TREBALL FINAL DE GRAU

GRAU EN ENGINYERIA INFORMÀTICA

FACULTAT DE MATEMÀTIQUES
UNIVERSITAT DE BARCELONA

HUMAN COMPUTER INTERACTION FOR 3D MODEL VISUALIZATION USING SENSOR FUSION

Àlex Pardo Fernández
alexpardo.5@gmail.com

Director: Dr. Oriol Pujol Vila
Departament de Matemàtica Aplicada i Anàlisi
Barcelona, 20 de Juny de 2013

Special thanks to Oriol Pujol for the given help during the realization of the project

Produced with LaTeX

Abstract

In the last years, three dimensional visualization has become a widely extended tendency, including this technology on televisions or even Smartphones. Moreover, 3D imaging has been also introduced on medical diagnosis in the last years. However, volumetric objects have a rough interaction because the devices used to control them are bi-dimensional. For that reason, wearable sensors will give a three-dimensional control for this visualization helping the user to interact in a more natural manner. This kind of control has been a common subject in computer vision research, even more with the introduction of Microsoft® Kinect ™, but in the wearable sensor area, this is not a commonly extended subject of research and there are even less works including sensor fusion. This project proposes a system for managing a volume visualizer including in the development the whole system from the sensor data acquisition to the model visualization. The system will consist on two wearable sensors (with accelerometer and gyroscope) that will be attached to the arms of the user, an android phone will stream the values acquired from the sensors (via Bluetooth ® communication) to the Server using a wireless TCP/IP protocol. The server will store the values in the computer and a 3D Engine will process this data using a complementary filter to fuse the acceleration (given by the accelerometer) and the angular velocity (given by the gyroscope) in order to obtain the performed action and apply the required transformations to the displayed model. In order to validate the system, a real-case example will be used, trying to find some anomalies on a 3D textured model of a heart in a simulated medical diagnose process. This project tries to build a complete system of natural interaction between the human and the computer by using wearable sensors.

Contents

1 Introduction

In this Undergraduate Thesis Project, I present a 3D model visualization controller using wearable sensors. This kind of application is useful for example in medical imaging, where 3D models are obtained and have to be visualized in an easy way. This project has the aim of achieving an easy-to-use application from the point of view of HCI (Human-Computer Interaction), that means, a natural gesture system in order to communicate the user and the computer and make this communication intuitive.

This system will be characterized by the fact of using wearable sensors and sensor fusion techniques in order to use the best features of each sensor (accelerometer and gyroscope) to get the best approximation of the orientation of the device. The application has to run in real-time because the user will be expecting an immediate output from the given input (i.e. a gesture). If the interaction is not shown in an small period of time, the user will think there is an error and the application will not be usable.

It is also important to notice the significance of using wearable sensors instead of computer vision techniques. That means the system will work even when the user is far from the computer or even when the light conditions are not optimal (e.g. outdoors). A prototypical application of this technology will come with the advent of all the wearable glasses for augmented reality which allow the user to wear a visualization system also integrated with other components like a CPU, Bluetooth technology, Wi-fi or even also a camera. It could be said that this kind of sensors will constitute an Egocentric Computing point of view in contrast to the Environment Computing formed by the computer vision. This Egocentric Computing isolates the user from the context, making the system not to be limited by the environmental conditions.

As it has been previously commented, the application is demonstrated in a medical imaging scenario but could also be used on other areas such as structural design, hardware design or any discipline not oriented to computer or 3D designer experts. That is because, 3D modeling experts have more complex tools to perform this visualization and they are used to them.

Medical imaging techniques such as MRI (Magnetic Resonance Image), SPECT (Single-Photon Emission Computed Tomography) or Ultrasound [1] , produce 3D models of the scanned parts of the human body and are really useful in disease diagnosis.

There are a lot of works on gesture recognition and 3D model interaction, but most of them are developed using computer vision techniques. In [1] a close-range gesture recognition using Microsoft Kinect is presented. It uses nearest-neighbour algorithm in classification. They use 8 different hand gestures for real-time classification. The system is validated by using a dataset of 20 subjects and achieving correct classification rates of 99%. If we focus on human-computer interaction, [2] presents a Harlet-based hand gesture recognition integrated on an interactive robot. The pointing gestures are translated into goals for the robot, telling it where to go. In the area of 3D object interaction we can find works like [3] where a system is designed for recognizing natural gestures (i.e. open or closed hand). The gestures are recognized by using hand tracking, pose classification is given by using modified SURF descriptors and finally the gesture is found making use of Hidden Markov Models. The gesture classification is used to

[1] Found at http://en.wikipedia.org/wiki/Medical_imaging

control a 3D virtual drag and drop software. Finally, [4] is similar to [3], however a Neural Network is used to identify the performed gesture and the software implemented allows 3D object handling or 3D navigation in a large immersive display.

Three-dimensional GUI (Graphical User Interface) are not only used on medical imaging or design but also on applications that try to reach a simpler way to present the data (values, objects, etc.) to the user. In [5] a virtual Greek vase museum is presented, displaying and browsing vases from a digital library collection.

There are several methods available to fuse sensor data (i.e. accelerometer, gyroscope and, sometimes, also magnetometer). In [6] a gradient-descend algorithm is used to obtain quaternion-based orientation using MARG (Magnetic, Angular Rate and Gravity) sensors which gives better accuracy than Kalman filters. As could be seen in [7], the Kalman filter implemented in order to develop a real-time body motion tracking system using MARG sensors does not performs well when fast movements are executed and lag appears. The results are visualized in a human-like avatar. If the system does not need a high precision but requires a good performance, complementary filter is the best choice as could be seen on [8], where the filter is used to obtain the rotation matrix of an UAV (Unmanned Aerial Vehicle) using a low-cost IMU (Inertial Movement Unit). Also [9] is a good example of complementary filter performance where it is used to estimate the attitude and the heading of a vehicle. Attitude achieves a RMS (Root Mean Square) error of 0.2214, 0.6720, 2.0788 deg for roll, pitch, and heading, respectively. In [10] a survey of different sensor fusion methods is presented and will be analysed in detail in the Design section.

In the area of HCI, some work has been developed using computer vision techniques. Focusing on medical imaging, there is a special interest on the development of touchless user interfaces for sterile environments in order to avoid physical interaction with the computer. In [11], a gesture-based navigation interface is developed for medical 3D models interaction. A ToF (Time of Flight) camera is used. A 94% of accuracy is achieved in the classification and, in a satisfaction survey, the users have given a 4 in a 1-5 range (where 1 is the worst value). Also in the same area, [12] uses a Kinect based system to recognize gestures and interact with 3D images. Several gestures are recognized such as pointing, animating, zooming, translation and among others.

In the same scope but using wearable sensors, in [13] a system using the Nintendo WiiMote controller is designed with the objective of interacting with volumetrical images in a semi-immersive virtual environment. The application allows to perform rotations, translations, zoom, pointing and viewing inside the object or as they name this, crop. In order to visualize the images, polarized projectors and glasses are used to create 3D images. Working also on medical images, [14] proposes the use of the widely used Smartphones to interact with a professional medical volumes software of visualization. The mobile phone is only used as a support for the interaction and does not replace at all the keyboard. The accelerometer of the phone is used to perform all the geometrical transformations. The validation tests where performed both with doctors and with non-specialists and achieved good results. However, the article is centred on the interface more than on the movement detection. As could be seen there is no much work on gestural control using wearable sensors as controller.

The application developed in this project, is not only focused on achieving orientation values from the raw data of the sensor but also on the graphical interface by itself. A fully working system is the final goal of this project and, for that reason, it should include a well defined set of actions, run in real-time, allow the user to perform an intuitive control of the three-dimensional

model and have the ability of being run using other platforms than the one used in this project, achieving a high compatibility level.

The rest of the Undergraduate Thesis Project is structured as follows: in section 2, the *Motivation* to carry out this project will be explained. The main *Objectives* will be introduced next.Then the proposed system will be analysed on the *Analysis* section. Once the project have been defined, the *Design* will be explained and lately how the *Development* has been made. The next section will be *Set-up* that will explain how to make the system run. Finally *Result and Validation* and *Conclusion* are presented.

2 Motivation

In this section, the motivation of the project will be introduced in order to understand all the decisions that will be made during the design and the development processes.

The main motivation to carry on this project is the introduction of 3D models in our lives by the simplification of the digitalization of real objects and the amenities to visualize them using stereo images or polarized 3D systems. These three-dimensional objects are frequently used in prototyping, disease diagnosis, structural design, video games and quite a lot of others.

In the lasts years, lot of new user interfaces for ease the interaction between humans and computers have appeared. There are even some films such as Minority Report (2002, Steven Spielberg) or Iron Man 2(2010, Jon Favreau) that use gestural interfaces to interact with computers as could be seen on Fig. 1. Also, the newest TV like Samsung Smart TV and video-game platforms like Microsoft Xbox using Kinect have a gesture recognizer.

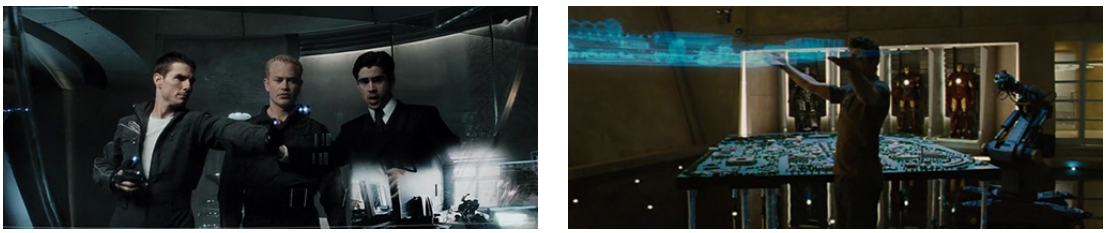

Figure 1: *Minority Report* and *Iron Man 2* examples of gesture interaction

There is a global need of making the human-computer interaction easier and much more natural, for example using voice commands, gestures or even eye or head tracking.

Since visualizing with 3D models could be difficult to non-expert users, it should help to introduce a natural way of interacting with them reducing the effort needed to learn how to use complex interfaces.

It could be seen that, the main area that uses these models as a support for their job is medical diagnosis. In this sector, the people that uses 3d visualization systems are not needed to be professionals or have an additional knowledge of these systems. For this reason, the design of an application easy to use and non-expert oriented will help to reduce the time needed to learn how to interact with the objects and will help them to perform rotations, translations, shears, and other transformations just having a basic intuition of what they pursue.

Nevertheless, this system is not only oriented to medical imaging but also to any area using 3D models. The main advantage of focusing the design in an area where the final users may have not experience with this kind of visualization applications, will make it simpler and much more natural.

There are three main reasons to use wearable sensors instead of vision-based systems. First of all, I have some experience with inertial sensors and wearable devices and this simplifies the project because there is a previous learning on this area. Another reason to use this devices is the few works that could be found of systems working with them and even less using wearable sensors for human-computer interaction and the huge amount of them in computer-vision area.

The third and possibly the most important one is the potential weakness of computer vision systems. If markers are used, they are not so accurate as wearable sensors. Depth cameras such as Kinect are also constrained by the environment and the precision of measures will be conditioned by the occlusions, the illumination, if there is other people in the range of vision or even the distance from the user to the camera. Additionally, notice the environments based on vision are static and, with the appearance of wearable systems that allow interactions in non-static environments (e.g. Google Glasses), wearable systems are the best choice.

3 Objectives

The main objective of this project is to design and develop an easy-to-use interface for 3D models visualization, including interactions with an object for visualization purposes.

The main requirements that are pretended to be achieved by the end of this project are:

- Design and develop a communication pipeline between some sensors, a Smartphone and a PC
- Design and develop a vocabulary of gestures to control the graphical interface
- Design and develop an algorithm to recognize the defined gestures
- Design and develop a user interface to validate the technology in a medical image example case

The skills and knowledges expected to be acquired by the realization of this project are:

- Familiarize with LaTeX language
- Familiarize with the wireless communications in Android environment
- Familiarize with wearable sensors, specifically inertial measurement units
- Study and analyse different Sensor Fusion techniques

Secondary goals of the project are the following:

1. Analyse different types of interaction between the application and the user in order to compare them and extract the best features of each type
2. Add more than one method to classify the gestures in order to compare them and check their performance

4 Analysis

The proposed application aims to improve the interaction between a human and a 3D model visualization engine by using wearable sensors and a modular communication system going from the sensor, to a 3D engine through a Smartphone and a PC Server. For that reason a set of gestures will be recognized by the application and translated into spatial transformations (e.g. zoom, translation, etc.) and visualization modifications (i.e. wire-frame/textured model).

4.1 Gestural vocabulary

First of all, a set of actions has to be defined in order to specify the use cases that will be implemented.

- Zoom: change the scale of the model in order to see the smallest details or have a full view of the object.
- Rotate: transform the object around a coordinate center in order to see all the surface and have different point of view in order to explore the model.
- Pan: translate the object with respect to the coordinate origin in the screen in order to, for example, search a small detail when the scale is big enough to fill the all the screen with the model.
- Center: move the model to its initial position, set the default scale and the initial rotation.
- Texture: alternate between the textured model and the wire-frame (see Fig. 2 in order to see the structural details.

Then, a gesture could be defined for every action defined above. In this system, a gesture will be defined as a movement of one of the two sensors in a specific axis. Every action will have zero, one, two or four gestures. In essence, *Center* has no gesture defined, *Texture* has only one, two gestures are defined for *Zoom* and *Pan* or *Rotate* have four different gestures.

Looking the system from the point of view of the user that is interacting with it, the use-cases will be defined as follows:

- **Rotate** The user places the sensors in the rotation mode position. A rotation icon will be displayed on the screen. Then, the user moves each sensor in the axis defined for the rotation gesture. The model will be rotated in the movement direction.
- **Pan** The user places the sensors in the pan mode position. A pan icon will be displayed on the screen. Then, the user moves each sensor in the axis defined for the pan gesture. The model will be moved in the movement direction.
- **Zoom** The user places the sensors in the zoom mode position. A zoom icon will be displayed on the screen. Then, the user moves the specified sensor along the zoom axis defined for the rotation gesture. The model will be scaled.
- **Center** The user places the sensors in the center mode position. A center icon will be displayed on the screen. The model will return to the starting position.

- **Toggle Wire-frame** The user places the sensors in the toggle wire-frame mode position. A wire-frame icon will be displayed on the screen. Then, the user moves the specified sensor along the toggle axis defined for the wire-frame gesture. The texture of the model will be changed.

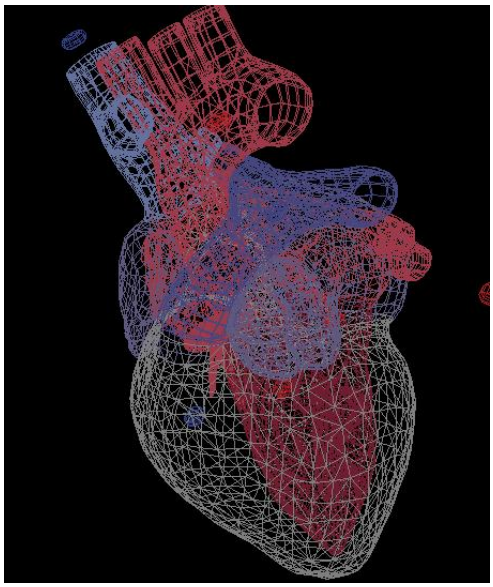

Figure 2: Wire-frame of a heart model.

4.2 Hardware

The hardware used in this system is composed by two 9-DOF (Degrees Of Freedom) Shimmer ® wearable sensors tied to the arms of the user as could be seen in Fig. 3.
In order to send the data obtained by the sensors, a mobile phone is used as a HUB to send the information over Internet to the server running on a PC.

Figure 3: Shimmer ® sensors attached to the arms.

4.2.1 Accelerometer

Accelerometer is a measurement device used to measure the acceleration of any body. This device is characteristic by measuring zero on free fall but g (Earth Gravity) standing still, that is

because the accelerometer uses the phenomenon of weight of a test mass at rest in the frame of reference of the device. Its units, specified by International System of Units (SI), are m/s^2

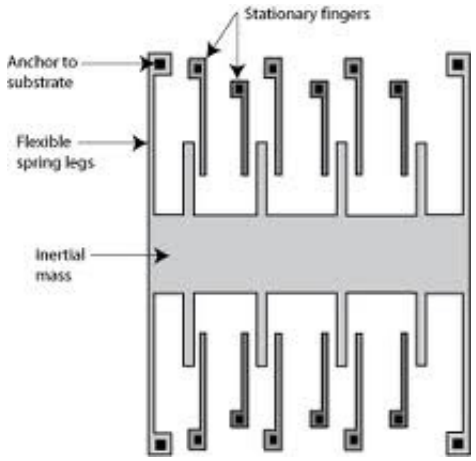

Figure 4: MEMS Accelerometer working scheme

In the Shimmer Device, a MEMS (Micro Electro-Mechanical System) accelerometer is used. The principle of this type of sensor is the same as a classic accelerometer, it has a housing or a frame of reference and a mass attached to a spring, then the acceleration is measured by the elongation of the spring (See Fig. 4). In a MEMS accelerometer, the housing and the mass are embedded in a tiny chip and are made of silicon.

4.2.2 Gyroscope

A classical Gyroscope is a device for measuring orientation based on the angular momentum principles. A MEMS Gyroscope, as used in Shimmer Dveice, is based on the physical principle that a vibrating mass tends to continue vibrating in the same plane of its rotation support (See Fig. 5).

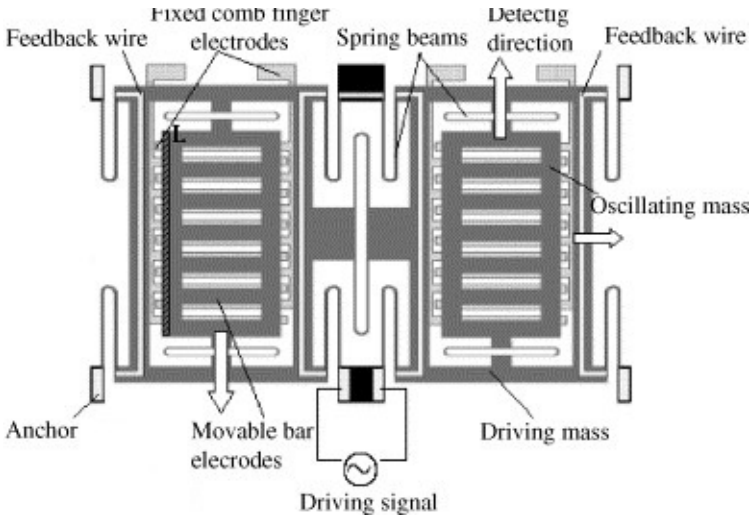

Figure 5: MEMS Gyroscope working scheme

Gyroscopes measure the angular speed of any object. Its units, specified by SI are rad/s. Gyroscope bias drift is a known problem of MEMS gyroscopes and causes big deviations in the measures by adding an offset to the measures that changes over the time.

4.2.3 Magnetometer

Magnetometer sensors are used to measure strength and direction of a magnetic field. A MEMS Magnetometer relies on the mechanical motion of the structure due to the Lorentz force acting on the conductor placed in the magnetic field. The movement produced by this force is sensed electronically. The Lorentz force is also the working principle of electric motors.

Shimmer Device magnetometer units are gauss (G), which is specified in the CGS (centimeter-gram-second System of Units).

4.3 System and Technologies

In this section, all the system components will be analysed and explained.

First of all, **Shimmer Device** uses a low-power MSP430F1611 microprocessor, has an 8-channel 12bit analog-to-digital converter (ADC) which is used to capture sensor data from the accelerometer, battery, or sensor expansions (such as kinematic module, i.e. gyroscope and magnetometer). The platform is equipped with Bluetooth and 802.15.4 radio modules, also has a microSD Flash socket for additional storage. The technology used to communicate the sensor and the mobile phone is **Bluetooth**. This allows short-distance wireless data exchanging using short-wavelength radio transmissions in the ISM band from 2400-2480 MHz. This standard creates a Personal Area Network (PAN) with high levels of security. Bluetooth uses a packet-based protocol with a master-slave structure. Bluetooth 3.0 version + High Speed has a theoretical data transfer of 24Mbit/s [2].

Android phone is connected with the Server using a Local Area Network (LAN) or Internet, depending on the Server and Phone proximity. The connection goes through a **Socket** interface (i.e. Java Sockets), using TCP connection and a custom port, 1234. This implementation uses a traditional client-server connection where the client produces (in this case the data is actually produced by the sensors) and the server stores the information. The connection is designed to be non-blocking and do not produce deadlocks since a wait time is specified for reading operations, so the process will not wait forever.

The third connection is between the **Server** and the **Engine** which will process all the data. This is a high-level connection since both are connected through a **Circular-Buffer** object. That raises the classic consumer-producer problem. The producer generates data and stores it in a fixed-size buffer and the consumer is reading the data stored. In order to synchronize both processes, the buffer ensures the reading pointer is always behind the writing one. This could cause the consumer to be frozen until a new element is written into the buffer but this could not be solved and in the performed tests had never happened.

[2]Extracted from Bluetooth® Core Specification Version 3.0 + HS available in: https://www.bluetooth.org/en-us/specification/adopted-specifications

However, the last connection is the most important: between the **3D Engine** and the user in order to display the results. The engine I decided to use is JMonkeyEngine 3 RC2, which is a fast and easy to code OpenSource Engine and allows managing 3D models in an easy way. Its written in Java and it is not as efficient as others and the quality of the models has to be lower in order to compensate it but for this project is the best option.

All the protocols used and the 3D Engine will be explained in the following sections.

Additional software and libraries used in this project are:

- **Shimmer Android Library**: used to manage the connection between the sensor and the Smartphone
- **Java Swing Library**: used to design a preference panel for the 3D Engine
- **Shimmer 9DOF Calibration Software**: used to calibrate the sensors and test the output values
- **MathWorks Matlab**: used to develop a prototype of the sensor fusion algorithm

5 Design

5.1 Architectural design

5.1.1 Architecture

In this section, the whole architecture of the project will be described with the aim of making easier the task of understanding how it works and give a complete overview of the project. The system is composed by two wearable sensors, an Android phone and a computer running the Server and the 3D engine. Fig. 6 shows the system pipeline and the communication technology between the different modules. From left to right, the pipeline goes from the two **Shimmer sensors**, which send samples to the **Smartphone** via Bluetooth. The phone is also connected to the **Server** which will be listening for packets and storing them in a Circular Buffer. The stored samples will be red by the **3D Engine** which will process them in order to fuse the values and obtain the orientation, the gravity vector and also the linear acceleration of each sensor.

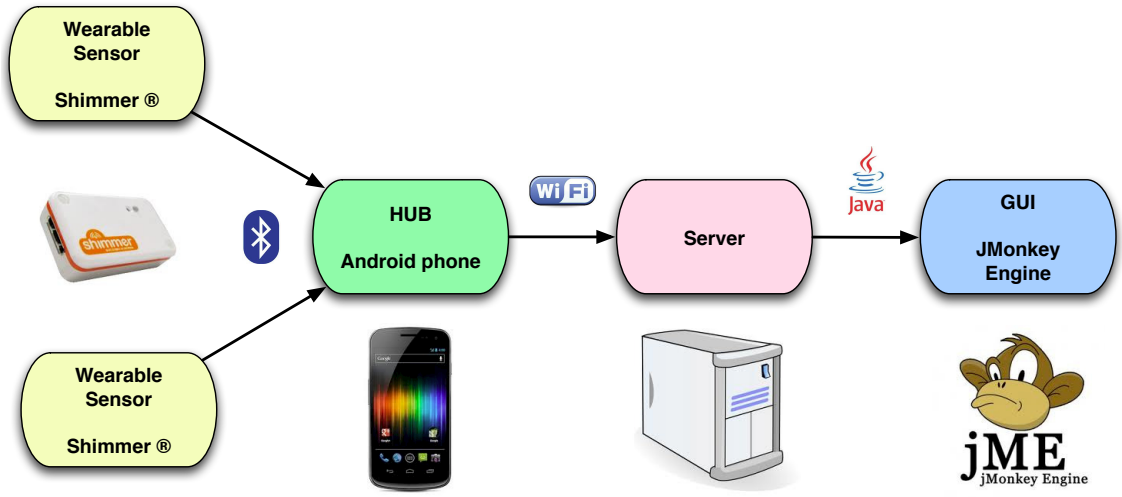

Figure 6: System diagram

The easiest way to understand the communication pipeline is taking a single data sample and follow all the process with it. We will suppose the phone is connected with the sensor and the server, the 3D Engine is running and the Sensor is sending samples.

1. A sample is taken by the Sensor firmware, packaged and sent using a specific protocol defined in the following sections.

2. The Android Phone receives a sample via Bluetooth® and is stored in a buffer. The application reads this value, generates a time-stamp in order to track the value, transforms the measures into a specific format defined in section *Android Application* and sends it to the server.
3. The Server is waiting for an incoming sample. Once the packet arrives and is read, the codified values(i.e. transformed into Strings) are converted into floating point values. A new Data object, containing the device number, the time-stamp an all the values is created and stored into the Circular Buffer that enables the communication between the Server and the 3D Engine.
4. Once the 3D Engine has processed a sample, will read another from the Buffer. Once the sample is read, a filter is applied in order to acquire the Sensor Orientation and Linear Acceleration. With these values the state machine will be updated and, if it is necessary, a transformation (Rotation, Translation or Scale) will be applied to the model displayed in the screen.

All the process is shown in Fig. 6. As could be seen, the system is divided into separate modules which will help if any part has to be changed. Also, it allows to create new applications modifying the 3D Engine part in order to, for example, control a robot, a computer or even an automated house.

5.1.2 Communication protocol

A new communication protocol specifically designed for this application has been implemented accordingly with the requirements of the communication, in essence, the device name, the time-stamp and the nine measurement values. In order to simplify the protocol and since both systems are running Java, I decided to send every field as a String. The packet structure for the samples is defined as follows:

Device	TimeStamp	Sensor Data			ENDL
		ACC	GYRO	MAG	
”B4CC”	”12:06:34”	”9.8,0.1,0.5,[...],0.8,1.2”			”\n”

Table 1: Example of packet sent by the Android application.

The communication protocol uses simple commands (as START or ERR) and acknowledgement packets in order to notify the command have been received without errors. Notice the received samples are not notified because it will generate too much traffic and will make the whole system slower.

1. The client sends the “START” command (which is an UTF String with value “START”).
2. The server answers with an “OK” or “ERR” + error-code (“OK” and “ERR” are also UTF Strings and error-code is an integer).
3. If the server has sent an “OK”, the client is allowed to begin streaming. Every data packet has the format specified in Table 1.

Notice the data has csv format (has comma separators and ends with an EOL (End Of Line) value).

4. To stop the communication the client or the server has only to close the socket.

5.2 Application design

5.2.1 Smartphone application

The Android Application work-flow is shown in Fig. 7

As could be seen, *MainActivity* is the AndroidActivity launched when the application starts. Then, from this screen, the user can go to the *Preferences* activity or to one of the connection options (transfer data to pc or save data into external storage) controlled by *ConnectionActivity*. When Streaming option is selected, two connections will be established, one with the computer and another with the sensor. In the case the user wants to store data into the phone, the communication will be established with the sensor and the received data will be stored on the external storage of the phone.

The selected Android SDK will be version 2.3.2 Gingerbread API level 9 since will be compatible with the most Android Devices on the market.

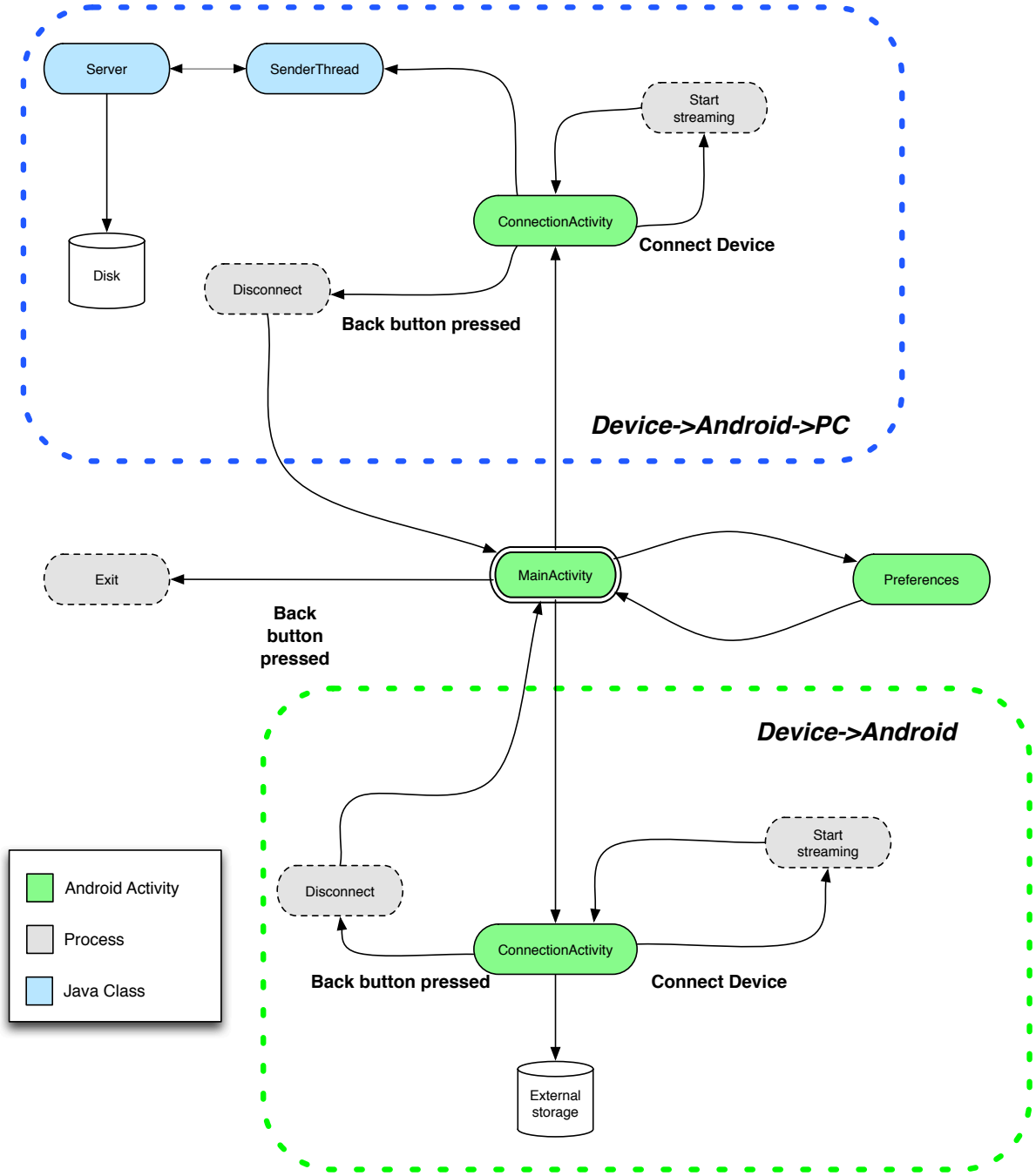

Figure 7: Android Application work-flow

5.2.2 Server

In Fig. 8 the Class diagram of the Server is explained. As can be seen, a main thread controls the Android-Server connection in a higher level and another thread manages the data streaming and its processing and parsing. Also this second server is the one that sends data to the 3D Engine.

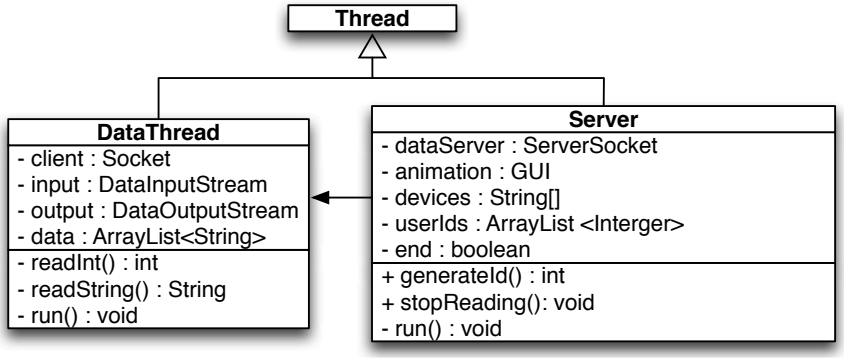

Figure 8: Class diagram of the Server

5.2.3 User interface

In the Fig. 9, the 3D Engine class diagram is shown. As can be seen, the main class is *GUI* where the main loop is defined as *simpleUpdate*. The data received is stored in a *Data* object and put in the buffer. In order to display some properties the user can choose, a *JFrame* have been designed and is launched before the application starts. Finally, once an action is detected, it is stored in an *Action* object in order to keep its type, value and axis in an easy-to-access place.

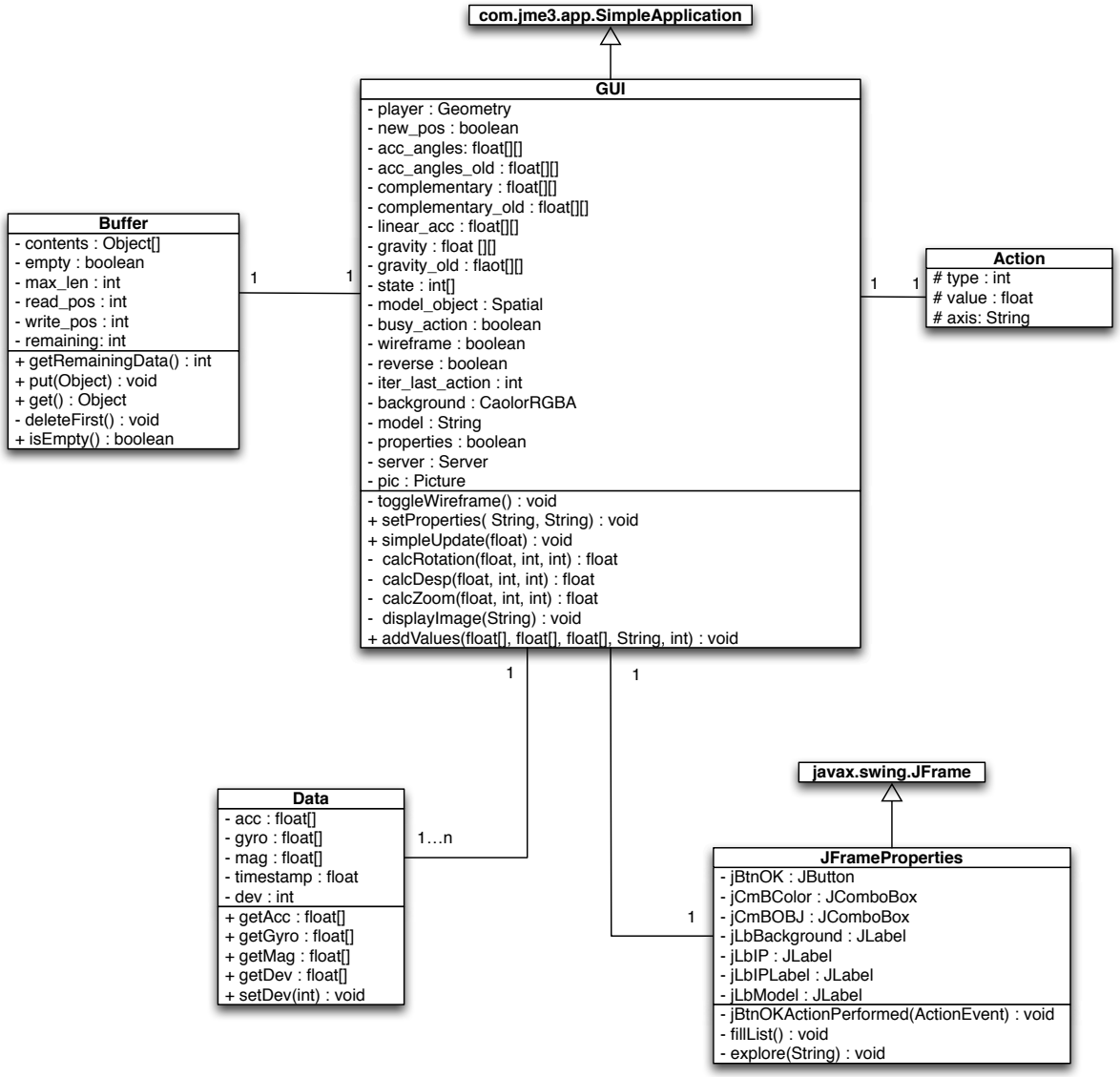

Figure 9: Class diagram of the 3D Engine

5.3 Interaction modes design

Although, the application was proposed with a single interaction method, I decided to implement two additional methods in order to compare them.

In this section, the design of these three interaction modes will be explained.

The first interaction mode is based on sensor pose and gesture detection. As could be seen in Fig. 10, there have been defined five different orientations, one for each mode:

(a) *Rotation mode*. Both sensors have the Earth gravity vector pointing downwards the y sensor axis.

(b) *Pan mode*. Left hand sensor will have the gravity vector pointing downwards the z sensor axis and right one will have this vector pointing downwards the y axis.

(c) *Zoom mode*. Left hand sensor will have the gravity vector pointing downwards the y sensor axis and right one will have this vector pointing downwards the z axis.

(d) *Centring mode*. Both sensors have the Earth gravity vector pointing downwards the z sensor axis.

(e) *Toggle wire-frame mode*. Left hand sensor will have the gravity vector pointing downwards the y sensor axis and right one will have this vector pointing upwards the z axis.

When the application is in a specific state, the user has to move his hands in order to perform the action.

Rotation is only available for two axes, x and y because are enough for the visualization of the model. In order to rotate in the x-axis, the user have to move the left sensor along this axis. In the same way, a movement in the other sensor along the y-axis will result in a rotation in this coordinate.

The *pan* is defined as the movement of the object among the x-y plane. The movements are defined the same way as the rotation ones. The object will change its x coordinates when the left sensor is moved along this axis and a y movement will be produced by moving the right sensor the same way.

Zoom moves the object along the z axis in order to increase the detail or have a wider view of the model. This is achieved by moving the right sensor in the y axis, forward to increase the zoom and backward in order to decrease it.

Toggle Wire-frame is a switch for this option and is toggled by moving the left sensor in any direction of the y axis.

In the special case of *centering*, there is no movement since the action is performed when the state is reached.

For more detail on the gestures definition, see 6.4.

The second interaction mode also uses pose and gesture detection but the gesture will be split in order to use it as a start/stop flag. That is, with the first half of the gesture the transformation will be started and will be stopped when the hand returns to the starting position. The modes defined for this interaction method (Shown in Fig. 10) are:

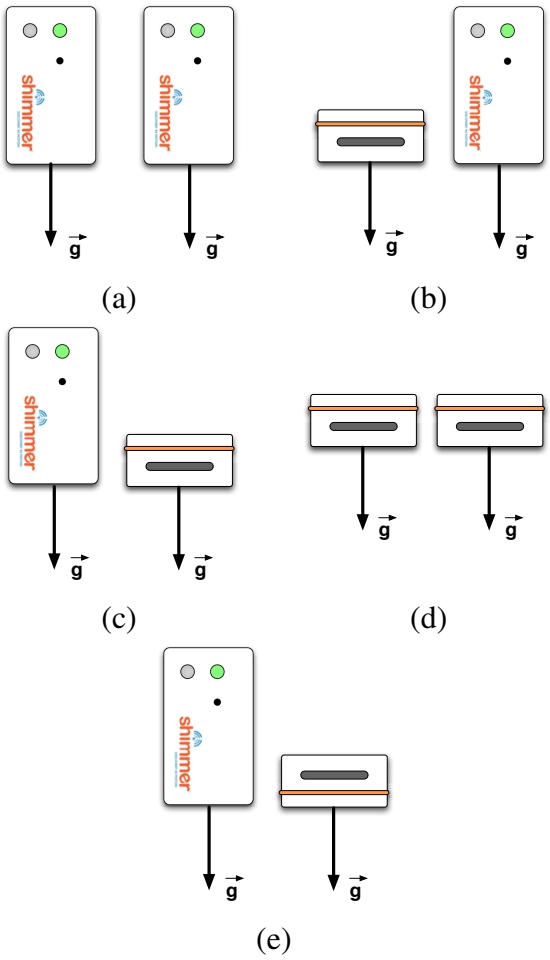

Figure 10: Sensor orientation modes.

(a) *Rotation mode*. Both sensors have the Earth gravity vector pointing downwards the y sensor axis.

(b) *Pan mode*. Left hand sensor will have the gravity vector pointing downwards the z sensor axis and right one will have this vector pointing downwards the y axis.

(d) *Centring mode*. Both sensors have the Earth gravity vector pointing downwards the z sensor axis.

Notice that Zoom and Toggle Wire-frame modes have been disregarded. The main reason for not defining the zoom action is the need for the scale transformation of being performed in small increments. For the Toggle Wire-frame action, defining start and stop flags makes no sense. The movements for this interaction method are the same as the ones defined for the first method.

Finally, The third interaction method uses the Shimmer device as a representation of the 3D model, so the rotations performed in the device will also be visualized in the virtual object. For this mode, only rotation has been defined since pan and scale require trajectory reconstruction which has not been implemented.

5.4 Data processing and algorithmic design

5.4.1 Preprocessing

All the samples (originally in millivolts) are translated into calibrated values with specific units such as m/s^2 rad/s and G by using a previously defined calibration matrix with the help of the Shimmer 9DOF Calibration Software (see 7 section).

The Android application, also modifies the packet the sensors had sent by adding a time-stamp in order to label every sample and maintain the reception order.

5.4.2 Fusion techniques

In order to obtain the orientation of the sensor and its linear acceleration, sensor fusion is performed. The main reason of using this method are the disadvantages of each sensor and their complementarity. First of all, accelerometers are known of being noisy but its measures have no big drift, on the other hand, gyroscope has important drift and varies in time so it is very difficult to eliminate. Notice magnetometer will not be used since there was no notable improvements with its inclusion in the complementary filter. With the goal of obtaining sensor orientation we need its rotation angles and for this purpose there are several options, shown in Table 2.

Description	Advantages	Disadvantages
Compute the angle directly from accelerometer measurements.	It is easy to implement and also intuitive.	Noisy measurements
Apply a low-pass filter to accelerometer values in order to reduce noise.	Easy to implement and will be effective.	There will be lag in angle changes.
Integrate gyroscope values to simply acquire the angles of rotation (gyroscope units are deg/s).	Easy to code and fast so there will be no lag on angle changes.	Gyroscope always have bias drift, that is the sensor does not measure zero when stationary.
Use a Kalman Filter.	Takes into account physical properties of the sensor. Is a theoretically-ideal filer and give clean results.	Really difficult to understand and code. Makes real-time processing difficult.

Use a Complementary Filter.	Fixes accelerometer noise and gyroscope drift, has no lag and is possible to use in real-time applications.	Theory has to be understood in order to code the filter.

Table 2: Sensor fusion solutions extracted from [10]

After reviewing all the filters, I have decided to implement a Complementary Filter because achieves a trade-off between effectiveness and efficiency.

How it works

Basically the application needs to obtain the orientation of the sensor. For this reason we are going to use the best properties of each sensor in order to get clean results in the least time.

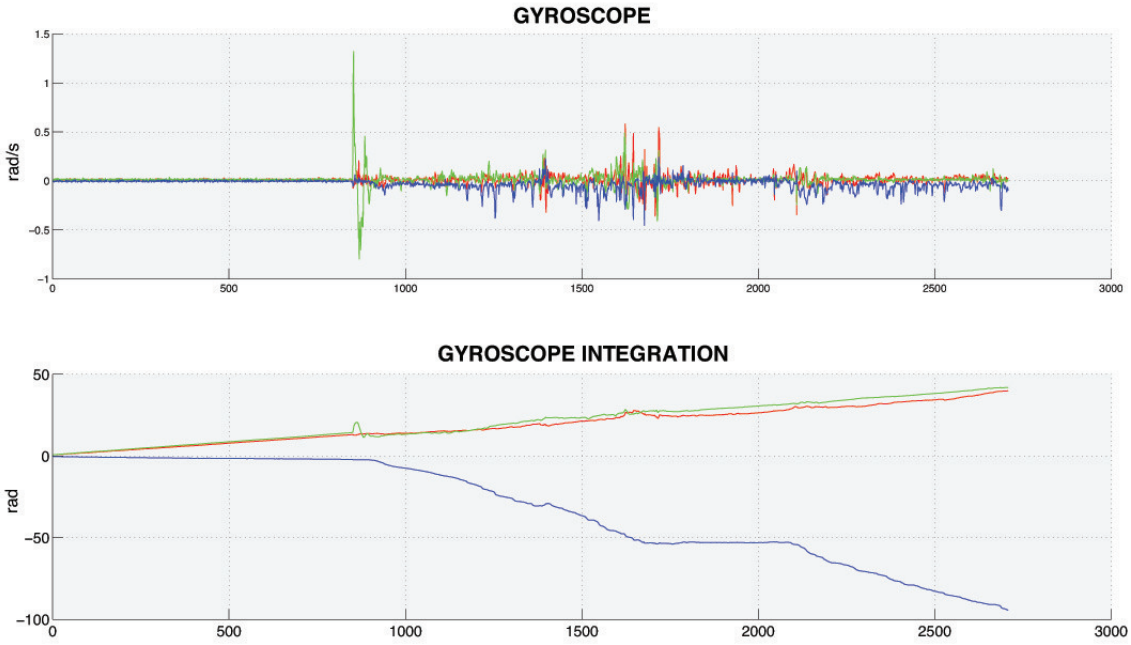

Figure 11: Gyroscope integration example.

First of all, the inputs given are: Accelerometer values which units are m/s^2 and Gyroscope ones measured in deg/s. Gyroscope values are converted into radians and afterwards are integrated in order to get the rotation of the sensor:

$$\theta_t = \int_{t_0}^{t} \omega \, \mathrm{d}t$$

Where θ_t is the rotation at t time , ω is the angular velocity and t is time.

Once we have the absolute rotation, if we plot it, we will obtain a big slope, almost exponential (if the time is big enough) as could be seen in Fig. 11.

If we have a look at the accelerometer input, first of all values need to be converted to radians in order to combine the same units in the following parts of the filter. These are the equations for obtaining the rotation angle of each axis:

$$\alpha = cos^{-1}\frac{x}{|\vec{a}|}$$
$$\beta = cos^{-1}\frac{y}{|\vec{a}|}$$
$$\gamma = cos^{-1}\frac{z}{|\vec{a}|}$$

Where x, y, and z are accelerometer measures in each axis and $\vec{a}$ is the acceleration vector: $\vec{a} = (x, y, z)$. Also notice α, β and γ are the angles between x, y, and z components and $\vec{a}$ respectively.

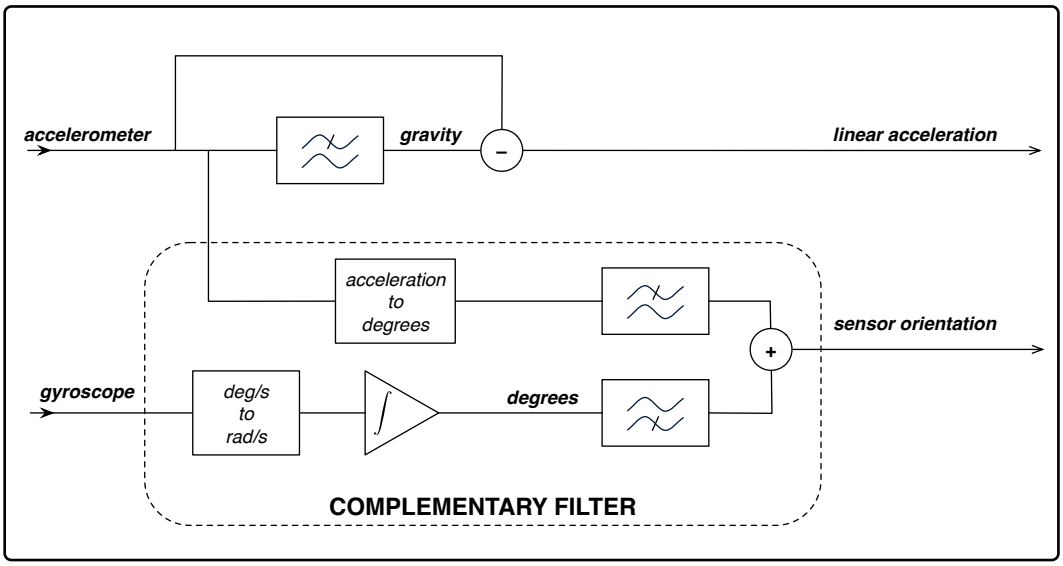

Figure 12: Filter used in the application.

The next task is eliminate gyroscope drift, which is a low-frequency component and accelerometer noise which is a high-frequency component, that is the essence of the Complementary filter, merge both sensors and eliminate the components we do not want such as drift or noise. As could be seen in Fig. 12, the gyroscope values go through a high-pass filter that will delete the bias drift because of its low frequency; in the other hand, in the accelerometer measurements, noise has to be reduced and a low-pass filter is used. That is because noise changes every sample and accelerometer measurements are quite stable.

Once we have both sensor filtered, we have low frequencies of the accelerometer, i.e. changes will take more time to appear, and the higher ones of the gyroscope that means the response will be faster. If we sum these values, we will obtain a precise orientation of the sensor that will have

a higher or lower response depending on the value given to σ:

$$C_{x_i} = \sigma * \theta_{x_i} + (1 - \sigma) * \alpha_i$$
$$C_{y_i} = \sigma * \theta_{y_i} + (1 - \sigma) * \beta_i$$
$$C_{z_i} = \sigma * \theta_{z_i} + (1 - \sigma) * \gamma_i$$

At that point we have the orientation of the sensor so it is really easy to obtain its position and, in consequence, its state. Although we know an important information about the sensor it is impossible with that information to understand towards where it is moving an which is the strength of this movement. For that reason, linear acceleration has to be obtained somehow. There is another high-frequency component of the acceleration I have not explained yet: linear acceleration. In fact, accelerometer values are composed by noise, linear acceleration and gravity measurements and all we want is removing this third component and, of course, the noise. The best option is using a low pass filter to obtain the gravity and subtract it from accelerometer values.

$$gravity_i = \lambda * gravity_{i-1} + (1 - \lambda) * acc_i$$
$$linear\_acc_i = acc_i - gravity_i$$

Where acc_i is the acceleration vector in m/s^2 and λ is a constant with value 0.95.

In fact we will have noisy linear acceleration, but analysing the charts generated by these values, it is easy to notice that noise is even higher frequency than linear acceleration and its values are really low (about ten times lower). This part of the filter could be seen in the upper side of the block diagram of the filter shown in Fig. 12.

If is needed, applying a median filter with a window of five samples will produce acceptable linear acceleration values, in the special case of this application that is not required because does not matter the intensity of the movement of the user, only the direction he is moving his hand. A threshold will be used in order to detect the movements.

Here you can see the MATLAB code of this filter (let M be the value matrix, each sample in a new row and columns distributed as accelerometer, gyroscope and magnetometer):

```
1
2  %% transform accelerometer into rotation
3  acc = zeros(size(M,1), 3);
4  for i=1:size(M,1)
5      x = M(i,1);
6      y = M(i,2);
7      z = M(i,3);
8      a = sqrt(x^2 + y^2 + z^2);
9      acc(i, 1) = acos(x/a);
10     acc(i, 2) = acos(y/a);
11     acc(i, 3) = acos(z/a);
12
13 end
14
```

```
15
16 %% gyroscope deg/s -> rad/s
17
18 M(:,4) = M(:,4).* (pi / 180);
19 M(:,5) = M(:,5).* (pi / 180);
20 M(:,6) = M(:,6).* (pi / 180);
21
22 %% apply complementary filter (integration is performed jointly)
23
24 alpha = 0.08;
25 complementary = zeros(size(M(:,1:3)));
26 for i=2:size(M, 1)
27     complementary(i,1) = (1-alpha) * (complementary(i-1,1) + M(i,4)*dt) +
           alpha * acc(i,1);
28     complementary(i,2) = (1-alpha) * (complementary(i-1,2) + M(i,5)*dt) +
           alpha * acc(i,2);
29     complementary(i,3) = (1-alpha) * (complementary(i-1,3) + M(i,6)*dt) +
           alpha * acc(i,3);
30 end
31
32
33 %% obtain linear acceleration from accelerometer
34
35 alpha = 0.95;
36 gravity = zeros(size(M(:,1:3)));
37 linear_acc = zeros(size(M(:,1:3)));
38
39 for i = 2:size(M,1)
40    gravity(i,1) = alpha*gravity(i-1,1) + (1-alpha) * M(i,1);
41    gravity(i,2) = alpha*gravity(i-1,2) + (1-alpha) * M(i,2);
42    gravity(i,3) = alpha*gravity(i-1,3) + (1-alpha) * M(i,3);
43
44    linear_acc(i,1) = M(i,1) - gravity(i,1);
45    linear_acc(i,2) = M(i,2) - gravity(i,2);
46    linear_acc(i,3) = M(i,3) - gravity(i,3);
47 end
```

Source code 1: Example filter code

5.4.3 Gesture detection and Angle computation

In the first and the second interaction methods, poses and gestures have to be detected in order to perform the desired transformations to the model. The pose detection is made by using the orientation vector given by the sensor fusion algorithm, where the rotation on each axis is used to define a state. Once we have the state, if the linear acceleration, in the axis defined on 5.3, is greater than a fixed threshold, the gesture is being performed, so it will be stored (with the axis, the type of gesture and the direction) in order to continue the transformation in the following frames until the end is reached. This end will be prefixed in the first interaction method and detected by the same gesture in the opposite direction in the second interaction method.

In the third interaction method, the rotation angles (pitch and roll) should be obtained using the gravity vector.

$$pitch = arc\,tan\frac{x}{\sqrt{y^2 + z^2}}$$

$$roll = arc\,tan\frac{y}{\sqrt{x^2 + z^2}}$$

In essence, the pitch rotation will be obtained by using the y and z gravity variation when a rotation is performed around the x axis. For the roll, the proceeding is the same, but changing the axis.

6 Development

6.1 Sensor Firmware

Shimmer Research provide some working firmwares. I come across with Shimmer Boilerplate Firmware, which allows to select the set-up of the device, the parameters I have used are:

- 50Hz sampling rate.
- 6G of accelerometer resolution.
- Include accelerometer, gyroscope and magnetometer data.

The packet format is defined by the developer of the sensor (Fig. 13) but since an Android library have been used, the sensor-phone communication have become transparent.

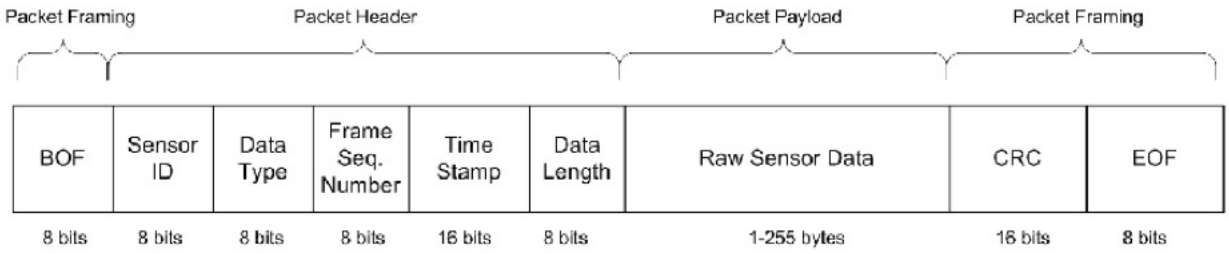

Figure 13: Packet format extracted from Shimmer Research documentation.

In order to achieve lately a precise orientation of the sensor, there is the need of establishing a calibration matrix in the sensor memory to perform the transformation at the lowest level increasing the speed of the computation and making the system more efficient.
For this purpose, Shimmer have developed an application named *Shimmer 9DOF Calibration Tool* which runs on a Windows PC and allows to obtain the calibration values and store them into the device. This tutorial have been provided by the support team of the developer: Tutorial video [3].

6.2 Android Application

An Android application have been developed in order to manage the connection to the sensors and the data retransmission to the server.

This application includes:

- Data logging.
- Retransmission from the sensor to the server.

[3] http://www.youtube.com/watch?v=pgxc8VJOQjM

- The option of deciding whether to use calibrated data, the values will be in m/s^2 for the accelerometer, rad/s for the gyroscope and μT for the magnetometer, or raw data which units will be mV.

Shimmer Android Library have been used in order to simplify the connection between the phone and the sensors. This library manages the connection and the set-up of the sensor parameters, parses the data obtained and allows to control the state of the communication.

Here there are some images illustrating the application usage:

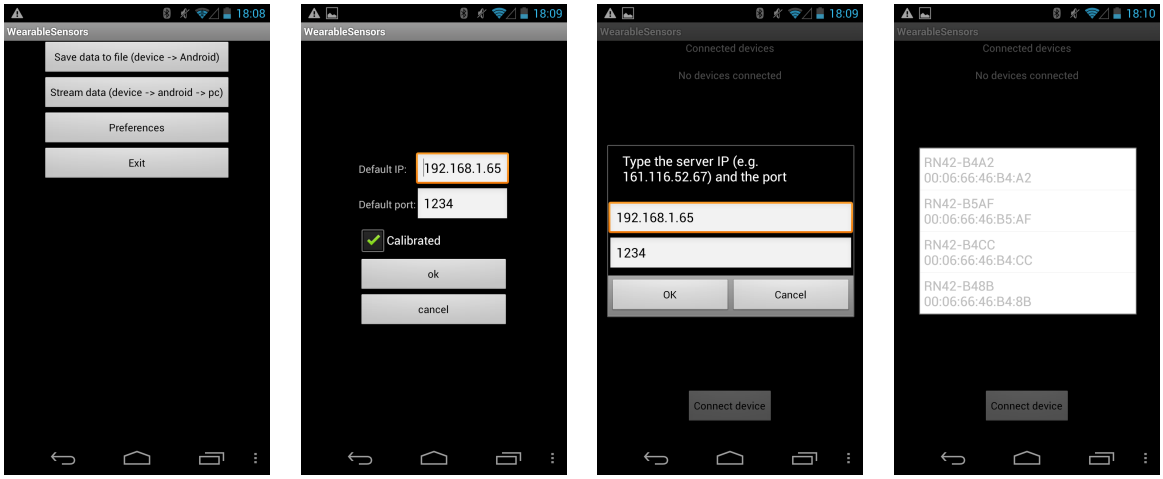

Figure 14: Left to right: Main window, Preferences, Server Connection pop-up, Sensor connection.

As it have been said, a new protocol had to be defined since the Android application was designed to include a time stamp in every packet the sensor sends, that is the way of knowing which package arrived the first since Shimmer device sends a time-stamp according to its processor clock.

6.3 Server

The Server have been designed using Java Sockets since the communication will be between two Java applications.

All the readings are defined as non-blocking by defining a maximum wait time for the read so the server or the client will be waiting to read a value not more than 2 seconds, for example, and if by then there is no value to read, an exception will be thrown. This allows to define a client/server connection without dead-locks.

Since the application has been defined for two sensors, the Server will be expecting packets from two different named devices. If there are more than two sensors, the application will fail since the initial conditions are not accomplished.

In order to store the received values a Circular Buffer for generic Java Object have been defined. The data will be produced by the Server (seen from the perspective of the buffer) and consumed

by the engine, this temporal information storing method is the best option because of the possibility of allocating a fixed amount of memory and use this space efficiently only with new values. Also has the problem of fixing an small array to store the values and fall into a buffer overflow. For this reason is really important to establish the correct value for not having buffer overflow and not allocate a high amount of memory and reduce the program efficiency.

6.4 Gesture recognition

Once the information have arrived to the Server and is stored in a Buffer, it is the time to process it and obtain useful values to calculate the sensor orientation and its accelerations in order to classify the gesture and apply the transformations to the displayed model.

The 3D viewer will be controlled by a simple state-machine defined in Fig. 15. This finite state machine defines five different modes and an additional NULL state for all which could not be classified as one of these. From every state there could be reached any other. The transitions are defined as the movements of the sensors but are too complicated to be described in the figure so will be detailed lately.

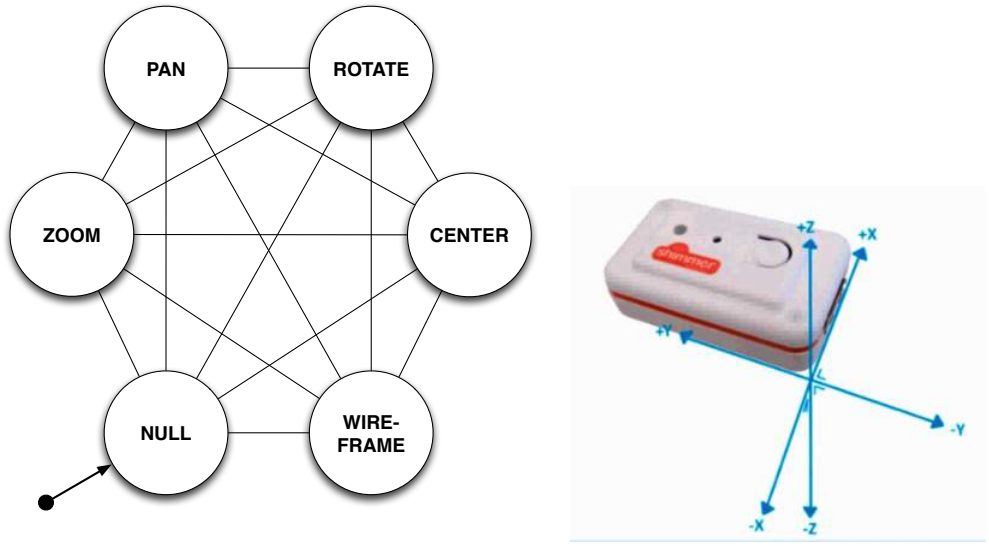

Figure 15: On the left, finite state machine for application mode controlling. On the right, sensor axes.

Once a movement is performed and the object have been moved, there is an small period of time (about a second) where the engine will not be recognizing the gestures allowing the user to move the hand back to the starting position. If this period of time was not established, the opposite action will also be recognized and the object would return to the starting position.

6.5 3D Engine

In order to visualize the gestures the user is performing, a 3D engine based on JMonkeyEngine 3 RC2 is used.

When a new sample is stored in the buffer, the GUI class is notified so the main loop (the one which updates the scene) can perform the required operations using the modified complementary filter explained in the previous section.

Once the linear acceleration and orientation values are obtained, the device state can be recognized by the orientation vector formed by the yaw, pitch and roll rotations (see Fig. 16).

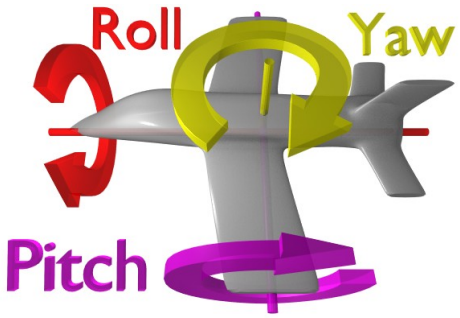

Figure 16: Yaw, Pitch and Roll rotations

The state is defined as the axis pointing to the $\overrightarrow{g}$ vector (pointing through the Earth), so there are six possible states and two devices, but we are only interested on five of all the possible combinations; the five that will define the possible actions (i.e. Pan, Zoom, Rotate, Center and Toggle wireframe). With the objective of detecting the action, the states are filtered.

If, for example, the rotation state is detected, the system will check the linear accelerations in the required axis and will rotate the model in case it is necessary.

All this computations are enqueued as a task so the engine can manage them and increase the performance of the whole system. Additionally, the frame rate is not fixed so the engine can manage itself the value according to the processing requirements of every single moment.

Since there will be two sensors connected and streaming data through the phone, the engine has to distinguish between them with the purpose of not mixing the readings of both devices. For this reason, the server will store the values adding an identifier (0 or 1) to the value which will be related with a device, then, when the engine gets this value, it will be easy to know from which sensor comes this sample. The performance will be increased if the values from both sensors are processed in the same task because the buffer will have less memory in use, but this also has problems. For example, if one sensor sends less samples per second than the other (notice the established sample rate is an approximate value), the readings will be unbalanced and the samples of one of the sensors will be processed before than the other so the system will not work as is required.

In order to help the user in the visualization tasks, a icon per action have been established in order to recognize which is the actual state.

All the models included with the application have been found under a CreativeCommons license and one of them, a 3D textured Heart have been modified in order to add some deficiency and use it in the example shown in the *Results and Validation* section. A GUI designed with Java Swing is used to help the user to choose the available models and the background solid color. Also the server IP is shown with the objective of helping the user to connect the Android device.

Figure 17: From left to right, Pan, Zoom, Rotate, Center and Toggle Wireframe icons used in the 3D visualization display.

Only the model used in the real-case scenario is textured but the others will have a texture based on its structure with shading so the 3D perception is maintained. The *toggle wire-frame* action allows to switch between the texture (a solid object) and its wire-frame, which will allow to see the structure of the object and also its interior (in case it has).

Finally, the communication is designed to avoid deadlocks and to be closed from any of its to sides. If the user presses the "ESC" key in the 3D engine, the application will stop the server which will send the "END" command to the phone and the communication will be closed. Also the phone will close the connection with the sensors. In the other hand, if the communication is closed from the phone (by pressing the "back" key), the server will be notified and will do the same with the GUI. The connection with the sensors will also be closed.

6.5.1 Interaction modes

The first interaction mode implemented allows the user to move, rotate or scale the model by fixed increments. The fixed values are low to increase the precision. This implies that the quantity of movements in order to, for example, rotate the model 180 degrees, is high.

The second interaction mode uses the beginning of the gesture as a *start* signal and the end of it as a *stop* signal. That implies the model will be moving, rotating or scaling until the hand reaches its initial position. This will decrease the precision of the system.

Finally, the third interaction mode developed in this project is focused on a more manipulative interaction, using the sensor as a real-life representation of the object. This helps the user to understand the rotations. Only rotations are implemented because of the lack of trajectory reconstruction which will be necessary to introduce 3D translations.

7 Set-up

The general procedure in order to run the whole application will be detailed. Notice the order of the set-up is not the same as the pipeline.

7.1 Shimmer® Sensor

First of all we will suppose the sensor is fully charged and has BoilerPlate Firmware running on it. Also the Sensor has to be calibrated using *Shimmer 9DOF Calibration Tool*. In the special case of this application, it is recommended to tie the sensor to the arms in order to achieve better results.

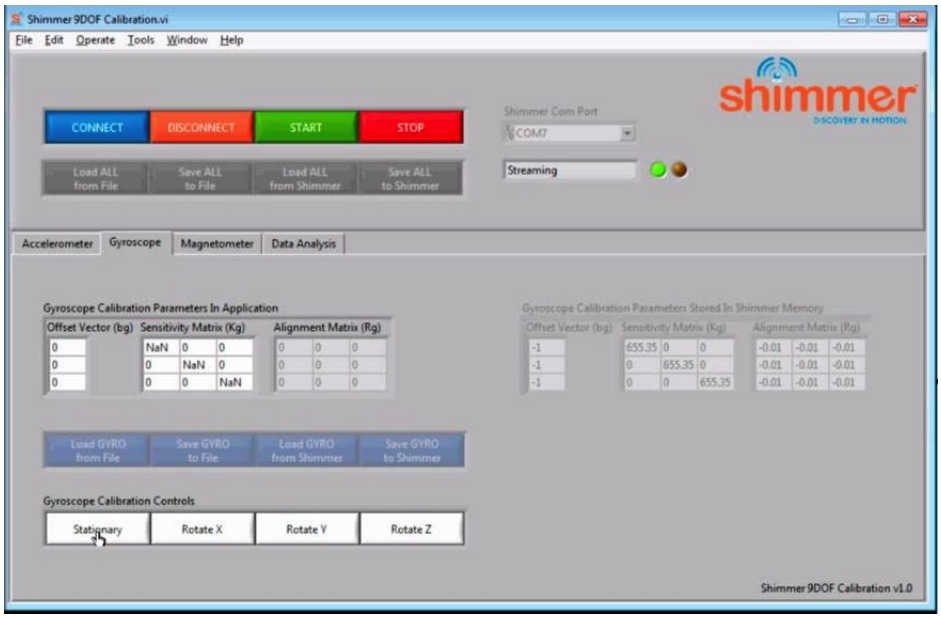

Figure 18: Shimmer 9DOF Calibration Tool

7.2 Server

Before connecting the Android application, we need to run the Server. That is because the Android Application is only a "bridge" between sensor and server.

In order to run the server, a LAN connection is required and also have the proper rights for listen to a specific port, in that case *1234*.

Once the Server is running, a "Waiting for incoming connections..." message will be displayed.

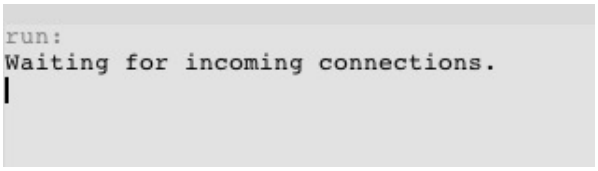

Figure 19: Server output message

7.3 3D Engine

The best moment of running the engine is after running the server and before the Android application because of the Circular Buffer. This will help to reduce the possibility of buffer overflow because the sensors are not connected yet. If we want to allocate the minimum needed space for the incoming values we need to consume them fast. If the Server is receiving data before the Engine is running, all these values will be stored on the Buffer and could cause an overflow. If the buffer is empty, the engine will wait for a new value but will not crash.

In order to run the 3D viewer, a small configuration window will be displayed. It will show the option of changing the background color and also the choice of the 3D model to be displayed. All the 3D models are stored on the Assets/Models path and are read automatically, so if a new model is added, the system will recognize it. Notice the server IP is displayed on the bottom of the window in case you need it to configure the Android Phone (explained in next section). When the *OK* button is pressed, the model will be loaded on the screen displaying the frame-rate. A minimum of 150 frames per second are recommended in order to have a fast response of the sensor movements (this value is automatically set by the engine).

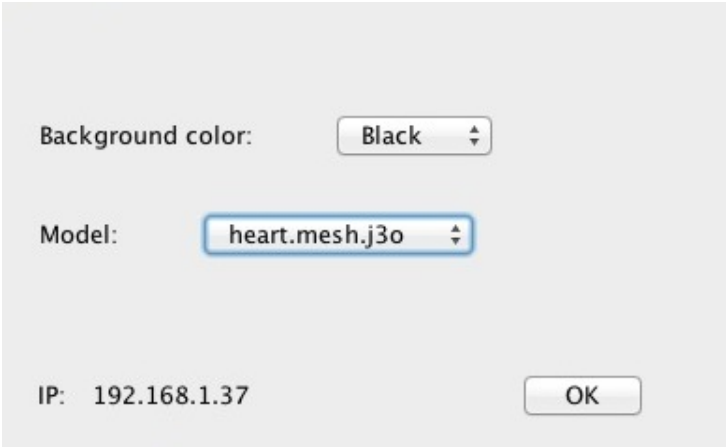

Figure 20: Configuration screen

7.4 Android Application

Once we have both sensor and server running, we could launch the *WearableSensors* application. Remember to pair the Shimmer sensor and the mobile phone and also activate the Internet connection before starting this procedure. Enabling Bluetooth is also recommended but in case it is not activated, the application will ask you to do it.

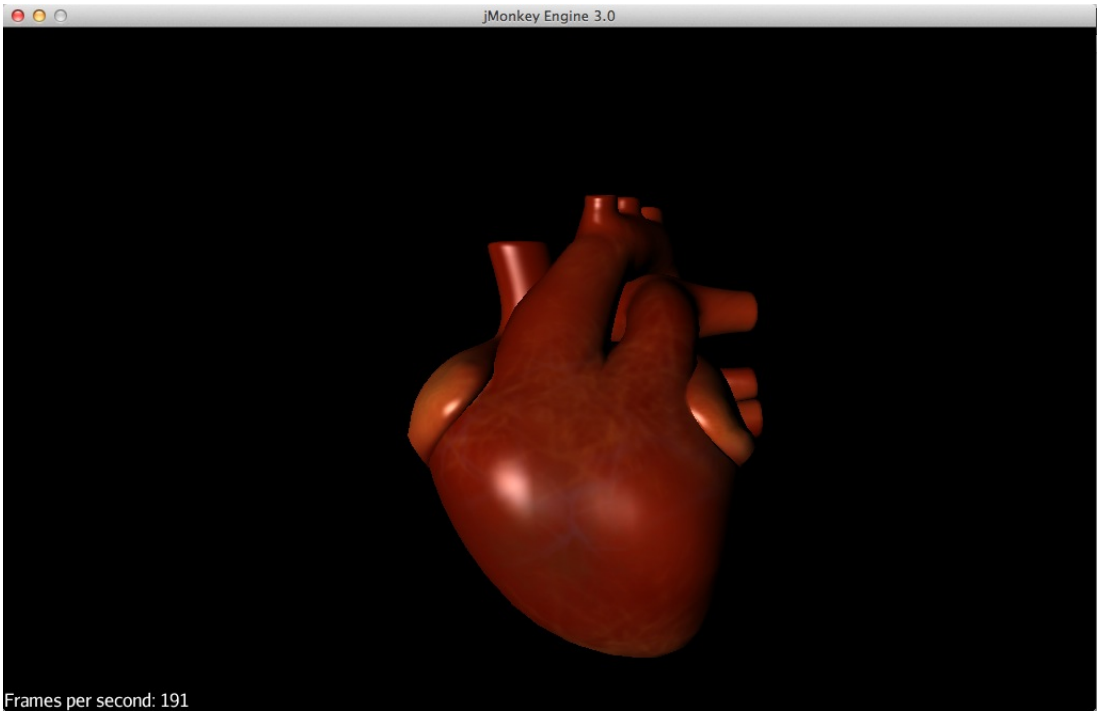

Figure 21: 3D visualizer example

When the main window is displayed, you will see four different options (Save data to file, Stream data, Preferences and Exit). We will begin by setting up the IP and the Port of our server, so you shall open Preferences. In this screen you have to write the server port and IP address. Notice the the system will check if it has the correct format (i.e. four groups of three digits with a dot between them). Also you have to set check the *calibrated* check-box and then press *OK*.

The next step is connecting to both sides (Server and Sensor). In order to do this you have to touch the *Stream Data* button. A pop-up will appear in order to change the Server IP and port (in case it is needed). If the values are correct, you could touch *OK* button and the mobile phone will connect to the server, notice a message will appear on Server display announcing a new connection have been created with the shown IP.
The next step is connecting to the Sensor. To do this touch the *Connect device* button and a list with the paired devices will appear. Select the one you want to pair (the device name and MAC address are in the back of the sensor device). Then, the phone will ask the sensor to establish a new connection and will set it up by selecting the accelerometer range (4G, i.e. the maximum value will be four times Earth gravity), the sensors we want to stream (accelerometer, gyroscope and magnetometer) and calibrated or uncalibrated values. Once the sensor has the parameters, it will begin to stream data and we will notice that by the blinking leds on the sensor, a green led notifies the connection and the orange one the data streaming. In the case the connection could not be established, reset the Shimmer device by pressing the small button hidden in a small hole in the middle of the device. All the screens described are shown on Fig. 22.

Right now we should have the application running and all the devices connected.

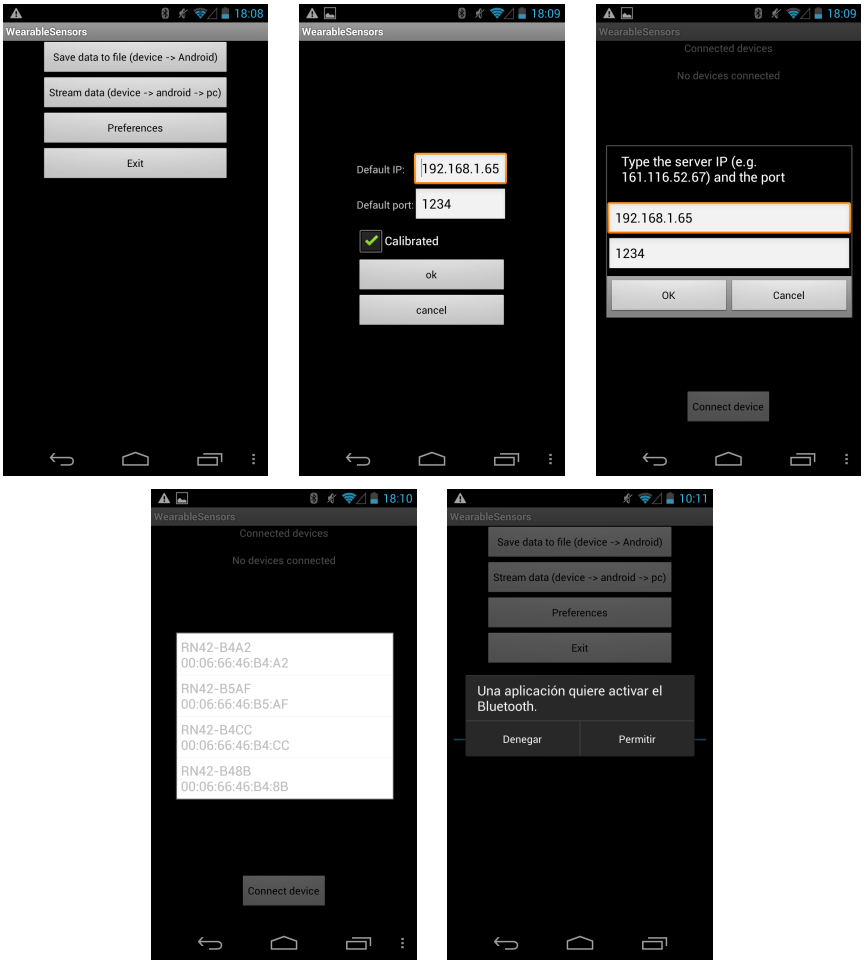

Figure 22: On first row: Main window, Preferences and Server Connection pop-up. On second one: Sensor connection and Bluetooth activation request pop-up.

8 Validation and Results

In this section, the overall results of the application will be explained as well as a real-case example of use.

In a general overview of the application, a cross-platform have been developed, with an easy-to-use user interface. Focusing on the 3D visualizer area, five gestures have been included and also five 3D models, one of them fully textured. The user has also the possibility of changing the background color. Three different interaction methods have been developed and will be discussed later.

A especial scenario have been designed for the validation of the system. This involves a Doctor interacting with the interface in order to check a heart and detect its anomalies. This case allows to validate the system in all of its features in a possible real case scenario of a medical diagnosis.

In this validation the two Shimmer® devices will be tied to the user arms, a Samsung GT-I9250 phone running Android 4.2.2 (Jelly Bean) will connect the wearable sensors and the Server, running on a MacBook Pro running Mac OS 10.8.3 on an Intel Core 2 Duo at 2.26 GHz and 8GB of DDR3 RAM memory. Both the server and the phone will be connected through a LAN (Local Area Network) connection. The 3D Engine will be running on the same computer than the server. The interaction mode used in this validation is the one that uses small increments in the transformations due to the precision needed.

It is supposed, the server is running and all the modules are connected (i.e. the sensors, the phone and the server). In the GUI, the heart model is selected using a black background.

In the first screen will appear the model on its initial position as could be seen on Fig. 23.

Then the user will move the heart to a better position. Notice the lower left corner displays an icon to help the user to identify the current movement he is performing. In the example seen on Fig. 24, the user have moved several times the model to the left and also to an upper position. Notice the illumination changes depending on the object position.

Another action the user can perform is toggling between a textured material and a wire-frame as could be seen on Fig. 25. The wires will also have the same color of the texture, that will be helpful in the cases where the texture shows the density of each point.

Now, the user will have a closer look in order to search for possible anomalies on the studied heart. The 3D model have been modified in order to include two hidden anomalies so the user will have to rotate the model several times. In the next figure, we have an example of a closer look obtained by rotating the model and performing a zoom.

After rotating the model several times, the user have found the first of the anomalies, a kind of blockage in the superior cava vein [4] as could be seen on Fig. 27 marked with a gray arrow.

If the user keeps searching, he or she will find another anomaly inside of the pulmonary arteria, also marked with a gray arrow in Fig. 28.

[4] All of the veins and arteries names have been found on http://upload.wikimedia.org/wikipedia/commons/f/f5/Wiki_Heart_Antomy_Ties_van_Brussel.jpg, an image created by Tvanbr under a Creative Commons license.

Finally, the user will have a look at the interior of the heart by using the wire-frame structure. As could be seen on Fig. 29, the inner part of the left pulmonary veins is fine and have no anomalies.

Finally, the user will restart the heart to its initial position by using the center action. The result will be the shown in Fig. 30.

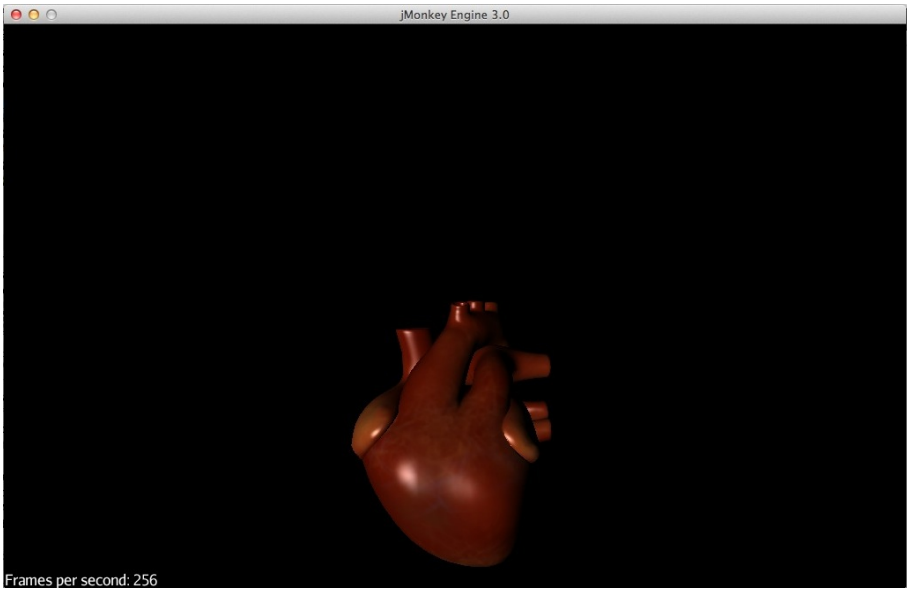

Figure 23: Initial position

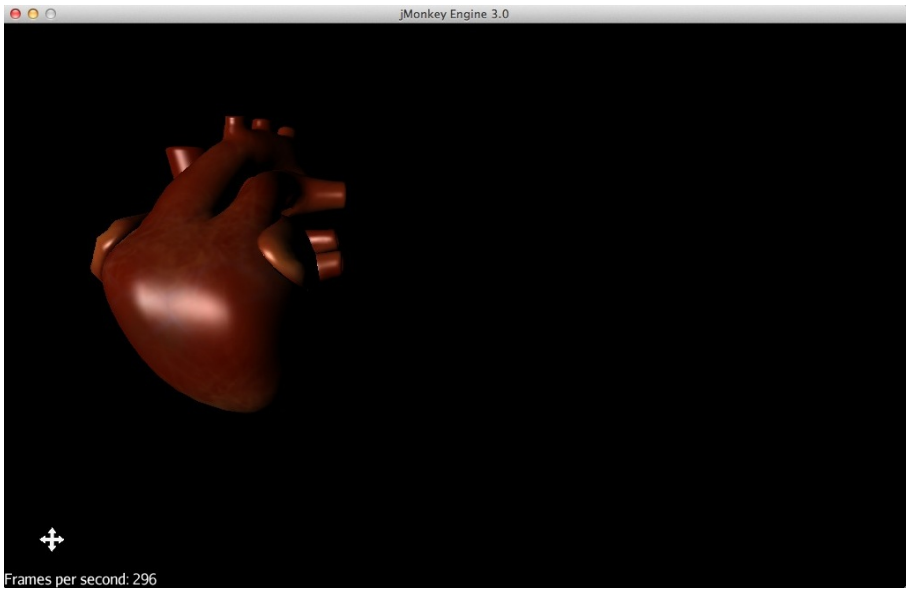

Figure 24: Moving the model

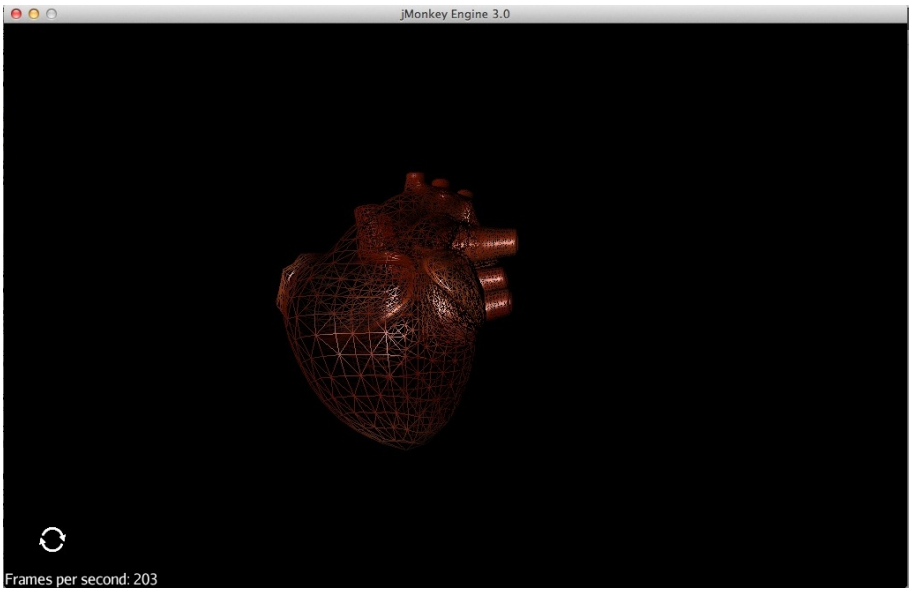

Figure 25: Toggle Wire-frame

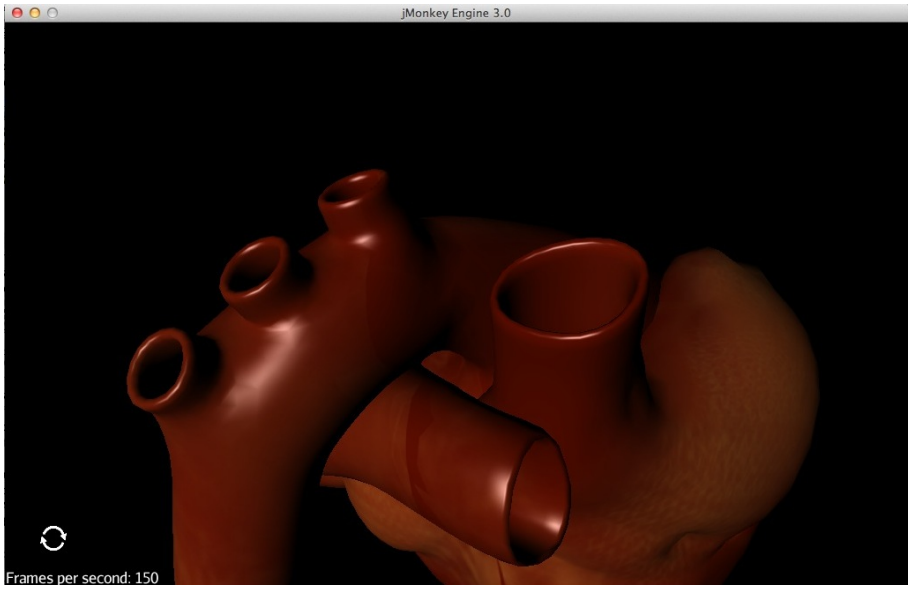

Figure 26: Rotate and Zoom

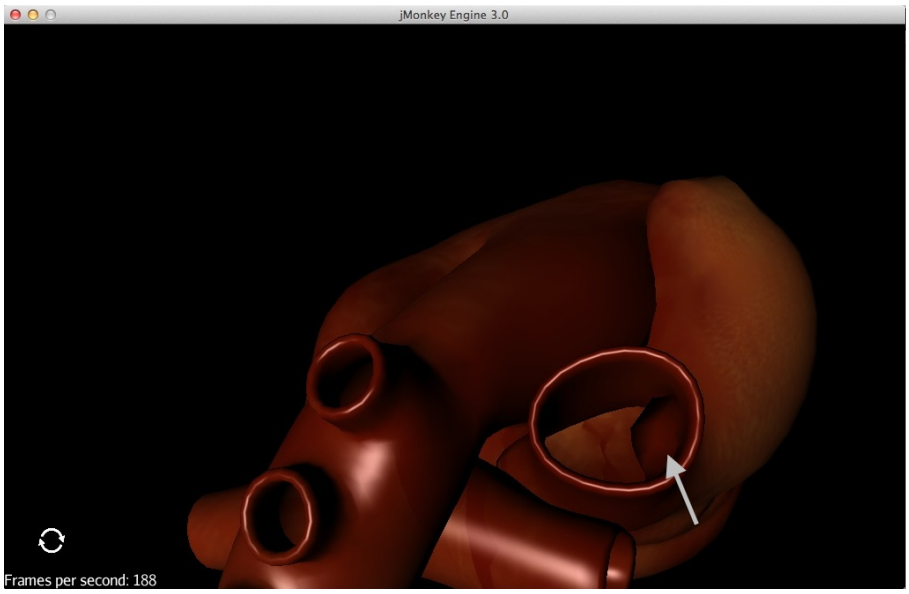

Figure 27: First anomaly found

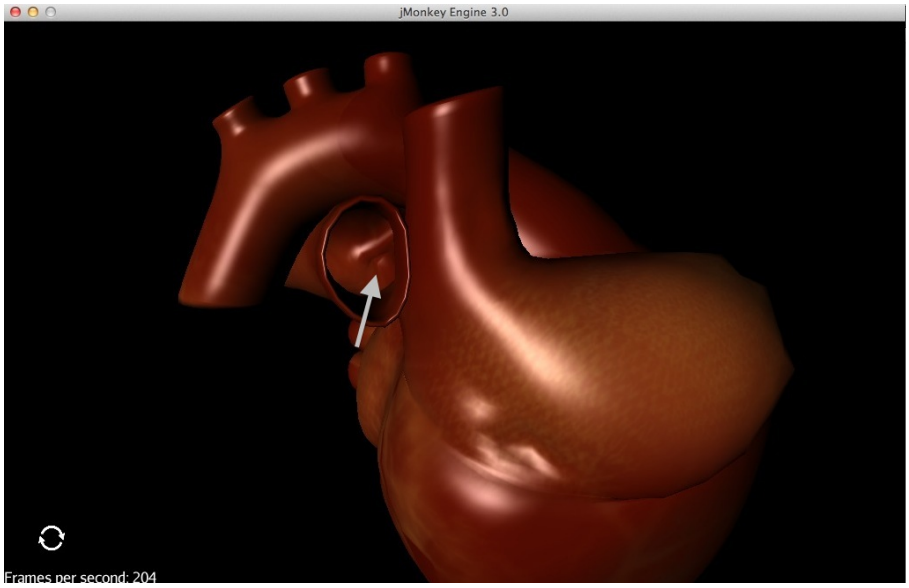

Figure 28: Secon anomaly found on the heart

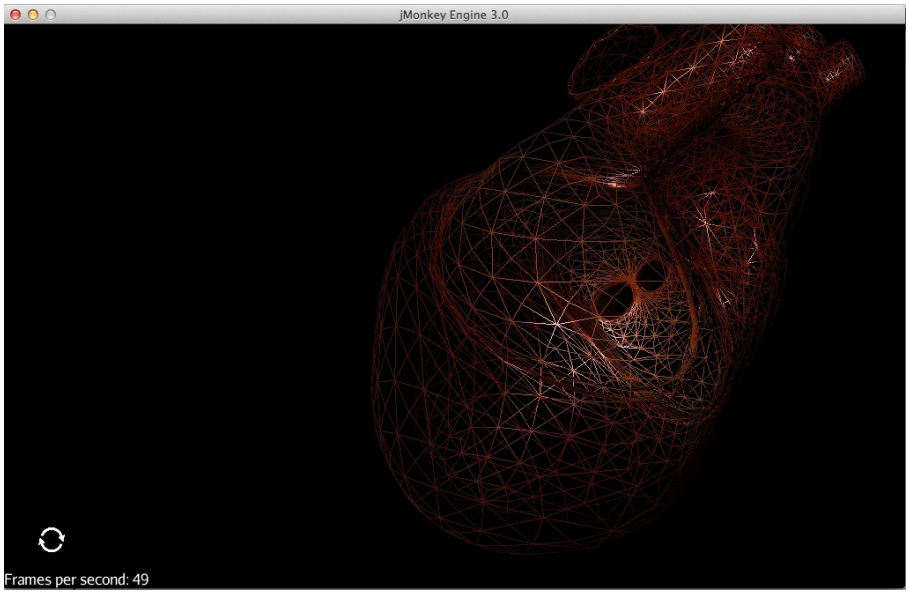

Figure 29: The wire-frame allows to visualize the internal structure of the heart

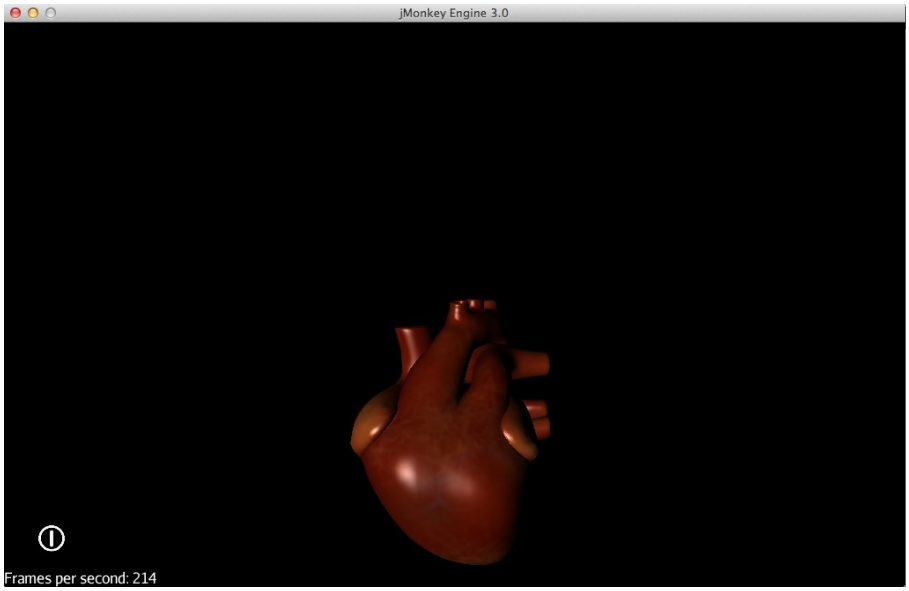

Figure 30: When the user centres the model, it will return to the initial position

Finally, as a result of the project, the three interaction methods should be explained and discussed.

The first one uses two Shimmer devices and has five different gestures defined (rotate, pan, zoom, center and toggle wire-frame). In order to perform a transformation, the user has to move his or her hands which will result on an small transformation. It is easy to notice the need of performing a lot of movements in order to make a *big* transformation (e.g. rotating the model 180 degrees), that is why this second interaction method was implemented, it uses a start/stop method, that is, when the user moves the hand, a continuous transformation begins and it is ended when the hand returns to the starting position. The included gestures are only rotate and pan since zoom becomes not usable (based on the tests performed during the implementation). Thinking in a more manipulative interaction, a third method have been developed. This uses the Shimmer device as a real representation of the 3D model so the rotations performed in the device will be also applied to the 3D model.

The comparative table (see Table 3) shows the advantages and disadvantages of each method as well as a small description.

Description	Advantages	Disadvantages
Use pose to determine the action and use gestures as small transformations to the model.	High precision, useful to accurate examinations of the 3D object.	High number of movements needed to perform big rotations and displacements.
Use pose to determine the action and use gestures as start/stop flags.	Interaction simple and allows big movements in a simple gesture. Useful for model overview.	Low precision since becomes more difficult to perform the stop gesture in the precise moment.
Use the device as a real representation of the 3D model.	Becomes more manipulative and the interaction is simplified.	Only rotation is available and is very limited since yaw is not available (magnetometer is needed).

Table 3: Comparative table of the different interaction methods

9 Conclusions

The developed system have reached most of the objectives set at the beginning of the project. The application includes a working 3D visualizer system which is controlled by using two wearable sensors with 6DOF each one. Sensor fusion, more specifically complementary filter, is used in order to obtain sensor orientation. This orientation is used to define an state in the finite-state machine that will determine the action the user is performing. In order to toggle an action, the linear acceleration is measured. Five actions have been defined: zoom, pan, rotate, toggle wire-frame and center the model. Finally, two additional interaction modes have been implemented.

As a result of the design and development process, I have obtained a simple application that works in real-time and is cross-platform (the server and the 3D engine) and modular so the conversion to other platforms of the android and the wearable sensor parts will be straightforward. Although the application have been presented as a solution for medical volumes visualization and interaction, it is not restricted to this area and could be also useful in education, engineering design, and a huge bunch of disciplines that use, or could use, 3d models.

After the implementation of three different types of interaction methods, it could be noticed that the first one, that implies small movements, have higher precision but requires much more movements in order to move the model. On the other hand, the second method is better for high displacement moves because it could be done with one single gesture, but the precision will be depleted due to the largest movements. The third one tries to establish a relation between the 3D model and the sensor device so the transformations will be more intuitive and the interaction will become more manipulative.

In the development process I have found some issues which presented an additional difficulty. First of all, when I was testing the wearable sensor and the Android phone connectivity, some problems with the Shimmer® firmware and the device calibration appeared but the support team provided me the solutions. Also Sensor Fusion was a difficult part of the system because of the complexity of representing the values obtained in order to check them and evaluate its performance.

Some of the additional objectives have not been achieved because of a lack of time. Another gesture classification method have not been implemented. Instead of that, only a finite-state machine have been developed which was simpler but worked as well. Also, not all the models have been textured, only the one used to the examples albeit it includes some anomalies in order to give the validation example more realism. Not all the features of the Shimmer® devices have been used and only a 6DOF-based sensor fusion was implemented, disregarding the magnetometer.

As future work, a classifier should be introduced in order to recognize gestures not only by the device orientation but also by its movement (for example changing the zoom action by a gesture like the widely used “pinch to zoom”). Also will be really useful the option of viewing slices of the model in order to see its section and its interior. If we think in medical imaging, adding animated models like a beating heart could be useful in the detection of some diseases but this feature will also be useful in education or game development. Additionally a more accurate Sensor Fusion algorithm should be used including the magnetometer to the computations. Finally, taking a look at the interaction modes, will be interesting performing an usability test with potential users in order to get the best features of each interaction method and fuse them in a more

usable application, also verified by the same users.

This work is a proof of concept of a more intuitive way of interacting with computer and especially with the visualization of volumes by using small and precise wearable sensors.

References

[1] Kam Lai, Janusz Konrad, , and Prakash Ishwar. A gesture-driven computer interface using kinect. *Image Analysis and Interpretation (SSIAI), IEEE Southwest Symposium*, pages 185 – 188, 2012.

[2] Michael Van den Bergh, Daniel Carton, Roderick De Nijs, Nikos Mitsou, Christian Landsiedel, Kolja Kuehnlenz, Dirk Wollherr, Luc Van Gool, and Martin Buss. Real-time 3d hand gesture interaction with a robot for understanding directions from humans. *RO-MAN, IEEE*, pages 357 – 362, 2011.

[3] Matthew Tang. Recognizing hand gestures with microsoft's kinect. Technical report, Palo Alto: Department of Electrical Engineering of Stanford University, 2011.

[4] Yoichi Sato, Makiko Saito, and Hideki Koike. Real-time input of 3d pose and gestures of a user's hand and its applications for hci. *Virtual Reality. Proceedings. IEEE*, pages 79 – 86, 2001.

[5] Horn yeu Shiaw, Robert J.K. Jacob, and Gregory R. Crane. The 3d vase museum: A new approach to context in a digital library. *Proceedings of the Joint ACM/IEEE Conference on Digital Libraries*, pages 125–134, 2004.

[6] Sebastian O.H. Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays. *Report x-io and University of Bristol (UK)*, 2010.

[7] Xiaoping Yun, Conrado Aparicio, Eric R. Bachmann, , and Robert B. McGhee. Implementation and experimental results of a quaternion-based kalman filter for human body motion tracking. *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 317–322, 2005.

[8] Najib Metni, Jean-Michel Pflimlin, Tarek Hamel, and Philippe Soures. Attitude and gyro bias estimation for a flying uav. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 295–301, 2005.

[9] Tae Suk Yoo, Sung Kyung Hong, Hyok Min Yoon, and Sungsu Park. Gain-scheduled complementary filter design for a mems based attitude and heading reference system. *Sensors*, pages 3816–3830, 2011.

[10] Shane Colton. The balance filter: a simple solution for integrating accelerometer and gyroscope measurements for a balancing platform. Technical report, Massachusetts Institute of Technology, 2011.

[11] Stefan Soutschek, Jochen Penne, Joachim Hornegger, and Johannes Kornhuber. 3-d gesture-based scene navigation in medical imaging applications using time-of-flight cam-

eras. *Computer Vision and Pattern Recognition Workshops. IEEE Computer Society Conference on. IEEE*, pages 1–6, 2008.

[12] Luigi Gallo, Alessio Pierluigi Placitelli, and Mario Ciampi. Controller-free exploration of medical image data: experiencing the kinect. *Computer-Based Medical Systems (CBMS)*, pages 1–6, 2011.

[13] Luigi Gallo, Giuseppe De Pietro, and Ivana Marra. 3d interaction with volumetric medical data: experiencing the wiimote. *Proceedings of the 1st international conference on Ambient media and systems (Ambi-Sys)*, pages 14:1–14:6, 2008.

[14] Mateus de Souza, Diego Dias Bispo Carvalho, Peter Barth, Jeferson Vieira Ramos, Eros Comunello, and Aldo von Wangenheim. Controller-free exploration of medical image data: experiencing the kinect. *SIBGRAPI - Conference on Graphics, Patterns and Images*, pages 339–345, 2010.