),-10,10,0,100);
198       //float accy=map(theOscMessage.get(7).floatValue(),-10,10,0,100);
199       //float accz=map(theOscMessage.get(8).floatValue(),-
10,10,0,100); 200
201       d_xSens21.x=accXsens
(theOscMessage.get(6).floatValue(),theOscMessage.get
(7).floatValue(),theOscMessage.get(8).floatValue());
202
203       //float sum=pow(rotx,2)+pow(roty,2)+pow(rotz,2);
204       //float raz=sqrt(sum);
205       //raz=log(raz);
206       d_xSens21.y=rotXsens(theOscMessage.get(4).floatValue());
207     }
208     if (id==22.0) {
209       //float rotx=theOscMessage.get(3).floatValue();
210       //float roty=theOscMessage.get(4).floatValue();
211       //float rotz=theOscMessage.get(5).floatValue();
212       //float accx=map(theOscMessage.get(6).floatValue(),-10,10,0,100);
213       //float accy=map(theOscMessage.get(7).floatValue(),-10,10,0,100);
214       //float accz=map(theOscMessage.get(8).floatValue(),-10,10,0,100);
215
216       d_xSens22.x=accXsens    (theOscMessage.get(6).floatValue(),theOscMessage.get↙
(7).floatValue(),theOscMessage.get(8).floatValue());
217
218       //float sum=pow(rotx,2)+pow(roty,2)+pow(rotz,2);
219       //float raz=sqrt(sum);
220       //raz=log(raz);
221       d_xSens22.y=rotXsens(theOscMessage.get(4).floatValue());
222     }
223     if (id==23.0) {
224       //float rotx=theOscMessage.get(3).floatValue();
225       //float roty=theOscMessage.get(4).floatValue();
226       //float rotz=theOscMessage.get(5).floatValue();
227       //float accx=map(theOscMessage.get(6).floatValue(),-10,10,0,100);
228       //float accy=map(theOscMessage.get(7).floatValue(),-10,10,0,100);
229       //float accz=map(theOscMessage.get(8).floatValue(),-10,10,0,100);
230
231       d_xSens23.x=accXsens    (theOscMessage.get(6).floatValue(),theOscMessage.get
(7).floatValue(),theOscMessage.get(8).floatValue());
232
233       //float sum=pow(rotx,2)+pow(roty,2)+pow(rotz,2);
234       //float raz=sqrt(sum);
235       //raz=log(raz);
236       d_xSens23.y=rotXsens(theOscMessage.get(4).floatValue());
237     }
```

```
238     if (id==24.0) {
239       //float rotx=theOscMessage.get(3).floatValue();
240       //float roty=theOscMessage.get(4).floatValue();
241       //float rotz=theOscMessage.get(5).floatValue();
242       //float accx=map(theOscMessage.get(6).floatValue(),-10,10,0,100);
243       //float accy=map(theOscMessage.get(7).floatValue(),-10,10,0,100);
244       //float accz=map(theOscMessage.get(8).floatValue(),-10,10,0,100);
245
246       d_xSens24.x=accXsens    (theOscMessage.get(6).floatValue(),theOscMessage.get
(7).floatValue(),theOscMessage.get(8).floatValue());
247
248       //float sum=pow(rotx,2)+pow(roty,2)+pow(rotz,2);
249       //float raz=sqrt(sum);
250       //raz=log(raz);
251       d_xSens24.y=rotXsens(theOscMessage.get(4).floatValue());
252     }
253     if (id==19.0) {
254       //float rotx=theOscMessage.get(3).floatValue();
255       //float roty=theOscMessage.get(4).floatValue();
256       //float rotz=theOscMessage.get(5).floatValue();
257       //float accx=map(theOscMessage.get(6).floatValue(),-10,10,0,100);
258       //float accy=map(theOscMessage.get(7).floatValue(),-10,10,0,100);
259       //float accz=map(theOscMessage.get(8).floatValue(),-10,10,0,100);
260
261       d_xSens25.x=accXsens    (theOscMessage.get(6).floatValue(),theOscMessage.get
(7).floatValue(),theOscMessage.get(8).floatValue());
262
263       //float sum=pow(rotx,2)+pow(roty,2)+pow(rotz,2);
264       //float raz=sqrt(sum);
265       //raz=log(raz);
266       d_xSens25.y=rotXsens(theOscMessage.get(4).floatValue());
267     }
268   }
269   if (theOscMessage.addrPattern().equals("/sensor/1D/gloves")) {
270     float glove=theOscMessage.get(3).floatValue();
271     //println("glove=", glove);
272   }
273   else if (theOscMessage.addrPattern().equals("/sensor/2D/ubisense")) {
274     String id=theOscMessage.get(1).stringValue();
275     //println("id=", id);
276     if (id.equals("089")) {
277       //println("tag 089  x=", theOscMessage.get(3).floatValue(),"    y=",
theOscMessage.get(4).floatValue());
278       //if (fltMinX1 > theOscMessage.get(3).floatValue()){fltMinX1 =
theOscMessage.get(3).floatValue();}
279       //if (fltMaxX1 < theOscMessage.get(3).floatValue()){fltMaxX1 =
theOscMessage.get(3).floatValue();}
280       //if (fltMinY1 > theOscMessage.get(4).floatValue()){fltMinY1 =
theOscMessage.get(4).floatValue();}
281       //if (fltMaxY1 < theOscMessage.get(4).floatValue()){fltMaxY1 =
theOscMessage.get(4).floatValue();}
282       //println("minX: "+fltMinX1+"\tmaxX: "+fltMaxX1);
283       //println("minY: "+fltMinY1+"\tmaxY: "+fltMaxY1);
284       tag089.x=map(theOscMessage.get(3).floatValue(), 0.007, 0.91, 0, w);
285       tag089.y=map(theOscMessage.get(4).floatValue(), 0.79, 0.17, 0, h/2); 286
          d_tag089.x=tag089.x;
287       d_tag089.y=tag089.y;
```

```
288       //println("X1="+d_tag089.x+"\tY1="+d_tag089.y+"\tZ1="+d_tag089.z);
289     }
290     if (id.equals("143")) {
291       tag143.x=map(theOscMessage.get(3).floatValue(), 0.007, 0.91, 0, w);
292       tag143.y=map(theOscMessage.get(4).floatValue(), 0.79, 0.17, 0, h/2); 293
          d_tag143.x=tag143.x;
294       d_tag143.y=tag143.y;
295       //println("X2="+d_tag143.x+"\tY2="+d_tag143.y+"\tZ2="+d_tag143.z);
296     }
297     if (id.equals("248")) {
298       tag248.x=map(theOscMessage.get(3).floatValue(), 0.007, 0.91, 0, w);
299       tag248.y=map(theOscMessage.get(4).floatValue(), 0.79, 0.17, 0, h/2); 300
          d_tag248.x=tag248.x;
301       d_tag248.y=tag248.y;
302       //println("X3="+d_tag248.x+"\tY3="+d_tag248.y+"\tZ3="+d_tag248.z);
303     }
304   }
305 } 306
307 void keyPressed() {
308   switch(key) {
309     case 32:
310       if ( displaysClient.isDrawing ){ displaysClient.endDraw(); }
311       else { displaysClient.beginDraw(""); }
312       break;
313     case ESC:
314       if ( displaysClient.isDrawing ){ displaysClient.endDraw(); }
315       displaysClient.disconnect();
316       super.stop();
317       break;
318     case '1'://rain
319       displaysClient.beginDraw("1");
320       break;
321     case '2'://rainTransition
322       displaysClient.beginDraw("2");
323       break;
324     case '3'://bubbles
325       displaysClient.beginDraw("3");
326       break;
327     case '4'://camBubblesTransition
328       displaysClient.beginDraw("4");
329       break;
330     case '5'://traces
331       displaysClient.beginDraw("5");
332       break;
333     case '6'://fadeCurtain
334       displaysClient.beginDraw("6");
335       break;
336     case '7'://smoke
337       displaysClient.beginDraw("7");
338       break;
339     case '8'://createMesh
340       displaysClient.beginDraw("8");
341       break;
342     case '9'://meshSphere
343       displaysClient.beginDraw("9");
344       break;
345     default:
```

```
346         break;
347     }
348 }
```

II.2. Displays Host

```
 1 import oscP5.*;
 2 import netP5.*;
 3
 4 class DisplaysHost { 5
 6   public class Display {
 7     public int x;
 8     public int y;
 9     public int w;
10   public int h; 11
     }
12
13   OscP5 oscP5;
14   NetAddress myBroadcastLocation;
15
16
17   //String ipAPI="155.210.155.229";
18   //String remoteIP="127.0.0.1";
19   String remoteIP="192.168.0.2";
20   //String remoteIP="155.210.155.229";
21   int listenPort=12000; // el puerto en el que se queda escuchando mensajes del
broadcaster
22
23   private XML file;
24   private int w, h;
25   public Display[] displayList;
26   private OscMessage myMessage; //Contiene el identificador del display que debe ↙
pintarlo y el identificador, el color y la posicion x e y de la mariposa a pintar
27   private String idDisplay;
28   private String tipo; //Posibles valores: host, sensor o display
29   public int id;
30
31   public boolean isDrawing = false;
32
33   DisplaysHost(String tipo) {
34
35     this.tipo = tipo;
36
37     file = loadXML("displays.xml"); // Se abre el fichero
38     if (file==null) {
39       println("Fail to load displays.xml file");
40     } else {
41     println("displays.xml succesfully loaded"); 42 }
43     XML[] currentDisplays = file.getChildren("currentDisplay");
44     for (int i = 0; i<currentDisplays.length; i++) {
45     this.id=currentDisplays[i].getInt("id"); 46
       }
47     println("display actual=", this.id);
48
49     XML[] virtualDisplays = file.getChildren("virtualDisplay");
50
```

```
 51      for (int i = 0; i<virtualDisplays.length; i++) {
 52        println("virtualdisplay=", virtualDisplays[i].getInt("width"));
 53        this.w=virtualDisplays[i].getInt("width");
 54        this.h=virtualDisplays[i].getInt("height");
 55        XML[] displays = virtualDisplays[i].getChildren("display");
 56        displayList=new Display[displays.length+1];
 57
 58        for (int j = 0; j<displays.length; j++) {
 59          displayList[displays[j].getInt("id")]=new Display();
 60          displayList[displays[j].getInt("id")].h=displays[j].getInt("height");
 61          displayList[displays[j].getInt("id")].w=displays[j].getInt("width");
 62          displayList[displays[j].getInt("id")].x=displays[j].getInt("x");
 63        displayList[displays[j].getInt("id")].y=displays[j].getInt("y"); 64
           }
65       }
66
 67      // OSC, aqui esta el puerto donde escucha mensajes
 68      oscP5 = new OscP5(this,
listenPort); 69
 70      // set the remote location to be the localhost on port
 71      myBroadcastLocation = new NetAddress(remoteIP, 33000);
 72    connect();
 73    }
74
75    //Mandar
Osc 76
77    public void beginDraw(String strPart)
{ 78      for(int i =1;i<4;i++){
 79        myMessage = new OscMessage("/display/BeginDrawDance");
 80        isDrawing = true;
 81        println(myMessage+"-"+strPart);
 82        if(!strPart.isEmpty()){
 83          myMessage.setAddrPattern("/display/BeginDrawDance");
 84          idDisplay = str(i);
 85          myMessage.add(idDisplay);
 86          println(idDisplay);
 87        myMessage.add(strPart);
 88        }
89          mandarMensaje(myMessage);
90        }
91    }
92
 93    public void endDraw() {
 94      myMessage = new OscMessage("/display/EndDrawDance");
 95      isDrawing = false;
 96      println(myMessage);
 97    mandarMensaje(myMessage);
 98    }
99
100    public void dibujar(String id, float x, float y, int ancho, int alto,
float angle, String pivot) {
101      OscBundle myBundle = new OscBundle();
102      OscMessage mensajeTransformado = new OscMessage("/display/DibujarDance");
103      idDisplay = ""; //ninguna pantalla pinta el dibujable (fuera de
las coordenadas de la pantalla total)
104      //Identificar el display que debe pintar el dibujable
105      for (int j = 1; j < displayList.length; j++)
```

```
106     {
107       if (pivot.equals("center")) {
108         if ((x + ancho/2) >= displayList[j].x && (x -ancho/2) <= (displayList[j].x
+ displayList[j].w)
109           && (y + alto/2) >= displayList[j].y && (y-alto/2) <= (displayList[j].y
+ displayList[j].h))
110         {
111           idDisplay = str(j);
112           PVector postTrasformada = new
PVector(abs(x),abs(y));//cambioCoordenadas (j, x, y);
113           mensajeTransformado.setAddrPattern("/display/DibujarDance");
114           mensajeTransformado.add(idDisplay);
115           mensajeTransformado.add(id);
116           mensajeTransformado.add(postTrasformada.x);
117           mensajeTransformado.add(postTrasformada.y);
118           mensajeTransformado.add(ancho);
119           mensajeTransformado.add(alto);
120           mensajeTransformado.add(angle);
121           if (pivot.equals("center"))
122             mensajeTransformado.add("C");
123           else
124           mensajeTransformado.add("E");
125           myBundle.add(mensajeTransformado);
126           mensajeTransformado.clear();
127         }
128       } else {
129         if ((x + ancho) >= displayList[j].x && x <= (displayList[j].x
+ displayList[j].w)
130           && (y + alto) >= displayList[j].y && y <= (displayList[j].y
+ displayList[j].h))
131         {
132           idDisplay = str(j);
133           PVector postTrasformada = new
PVector(abs(x),abs(y));//cambioCoordenadas (j, x, y);
134           mensajeTransformado.setAddrPattern("/display/DibujarB");
135           mensajeTransformado.add(idDisplay);
136           mensajeTransformado.add(id);
137           mensajeTransformado.add(postTrasformada.x);
138           mensajeTransformado.add(postTrasformada.y);
139           mensajeTransformado.add(ancho);
140           mensajeTransformado.add(alto);
141           mensajeTransformado.add(angle);
142           if (pivot.equals("center"))
143             mensajeTransformado.add("C");
144           else
145           mensajeTransformado.add("E");
146           myBundle.add(mensajeTransformado);
147           mensajeTransformado.clear();
148         }
149       }
150     }
151     mandarPaquete(myBundle);
152  }
153
154   PVector cambioCoordenadas(int id, float x, float y){

155     PVector resul = new PVector(0, 0);
```

```
156     //println("**cambio coordenadas**");
157     //println("original X=", x, " **** ", "y=",y);
158   //println("el display es", id, "***** x=", displaysClient.displayList[id].x, "
y=",  displaysClient.displayList[id].y);
159     resul.x = abs(x - displaysClient.displayList[id].x);
160     resul.y = abs(y   - displaysClient.displayList[id].y);
161     //println("cambiado X=", resul.x, " **** ", "y=", resul.y);
162     return resul;
163  }
164
165   public void connect() {
166     OscMessage m;
167     println("conectar");
168     m = new OscMessage("/" + tipo + "/connect", new Object[0]);
169     m.add(listenPort);
170     mandarMensaje(m);
171     delay(1000);
172   }
173
174   public void disconnect() {
175     OscMessage m;
176     m = new OscMessage("/" + tipo + "/disconnect", new Object[0]);
177     m.add(listenPort);
178     mandarMensaje(m);
179     println("disconneted");
180   }
181
182   void mandarMensaje(OscMessage myMessage) {
183     // send the message
184     oscP5.send(myMessage, myBroadcastLocation);
185   }
186
187   void mandarPaquete(OscBundle myMessage) {
188     // send the message
189     oscP5.send(myMessage, myBroadcastLocation);
190   }
 191
192   /* incoming osc message are forwarded to the oscEvent method. */
193   void oscEvent(OscMessage theOscMessage) {
194     mensajeRecibido(theOscMessage); //es necesarioque esta funcion este
impremetada en el main
195   }
196 }
```

II.3. Proces Signals

```
boolean modoDebug = true;
float   ubisenseOrigTime = 0;
float[] ubisenseOrigVal  = new float[2];
public float[] getUbisenseOrig() {
  if (modoDebug) {
    float t = getTimeStamp();
    if (t>ubisenseOrigTime+1000) {
      ubisenseOrigTime = t;
      ubisenseOrigVal[0] = mouseX;
    ubisenseOrigVal[1] = mouseY; 11   }
    return ubisenseOrigVal; 13   }
  // Valores de los sensores
  return null; 16 }

//////////////////
float[] ubiTimes = new float[2];
float[] ubiVals0 = new float[2];
float[] ubiVals1 = new float[2];
float   ubiDifTime = 0;
float[] ubiDifVals = new float[2]; 24

public float getTimeStamp() {
  return millis()+1000*(second()+60*(minute()+60*hour())); 28 }


public float[] getUbisense() {
  float[] inNew = getUbisenseOrig();
  float t = getTimeStamp();
  if (inNew[0]!=ubiVals1[0] || inNew[1]!=ubiVals1[1]) {
    ubiTimes[0] = ubiTimes[1];
    ubiTimes[1] = t;
    ubiVals0[0] = ubiVals1[0];
    ubiVals0[1] = ubiVals1[1];
    ubiVals1[0] = inNew[0];
    ubiVals1[1] = inNew[1];
    ubiDifTime = ubiTimes[1]-ubiTimes[0];
    ubiDifVals[0] = ubiVals1[0]-ubiVals0[0];
  ubiDifVals[1] = ubiVals1[1]-ubiVals0[1]; 44     }
  float[] res = new float[2];
  float lambda = 2*(t-ubiTimes[1])/ubiDifTime;
  if (lambda>1) { lambda=1; }
  res[0] = (1-lambda)*ubiVals0[0] + lambda*ubiVals1[0];
  res[1] = (1-lambda)*ubiVals0[1] + lambda*ubiVals1[1];
  return res; 51 }


////////////// Procesado Xsens 55
float maxAccX = 12;
float minAccX = -12;
float maxAccY = 12;
```

```
59 float minAccY = -12;
60 float maxAccZ = 12;
 61 float minAccZ = -12;
 62 float pitchMax = 85;
 63 float pitchMin = -85;
 64
 65 public float accXsens(float accX, float accY, float accZ){ 66
 67   if(accX < minAccX){accX = minAccX;}     //Ajuste alos maximos y minimos
 68   if(accX > maxAccX){accX = maxAccX;}
 69   if(accY < minAccY){accY = minAccY;}
 70   if(accY > maxAccY){accY = maxAccY;}
 71   if(accZ < minAccZ){accZ = minAccZ;}
 72   if(accZ > maxAccZ){accZ = maxAccZ;}
 73
 74   if(accX > 0){accX = map(accX,0,maxAccX,0,50);}   //mapeo devalores
 75   if(accX < 0){accX = map(accX,0,minAccX,0,-50);}
 76   if(accY > 0){accY = map(accY,0,maxAccY,0,50);}
 77   if(accY < 0){accY = map(accY,0,minAccY,0,-50);}
 78   if(accZ > 0){accZ = map(accZ,0,maxAccZ,0,50);}
 79   if(accZ < 0){accZ = map(accZ,0,minAccZ,0,-50);} 80
 81
 82
 83    float deltaAcc = sqrt(pow(accX,2)+pow(accY,2)+pow(accZ,2));   //calculo de la↙
intensidad de la aceleracion
 84    return deltaAcc;
 85
 86
 87 }
 88
 89
 90
 91 public float rotXsens(float pitch){ 92
 93   if(pitch > pitchMax){pitch = pitchMax;}
 94   if(pitch < pitchMin){pitch = pitchMin;}
 95   if(pitch > 0){pitch = map(pitch,0,pitchMax,0,100);}
 96   if(pitch < 0){pitch = map(pitch,0,pitchMin,0,-100);} 97
 98   return pitch;
 99
100 }
101
102
103 //float[] listaPosicionesX = new float[24];
104 //float[] listaPosicionesY = new float[24];
105
106 //float pct = 0.0;       // Percentage traveled (0.0 to 1.0)
107 //float step = 0.02;     // Size ofeach step along the path
108 //float exponent = 4;
109
110 ////float distX = sensorX - beginX;    // X-axis distanceto move
111 //float distY = sensorY - beginY;    // Y-axis distanceto move
112
113 //public void procesadoUbiSens(float posX, float posY){ 114
115 //for (int i=0; i<24; i++){ 116
117 //   pct += (1/(24-i));
118 //  if (pct < 1.0) {
119 //    listaPosicionesX[i] = posX + (pct * distX);
120 //    listaPosicionesY[i] = posY + (pow(pct, exponent) * distY);
```

```
121 //    }
122 //  }
123 // beginX=listaPosicionesX[23];
124 // beginY=listaPosicionesX[23];
125
126 //}
```

Anexo III

Display

III.1. Pintar Dance

```
 1 import ddf.minim.*;
 2 import deadpixel.keystone.*;
 3 import netP5.*;
 4 import oscP5.*; 5
 6 int RATE = 24;
 7 int intAudioPosition;
 8 int intBackGround; 9
10 //Sound
11 Minim sound;
12 AudioPlayer audio;
13 AudioListener audLis;
14 WaveRenderer wavRen;
15
16 //Dancers
17 Fan fan1;
18 float posX1;
19 float posY1;
20 //hands
21 int intTamanio1A;
22 int intTamanio1B;
23 int intRota1A;
24 int intRota1B;
25 Boolean blnRota1 = true;//si la rotacion es positiva (brazos arriba) o negativa ↙
(brazos abajo)
26 Fan fan2;
27 float posX2;
28 float posY2;
29 //hands
30 int intTamanio2A;
31 int intTamanio2B;
32 int intRota2A;
33 int intRota2B;
34 Boolean blnRota2 = true;
35 Fan fan3;
36 float posX3;
37 float posY3;
38 //hands
```

```
int intTamanio3A;
int intTamanio3B;
int intRota3A;
int intRota3B;
Boolean blnRota3 = true;

ArrayList<Part> arrPart;
int intDrops;
int intBubbles;
int intFadeFactorStart, intFadeFactorEnd;
ArrayList<PVector> arrStartEndPoints;
int intLines;
int intTreads;
                  //  1,  2,  3,  5,  8, 13, 21, 34

int[] intDeform = {515,516,518,521,526,534,547,555,568,578,590,602};
int[] intPulse  = {2,3,5,8,13,21,34,55,89,144};
float fltFactorTam = 0.6f; 57
Rain rain;
Bubbles bubbles;
Traces traces1;
Traces traces2;
Traces traces3;
FadeCurtain fadeCurtain;
SmokeCA smoke1;
SmokeCA smoke2;
SmokeCA smoke3;
OrganicMesh organicMesh;
WaveOnSphere wave; 69
//virtual screen Draw
Displays displaysClient;
int ID=2;
Keystone ks;
CornerPinSurface surface;
PGraphics offscreen;
boolean calibration=false;
PImage calibrationSheet ; 78
void setup() 80 {
  size(1280, 768, P3D);
  frameRate(RATE);
  smooth(); 84
  displaysClient = new Displays("display");
  displaysClient.id=ID; 87
  ks = new Keystone(this);
  surface = ks.createCornerPinSurface(width, height, 20);
  // We need an offscreen buffer to draw the surface we
  // want projected
  // note that we're matching the resolution of the
  // CornerPinSurface.
  // (The offscreen buffer can be P2D or P3D)
  offscreen = createGraphics(width, height, P2D);
  // loads the saved layout
  XML f = loadXML("keystoneDance2.xml"); 98
  if (f != null) {
     ks.load("keystoneDance2.xml");
  }
  else println("no existe");
```

```
103   calibrationSheet = loadImage("calibration.png"); 104
105   // Load a soundfile from the /data folder of the sketch and play it back
106   sound = new Minim(this);
107   audio = sound.loadFile("audio.mp3");
108   wavRen = new WaveRenderer(); 109
110   initialize(); 111
112   colorMode(RGB, 255);
113 } 114
115 void initialize(){
116   intAudioPosition=0;
117   intBackGround=0; 118
119   //Dancers
120   posX1=0;
121   posY1=0;
122   intTamanio1A=0;
123   intTamanio1B=0;
124   fan1 = new Fan(posX1,posY1);
125   posX2=0;
126   posY2=0;
127   intTamanio2A=0;
128   intTamanio2B=0;
129   fan2 = new Fan(posX2,posY2);
130   posX3=0;
131   posY3=0;
132   intTamanio3A=0;
133   intTamanio3B=0;
134   fan3 = new Fan(posX3,posY3);
135
136   arrPart = new ArrayList<Part>();
137   arrPart.add(new Part(1,"rain",1,147));//min-> 0:01 - 2:28
138   arrPart.add(new Part(2,"rainTransition",148,161));//min-> 2:29 - 2:55
139   arrPart.add(new Part(3,"bubbles",162,257));//min-> 2:56 - 4:17
140   arrPart.add(new Part(4,"camBubblesTransition",258,305));//min-> 4:18 - 4:45
141   //JCC
142   arrPart.add(new Part(5,"traces",306,395));//min-> 4:46 - 6:35
              143   arrPart.add(new Part(6,"fadeCurtain",396,422));//min-> 6:36 - 7:02
  144   arrStartEndPoints = new ArrayList<PVector>();
                    145   for(int i=1; i<= 3; i++){
146     Part p = arrPart.get(5);
147     intFadeFactorStart = p.intInitTime+(int((p.intEndTime-p.intInitTime)/3)*(i-
1));//396-404-412
148     intFadeFactorEnd = p.intInitTime+(int((p.intEndTime-p.intInitTime)/3)*i)+1;
//404-412-420
149     arrStartEndPoints.add(new    PVector(i,intFadeFactorStart,intFadeFactorEnd));
150   }
151   //JCC
152   arrPart.add(new Part(7,"smoke",420,464));//min-> 7:03 - 7:44
153   arrPart.add(new Part(8,"createMesh",465,514));//min-> 7:45 - 8:34 154
      arrPart.add(new Part(9,"meshSphere",515,619));//min-> 8:35 - 10:19
155   rain = new Rain();
156   bubbles = new Bubbles();
157   traces1 = new Traces();
158   traces2 = new Traces();
159   traces3 = new Traces();
160   fadeCurtain = new FadeCurtain();
161   smoke1 = new SmokeCA();
162   smoke2 = new SmokeCA();
```

```
163   smoke3 = new SmokeCA();
164   organicMesh = new OrganicMesh();
165   wave = new WaveOnSphere();
166
167   intDrops=0;
168   intBubbles=0;
169   intFadeFactorStart=0;
170   intFadeFactorEnd=0;
171   intLines=0;
172   intTreads=0;
173 }174
175 void draw(){
176   // Convert the mouse coordinate into surface coordinates
177   // this will allow you to use
mouse events inside the// surface from
your screen.
179
180   PVector surfaceMouse = surface.getTransformedMouse(); 181
182   // Draw the scene, offscreen
183   offscreen.beginDraw(); 184
185   receiveMessage();
186   intAudioPosition = int(audio.position()/1000);
187   //println("Audio time: "+intAudioPosition);
188   for(int i = 0; i<arrPart.size();i++){
189     Part p = arrPart.get(i);
190     if(p.intInitTime<=intAudioPosition && p.intEndTime>intAudioPosition){
191       switch(p.intId){
192         case 1:
193           //println("rain");
194           if(intDrops<rain.maxDrops){
195             intDrops = int(map(intAudioPosition, p.intInitTime, int(p↙
intEndTime/2), 0, rain.maxDrops));
196           } else {
197             intDrops=rain.maxDrops;
198           }
199           rain.display(intDrops, false, fan1, fan2, fan3);//rain
200           if(displaysClient.id==2){//only central display
201             //Inlcuye el pulso de los fan
202             for(int k=0;k<intPulse.length;k++){
203               if(((intAudioPosition%intPulse[k])==0)&&intAudioPosition<p.
intEndTime){
204                 if(frameCount%int(RATE)==0){
205                   intTamanio1A+= fan1.intMaxFanSize*fltFactorTam;
206                   intTamanio1B+= fan1.intMaxFanSize*fltFactorTam;
207                   intTamanio2A+= fan2.intMaxFanSize*fltFactorTam;
208                   intTamanio2B+= fan2.intMaxFanSize*fltFactorTam;
209                   intTamanio3A+= fan3.intMaxFanSize*fltFactorTam;
210                   intTamanio3B+= fan3.intMaxFanSize*fltFactorTam;
211                 }
212               }
213             }
214             //Ajusta posiciones a la pantalla
215             posX1 = posX1-100;
216             posY1 = posY1 - ((posY1)*(((intRota1A+intRota1B)/2)/100.0));
217             posX2 = posX2-60;
218             posY2 = posY2 - ((posY2)*(((intRota2A+intRota2B)/2)/100.0));
219             posX3 = posX3-20;
```

```
220             posY3 = posY3 - ((posY3)*(((intRota3A+intRota3B)/2)/100.0));
221             //Envia los datos ajustados para pintar el objeto
222             fan1.display(posX1,posY1-100,int((intTamanio1A+intTamanio1B)/2));
223             fan1.update(posX1,posY1-100);
224             fan2.display(posX2,posY2-100,int((intTamanio2A+intTamanio2B)/2));
225             fan2.update(posX2,posY2-100);
226             fan3.display(posX3,posY3-100,int((intTamanio3A+intTamanio3B)/2));
227             fan3.update(posX3,posY3-100);
228           }
229           break;
230         case 2:
231           //println("rainTransition");
232           //Fade black to white backgroud
233           rain.display(int(rain.maxDrops*0.9), true, fan1, fan2, fan3);
234           break;
235         case 3://println("bubbles");
236           if(intBubbles<bubbles.intMaxBubbles){
237             intBubbles = int(map(intAudioPosition, p.intInitTime, int(p.
intInitTime+((p.intEndTime-p.intInitTime)/4)), bubbles.intMinBubbles, bubbles.
intMaxBubbles));
238           } else {
239             intBubbles=bubbles.intMaxBubbles;
241           }
242           //Si la rotacion es positiva (brazos arriba), zoomIn
243           //sino (brazos abajo), zoomOut
244           //Se cambia la velocidad de movimiento en funcion de la aceleracion
245           if(((intRota2A+intRota2B)/2)>0){
246             blnRota2 = true;
247           } else {
248             blnRota2 = false;
249           }
250           //ajusta los datos de la pantalla
251           //posY2 = posY2 - ((posY2)*(((intRota2A+intRota2B)/2)/100.0));
252           //envia los datos ajustados para pintar
253           bubbles.display(intBubbles, posX2, posY2, false, blnRota2, int
((intTamanio2A+intTamanio2B)/2));
254           break;
255         case 4:
256           //println("bubblesTransition");
257           if(intBubbles>0){
258             intBubbles = int(map(intAudioPosition, p.intInitTime, p.intEndTime-1,
bubbles.intMaxBubbles, 0));
259           } else {
260             intBubbles=0;//bubbles.intMinBubbles;
261           }
262           if(((intRota2A+intRota2B)/2)>0){
263             blnRota2 = true;
264           } else {
265             blnRota2 = false;
266           }
267           bubbles.display(intBubbles, posX2, posY2, false, blnRota2, int
((intTamanio2A+intTamanio2B)/2));
268           break;
269         case 5:
270           //println("traces");
271           if(displaysClient.id==2){//only central display
272             traces1.display(posX1+100, posY1+300);
```

```
273             traces2.display(posX2+200, posY2+300);
274             traces3.display(posX3+500, posY3+300);
275           } else {
276             offscreen.noStroke();
277             offscreen.fill(255,12);
278             offscreen.rect(0,0,width,height);
279           }
280           break;
281         case 6:
282           //println("fadeCurtain");
283           //Barrido De fondo blanco a fondo negro de 396 a 422 segundos
284           intFadeFactorStart = int(arrStartEndPoints.get(ID-1).y);
285           intFadeFactorEnd = int(arrStartEndPoints.get(ID-1).z);
286           switch(int(arrStartEndPoints.get(ID-1).x)){
287             case 2:
288               intFadeFactorStart -= 1;
289               break;
290             case 3:
291               intFadeFactorStart -= 2;
292               break;
293             default:
294               break;
295           }
296           if(intFadeFactorStart<=intAudioPosition &&↙
intFadeFactorEnd>intAudioPosition){
297             if(intLines<width){
298               intLines = int(map(intAudioPosition, intFadeFactorStart↙
intFadeFactorEnd, 10, width));
299             } else {
300               intLines=width;
301             }
302             println(intLines);
303             fadeCurtain.display(intLines);
304           }
305           break;
306         case 7:
307           //println("smoke");
308           if(displaysClient.id==2){//only central display
309             if (intAudioPosition<(p.intEndTime-7)){
310               smoke1.display(posX1, posY1+400, false);
311               smoke2.display(posX2, posY2+400, false);
312               smoke3.display(posX3, posY3+400, false);
313             } else {
314               smoke1.display(posX1, posY1+400, true);
315               smoke2.display(posX2, posY2+400, true);
316               smoke3.display(posX3, posY3+400, true);
317             }
318           } else {
319             offscreen.noStroke();
320             offscreen.fill(5, 15);
321             offscreen.rect(0, 0, width, height);
322           }
323           break;
324         case 8:
325           //println("createMesh");
326           if(intTreads<organicMesh.intMaxTreads){
327             intTreads = int(map(intAudioPosition, p.intInitTime, p.intEndTime, 0↙
```

```
organicMesh.intMaxTreads));
328           } else {
329             intTreads=organicMesh.intMaxTreads;
330           }
331           //ajusta los datos de la pantalla
332           posY2 = posY2 - ((posY2)*(((intRota2A+intRota2B)/2)/100.0));
333           //envia los datos ajustados para pintar
334           organicMesh.display(posX2, posY2, intTreads);
335          break;
336         case 9:
337           //println("meshSphere");
338           if(displaysClient.id==2){//only central display
339             wave.display(wave.intValor1, wave.intValor1);
340             for(int k=0;k<intDeform.length;k++){
341               if(((intAudioPosition%intDeform[k])==0)&&intAudioPosition<p.
intEndTime){
342                 if(frameCount%int(RATE)==0){
343                   wave.change(posX2, posY2);
344                 }
345               }
346             }
347           } else {
348             offscreen.background(0) ;
349           }
350          break;
351         default:
352           supendAll();
353          break;
354       }
355     }
356   } 357
358   if(intAudioPosition>=618){
359     supendAll();
360   } 361
362   if (calibration) offscreen.image(calibrationSheet,0,0,width,height);
363   offscreen.endDraw(); 364
365   // most likely, you'll want a black background to minimize
366   // bleeding around your projection area
367   background(0); 368
369   // render the scene, transformed using the corner pin surface
370   surface.render(offscreen); 371
372 } 373
374 void supendAll(){
375   audio.removeListener( wavRen );
376   audio.pause();
377   audio.rewind();
378   initialize();
379   offscreen.fill(0,255);
380   offscreen.rect(0,0,width,height);
381 } 382
383 void changeTo(int intMillis){
384   if (audio.isPlaying()){
385       supendAll();
386       startIn(intMillis);
387     } else {
388       startIn(intMillis);
389     }
```

```
} 391
void startIn(int intMillis){
  audio.play(intMillis);
  audio.addListener( wavRen );
} 396
void stop(){
  // always close Minim audio classes when you are done with them
  audio.close();
  sound.stop();
  super.stop();
} 403
void receiveMessage(){
  if((int(posX1)==0||int(posX1)==1280)&&int(posY1)==0){
    //println("1 no play");
    intTamanio1A=0;
    intTamanio1B=0;
  } else {
    if(intTamanio1A>fan1.intMaxFanSize){intTamanio1A=fan1.intMaxFanSize;}
    if(intTamanio1A<fan1.intMinFanSize){intTamanio1A=fan1.intMinFanSize;}
    if(intTamanio1B>fan1.intMaxFanSize){intTamanio1B=fan1.intMaxFanSize;}
    if(intTamanio1B<fan1.intMinFanSize){intTamanio1B=fan1.intMinFanSize;}
  }
  if((int(posX2)==0||int(posX2)==1280)&&int(posY2)==0){
    //println("2 no play");
    intTamanio2A=0;
    intTamanio2B=0;
  } else {
    if(intTamanio2A>fan2.intMaxFanSize){intTamanio2A=fan2.intMaxFanSize;}
    if(intTamanio2A<fan2.intMinFanSize){intTamanio2A=fan2.intMinFanSize;}
    if(intTamanio2B>fan2.intMaxFanSize){intTamanio2B=fan2.intMaxFanSize;}
    if(intTamanio2B<fan2.intMinFanSize){intTamanio2B=fan2.intMinFanSize;}
  }
  if((int(posX3)==0||int(posX3)==1280)&&int(posY3)==0){
    //println("3 no play");
    intTamanio3A=0;
    intTamanio3B=0;
  } else {
    if(intTamanio3A>fan3.intMaxFanSize){intTamanio3A=fan3.intMaxFanSize;}
    if(intTamanio3A<fan3.intMinFanSize){intTamanio3A=fan3.intMinFanSize;}
    if(intTamanio3B>fan3.intMaxFanSize){intTamanio3B=fan3.intMaxFanSize;}
    if(intTamanio3B<fan3.intMinFanSize){intTamanio3B=fan3.intMinFanSize;}
  }
} 436
void mensajeRecibido(OscMessage theOscMessage){
  //println("THE_OSC_MESSAGE=", theOscMessage.addrPattern());
  if(theOscMessage.addrPattern().equals("/display/BeginDrawDance")){
    println("Start draw: ", theOscMessage.addrPattern());
    if (theOscMessage.get(0).stringValue().contains(str(displaysClient.id))) {
      String strPart = theOscMessage.get(1).stringValue();
      resolveKey(strPart.charAt(0));
    }
  }

  if(theOscMessage.addrPattern().equals("/display/EndDrawDance")){
    println("Stop draw: ", theOscMessage.addrPattern());
    supendAll();
  }
```

```
451   if (theOscMessage.addrPattern().equals("/display/DibujarDance") ){
452     if (theOscMessage.get(0).stringValue().contains(str(displaysClient.id))) {
453       if(theOscMessage.get(1).stringValue().equals("tag089")){
454         posX1 = theOscMessage.get(2).floatValue();
455         posY1 = theOscMessage.get(3).floatValue();
456         //println(frameCount+" -> Dancer 1 posX: " + posX1 +" posY: " + posY1);
457       }
458       if(theOscMessage.get(1).stringValue().equals("tag143")){
459         posX2 = theOscMessage.get(2).floatValue();
460         posY2 = theOscMessage.get(3).floatValue();
461         println(frameCount+" -> Dancer 2 posX: " + posX2 +" posY: " + posY2);
462       }
463       if(theOscMessage.get(1).stringValue().equals("tag248")){
464         posX3 = theOscMessage.get(2).floatValue();
465         posY3 = theOscMessage.get(3).floatValue();
466         //println(frameCount+" -> Dancer 3 posX: " + posX3 +" posY: " + posY3);
467       }
468       if(theOscMessage.get(1).stringValue().equals("xSens20")){
469         //recibe valores entre 1 y 100, es decir, entre 1% y 100%
470         intTamanio1A = int(theOscMessage.get(2).floatValue());
471         intRota1A = int(theOscMessage.get(3).floatValue());
472         //println(frameCount+" -> Dancer 1 valor1: "+ theOscMessage.get(2).↙
floatValue()+ "\tTam=" +intTamanio1A);
473       }
474       if(theOscMessage.get(1).stringValue().equals("xSens21")){
475         intTamanio1B = int(theOscMessage.get(2).floatValue());
476         intRota1B = int(theOscMessage.get(3).floatValue());
477         //println(frameCount+" -> Dancer 1 valor2: "+ theOscMessage.get(2).↙
floatValue()+ "\tTam=" +intTamanio1B);
478         //println(frameCount+" -> Dancer 1 valor2: "+ theOscMessage.get(2).↙
floatValue()+ "\tTam=" +intTamanio1B);
479       }
480       if(theOscMessage.get(1).stringValue().equals("xSens22")){
481         intTamanio2A = int(theOscMessage.get(2).floatValue());
482         intRota2A = int(theOscMessage.get(3).floatValue());
483         //println(frameCount+" -> Dancer 2 valor1: "+ theOscMessage.get(2).↙
floatValue()+ "\tTam=" +intTamanio2A);
484         //println(frameCount+" -> Dancer 2 rota1: "+ theOscMessage.get(3).↙
floatValue()+ "\tTam=" +intRota2A);
485       }
486       if(theOscMessage.get(1).stringValue().equals("xSens23")){
487         intTamanio2B = int(theOscMessage.get(2).floatValue());
488         intRota2B = int(theOscMessage.get(3).floatValue());
489         //println(frameCount+" -> Dancer 2 valor2: "+ theOscMessage.get(2).↙
floatValue()+ "\tTam=" +intTamanio2B);
490         //println(frameCount+" -> Dancer 2 rota2: "+ theOscMessage.get(3).↙
floatValue()+ "\tTam=" +intRota2B);
491       }
492       if(theOscMessage.get(1).stringValue().equals("xSens24")){
493         intTamanio3A = int(theOscMessage.get(2).floatValue());
494         intRota3A = int(theOscMessage.get(3).floatValue());
495         //println(frameCount+" -> Dancer 3 valor1: "+ theOscMessage.get(2).↙
floatValue()+ "\tTam=" +intTamanio3A);
496       }
497       if(theOscMessage.get(1).stringValue().equals("xSens25")){
498         intTamanio3B = int(theOscMessage.get(2).floatValue());
499         intRota3B = int(theOscMessage.get(3).floatValue());
```

```
500         //println(frameCount+" -> Dancer 3 valor2: "+ theOscMessage.get(2).↙
floatValue()+ "\tTam=" +intTamanio3B);
501       }
502     }
503   }
504 }505
506 void keyPressed() {
507   resolveKey(key);
508 }509
510 void resolveKey(char cKey) {
511   switch(cKey) {
512     case 'c':
513       // enter/leave calibrationmode, where surfaces can be warped
514       // and moved
515       calibration=!calibration;
516       ks.toggleCalibration();
517       if (calibration==false) {ks.save("keystoneDance2.xml");
518         supendAll();
519         noLoop();
520       } else {
521         loop();
522         intBackGround=0;
523         changeTo(1000);
525       }
526       break;
527     case ESC:
528       displaysClient.disconnect();
529       this.stop();
530       break;
531     case '1'://rain
532       intBackGround=0;
533       changeTo(1000);
534       break;
535     case '2'://rainTransition
536       intBackGround=0;
537       changeTo(148000);
538       break;
539     case '3'://bubbles
540       intBackGround=255;
541       changeTo(162000);
542       break;
543     case '4'://camBubblesTransition
544       intBackGround=255;
545       changeTo(258000);
546       break;
547     case '5'://traces
548       intBackGround=255;
549       changeTo(306000);
550       break;
551     case '6'://fadeCurtain
552       intBackGround=255;
553       changeTo(396000);
554       break;
555     case '7'://smoke
556       intBackGround=0;
557       changeTo(423000);
558       break;
```

```
559     case '8'://createMesh
560       intBackGround=0;
561       changeTo(465000);
562       break;
563     case '9'://meshSphere
564       intBackGround=0;
565       changeTo(515000);
566       break;
567     case 32:
568       intBackGround=0;
569       changeTo(0);
570       break;
571   }
572 }
```

III.2. Bubble

```
 1 class Bubble{
 2   float ZSTEP = 0.008;
 3   float RADIUS = height/10;
 4   float SPEED = 0.001;
 5   int intMaxSize=80;
 6   int intMinSize=30;
 7   PVector vecPosition;
 8   PVector vecSpeed;
 9   int intSize;
10   color colBubble;
11   color
colShaded; 12
13   Bubble(float x, float y, float z){
14     this.vecPosition = new PVector(x,y,z);
15     setColor();
16     this.vecSpeed = new PVector(random(-1.0, 1.0),random(-1.0, 1.0),random(-1.0↙
1.0));
17     float magnitude = sqrt(pow(this.vecSpeed.x,2)+pow(this.vecSpeed.y,2)+pow(this.↙
vecSpeed.z,2));
18     this.vecSpeed.mult(SPEED/magnitude);
19   this.intSize = int(random(intMinSize,intMaxSize));
20   }
21
22   float getFltZ(){
23   return this.vecPosition.z;
24   }
25
26   void setColor() {
27     float shade = this.vecPosition.z;
28     float shadeinv = 1.0-shade;
29     this.colShaded = color( (red(this.colBubble)*shade)+(red(255)*shadeinv),
30                     (green(this.colBubble)*shade)+(green(255)*shadeinv),
31   (blue(this.colBubble)*shade)+(blue(255)*shadeinv)); 32  }
33
34   void display(float xoffs, float yoffs, int intZoom){
35     float a = pow(this.vecPosition.z,2);
36     float posX = (intZoom*this.vecPosition.x*width*(1+a))- intZoom*xoffs*width*a;
37     float posY = (intZoom*this.vecPosition.y*height*(1+a))- ↙
intZoom*yoffs*height*a;
38     float radius = this.intSize+(this.vecPosition.z*RADIUS);
39     float diam =
RADIUS*2; 40
41     if ( posX > -diam && posX < width+diam
42       && posY > -diam && posY < height+diam) {
43     blurred_circle(posX, posY, radius);
44     }
45   }
46
47   //void blurred_circle(){
```

```
 48   void blurred_circle(float xx, float yy, float rr){
 49     offscreen.noStroke();
 50     offscreen.fill(50, 150);
 51     offscreen.ellipse(xx, yy, rr, rr);
 52     offscreen.fill(230, 50);
 53     offscreen.pushMatrix();
 54     offscreen.translate(xx, yy);
 55     offscreen.rotate(radians(40));
 56     offscreen.ellipse(-35, -15, 10, 30);
 57     offscreen.popMatrix();
 58   }
 59
 60   void zoomIn(float step) {
 61     this.vecPosition.z += step;
 62     if (this.vecPosition.z > 1.0) {
 63     this.vecPosition.z = 0.0 + (this.vecPosition.z-1.0); 64
        }
 65   }
 66
 67   void zoomOut(float step) {
 68     this.vecPosition.z -= step;
 69     if (this.vecPosition.z < 0.0) {
 70     this.vecPosition.z = 1.0 - (0.0-this.vecPosition.z); 71
        }
 72   }
 73
 74   void update(boolean doZoomIn, boolean doZoomOut, float fltAcceleration){
 75     float fltzStep = ZSTEP + (5*(ZSTEP*(fltAcceleration)));
 76     if (doZoomIn) {
 77     zoomIn(fltzStep); 78
        }
 79     if (doZoomOut) {
 80     zoomOut(fltzStep); 81 }
 82     if (this.vecPosition.x <= 0) {
 83         this.vecSpeed.x = abs(this.vecSpeed.x);
 84     this.vecPosition.x = 0.0f; 85
        }
 86     if (this.vecPosition.x >= 1.0) {
 87         this.vecSpeed.x = -1.0 * abs(this.vecSpeed.x);
 88     this.vecPosition.x = 1.0; 89
        }
 90     if (this.vecPosition.y <= 0) {
 91         this.vecSpeed.y = abs(this.vecSpeed.y);
 92     this.vecPosition.y = 0.0f; 93
        }
 94     if (this.vecPosition.y >= 1.0) {
 95         this.vecSpeed.y = -1.0 * abs(this.vecSpeed.y);
 96     this.vecPosition.y = 1.0; 97
        }
 98     if (this.vecPosition.z < 0 || this.vecPosition.z > 1.0) {
 99         this.vecPosition.z = this.vecPosition.z% 1.0;
100     }
101     //this.vecPosition.add(this.vecSpeed);
102     this.vecPosition.x += this.vecSpeed.x;
103     this.vecPosition.y += this.vecSpeed.y;
104     this.setColor();
105   }
```

```
106
107   void move(){
108     float time = frameCount*0.0001f;
109     float cy = map(sin(time), -1, 1, -height / 4, height / 4);
110     //this.velocity.y = cy;
111     this.vecPosition.add(this.vecSpeed);
112   }
113
114   void wallCollide(){
115     int intRatio = int(this.intSize/2);
116     if(this.vecPosition.x-intRatio < 0 || this.vecPosition.x+intRatio > width){
117       this.vecSpeed.x *= -1;} else if(this.vecPosition.y-intRatio < 0 ||
this.vecPosition.y+intRatio > height){
118         this.vecSpeed.y *= -1;
120     }
121   }
122 }
```

III.3. Bubbles

```
 1 class Bubbles{
 2   int intZoom = 2;
 3   int intMinBubbles = 50;
 4   int intMaxBubbles = 300;
 5   ArrayList<Bubble> arrBubbles;
 6   float xoffs = 0;
 7   float yoffs = 0;
 8   float fltFactor = 0.03;
 9   int intDesfase = 300;
10   float fltAjusteX;
11   float fltAjusteY;
12   boolean zoomIn = false;
13   boolean zoomOut =
false; 14
15   Bubbles(){
16     arrBubbles = new ArrayList<Bubble>();
17     Bubble b;
18     float posX, posY, posZ;
19     for(int i = 0; i < intMaxBubbles; i++){
20       posX = random(1.0f);
21       posY = random(1.0f);
22       posZ = random(1.0f);
23       b = new Bubble(posX, posY, posZ);
24     arrBubbles.add(b);
25     }
26
     sortBubbles();
27   }
28
29   void sortBubbles() {
30     // Sort them (this ensures that they are drawn in the right order)
31     float last = 0;
32     ArrayList temp = new ArrayList();
33     for (int i=0; i<intMaxBubbles; i++) {
34         int index = 0;
35         float lowest = 100.0;
36         for (int j=0; j<intMaxBubbles; j++) {
37             Bubble current = (Bubble)arrBubbles.get(j);
38             if (current.getFltZ() < lowest && current.getFltZ() > last) {
39                 index = j;
40             lowest = current.getFltZ(); 41
               }
42         }
43         temp.add(arrBubbles.get(index));
44     last = ((Bubble)arrBubbles.get(index)).getFltZ(); 45}
46     arrBubbles =
temp; 47    }
48
49   void display(int numBubbles, float fltPosX, float fltPosY, boolean
```

```
blnTransition, boolean blnZoom, int intAcelera){
50      offscreen.fill(255,255);
51      offscreen.rect(0,0,width,height);
52
53      fltAjusteX = intDesfase*(sin(frameCount*fltFactor*PI/4));
54      fltAjusteY = intDesfase*(cos(frameCount*fltFactor*PI/4));
55      //if (numBubbles==intMaxBubbles){
56        //if(fltAjusteX<0&&fltAjusteY<0){
57        if(blnZoom){//Brazos arriba
58          zoomIn=true;
59        zoomOut=false;} else {
60          zoomIn=false;
61        zoomOut=true; 63      }
64      //}
65      fltPosX+=fltAjusteX;
66      fltPosY+=fltAjusteY; 67
68      xoffs = xoffs*0.9 + 0.1*fltPosX/width;
69      yoffs = yoffs*0.9 + 0.1*fltPosY/height;
70      //println("xoffs: "+ xoffs + "\tyoffs " +yoffs); 71
72      for (int i=0; i<intMaxBubbles; i++) {
73        Bubble current = (Bubble)arrBubbles.get(i);
74        //println("intAcelera: "+intAcelera);
75      current.update(zoomIn, zoomOut, intAcelera/100.0); 76      }
77
78      sortBubbles(); 79
80      for(int i = 0; i < numBubbles; i++){
81        Bubble b = arrBubbles.get(i); 82
          //b.move();
83        //b.wallCollide();
84        if(!blnTransition){
85        b.display(xoffs, yoffs, intZoom); 86   }
87      }
88    }
89 }
```

III.4. Displays

```
 1 class Displays{
 2   OscP5 oscP5;
 3   NetAddress myBroadcastLocation; 4
 5   //String remoteIP="127.0.0.1";
 6   String remoteIP="192.168.0.2";
 7   int listenPort=12001; // el puerto en el que se queda escuchando mensajes del ↙
broadcaster
 8
 9   private String tipo; //Posibles valores: host, sensor o display
10   public int id; 11
12   public class Display {
13       public int x;
14       public int y;
15       public int w;
16   public int h; 17   }
18
19   Displays(String tipo) {
20     this.tipo = tipo;
21     // OSC, aqui esta el puerto donde escucha mensajes 22
23     oscP5 = new OscP5(this,listenPort);
24     // set the remote location to be the localhost on port 25
26     myBroadcastLocation = new NetAddress(remoteIP,33000);
27   connect(); 28
     }
29
30   public void connect() {
31     OscMessage m;
32     println("conectar");
33     m = new OscMessage("/" + tipo + "/connect",new Object[0]);
34     m.add(listenPort);
35   OscP5.flush(m,myBroadcastLocation); 36
     }
37
38   public void disconnect() {
39     OscMessage m;
40     m = new OscMessage("/" + tipo + "/disconnect",new Object[0]);
41     m.add(listenPort);
42     OscP5.flush(m,myBroadcastLocation);
43   println("disconneted"); 44}
45
46   void mandarMensaje(OscMessage myMessage) {
47        // send the message
48   oscP5.send(myMessage, myBroadcastLocation); 49   }
50
51  /* incoming osc message are forwarded to the oscEvent method. */
52   void oscEvent(OscMessage theOscMessage) {
53      mensajeRecibido(theOscMessage); //es necesarioque esta funcion este ↙
impremented en el main
54   }
55 }
```

III.5. Drop

```
class Drop{
  PVector vecOrigin = new PVector(0, 0);
  PVector vecPosition;
  PVector vecSpeed;
  float speedFactor = 8.0f;
  float z, onde, d, d1;
  float accFactor = 0.01f;
  float accFactor2 = 0.5f;
  boolean s;
  color myColor;

  Drop(int x,int y, int z, int d){
    this.vecOrigin = new PVector(x,y);
    this.vecPosition = new PVector(x,y);
    this.vecSpeed = new PVector(0,0);
    this.d=d;
    this.z=z;
    onde=0;
    d1=d;
    myColor = color(255);
  this.vecSpeed = new PVector(0,0);
  }

  void fall(Boolean blnWithOnde){
    if(this.vecPosition.y > 0.0f){
      if(blnWithOnde){
        this.vecSpeed.y+=accFactor;
      } else {
      this.vecSpeed.y+=accFactor2; 30
      }
    }
    offscreen.stroke(myColor,map(z,0,height,0,255));
    offscreen.strokeWeight(2);
    if (this.vecPosition.y<z){
      this.vecPosition.y=this.vecPosition.y+this.vecSpeed.y+speedFactor;
      this.vecPosition.x=this.vecPosition.x+this.vecSpeed.x;
      offscreen.line(this.vecPosition.x,this.vecOrigin.y,this.vecPosition.x,this.
vecPosition.y);
      this.vecOrigin.y=this.vecPosition.y;
    } else {
      offscreen.noFill();
      offscreen.stroke(175,175,175,175-map(onde,0,d,0,255));
      offscreen.strokeWeight(map(onde,0,d,0,4));
      d=d1+(this.vecPosition.y-height)*4;
      //if(blnWithOnde){offscreen.ellipse(this.vecPosition.x,this.vecPosition.y,
onde/5,onde/20);}
      onde=onde+7;
      if(onde>d){
        onde=0;
```

```
48          this.vecSpeed.x=0;
49          this.vecSpeed.y=0;
50          this.vecPosition.x=int(random(width));
51          this.vecPosition.y=-int(random(height*2));
52          this.vecOrigin.y=this.vecPosition.y; 53
            d=d1;
54        }
55      }
56    }
57
58    void wind(float fltPosX, float fltPosY, int fltFanSize, float fltFanSpeed){
59
60      float fltDistance = dist(fltPosX,fltPosY,this.vecPosition.x,this.vecPosition.↙
y);
61      float fltRat = atan2(fltPosY-this.vecPosition.y,fltPosX-this.vecPosition.x)+PI
62
63      if(fltDistance < fltFanSize){
64          this.vecSpeed.x = this.vecSpeed.x + (fltDistance * cos(fltRat))↙
(100/fltFanSpeed);
65          this.vecSpeed.y = this.vecSpeed.y + (fltDistance * sin(fltRat))↙
(100/fltFanSpeed);
66      }
67    }
68 }
```

III.6. Fade Curtain

```
class FadeCurtain{
  int lines;
  float fade;
  int difAngle;  //velocity
  int num;  //number of points in a row (or column)
  color cor; //main color
  int mX, mY;  //variables used to allows changing the main color 8
  FadeCurtain(){
    difAngle = 9;
    num = 10;
    lines=0;
  fade =0; 14  }

  void display(int numLines){
    cor = color(0);
    lines += 9;
    fade += 1.5f; 20
    //draw background
    offscreen.fill(255);
    offscreen.rect(0,0,lines,height); 24
    for (int i = 0; i < lines; i +=2){
      offscreen.strokeWeight(2);
      offscreen.stroke(cor, map(i, 0, lines, 255, fade));
    offscreen.line(i, 0, i, height); 29     }

    //draw the pattern
    //float cellsize = lines / (num - 1);
    float cellsize = height;
    int circleNumber = 0;
    for (int i=0; i<num; i++) 36 {
      for (int j=0; j<num; j++)
      {
        circleNumber++; 40
        float tx = cellsize * i;
        float ty = cellsize * j; 43
        movingCircle(tx, ty, cellsize, circleNumber); 45}
    }
  }

  void movingCircle(float x, float y, float size, int circleNum){
    float finalAngle = millis()/100 + circleNum;
    float tempX = x + (size / 2) * sin(PI / difAngle * finalAngle);
    float tempY = y + (size / 2) * cos(PI / difAngle * finalAngle); 53
    offscreen.noStroke();
    offscreen.fill(cor, circleNum/10);
    ellipse(tempX, tempY, 7, size*8); 57
    offscreen.fill(cor, circleNum/6);
  ellipse(tempX, tempY, 5, size*5); 60}61 }
```

III.7. Fan

```
1 class Fan {
2   private float fltFanSpeed = 1;//0.01;
3   private int intFanSize = 0;
4   private int intMinFanSize = 20;
5   private int intMaxFanSize = 70;
6   PVector vecPosition;
7   float fltRotation;
8   color myColor; 9
10
11  int intCount = 0;
12  PVector vecPosition2;
13  PVector vecNewPosition;
14  ArrayList<PVector> arrPositions;
15  int intConta;
16  float fltFrameRate = RATE;
17  boolean blnDebug = false; 18
19  Fan(float fltPosX, float fltPosY) {
20    this.vecPosition = new PVector(fltPosX, fltPosY);
21    vecPosition2 = new PVector(0,0);
22    arrPositions = new ArrayList<PVector>();
23    for (int a = 0; a<=int(fltFrameRate); a++){
24    arrPositions.add(new PVector(0,0)); 25   }
26  }
27
28  Fan(float fltPosX, float fltPosY, float r, float g, float b) {
29    this.vecPosition = new PVector(fltPosX, fltPosY);
30    this.myColor = color(r,g,b);
31    vecPosition2 = new PVector(0,0);
32    arrPositions = new ArrayList<PVector>();
33    for (int a = 0; a<=int(fltFrameRate); a++){
34    arrPositions.add(new PVector(0,0)); 35   }
36  }
37
38  void update(float fltPosX, float fltPosY){
39    //this.vecPosition.x = fltPosX;
40    //this.vecPosition.y = fltPosY;
41    if(intCount%int(fltFrameRate)==0){
42      interpolar(fltPosX,fltPosY,true);
43    } else {
44    interpolar(0,0,false); 45     }
46    this.fltRotation += this.fltFanSpeed; 47}
48
49  void display(float fltPosX, float fltPosY, int intSize){
50    if(intCount%int(fltFrameRate)==0){
51      interpolar(fltPosX,fltPosY,true);
52    } else {
53    interpolar(0,0,false); 54     }
55    this.intFanSize = intSize;
```

```
56      offscreen.noStroke();
57      offscreen.pushMatrix();
58      offscreen.translate(this.vecPosition.x, this.vecPosition.y);
59      offscreen.rotate(this.fltRotation);
60      drawObjetive();
61    offscreen.popMatrix();
62    }
63
64    void drawObjetive(){
65    offscreen.rect(-1, 0-(this.intFanSize/2), 3, this.intFanSize); 66 }
67
68    void interpolar(float fltPosX, float fltPosY, boolean blnInterpolate){
69      if(blnInterpolate){
70        println("interpolar");
71        asignarPosicion(this.intConta);
72        this.arrPositions = new ArrayList<PVector>(int(this.fltFrameRate));
73        for(int i = 0; i<int(fltFrameRate); i++){
74          this.vecNewPosition = new PVector(0,0);
75          this.vecNewPosition.x = lerp(this.vecPosition2.x, fltPosX, i/this↙
fltFrameRate);
76          this.vecNewPosition.y = lerp(this.vecPosition2.y, fltPosY, i/this↙
fltFrameRate);
77        this.arrPositions.add(vecNewPosition); 78     }
79        this.arrPositions.add(new PVector(fltPosX,fltPosY));
80        this.vecPosition2.x = fltPosX;
81        this.vecPosition2.y = fltPosY;
82        this.intConta =0;
83      } else {
84        asignarPosicion(this.intConta);
85        this.intConta++;
86        if(this.intConta==(this.arrPositions.size()-2)){
87          asignarPosicion(this.intConta);
88          }
89      }
90    }
91
92    void asignarPosicion(int intItem){
93      this.vecPosition.x = arrPositions.get(intItem).x;
94    this.vecPosition.y = arrPositions.get(intItem).y; 95   }
96 }
```

III.8. Organic Mesh

```
class OrganicMesh{
  ArrayList<PVector> arrOrigins;
  ArrayList<PVector> arrEnds;
  float easing = 0.03f;
  float targetX=width/10;
  float targetY=height/2;
  int intMaxTreads=400; 8
  OrganicMesh() {
    this.arrOrigins = new ArrayList<PVector>();
    this.arrEnds = new ArrayList<PVector>();
    for (int i=0;i<this.intMaxTreads;i++) {
      this.arrOrigins.add(new    PVector(random(-60,width),random(-60,height)));
    this.arrEnds.add(new PVector(0.0f,0.0f)); 15
    }
  }

  void display(float fltPosX, float fltPosY, int intNumTreads){
    offscreen.noStroke();
    offscreen.background(0);
    for (int i=0;i<intNumTreads;i++) {
      this.arrEnds.get(i).x = targetX - this.arrOrigins.get(i).x;
      this.arrEnds.get(i).y = targetY - this.arrOrigins.get(i).y;
      this.arrOrigins.get(i).x += this.arrEnds.get(i).x * easing;
      this.arrOrigins.get(i).y += this.arrEnds.get(i).y * easing;
      for (int j=0;j<intNumTreads;j++) {
        float dist = dist(this.arrOrigins.get(i).x,this.arrOrigins.get(i).y,this.↙
arrOrigins.get(j).x,this.arrOrigins.get(j).y);
        if(i!=j&&dist<=60.3) {
          this.arrOrigins.get(j).x += this.arrEnds.get(i).x * easing/10;
          this.arrOrigins.get(j).y += this.arrEnds.get(i).y * easing/10;
          this.arrOrigins.get(i).x -= this.arrEnds.get(i).x * easing;
        this.arrOrigins.get(i).y -= this.arrEnds.get(i).y * easing; 33
        }
        if(i!=j&&dist<=135) {
          offscreen.strokeWeight(2);
          offscreen.stroke(255, 255, 255,50);
          offscreen.line(this.arrOrigins.get(i).x,this.arrOrigins.get(i).y,this.   ↙
arrOrigins.get(j).x,this.arrOrigins.get(j).y);
        }
        dist = dist(this.arrOrigins.get(i).x,this.arrOrigins.get(i).y,fltPosX↙
fltPosY);
        if(i!=j&&dist<=90) {
          arrOrigins.get(i).x -= arrEnds.get(i).x * easing;
        arrOrigins.get(i).y -= arrEnds.get(i).y * easing; 43
        }
      }
    }
  }
}
```

III.9. Part

```
 1 class Part{
 2   int intId;
 3   String strName;
 4   int intInitTime;
 5   int intEndTime; 6
 7   Part(int id, String name, int intTime, int endTime){
 8     this.intId = id;
 9     this.strName = name;
10     this.intInitTime = intTime;
11   this.intEndTime = endTime; 12  }
13 }
```

III.10. Rain

```
class Rain{
  int maxDrops = 8000;
  //int minDrops = 500;
  int intFloor = 700;
  //int h,h1;
  int x1, y1, z1;
  Drop[] drops=new Drop[maxDrops];

  Rain(){
    for (int i = 0; i < maxDrops; i++){
      x1 = int(random(width));
      y1 = -int(random(height*2));
      z1 = int(random(intFloor, height));
    drops[i] = new Drop(x1,y1,z1,width);
    }
  }

  void display(int numDrops, boolean blnTransition, Fan fanA, Fan fanB, Fan fanC){
    if(!blnTransition){
    gradient();
    }
    for (int i=0;i<numDrops;i++){
      drops[i].fall(!blnTransition);
      if(!blnTransition){
        drops[i].wind(fanA.vecPosition.x,    fanA.vecPosition.y,fanA.intFanSize,fanA↙
fltFanSpeed);
        drops[i].wind(fanB.vecPosition.x,    fanB.vecPosition.y,fanB.intFanSize,fanB↙
fltFanSpeed);
        drops[i].wind(fanC.vecPosition.x,    fanC.vecPosition.y,fanC.intFanSize,fanC↙
fltFanSpeed);
      }
    }
  }

  void gradient(){
    offscreen.noStroke();
    offscreen.beginShape(QUADS);
    //fill(188,190,192);
    //fill(0);
    offscreen.vertex(0,0);
    offscreen.vertex(width,0);
    offscreen.fill(0,5,10);
    offscreen.vertex(width,height);
    offscreen.vertex(0,height);
  offscreen.endShape(); 43
    }
}
```

III.11. Smoke

```
PVector rootnCA = new PVector(random(123), random(123));//noise root
ArrayList<SmokePartCA> toAddCA = new ArrayList<SmokePartCA>();
class SmokeCA{
PVector speedn = new PVector(random(-.01, .01), random(-.01, .01));//noise speed
ArrayList<SmokePartCA> parts = new ArrayList<SmokePartCA>();//Parts
PVector m, pm;//mouse, previous mouse
float maxD = 10;//max distance between two smokes
int b = 10;

int intMinAge = 0;
int intMaxAge = 30;
int intMinLife = 50;
int intMaxLife = 180;

  SmokeCA(){

  }

  void display(float posx, float posy, boolean blnTransition){
    offscreen.noStroke();
    offscreen.fill(5, 15);
    offscreen.rect(0, 0, width, height);
    for (SmokePartCA p : toAddCA)
    {
      parts.add(p);
    }
    toAddCA = new ArrayList<SmokePartCA>();

    rootnCA.add(speedn);
    m = new PVector(posx, posy);
    int nb = parts.size()-1;
    if (blnTransition){pm=null;}
    if (!blnTransition && nb < 7000)
    {
      if (pm == null) pm = m.get();
      else
      {
        float d = PVector.dist(pm, m);
        if ((pm.x != m.x || pm.y != m.y) && d > maxD)
        {
          int n = int(d / maxD);
          PVector tmp = PVector.sub(m, pm);
          tmp.normalize();
          tmp.mult(maxD);
          PVector tmp2 = m.get();
          for (int i = 0; i < n; i++)
          {
            tmp2.sub(tmp);
            parts.add(new SmokePartCA(tmp2, (int)random(intMinAge, intMaxAge)↙
```

```
(int)random(intMinLife, intMaxLife), 0));
50            }
51          }
52        }
 53       parts.add(new SmokePartCA(m, (int)random(intMinAge, intMaxAge), (int)random
(intMinLife, intMaxLife), 0));
 54     pm = m.get();
 55     }
 56     nb = parts.size()-1;
 57     for (int i = nb; i > -1; i--)
 58     {
 59       if (parts.get(i).display())
 60     parts.remove(i); 61
        }
62    }
63 }
64
65 class SmokePartCA 66
{
 67   float rad, nx, ny;//
 68   float c = random(.6, .8);
 69   float theta = random(TWO_PI);
70    int life;// = (int)random(300000, 400000);
71    int age;// = (int)random(300, 400);
 72   int  mod = (int)random (30,40);
 73   PVector pos;
 74   float fltNoiseXFactor = 40.0f;
 75   float fltNoiseYFactor = 3.0f;
 76
77    SmokePartCA(PVector p, int a, int l, float r)
78    {
 79     pos = p.get();
 80     age = a;
 81     life = l;
 82   rad = r;
 83   }
84
85    Boolean display()
86    {
 87     nx = noise(rootnCA.x + pos.x/500)-.5;
 88     ny = -noise(rootnCA.y + pos.y/500)-.7;
 89     pos.add(new PVector(random(-fltNoiseXFactor,fltNoiseXFactor)*nx,
fltNoiseYFactor*ny));
 90     rad += cos(map(age, 0, life, 0, HALF_PI)) * c;
 91     offscreen.stroke(200, 200 * sq(map(age, 0, life, 1, 0)));
 92     offscreen.strokeWeight(rad);
 93     offscreen.point(pos.x, pos.y);
 94     if (age++ % mod == 0)//split the Part in two
 95     {
 96       toAddCA.add(new SmokePartCA(new PVector(pos.x + rad/2 * (cos(theta)), pos.y
+ rad/2 * (sin(theta))), age, life, rad * random(.6, .8)));
 97       toAddCA.add(new SmokePartCA(new PVector(pos.x - rad/2 * cos(theta), pos.y
rad/2 * (sin(theta))), age, life, rad * random(.6, .8)));//.6
 98     age = life+1;
 99     }
100     return age > life; 101     } 102 }
```

III.12. Traces

```
1 class Traces{
2   int intLineSize = 80;
3   int intLineWeight = 20;
4   float PY, PX;
5   float[] x = new float[intLineSize];
6   float[] y = new float[intLineSize];
7   int segLength = 10;
8   float fltFactor = 0.03;
9   int intDesfase = 300;
10  float fltAjusteX;
11  float fltAjusteY;
12  int intCount = 0; 13
14  PVector vecPosition;
15  PVector vecNewPosition;
16  ArrayList<PVector> arrPositions;
17  int intConta;
18  float fltFrameRate = RATE;
19  boolean blnDebug = false; 20
21  Traces () {
22    //size(600, 600, P3D);
23    //background(255);
24    //createCanvas(windowWidth,windowHeight);
25    for(int i=0; i<intLineSize; i++) { 26
      x[i]=1;
27        y[i]=1;
28    }
29    vecPosition = new PVector(0,0);
30    arrPositions = new ArrayList<PVector>();
31    for (int a = 0; a<=int(fltFrameRate); a++){
32    arrPositions.add(new PVector(0,0)); 33  }
34  }
35
36  void segment(float x, float y, float a){
37    //offscreen.strokeWeight(intLineWeight);
38    //offscreen.stroke(0, 0, 0,50);
39    offscreen.pushMatrix();
40    offscreen.translate(x, y);
41    offscreen.rotate(a);
42    offscreen.strokeWeight(intLineWeight);
43    offscreen.stroke(0,100);
44    offscreen.line(0, 0, segLength, 0);
45    offscreen.strokeWeight(intLineWeight-5);
46    offscreen.stroke(50,50);
47    offscreen.line(10, 10, segLength, 10);
48    offscreen.strokeWeight(intLineWeight-10);
49    offscreen.stroke(100,25);
50    offscreen.line(15, 15, segLength, 15);
51    offscreen.stroke(150,10);
52    offscreen.line(20, 20, segLength, 20);
```

```
53   offscreen.popMatrix(); 54}
55
56   void dragSegment(int i, float xin, float yin){
57     float dx = xin - x[i];
58     float dy = yin - y[i];
59     float angle = atan2(dy, dx);
60     x[i] = xin - cos(angle) * segLength;
 61     y[i] = yin - sin(angle) * segLength;
 62   segment(x[i], y[i], angle); 63
      }
 64
 65   void display(float posX, float posY) {
 66     offscreen.noStroke();
 67     offscreen.fill(255,12);
 68     offscreen.rect(0,0,width,height);
 69
 70     if(intCount%int(fltFrameRate)==0){
 71       fltAjusteX=0;
 72       fltAjusteY=0;
 73       if((sin(intCount*fltFactor*PI/4))>0){
 74         fltAjusteX += random(0,intDesfase);
 75       } else {
 76       fltAjusteX -= random(0,intDesfase); 77
         }
 78       if((cos(intCount*fltFactor*PI/4))>0){
 79         fltAjusteY += random(0,intDesfase);
 80       } else {
 81       fltAjusteY -= random(0,intDesfase); 82
         }
 83       if(blnDebug)println(intCount+" -> posX: "+posX+"\tfltAjusteX: "+fltAjusteX);
 84       if(blnDebug)println(intCount+" -> posY: "+posY+"\tfltAjusteY: "+fltAjusteY);
 85       posX+=fltAjusteX;
 86       posY+=fltAjusteY;
 87
 88       PX = posX;
 89       PY = posY;
 90
 91       if(blnDebug)println("Entrada: [X="+PX+", Y="+PY+"]");
 92       interpolar(PX,PY,true);
 93     } else {
 94     interpolar(0,0,false); 95
       }
 96     //PX = posX;
 97     //PY = posY;
 98
 99     dragSegment(0, PX, PY);
100     for(int i=0; i<x.length-1; i++) {
101       dragSegment(i+1, x[i], y[i]);
102     }
103
104     this.intCount++;
105   }
106
107   void interpolar(float fltPosX, float fltPosY, boolean blnInterpolate){
108     if(blnInterpolate){
109       asignarPosicion(this.intConta);
110       this.arrPositions = new ArrayList<PVector>(int(this.fltFrameRate));
```

```
111       for(int i = 0; i<int(fltFrameRate); i++){
112         this.vecNewPosition = new PVector(0,0);
113         this.vecNewPosition.x = lerp(this.vecPosition.x, fltPosX, i/this↙
fltFrameRate);
114         this.vecNewPosition.y = lerp(this.vecPosition.y, fltPosY, i/this↙
fltFrameRate);
115         this.arrPositions.add(vecNewPosition);
116       }
117       this.arrPositions.add(new PVector(fltPosX,fltPosY));
118       this.vecPosition.x = fltPosX;
119       this.vecPosition.y = fltPosY;
120       this.intConta =0;
121       //Imprimir lineas
122       for(int x = 0; x<this.arrPositions.size()-1;x++){
123           if(blnDebug)mostrarCoordenadas(x,false);
124       }
125     } else {
126       asignarPosicion(this.intConta);
127       if(blnDebug)mostrarCoordenadas(this.intConta,true);
128       this.intConta++;
129       if(this.intConta==(this.arrPositions.size()-2)){
130         asignarPosicion(this.intConta);
131         if(blnDebug)mostrarCoordenadas(this.intConta,true);
132         }
133     }
134   }
135   void mostrarCoordenadas(int intItem, boolean blnPinta){
136     if(blnPinta){print("Pinta ");}
137     print("LÃ-nea "+intItem);
138     print("\tdesde: [X"+intItem+"="+round(this.arrPositions.get(intItem).x)↵, Y" ↙
+intItem+"="+round(this.arrPositions.get(intItem).y)↵]");
139     println("\thasta:     [X"+(intItem+1)+"="+round(this.arrPositions.get(intItem+1).↙
x)+",     Y"+(intItem+1)+"="+round(this.arrPositions.get(intItem+1).y)↵]");
140   }
141
142   void asignarPosicion(int intItem){
143     PX = arrPositions.get(intItem).x;
144     PY = arrPositions.get(intItem).y;
145   }
146 }
```

III.13. Wave On Sphere

```
class WaveOnSphere{
  int intValor1 = 400;//radio
  int intValor2 = 300;//centro
  int intValor3 = 10000;//velocidad
  float fltFactor = 1.2f;

  int Nmax = 1000; float M = 50; float H = 0.99; float HH = 0.01;
  float R = 2*sqrt((4*PI*(intValor1*intValor1)/Nmax)/(2*sqrt(3)));
  float X[] = new float[Nmax+1] ; float Y[] = new float[Nmax+1] ; float Z[] = new float[Nmax+1] ;
  float V[] = new float[Nmax+1] ; float dV[] = new float[Nmax+1] ;
  float L ;
  float Lmin ; int N ; int NN ;
  float KX ; float KY ; float KZ ;
  float KV ; float KdV ; int K ;

  WaveOnSphere(){
    offscreen.stroke(255,255,255) ;
    offscreen.fill(50,50,50) ;
    for ( N = 0 ; N <= Nmax ; N++ ){
      X[N] = random(-intValor2,+intValor2) ;
      Y[N] = random(-intValor2,+intValor2) ;
    Z[N] = random(-intValor2,+intValor2) ;
    }
  }

  void display(float posX, float posY){
    offscreen.background(0) ;
    for ( N = 0 ; N <= Nmax ; N++ ){
       for ( NN = N+1 ; NN <= Nmax ; NN++ ){
          L = sqrt(((X[N]-X[NN])*(X[N]-X[NN]))+((Y[N]-Y[NN])*(Y[N]-Y[NN]))) ;
          L = sqrt(((Z[N]-Z[NN])*(Z[N]-Z[NN]))+(L*L)) ;
          if ( L < R ){
            X[N] = X[N] - ((X[NN]-X[N])*((R-L)/(2*L))) ;
            Y[N] = Y[N] - ((Y[NN]-Y[N])*((R-L)/(2*L))) ;
            Z[N] = Z[N] - ((Z[NN]-Z[N])*((R-L)/(2*L))) ;
            X[NN] = X[NN] + ((X[NN]-X[N])*((R-L)/(2*L))) ;
            Y[NN] = Y[NN] + ((Y[NN]-Y[N])*((R-L)/(2*L))) ;
            Z[NN] = Z[NN] + ((Z[NN]-Z[N])*((R-L)/(2*L))) ;
            dV[N] = dV[N] + ((V[NN]-V[N])/M) ;
            dV[NN] = dV[NN] - ((V[NN]-V[N])/M) ;
            offscreen.stroke(125+(Z[N]/2),125+(Z[N]/2),125+(Z[N]/2)) ;
            float orgX = X[N]*fltFactor*(intValor1+V[N])/intValor1+intValor2;
            float orgY = Y[N]*fltFactor*(intValor1+V[N])/intValor1+intValor2;
            float destX = X[NN]*fltFactor*(intValor1+V[NN])/intValor1+intValor2;
            float detsY = Y[NN]*fltFactor*(intValor1+V[NN])/intValor1+intValor2;
            line(orgX, orgY, destX, detsY) ;
          offscreen.line(orgX, orgY, destX, detsY) ;
          }
```

```
49            if ( Z[N] > Z[NN] ){
50              KX = X[N] ; KY = Y[N] ; KZ = Z[N] ; KV = V[N] ; KdV = dV[N] ;
51              X[N] = X[NN] ; Y[N] = Y[NN] ; Z[N] = Z[NN] ; V[N] = V[NN] ; dV[N] = dV
[NN] ;
52              X[NN] = KX ; Y[NN] = KY ; Z[NN] = KZ ; V[NN] = KV ; dV[NN] = KdV ;
53             }
54         }
55         L = sqrt((X[N]*X[N])+(Y[N]*Y[N])) ;
56         L = sqrt((Z[N]*Z[N])+(L*L)) ;
57         X[N] = X[N] + (X[N]*(intValor1-L)/(2*L)) ;
58         Y[N] = Y[N] + (Y[N]*(intValor1-L)/(2*L)) ;
59         Z[N] = Z[N] + (Z[N]*(intValor1-L)/(2*L)) ;
60         KZ = Z[N] ; KX = X[N] ;
61         Z[N] = (KZ*cos((intValor2-posX)/intValor3))-(KX*sin((intValor2-posX)↙
/intValor3)) ;
62         X[N] = (KZ*sin((intValor2-posX)/intValor3))+(KX*cos((intValor2-posX)↙
/intValor3)) ;
63         KZ = Z[N] ; KY = Y[N] ;
64         Z[N] = (KZ*cos((intValor2-posY)/intValor3))-(KY*sin((intValor2-posY)↙
/intValor3)) ;
65         Y[N] = (KZ*sin((intValor2-posY)/intValor3))+(KY*cos((intValor2-posY)↙
/intValor3)) ;
66         dV[N] = dV[N] - (V[N]*HH) ;
67         V[N] = V[N] + dV[N] ; dV[N] = dV[N] * H ;
68       }
69     }
70
71     void change(float posX, float posY){ 72
       Lmin = 600 ; NN = 0 ;
73       for ( N = 0 ; N <= Nmax ; N++ ){
74         L = sqrt(((posX-(intValor2+X[N]))*(posX-(intValor2+X[N])))+((posY↙
(intValor2+Y[N]))*(posY-(intValor2+Y[N])))) ;
75       if ( Z[N] > 0 && L < Lmin ){ NN = N ; Lmin = L ; } 76       }
77       if ( K == 0 ){ dV[NN] = -intValor1 ; K = 1 ; }
78              else{ dV[NN] = +intValor1 ; K = 0 ; } 79
80     }
81 }
```

III.14. Wave Renderer

```
1 class WaveRenderer implements AudioListener 2 {
3   private float[] left;
4   private float[] right; 5
6   WaveRenderer() 7   {
8     left = null;
9   right = null; 10   }
11
12   public synchronized void samples(float[] samp) 13
    {
14     left = samp; 15  }
16
17   public synchronized void samples(float[] sampL, float[] sampR) 18 {
19     left = sampL;
20   right = sampR; 21  }
22
23   public synchronized int getSize(){
24     if ( left != null && right != null ){
25       if(left.length<right.length){return left.length;}
26       else{return right.length;}
27   }else{return 0;} 28
    }
29
30   public synchronized float getLeft(int pos){
31     if (pos>left.length-1){return 0;}
32   else{return left[pos];} 33     }
34   public synchronized float getRight(int pos){
35   if (pos>right.length-1){return 0;}
36   else{return right[pos];} 37    }
38 }
```

Anexo IV

Estudio estadístico Xsens

Se ha realizado un estudio estadístico de las posibles magnitudes de interés que son capaces de medir los Xsens. El MT Manager permite entre sus diferentes aplicaciones, grabar los datos que se seleccionen, durante el periodo de tiempo que decida el usuario y posteriormente exportar esos datos medidos, a un archivo de texto para su posible evaluación o uso posterior. Lo que se ha llevado a cabo es medir estas variables durante un ensayo del baile y proceder a su evaluación en una hoja excel. Debido a la gran cantidad de medidas que los sensores graban por minuto no se va a incluir todos los datos con los que se realizó el estudio, se explicará el procedimiento realizado incluyendo una serie de capturas de la hoja de cálculo utilizada. Las gráficas donde se aprecían las distintas distribuciones de cada variable se exponen en el apartado 4.3 por lo que no se encuentran otra vez en este anexo.

En primer lugar una vez disponemos de los datos en el archivo de texto se ordenan en una tabla. El packet Counter es simplemente un contador de las mediciones.

Packet Counter	Acc X	Acc Y	Acc Z	Free Acc X	Free Acc Y	Free Acc Z	Roll	Pitch	Yaw
18781	-0.66953	-2.01992	9.716392	-0.181185	-0.077156	0.132044	-10.706781	4.410975	92.6245
18782	-0.71700	-1.879793	9.797754	-0.021708	-0.098766	0.18898	-10.803109	4.597984	92.460386
18783	-0.754758	-1.891764	9.677196	-0.017867	-0.086415	0.076125	-10.974726	4.792669	92.312485
18784	-0.82163	-1.90847	9.73591	0.029084	-0.049259	0.142303	-11.20079	4.917798	92.22433
18785	-0.849172	-2.065934	9.657749	-0.118525	0.001734	0.099285	-11.33402	5.000987	92.190785
18786	-0.768337	-2.026757	9.662242	-0.084234	-0.085531	0.088957	-11.423496	5.02266	92.190808
18787	-0.859858	-2.058861	9.669273	-0.099798	0.010654	0.110167	-11.436765	5.025463	92.185618
18788	-0.818753	-1.969421	9.646426	-0.01529	-0.044983	0.066597	-11.482865	5.027554	92.179943
18789	-0.772246	-2.046662	9.632645	-0.081673	-0.080357	0.064554	-11.577474	5.033497	92.173076
18790	-0.828663	-2.004453	9.671886	-0.009049	-0.046688	0.099306	-11.667802	5.059078	92.179353
18791	-0.82395	-1.856834	9.682912	0.185442	-0.098835	0.078784	-11.907804	5.104133	92.167655
18792	-0.758193	-1.90389	9.838826	0.282617	-0.217145	0.23093	-12.493268	5.181111	92.089363
18793	-0.669642	-1.556219	10.197559	1.041845	-0.505935	0.459213	-14.114327	5.278509	92.024489
18794	-0.491147	-3.218776	10.472673	-0.316833	-0.621809	1.131959	-15.451971	5.81725	91.990496
18795	-1.453909	-2.97722	10.521287	-0.016546	0.052116	1.21768	-15.589808	6.824047	91.810682
18796	-1.948548	-3.774313	10.435676	-1.183622	0.435947	1.382895	-14.148442	8.341853	91.68194
18797	-2.563373	-3.461089	10.311833	-1.271359	0.691737	1.267319	-12.222104	10.229457	92.00553
18798	-2.884387	-3.488424	9.341973	-1.881222	0.913544	0.354037	-9.987549	12.231214	93.127065
18799	-3.694817	-3.011717	8.588505	-1.922171	1.527467	-0.303368	-7.355685	14.107266	95.341911
18800	-4.111133	-2.008642	8.626801	-1.42471	1.591163	-0.285523	-4.67667	15.552084	98.761969
18801	-3.60337	-1.455032	8.529683	-1.426969	0.963307	-0.600316	-1.84185	16.331796	103.192183
18802	-3.723374	0.317923	9.408555	-0.118034	0.955485	0.263908	0.645585	16.249161	108.47721

Fig A. 1

El siguiente paso es obtener la media y la desviación estandar de cada variable, así como los máximos y los mínimos para estimar a partir de que valores filtrar los datos, dando las medidas fuera de este rango por erróneas.

	Media	Desviación Estándar	Máximos	Mínimos
Acc X	5.472749	7.279602	38.610386	-9.0572
Acc Y	-4.571088	5.219101	20.975487	-25.772122
Acc Z	3.081231	6.235729	19.090963	-22.797124
Free Acc X	-0.280457	5.157196	24.173292	-33.918068
Free Acc Y	-0.216841	5.552531	32.756066	-23.727958
Free Acc Z	-0.183611	5.332864	28.963514	-19.096668
Roll	-39.275031	52.955055	172.610241	-171.809665
Pitch	-9.133554	33.427515	85.061673	-88.976736
Yaw	-33.150029	93.127112	179.411548	-179.858524

Fig A. 2

Por último a partir de la media y la desviación estándar obtenida, se porcede a calcular la distribución normal de cada magnitud para posteriormente poder obtener las curvas pertinentes.

T4 =DISTR.NORM(Tabla3[Acc X];O6;P6;FALSO)

	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB
1														
2						Distribución Normal								
3						Acc X	Acc Y	Acc Z	Free Acc X	Free Acc Y	Free Acc Z	Roll	Pitch	Yaw
4						0.0383890	0.06783119	0.03632166	0.0773421	0.0718260	0.074677323	0.006513	0.010994	0.001721
5	Media	Desviación Estándar	Máximos	Mínimos		0.0381775	0.06692266	0.03581782	0.0772591	0.0718325	0.074625893	0.006520	0.010969	0.001725
6	5.472749	7.279602	38.610386	-9.0572		0.0380090	0.06700169	0.03656468	0.0772562	0.0718289	0.074719578	0.006531	0.010943	0.001729
7	-4.571088	5.219101	20.975487	-25.772122		0.0377099	0.06711153	0.03620071	0.0772172	0.0718160	0.074668682	0.006546	0.010925	0.001731
8	3.081231	6.235729	19.090963	-22.797124		0.0375865	0.0681215	0.03668533	0.0773183	0.0717931	0.074703071	0.006555	0.010914	0.001732
9	-0.280457	5.157196	24.173292	-33.918068		0.0379483	0.06787459	0.03665745	0.0773005	0.0718286	0.074710606	0.006560	0.010911	0.001732
10	-0.216841	5.552531	32.756066	-23.727958		0.0375385	0.06807714	0.03661383	0.0773090	0.0717884	0.07469483	0.006561	0.010911	0.001732
11	-0.183611	5.332864	28.963514	-19.096668		0.0377228	0.06750797	0.03675559	0.0772542	0.0718143	0.074725961	0.006564	0.010910	0.001732
12	-39.275031	52.955055	172.610241	-171.809665		0.0379309	0.06800041	0.03684112	0.0772990	0.0718270	0.074727299	0.006570	0.010909	0.001732
13	-9.133554	33.427515	85.061673	-88.976736		0.0376784	0.06773271	0.03659762	0.0772494	0.0718150	0.074703055	0.006576	0.010906	0.001732
14	-33.150029	93.127112	179.411548	-179.858524		0.0376995	0.06677038	0.03652923	0.0770414	0.0718325	0.074717755	0.006592	0.010900	0.001732
15						0.0379937	0.06708147	0.03556385	0.0768967	0.0718487	0.074582582	0.006629	0.010889	0.001734
16						0.0383885	0.06469253	0.03335913	0.0748550	0.0717514	0.074266745	0.006729	0.010875	0.001736
17						0.0391792	0.07391554	0.03169025	0.0773545	0.0716579	0.072566253	0.006809	0.010799	0.001737
18						0.0348499	0.07295623	0.03139779	0.0772552	0.0717645	0.072269736	0.006817	0.010649	0.001741
19						0.0325926	0.07555329	0.03191334	0.0761792	0.0713539	0.071649412	0.006732	0.010410	0.001744
20						0.0297973	0.07472952	0.03266323	0.0759416	0.0708932	0.072090068	0.006612	0.010091	0.001736
21						0.0283539	0.07481178	0.03864827	0.0737183	0.0703752	0.074429034	0.006465	0.009730	0.001708
22						0.0247979	0.07310205	0.0433158	0.0735345	0.0683895	0.074789394	0.006282	0.009372	0.001654

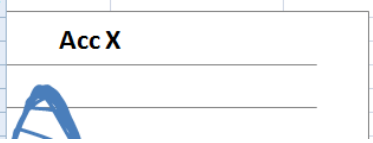


Fig A. 3

Bibliografía

https://shop.xsens.com/

https://es.wikipedia.org/wiki/Sistema_embebido

http://huribroadcast.com/que-es-broadcast/

https://es.wikipedia.org/wiki/Processing

https://processing.org/

https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)

https://www.visualstudio.com/es/vs/

https://es.wikipedia.org/wiki/C%2B%2B

https://es.mathworks.com/products/matlab.html

https://es.wikipedia.org/wiki/MATLAB

http://processing.joan.cat/cs/

https://www.programarya.com/Cursos/Java-Avanzado/Sockets

http://openframeworks.cc/about/

https://www.openprocessing.org/

https://es.wikipedia.org/wiki/Neil_Harbisson

http://www.kinectfordevelopers.com/es/2012/11/06/que-es-el-dispositivo-kinect/

http://pdf.directindustry.com/pdf/ubisense/korean-certified-ip-sensors/124957-508109.html

https://ubisense.net/en/products/Dimension4

http://todoproductividad.blogspot.com.es/2008/04/aplicacin-de-ethernet-en-un-entorno.html

http://mlab.no/blog/wp-content/uploads/2009/11/ubisense-tag.png

https://www.researchgate.net/figure/304025734_fig2_Fig-5-Ubisense-UWB-Real-time-Location-System-UWB-RTLS-consisting-of-4-sensors-UWB

https://es.wikipedia.org/wiki/Isadora_Duncan

http://pop-picture.blogspot.com/2015/03/asfixia--una-fusion-sorprendente-de-danza-y-tecnologia-de-captura-de-movimiento-.html

http://www.margaritabali.com/prensa/EN%20MOVIMIENTO-DANZA%20Y%20TECOLOG%C3%8DA.pdf

http://www.rehabilitacionblog.com/2011/05/xsens-analisis-del-movimiento-humano.html

http://www.danza.unam.mx/images/Curso/2016/Diplomado2016/diplomado-danza-mediacion-teconologica-web.pdf

https://riunet.upv.es/bitstream/handle/10251/3838/tesisUPV2962.pdf

http://www.huffingtonpost.es/2014/11/20/zapatos-ballet-tecnologia_n_6186754.html

http://www.elfinanciero.com.mx/after-office/la-belleza-de-la-danza-se-fusiona-con-la-tecnologia.html

http://publicaciones.zemos98.org/spip.php?article115

https://www.onysus.com/dance-technology/

http://www.exile.at/apparition/project.html

http://www.am-cb.net/en/projets/cinematique

https://es.wikipedia.org/wiki/Danza_contempor%C3%A1nea

https://definicion.de/danza-contemporanea/

http://www.ulima.edu.pe/departamento/vida-artistica-en-la-universidad/danza-contemporanea