Universidade Nova de Lisboa

Faculdade de Ciências e Tecnologia

Departamento de Informática

Dissertação de Mestrado em Engenharia Informática

2º Semestre, 2008/2009

Interfaces Baseadas em Gestos e Movimento

Nº26215 Tarquínio Miguel Estêvão Mota

Orientador

Prof. Doutor Nuno Correia

Nº do aluno: 26215

Nome: Tarquínio Miguel Estêvão Mota

Título da dissertação:

Interfaces baseadas em gestos e movimento

Palavras-Chave:

Interacção Pessoa-Máquina
Detecção de Posição e Movimento
Sensores Infravermelhos
Wiimote
Interacção em tempo real

Keywords:

Human-Computer Interaction
Motion and Position Tracking
Infrared Sensors
Wiimote
Real-time Interaction

Resumo

Esta tese estuda novas formas de interacção pessoa-máquina, baseadas em sensores de infra-vermelhos. O objectivo foi criar uma interface que tornasse a interacção com o computador mais natural e divertida, utilizando gestos e movimentos que são usados intuitivamente no dia-a-dia.

Foi necessário o desenho e implementação de um sistema flexível e modular, que permite detectar as posições e movimentos das mãos e cabeça do utilizador. Adicionalmente, esta interface tambem permite a utilização de botões e fornece *feedback* háptico ao utilizador. Foram encontrados vários problemas durante a realização do hardware, que levaram à utilização de novas abordagens e à construcção e teste de vários protótipos

Paralelamente à construção dos protótipos do hardware, foi implementada uma biblioteca que permite detectar a posição das mãos e cabeça cabeça do utilizador, num espaço tridimensional. Esta biblioteca trata de toda a comunicação com o hardware, fornecendo funções e *callbacks* simples ao programador das aplicações.

Foram desenvolvidas quatro aplicações que permitiram testar e demonstrar as várias funcionalidades desta interface em diferentes cenários. Uma destas aplicações foi um jogo, que foi demonstrado publicamente durante o dia aberto da FCT/UNL, tendo sido experimentado e avaliado por um grande número de utilizadores.

Abstract

This thesis studies new approaches to Human-Computer interaction, based on infrared sensors. The goal was to create an interface that makes the interaction more natural and fun, using gestures and movements that are used intuitively in everyday situations.

It was necessary to design and implement a system that was flexible and modular, allowing the detection of positions and movements of the hands and head. Additionally, this interface also allows the use of buttons and supplies haptic feedback to the user. Several problems were encountered during the development of the hardware that led to the utilization of new approaches and the development of additional test prototypes.

In parallel with the construction of the hardware, a library was implemented, allowing the detection of the head and hands of the user, in a tridimensional space. This library processes all the communication with the hardware, providing the programmer of the applications with simple functions and callbacks.

Four applications were developed, allowing testing and demonstrating the different functionalities of this interface in different scenarios. One of these applications was a game that was exhibited publicly during the FCT/UNL open day, where it was tested and evaluated by a large number of users.

Index

Figure Index

Table Index

1. Introduction

The motivation for this thesis will be described over the next few pages. Some possible approaches to handle interaction in more natural ways will be described, and then the proposed solution will be presented, with a justification for its choice. It will also be presented a list of contributions this thesis brought.

1.1. Motivation

It is a well known fact that the computer processing and graphical capabilities increased substantially in the past years. The interfaces we use, however, in most cases are the ones that have been used for years. Since window-based operating systems became popular, the mouse and keyboard have been the best choice available. Today they and are still the best existing interface for many applications. If we look at the gaming console industry, the situation is the same. Console games have amazing graphics and AI (Artificial Intelligence), compared with the ones available ten years ago. There are games that can render real-like images, and enough processing power to have hundreds of in-game agents, taking individual decisions. All of this makes the games more realistic. However, the standard interfaces are still the gamepad and the joystick, as the ones in the first games [1]. Many games would be much more interesting if the character could mimic the player actions in real life.

The words “Ubiquitous Computing” are very popular today. This concept means having small computational devices seamlessly distributed in our environment, doing a wide variety of tasks. With this kind of approaches growing in popularity, it is natural that we’ll need to find new ways to interact with these devices, and in some cases motion recognition can be the solution.

This need for new forms of interaction, combined with the technology available today (cheaper and more accurate sensors, more processing power to process data), suggest the development of interfaces that use common motion patterns, like pointing or moving around.

Previous work being done in this area will be analyzed in chapter 2 (Related Work).

1.2. Possible Solutions

The goal is to use common gestures to interact with a computer. There are two main different approaches to achieve this. One option is to capture real-time images and process them to extract information. This can be done by simply detecting moving areas within the image, or using more complex approaches which involve analyzing the image and extracting extra information from it [2]. The other approach is to use data from one or more sensors that get data for a specific application.

1.2.1. Using live-image processing

The first approach (using real-time images) has the advantage of being less intrusive from the user point of view. To interact with the camera the user may not even be aware where it is. But from the computer and the programmer point of view, it has several disadvantages:

- The precision is limited, when compared to other approaches.
- The response time is also slower, when compared to other approaches.
- Uses much processor power, to extract small amounts of information.
- Usually requires a controlled environment, with specific lights, and no movement on the background.

There has been a big evolution in the image acquisition technology lately, mostly because of the digital camera industry. But while the resolutions for capturing still images are today over 10MPixels, the standard values for capturing live video are usually still 0.3Mpixels (640x480),

many times with low frame rates (15fps or less). This kind of technology can be used where precision and responsiveness are not required. For example, if the objective is to detect motion, in a specific area, this can be easily achieved. However, when an interface with high precision is needed (for example if is necessary to emulate a PC mouse using the hands), this technology may not be the best choice.

Response time is also an issue. If we acquire images at a slow speed, and they still need to be processed after that, this will make the system less responsive. If we have a real-time game, for example, where the game character responds to the player movements, this reaction should be as fast as possible. And even if the speed is not that important to the system itself, for the user it is more enjoyable to use a system where actions are more fluid.

Another problem is that it is hard to use this technology in public spaces, assuming there are moving objects/persons in the field view of the camera.

When doing real-time image processing, usually having moving elements in the background disturbs the whole process. Dealing with these issues would take even more processing power, and in many cases wouldn't even be possible. There are many projects being developed using this kind of interface, with some good results. However, they all require a controlled environment, with little interference. So, in conclusion, processing live-images can be used in certain applications. If we have a controlled environment, with little interference, where speed and precision are not a priority, this can be a good solution.

1.2.2. Using sensor data

The other approach for this problem is using one or more sensors to collect data. There are several types, using light, sound, or radio waves. However, for this objective, the one that has the

best performance is based on infrared light. It doesn't involve complicated hardware, compared to the others, it offers a good precision and range, and its signals (infrared light) are not even perceived by the human eye, so it doesn't disturb the users.

If the objective is to track moving points in real-time, at reasonable speeds, with a good precision, this approach has more advantages than disadvantages:

Advantages	Disadvantages
Faster tracking	Extra hardware needed
More accurate	Longer setup time
Low processing required	

Table 1.1 - Advantages and disadvantages of image processing

Using this approach requires the use of some extra hardware. It is necessary to have a sensor to receive the information (some kind of infrared receiver, or infrared camera), and we'll need emitters in every point to be tracked. This means that if we want to track both hands and feet of a person, four infrared emitters will have to be attached to that person. This is not a big problem, but it is still more intrusive then just filming the person. Also, it is not possible to use this type of interface in a public space, where the potential users can't have proper access to the necessary sensors.

While having these disadvantages, when it is possible to use this approach, it delivers much better results. When we use such a sensor to detect a specific feature, the data will require very little processing. Instead of getting a whole image that will have to be processed, all the data read is directly related to the purpose of the application. Less processing means faster speed, and more processor time for other tasks.

Having sensors dedicated to keep track of a specific feature also means that there will be more precision and faster speed than using data from an image, because they are built for the purpose of tracking that specific feature.

These factors make this approach a better solution if we want an interface that can be used as a pointing device, with good precision, and fast response time.

1.3. Proposed solution

Considering the study above, as the objective of this project is to deliver an interface with good precision and speed, the best solution is to use sensors. The sensor used will be the Nintendo Wii remote, also known as Wiimote. The reason behind this choice is that the Wiimote offers a good infrared camera sensor, with wireless capabilities, at an affordable price. It costs about 40€ and it comes with a 1024x768 infrared camera.

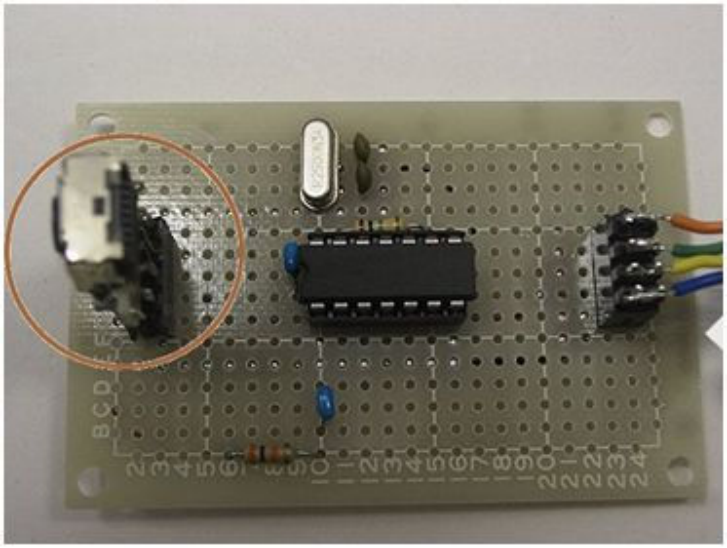

Figure 1.1 - Wiimote Camera connected to computer via I2C

The data is sent to the Wii Console (or to a computer) at 100fps. There are a few open-source projects of people that disassembled the Wiimote to extract the camera, and connected it to a computer via a I^2C(Inter-Integrated Circuit) interface, and managed to capture images at 200fps

This is an option that can be considered; however, even its default speed is already very good. 1024x760 at 100fps when compared to a good webcam, which usually captures at 640x480 at 30fps, offers a big improvement in quality.

Having 30fps would be enough to track an infrared point, but some of the requirements of this project imply to distinguish the points, and for each point to be able to send different signals. This means that the points will have to oscillate at different frequencies, to be able to encode that information. These frequencies are limited by the maximum frame rate of the camera. For a camera with a maximum of 30fps to perceive a blinking signal, the frequency would have to be less than 15Hz. At these frequencies, the user can easily perceive that the movement is not very smooth. With a capturing speed of 100fps, we should be able to encode data in every point and still have a final frequency close to 50Hz.

This controller was designed to be in the user hand, while the points emitting the infrared light were placed above or below the screen. In this project, this will be reversed. The Wiimote will be static, with the infrared camera directed to the user, which will have the infrared points on him. Some hardware was built, to make the infrared points blink at the desired moments, allowing their identification. The communication between the controllers and computer is made via a virtual COM port, either using USB or Bluetooth.

The Wiimote also uses Bluetooth, which is convenient when there are some constraints in terms of space to place the sensors. There are already some good open source libraries to deal with the communication part, like the Wii Yourself project [3].

The computer receives and decodes the data, calculating the user position, motion and button clicks. This is processed by a library that was implemented. This library processes the raw data incoming from the sensor, and tracks each individual point, which buttons are pressed, and deals with partial loss of sensor data the best way possible. It provides a useful set of information and functionality to the user:

- Automatic calibration routines
- Real-time position of each point
- Velocity and direction information, when the point is moving
- Dealing with physical events (like a button click)

This information will then be used by a higher-level library and applications to interact with the user.

1.4. Previous work

The inspiration to the work described in this project comes from some applications created by Jonny Lee [4]. He created three applications to demonstrate the possibilities of using the Wiimote as a receiver in a user interface, using very simple infrared sensors.

One of the applications tracks the user fingers, by reflecting infrared light on them. The next application used a simple pen with an infrared light on the top, allowing to use a normal projection as an Interactive Whiteboard. The final application tracks the user head in 3D space, using these coordinates to change the viewpoint of a group of targets on the screen.

These are interesting examples of building new interfaces, using few hardware items. The objective of this thesis is to create a similar system, but with extra functionality. In the second project that was mentioned, it would be useful for the user to see the mouse moving, while she

points, or to be able to click the right mouse button. In the first project, it could also be useful to be able to identify each finger. The proposed work will try to find the best way to implement these features, and develop applications to demonstrate how they can be used.

1.5. Contributions

The contributions derived from this work are:

- Creation of hardware controllers than allow to track the user hands and head, being able to identify them. Each controller can also send information via button clicks and provide haptic feedback for the user.
- Creation of a low-level software library, supplying higher level programmers simple methods and callbacks to deal with sensor information.
- Development of a suite of applications that demonstrates the versatility of the interface, using different combinations of sensors.
- Evaluation and analysis of user tests, allowing the understanding of user opinions and reactions to this type of interface.

The next part of this document includes related work, with a survey of interaction types and applications.

There is a detailed description of what was done in each phase of the project. One chapter is dedicated to the construction and testing of the hardware, followed by a chapter describing the implementation of the software. There is also a chapter dedicated to analyze the results obtained in the user tests, and then the conclusions and some ideas to explore in future work.

2. Related work

A survey of different interaction technologies will be presented next. Some are still in development, while others have been around as commercial applications for decades.

2.1. Types of interfaces and applications

The types of interface studied in this chapter usually are of one of the following types:

- Cursor interface
- Gesture or pattern recognition
- Physical position tracking

Some interfaces just focus in one of these aspects, while others can combine all three. For example, with an interface than can track the user hand in 2D space, and can recognize two different hand gestures, it is easy to emulate a two-button mouse.

To implement these types of interfaces, it is possible to use approaches based in image acquisition, or using sensor data. In the following pages some different solutions are presented using both approaches, and sometimes a combination of them.

2.1.1. Cursor type interface

These types of interfaces are widely used. Their objective is to allow the user to point to a specific part of the screen. The mouse is the best example of this type of interface. While being a good interface if we are using a computer for standard applications, it is not the best thing if we don't have a proper base, or if there is a need to be moving around. In the gaming industry, if the objective is to add realism to a game where the player needs to use a weapon, using a mouse or a gamepad interface is not the best way. It should be as close as possible with the real experience.

There have been some good proposals for this problem, and they will be explored in the next pages.

2.1.2. Gesture or pattern recognition

Several types of interfaces involving gesture or pattern recognition have been proposed in the last years. These types of interfaces usually involve image processing, that involves much computation, so it is natural that they are appearing more now, that there is enough processing power to get good results. These types of technologies usually require the computer to recognize a symbol or a shape, for example the user hand, doing a specific gesture.

An alternative to image processing is using special hardware, for example, a special glove in the hand, to detect specific finger positioning. Some projects [5-9] using this type of technology will be presented further along this chapter.

2.1.3. Physical position tracking

Another possibility for interaction is to track the user body in a 2D or 3D space. The whole body can be tracked, or just a specific part, like the head or hands. Like in the previous interfaces, this can be done by using image recognition, or using special hardware, with the same advantages and disadvantages [7].

2.2. Interface Survey: Techniques and Examples

Some sample interfaces will be presented, falling in one or more of the three types considered before.

2.2.1. The standard Wiimote interaction

The Nintendo Wii console and specially the Wiimote [10] generated a big revolution in the gaming community. While it does not have as much processing or graphical abilities as its competitor consoles, the Wii proposes a totally new type of interface, targeted to a different consumer. With the Wiimote, the player has to move the body and point to objects in the screen, in order to play, instead of just clicking buttons using a mouse or another type of standard controller.

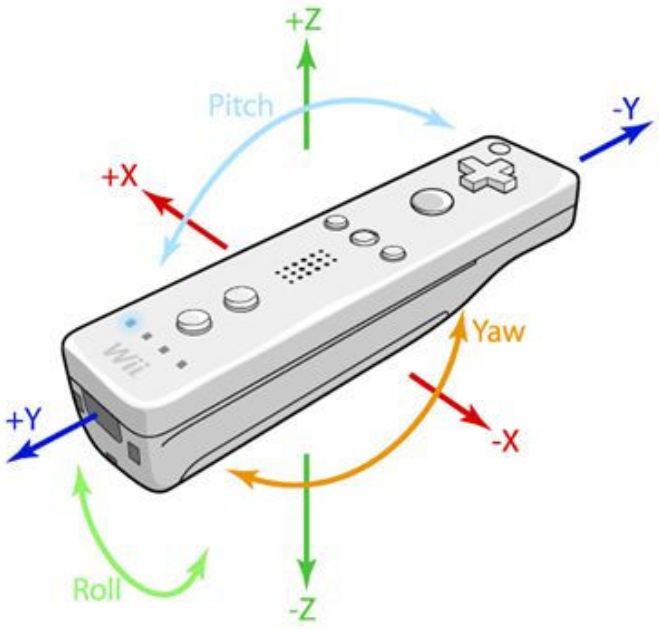

Figure 2.1 - Wii Remote (Wiimote)

The Wiimote has a 3D accelerometer that can sense its orientation, when static, and its acceleration when moving. It is interesting to be used in some games but, in many cases, it does not offer enough accuracy to make the games realistic.

The console comes with a game, Wii Sports, which is a compilation of several sports games, where the user needs to do certain movements. It contains games like tennis, bowling, boxing, and golf. In each game the user is supposed to do the same movements as if playing the real game. However, because of how the accelerometer works, most of the games can be cheated; any kind of movement can be done, as long as the timing is right. This would not happen if the user movement was tracked, instead of only the Wiimote acceleration.

The Wiimote can also be used as a pointing device. This can happen in menus, to select items, to control a character in a game, or to show targets in a shooting game. Actually, the user does not even need to point at the screen. The Wiimote has a infrared camera, that captures infrared light coming from a sensor bar (Figure 2.2). This bar is supposed to be placed above or below the screen, and in the console setup the relative position between the sensor bar and the screen must be configured. When the player is pointing at an object on the screen, the system doesn't know if she is really pointing properly, it only knows where she is pointing relatively to the sensor bar. This makes the targeting inaccurate, but it is easy for the user to adjust to this factor.

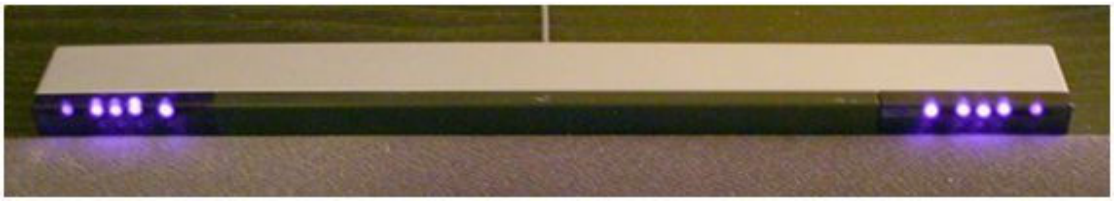

Figure 2.2 – Wii sensor bar

The only thing the system can track, in relation to the user position, is how far she is from the sensor bar, judging by the distance between the infrared points. Also, if the user is not directly in front of the bar, the measurements are not accurate (the light appears closer, which makes the system "think" the user if farther away than she really is).

This type of control is innovative and fun to use, but has a few limitations. It would be more interesting to have a system able to fully track the user positions and to react to that data.

2.2.2. WiimoteProject

The WiimoteProject consists of three applications than can demonstrate the potential of using the Wiimote just to do IR tracking:

- Tracking Your Fingers with the Wiimote
- Low-Cost Multi-point Interactive Whiteboards Using the Wiimote
- Head Tracking for Desktop VR Displays using the Wiimote

In the first project, an array of infrared LEDs is used as a light source, and the user has light reflectors in the fingers (Figure 2.3). When the user points the finger the light is reflected, and captured by the Wiimote. This is a simple system that allows the tracking of several fingers, at a low cost. The two fingers can also be used to control a 2D simulation of a grid, by zooming, panning and rotating it.

Figure 2.3 - Tracking fingers with a Wiimote

The second project uses a normal projector, and a Wiimote, pointed at the projection area, to track a light pen the user has in her hand. This light pen is basically a pen with an infrared light in the end. The user can click a button in the pen, emulating the mouse left button. This can provide a very cheap alternative to an interactive board.

Figure 2.4 - Interactive whiteboard alternative

Finally, in the head tracking project, the user is facing the Wiimote, using a pair of glasses that have one infrared LED on each side. With this, the computer can triangulate the position of the user, and display a 3D image on the screen.

Without any type of tracking, even when using a 3D application, the same scene is always presented on the screen, no matter where the user is (Figure 2.5).

Figure 2.5 - 3D view of a scene without head tracking

When enabling the head sensor, the scene changes with the user movement, making the experience more realistic (Figure 2.6).

Figure 2.6 - 3D view of a scene with head tracking

2.2.3. Head and hands tracking

Many approaches to new interfaces usually involve using methods to track the user head and/or hands. It is appropriate to use them to interact with the computer, because we use them every day to interact with other people and objects. There are a few projects involving this technology, that will be described as examples of the different approaches that are used.

2.2.3.1. Head and hands tracking by color recognition

One way to follow the user movements is to have one or more cameras capturing images, and then try to identify the different parts of the body [11]. The goal of this interface is to be able to identify when the user is pointing to an object. This system considers that there is a pointing movement when the user moves the hand away from the body, in the direction of the object, holds it for a small amount of time (around one second), then pulls it back. The next figure presents how the system works (Figure 2.7).

Figure 2.7 – Processing stereoscopic images

a) left camera image b) disparity map c) skin probability map

The hardware consists of two parallel cameras, pointed at the user. Both cameras see a slightly different figure, in the same way that happens with the human eyes. Comparing the differences between both images, the system can estimate how far each object is; doing the same process our brain does to have depth perception. In Figure 2.7.b, it is shown the disparity map, which provides information of the relative distance from the user to the camera.

To try to identify the position of the user head and hands, the system uses a skin-color map, where it evaluates the probability of each pixel belonging to the skin of a person (Figure 2.7.c). Assuming there isn't much interference, it will be able to discover the hands and head. Combining this information with the disparity map, it will be possible to find an approximate angle of where the arm is pointing to.

Using only the information from the images, this system was able to identify the direction the user was pointing with an average error angle of 37º. To improve these values, an extra magnetic sensor was used. This sensor measured the direction the head was facing. To get this direction, it was necessary to use this sensor, because simply using image recognition was not enough to determine the head direction, at the required distance. Combining the information from the cameras with the one from the sensors, it was possible to reduce the average angle error to 28º.

2.2.3.2. Head and hands tracking based in stereoscopic images

Another way to track the user movement in a 3D space is by trying to emulate a person's view. Using a helmet with two cameras, it is possible to obtain an image similar to what the user is seeing. The project described in [8], developed by Yunde Jia et. al, implements a system where the user uses such a helmet, and the objective is also to determine where the user is pointing to. This system works in a similar way to the last one, only it uses the viewpoint of the user. This makes it easier for the program interpreting the images to be in context with the user. The way to process the images is the same: the system computes the differences between the images, creating a disparity map, and then will try to identify the user hand and index finger in that image.

Figure 2.8 - Captured images from the stereoscopic vision system

Figure 2.9 - Disparity map

This system does not rely on skin color detection to detect the user hand; instead, it uses the information from the disparity map. The hand will be closer to the user body than anything else. This makes this method more robust, when subject to changes in lighting conditions, which affect the skin color recognition.

This interface was used to select objects in a virtual environment, with good results. The user was able to select different objects just by pointing at them, as shown in the next figure.

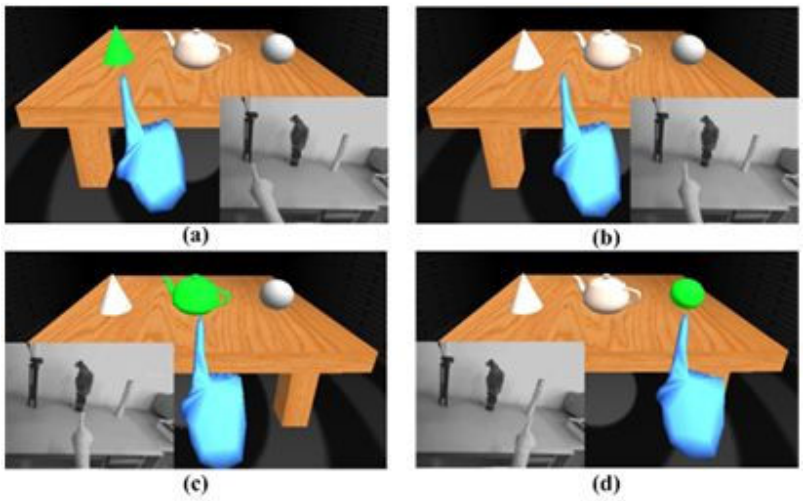

Figure 2.10 - Interacting with a 3D virtual world using pointing gestures

This system offers a good performance, and better response to lighting variation, when compared to other solutions based on skin color recognition, broadening the number of situations where it can be used.

2.2.3.3. Sony EyeToy: Using image recognition to interact with games

The Sony EyeToy [12] is the first commercial product that includes a large selection of games using augmented reality, relying only in real-time images from the user as input. It doesn't require the usage of any special symbols or extra hardware to be worn by the user.

Figure 2.11 - EyeToy for Playstation 3

In terms of hardware, the EyeToy uses a camera than can be connected to the Playstation game consoles. The innovation is that it processes the user movements, using it to trigger actions in the game. This allows the creation of interesting games, like Kinetic, for example.

Figure 2.12 - Playstation EyeToy Kinetic game

One of the objectives of the game is to hit targets on the screen. The games are interesting, but very prone to errors [2]. There are usually two types of mechanisms to interact with the games:

1. Triggering pre-defined hotspots: some areas of the game are considered hotspots, and the user has to "touch" specific hotspots with a part of her body.
2. Controlling an avatar: the user controls an avatar in the game, waving or leaning in a specific direction, to change the direction of the avatar movement in the game.

The problem with these methods is that they are very prone to errors, as the system assumes that only the user is on the screen, and playing fairly. Many of the games involve the user waiting for an object to appear, and then trying to hit it as fast as possible. However, in most of them it is easy to cheat, just waving both arms, or just moving a big object around the screen, as we can see in Figure 2.13. The system can easily recognize the movement in a specific hotspot, but it

cannot tell if the user is causing the movement, or if it is something else. This makes the system vulnerable to cheating and noise from the background.

Figure 2.13 - Cheating the Wishi Washy game using a paper bag

This is however an interesting system, where the user can be immersed in the game, but still has some limitations that are very hard to overcome without the use of additional sensors.

2.2.3.4. ZCam

Another commercial product is the Z-Camera, from 3DV Systems[13]. This device resembles a normal webcam, but it can capture images at 60fps, with depth information associated with each pixel, as shown in Figure 2.15.

Figure 2.14 - ZCam

Figure 2.15 - Image with computed depth map

This device is actually composed by two separate cameras. One is a normal RGB camera that processes the visible image. The other is an infrared camera that captures infrared light.

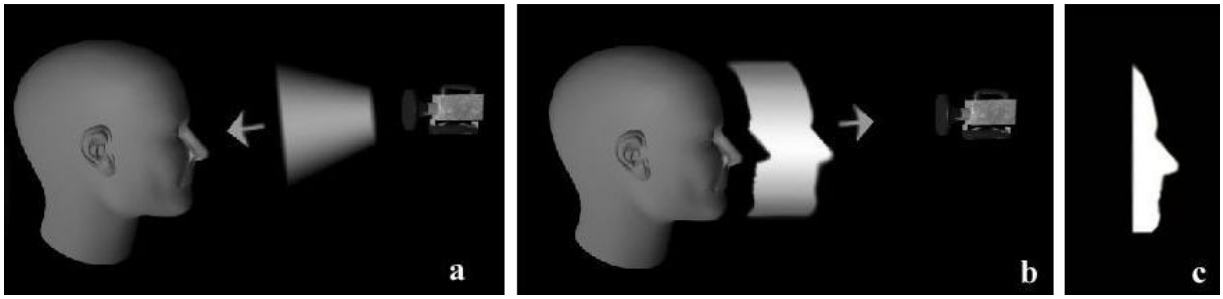

Figure 2.16 - Capturing the depth information

The depth information is calculated based on the amount of light reflected by the objects. The device starts by generating an “IR light wall”, moving along the field of view, as shown in Figure 2.16.a). As the light touched the objects, it is reflected back towards the camera carrying an imprint of the objects. The 3D information can now be extracted by deploying a fast image shutter in front of the camera at the right time, blocking part of the incoming IR light. This way, from the deformed “wall” in image b) it is possible to obtain the image in c). The collected light at each of the pixels is inversely proportional to the depth of that pixel.

This technology provides an efficient way to overcome some of the obstacles of using live images when interfacing with a computer. There is no need to limit the usage of colors in the image, as opposed to some other methods that involve identifying specific color areas, such as the user skin. The background movement also does not interfere with the interaction anymore, because it is simple to filter it all out using the depth information. It is not necessary to perform complex image analysis to extract the depth information, saving the processing power for the applications.

2.2.4. Gesture recognition with a Wii Controller

As the Wiimote has very good technical specs, when compared to the standard PC webcams, is has been used in many projects, since it appeared, only two years ago. The Wiimote has several different ways to track user movement, relying in accelerometers and infrared dots tracking. Since the Wiimote offers both acceleration and infrared sensing capabilities, with relatively good quality and has an affordable price, it has been the target of many other interesting projects[14-17].

The next project shows shat can be done using only the acceleration sensors. Its purpose is to identify the gesture the user is doing, from a group of pre-defined gestures. In this study, five reference gestures were used:

1. Drawing a circle
2. Drawing a square
3. Drawing 90º of a circle, then rolling back the same way
4. Drawing the shape of a Z
5. Simulating the serve of a tennis match.

To identify these gestures, the data coming from the sensors has to be processed. The Wiimote sends a constant stream of vector data, of 100 readings per second, which has to be filtered. For example, if the user is moving the Wiimote down, for 1/10 of a second, there will be ten vectors, pointing almost in the same direction. This means they can all be converted in only one vector, without any loss of useful information, as it is shown in Figure 2.17.

.

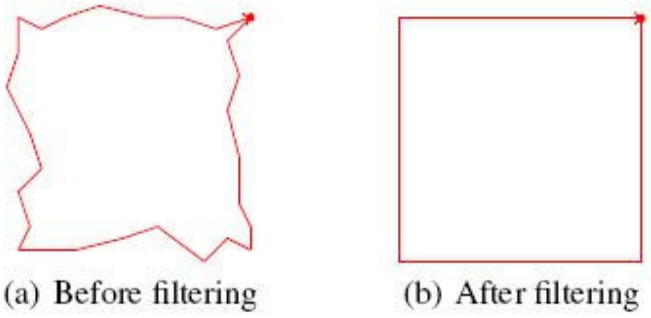

Figure 2.17 – The square movement, before and after the filtering

After this first phase, it uses a k-means algorithm to cluster the gesture data. After the filtering the number of vectors is reduced substantially, resulting in a simpler representation of the movement. The value of k had to be determined by experimentation. Several values were tried, as shown in Figure 2.18. Using k=8 it was impossible to recognize the five different gestures, because only two planes were being considered. Using k=18 resulted in over trained models, not improving the recognition and slowing down performance. The optimum results for a better combination of performance and results were obtained using k=14.

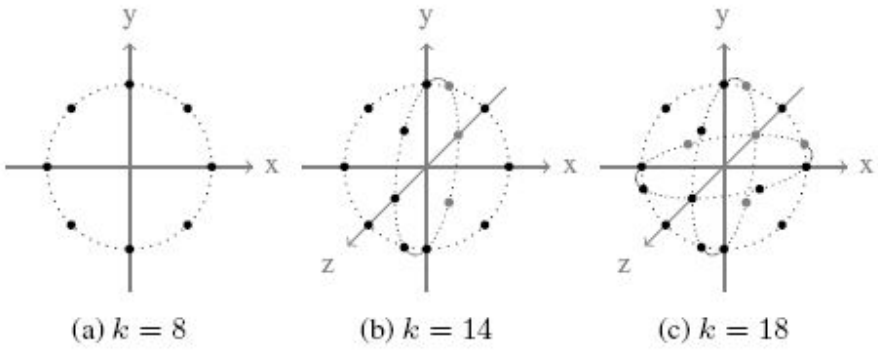

Figure 2.18 - Distribution of the cluster centers during quantization for k=8, 14, 18

The data was then used to build a Hidden Markov Model, representing an approximation of the movement. Then a Bayes classifier was used to match the processed movement with one of the reference gestures. The average recognition rates of the five gestures are shown in the following table:

Gesture	Recognition Rate
Square	88.8%
Circle	86.6%
Roll	84.3%
Z	94.3%
Tennis	94.5%

Table 2.1 – Gesture recognition rate

We can see the recognition results vary from 84 to 95%. This shows a promising start but still leaves room for some more optimization.

2.2.5. Character animation using infrared sensors

Character animation is a very important part of computer gaming and movie animation. Today it is possible to create realistic images, but it is hard to create algorithms that can mimic a person moving. It is often easier to record those motions from a living person, and apply them to a computer generated model (Figure 2.19).

This is a technique already used for some years in the movie industry, allowing the computer to track the actor motion in real time, and adding the special effects later. There are several companies developing hardware specifically for this type of image acquisition.

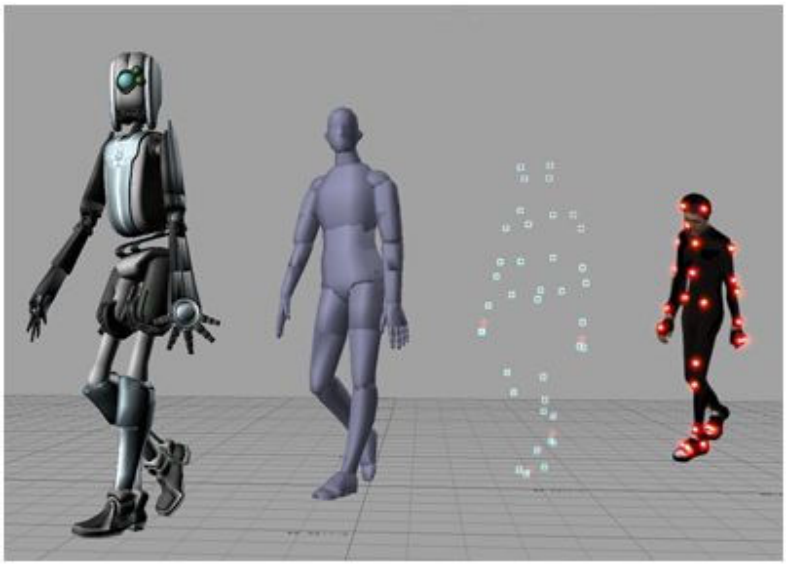

Figure 2.19 - Sampling movement from a real person and applying it to a computer character

When talking about cameras for motion capture, usually they are required to have much better precision and faster speeds than the cameras designed for just capturing images. Vicon Motion Systems is one of the companies that creates such hardware. One of their products is the Vicon T160 [18], that can capture images up to 16 Megapixels, at 120fps, or 8 Megapixels at 240fps. These are much better rates than the cameras used for normal image acquisition (Currently the new High Definition Televisions, HDTV, capture at 2 Megapixels, at 60fps).

Also, when capturing the movements of an actor for a character animation, it is required to use an array of cameras to capture the subtleties from every angle. Many times it is used an array of cameras covering a 360ª angle of vision (Figure 2.20).

Figure 2.20 - Motion capture using visible light and an array of 16 cameras

There are solutions using light sensors, using either infrared or visible light, like the one in the figure above. The number of used points varies substantially, depending how perfect the virtual model should be.

Another approach is the one used by Hideaki Nii, in a system developed in the Mitsubishi Electric Research Laboratories (MERL). This project [19] uses a different concept: the user has light sensors over her body, and there is an emitter that transmits light with spatial modulation, meaning that the light is only projected onto specific areas at each moment. By observing the moments when the sensor is capturing light and when it is not, it is possible to narrow down its position to a point.

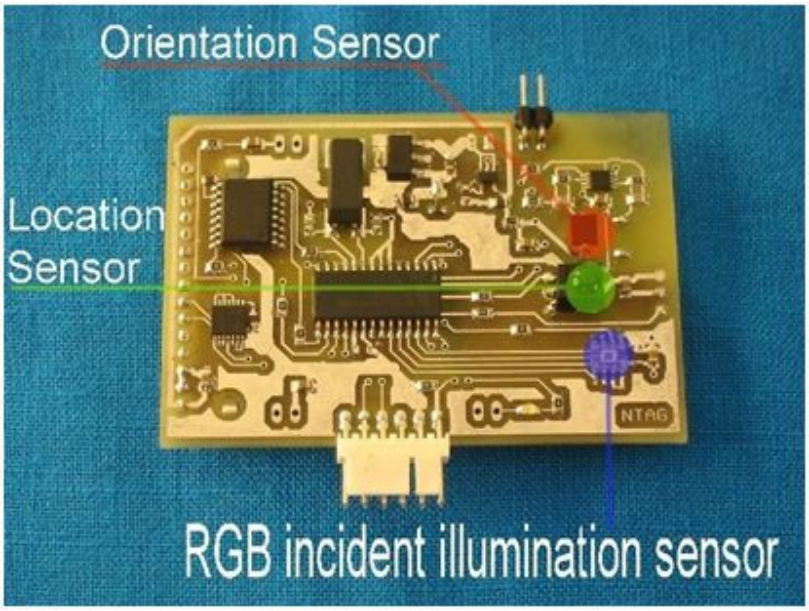

Figure 2.21 - Photosensing receiver tag

The receivers can also calculate the inclination in relation to the emitter by measuring the strength of the light they capture. When they are facing the emitter, they capture more light. When they are tilted sideways, the resulting photocurrent exhibits a cosine falloff.

The receivers are very small, and made so that they can be almost invisible when integrated in the clothes. They can be used to process data in live performances, or while shooting a movie, with minimum interference, as displayed in Figure 2.22.

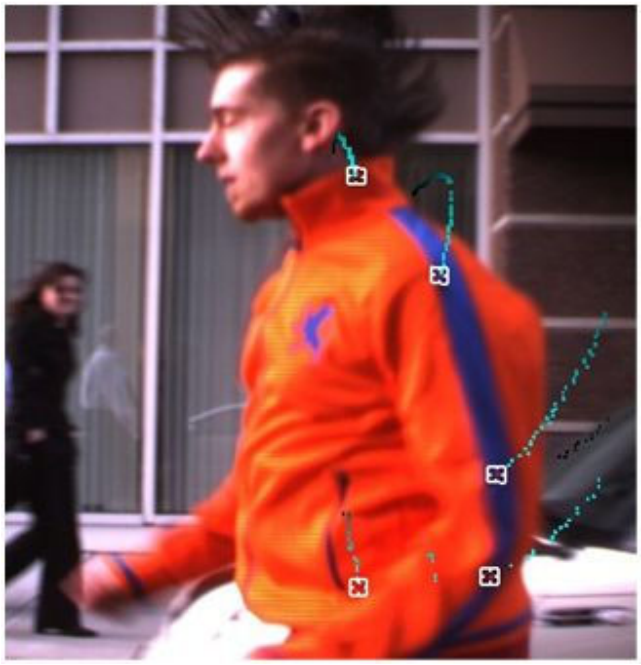

Figure 2.22 - Outdoor motion capture

This data is then sent wirelessly back to the emitter. Emitting the data is actually the bottleneck of the system, because the bandwidth is shared between all the sensors, meaning the more points to track, the slower it will be. The accuracy of location is 4mm at 2 meters range.

When compared with the previous systems, this one offers less information, because the points are only tracked from one perspective, so it is not the ideal for capturing movement to fully animate a computer generated character. However, it can be applied outdoors, with good precision, and it is easy to use while moving. This is something that would be really hard to do with systems using arrays of cameras, because they require a more complex setup, as presented in Figure 2.20.

2.2.6. Low-cost Hand Motion Capture

Another possible way to interact is tracking the hand and finger movements, but not the actual hand position. This next system[20] presents a design of a low-cost hand motion capture device that can track the hand orientation and acceleration, as well as the act of bending one of the fingers.

Figure 2.23 - 3dID glove

Each finger is tracked separately, allowing the user to perform complex gestures using combinations of movements on different fingers. By combining the data from all the sensors, a wide range of data is available to allow complex interactions.

2.2.7. Towards an Interface for Untethered Ubiquitous Gaming

This next system [21] uses another type of glove, with different objectives. It also uses accelerometers to detect hand movements, however instead of processing each finger individually, it detects when the user closes her wrist, by measuring the pressure in the palm of the hand. It also has a digital compass, in order to determine the direction where the hand is

pointing. A rumble motor is used to provide haptic feedback to the user. It is also able to identify objects that are being held, using an RFID antenna in the palm of the hand.

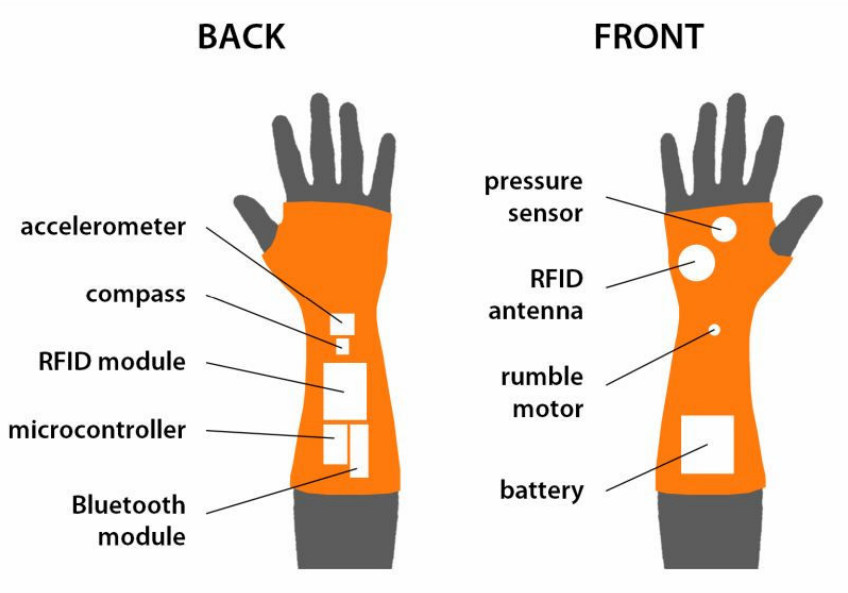

Figure 2.24 - Design sketch of the Gauntlet

For the objects to be identified, they have to be marked previously with different RFID tags.

The system proposed in this thesis can be considered a simplified version of the ones using an array of cameras; there is no need to capture all the details of the user movement, only a few key points. It makes sense to capture as many points as possible when doing character animation, but to interact with a game or an application it is not necessary to use nearly as much information. This means that it is possible to use more affordable hardware, and still get the same results, while providing some extra features, such as the identification of all the points, and the possibility of haptic feedback.

3. Hardware

The first thing to be developed was be the hardware used to communicate with the sensors. The sensor used was the Nintendo Wii remote control, which can track moving infrared points. Simply tracking a number of points is easy, and requires almost no hardware. However, the work proposed here includes additional features that require some extra hardware:

- Ability to distinguish the points from one another
- Ability of each point to send independent signals (for example indicating a certain button click)

When starting the implementation of the hardware, the plan was to create a system that offered good performance on a low budget. There were a few obstacles to overcome, and some features did not work as smoothly as expected, so different approaches had to be considered, and after some tests, three different versions of the hardware were developed, each one trying to solve problems and/or add features to the previous one.

Over the next pages there will be a detailed description of the planning and implementation of each one, as well as the problem encountered, and the solutions tested to overcome those problems.

3.1. Hardware V1.0

The initial approach involved using one-way communication, from the controller in the user hand, to the Wiimote. It was necessary for each controller to send information about its own ID, and a current state update on each of the associated buttons.

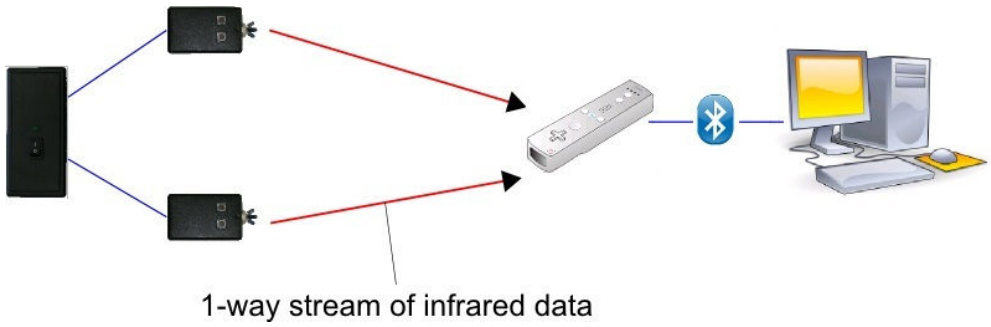

Figure 3.1 – Modules of the first version of the hardware

Considering a maximum of four points to track, and each point having a maximum of two buttons associated, it is only necessary to use four bits for each point, two for the point id (a number in [0, 3], coded in binary), and two more bits to report the state of each button. The software would track all the points in the Wiimote field of view, and register when they are on and off. Figure 3.2 shows the base of the signal all the points would send.

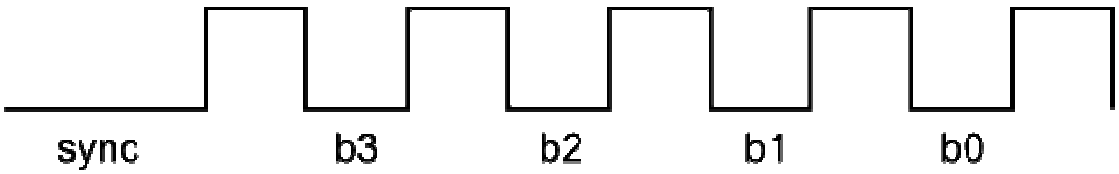

Figure 3.2 - Base signal to send

The signal starts off with a small period where the point is off, that works as a synchronization point. This was necessary, since the communication was asynchronous. During the rest of the

signal, there are allocated time slots for each bit to be transmitted. It was necessary to have assigned time slots where the point was always on, for two reasons: they would help keeping the communication in sync, and they were essential to make the motion capture smooth.

The bits b2 and b3 were used to send the number representing the id of the point, and the bits b0 and b1 represented the state of each possible button associated with that point.

In Figure 3.3 is shown an example of the signals that would be sent by controller with id=2, starting with all buttons released, then button 1 was pressed, then released again:

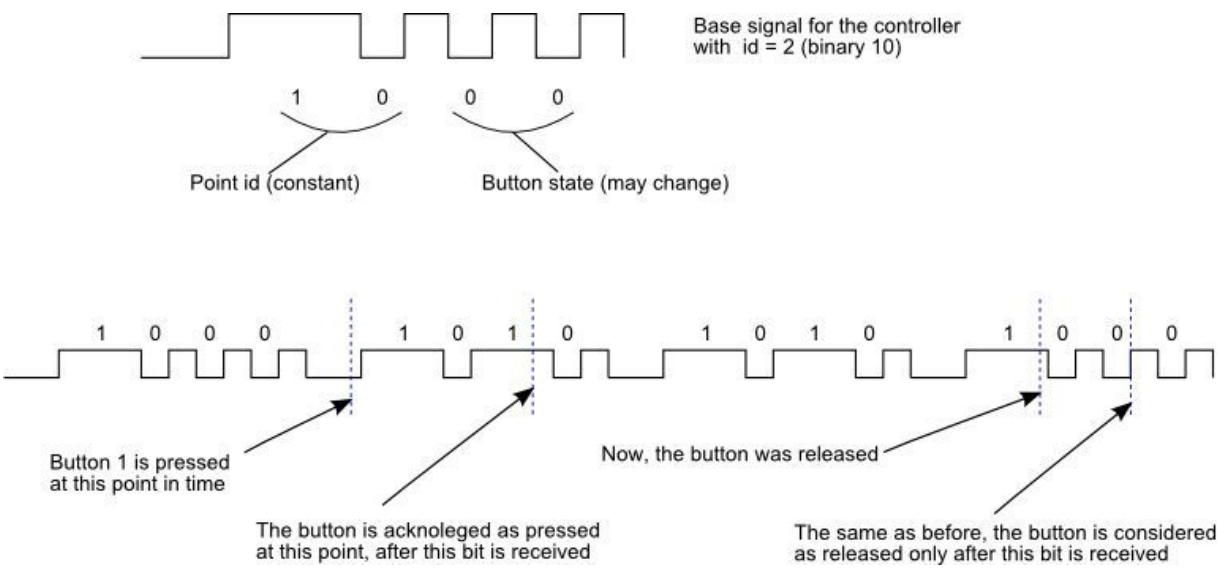

Figure 3.3 - Sample sequence

As we can see, there is a delay between the moment a button changes state, and that event is recognized. Ignoring all other delays, in the worse case, if the state change occurs just after the bit relative to that signal was transmitted, the total delay will be the length of the whole sequence.

To try out this approach, a prototype was built, that emitted the sequences as described above. To the prototype, a modular approach was chosen. There was a central device, used to power up and process the signals for all the sensors, as shown in Figure 3.4 below. This device allows to process signals from up to four controllers, connected via a RJ25 connector. In this initial phase, two similar controllers were also built, in two different sizes. The objective was to have a system that was flexible enough to allow to experiment with different controllers without having to build all the hardware for each one.

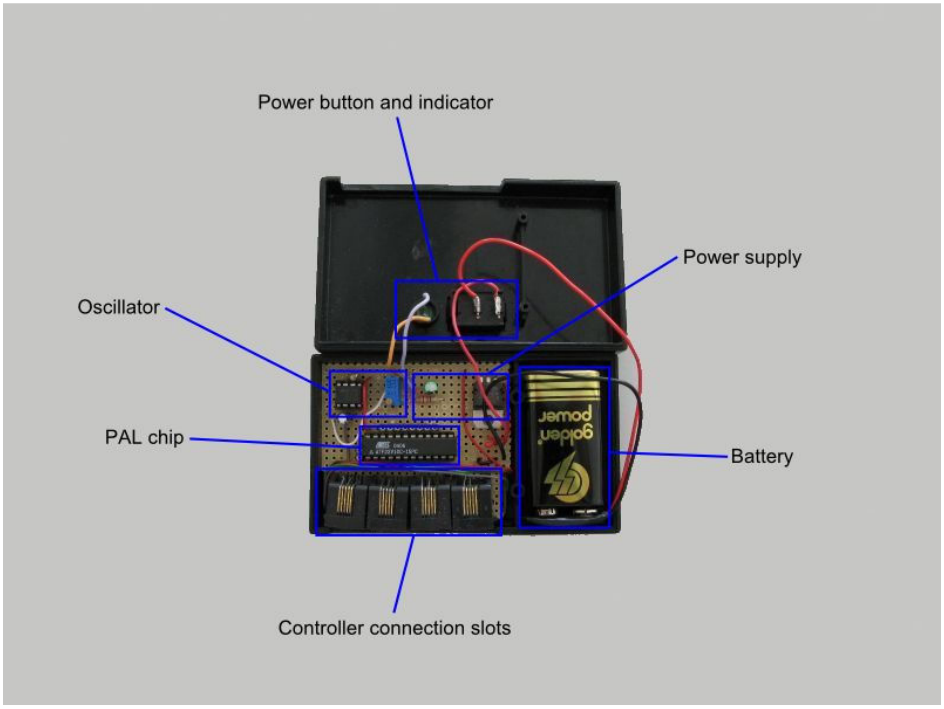

Figure 3.4 - Internal view of the central module V1.0

There is a 9V battery connected to a small power supply, to supply the rest of the circuit the necessary 5V. The core of this system is the PAL chip, used to create the 4 different signals. This system is an 11-state machine, with round-robin state transitions, where the outputs for each slot are constants, except when sending the bits matching the button state of the controllers.

The oscillator supplies the PAL with a stable clock signal, and works in the 5-250Hz range, ideal for the tests that were necessary to perform. This frequency can be set up by adjusting a small variable resistor in the oscillator.

Up to four different controllers are supported, connected via RJ25 connectors. This was made to simplify the testing with different controllers, without the need to change the central module. In this version of the hardware, the controllers can only have IDs in the range [0, 3], and each ID is assigned to a specific slot.

The controllers are much simpler than the central module. The only electronic inside each one is a small power buffer, so that the energy used to power up the LEDs comes directly from the main power supply, not passing through the chip in the central device.

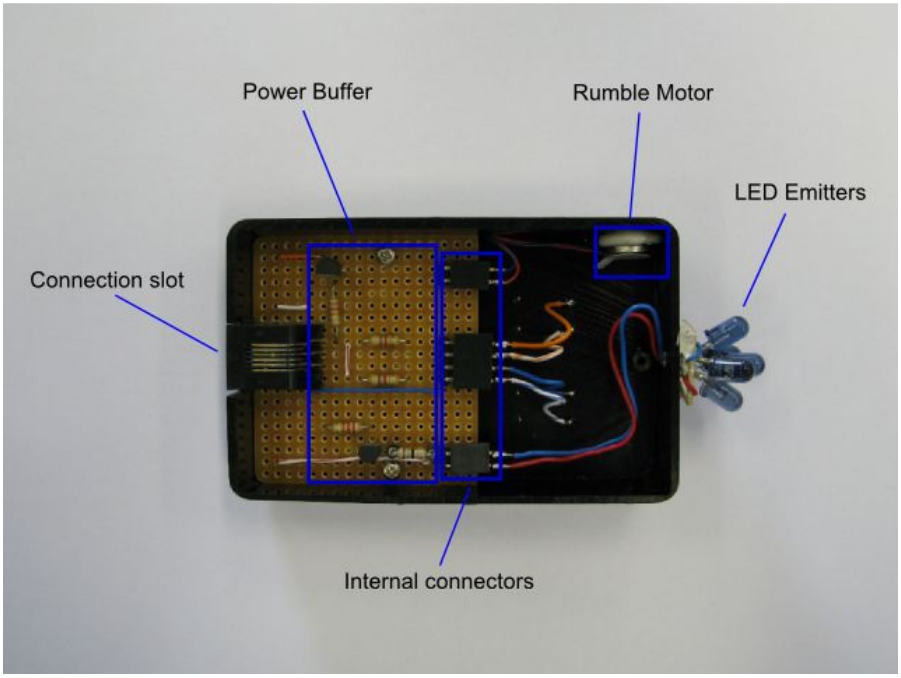

Figure 3.5 – Internal view of a controller V1.0

The power buffer was actually unnecessary for the components used in the final version of the hardware, because the power requirements are safely below the amount the chip can supply. However, projecting the hardware this way allowed testing different components, with higher power demands, and makes it simpler to change components in the future, without having to worry about those requirements.

This is actually a figure of the second version of the hardware. As shown in the figure, the controller has a rumble motor, capable of providing haptic feedback to the user. This feature was not supported in the first version of the hardware, since it is much simpler than the second version. However, the major changes are in the central module. The only difference between the controllers was the addiction of a rumble motor in the second version. Figure 3.6 presents the complete prototype, with two controllers. In this example, the controllers have the ID 0 and 3.

Figure 3.6 - Complete hardware V1.0

3.1.1. Problems encountered

In this first prototype, three big problems were encountered:

- The LEDs beam of light is narrow focused, restraining the user movements.
- The Wiimote does supply 100 updates per seconds, but sometimes the intervals are not constant, making the synchronization harder.
- To eliminate the communication errors, the system would have to work at much slower frequencies than initially planned, degrading performance.

The fact that the first LEDs had a narrow focused beam was the most important problem to overcome, not only in the first versions of the hardware, but also in the upcoming ones.

The initial vision for this system was that the user would hold a controller in her hand, and she would be able to do any movements, as long as the LEDs would remain in line of sight of the Wiimote. The limited beam angle of the LEDs limits the user movements, according to the value of that angle.

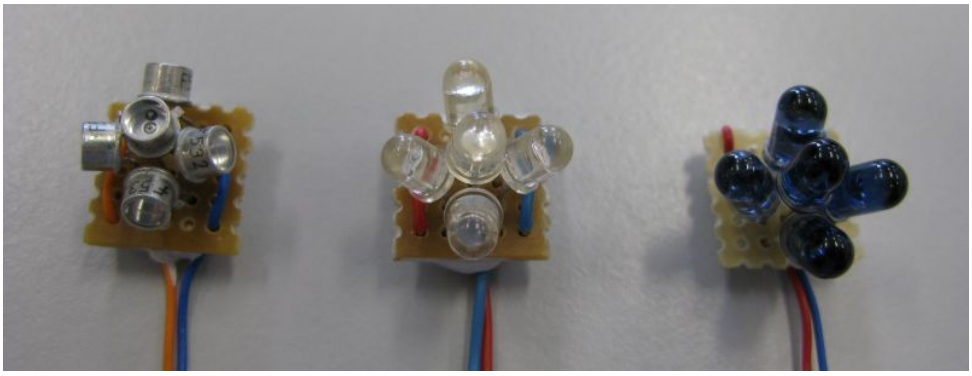

Figure 3.7 - Different types of LEDs tested

Different types of LEDs were tested, while trying to overcome this angle limitation. We explored solutions using a single one, with beam angle, also tried arrays of five LEDs, but the problem

still remained. The wider the angle, the more energy is necessary. Also, since the light is not focused, it doesn't reach as far as the narrow beam ones.

The experiences performed with higher beam angles did in fact increased the view angle, from the 80º-90º range to the 180º range. The problem was that this caused a 1000% increase in power consumption (from 50mA to 500mA), as well as substantial decrease in the range of view (around 1 meter, down from 3-4 meters range).

Some tweaking to the Wiimote was also tested, by trying to replace the standard IR filters by some weaker ones, in order to increase the range. This tweaking did not provide any substantial gain in quality, so that option was discarded. In the end, it was necessary to compromise, by sacrificing movement angle, in favor of power consumption and operating distance.

The second problem was that even though the Wiimote does supply 100 updates per second, sometimes it takes time to react to the fact that the point is off. Considering, for example, the base signal from Figure 3.2 again, the sync period takes 2 time slots, so the whole signal takes 11 time slots. To keep things simple, we can consider that one time slot is exactly 50ms. A simple program was used to process this signal, writing "1" or "0" as the signal was high or low. Since the Wiimote processes 100 frames per second, each time slot would be expected to result in 5 values being written. Sometimes it could be only 4, other times it could be 6, because there is nothing synchronizing the signal, but it should never be away from these values.

So the expected output of the program while reading that sequence would be:

00000000001111100000111110000011111000001111100000011111

If the synchronization was not perfect, we would expect to see something still very similar, for example:

000000000001111000001111100000111111000011111000001111111

However, it was common to find sequences like:

000000000001111110001111100000111111111111111000111111

This situation is very serious, because it creates errors in communication, and this problem is not easy to solve. Decreasing the frequency, the errors also decrease, but so does the system performance. The initial project involved using a frequency >50Hz, thus making each time slot last less than 20ms. A sequence using 11 time slots would be less then 220ms, roughly 1/5 of a second. These values seemed reasonable response times for a button click. However, with the problems encountered, to get more accurate and error-free readings, that value goes up substantially, making the interface seem unresponsive.

An alternative to lowering the frequency would be using more complex sequences, to deal with eventual transmission errors, but that would also degrade the performance, since it would be necessary to use many extra bits to ensure the errors would be properly dealt with. Since we did not want to sacrifice the performance of the system, a whole new approach was considered, with more complex hardware, as described in the next section.

3.2. Hardware V2.0

The approach used in the second version has a more complex model, and consequently requires more advanced hardware.

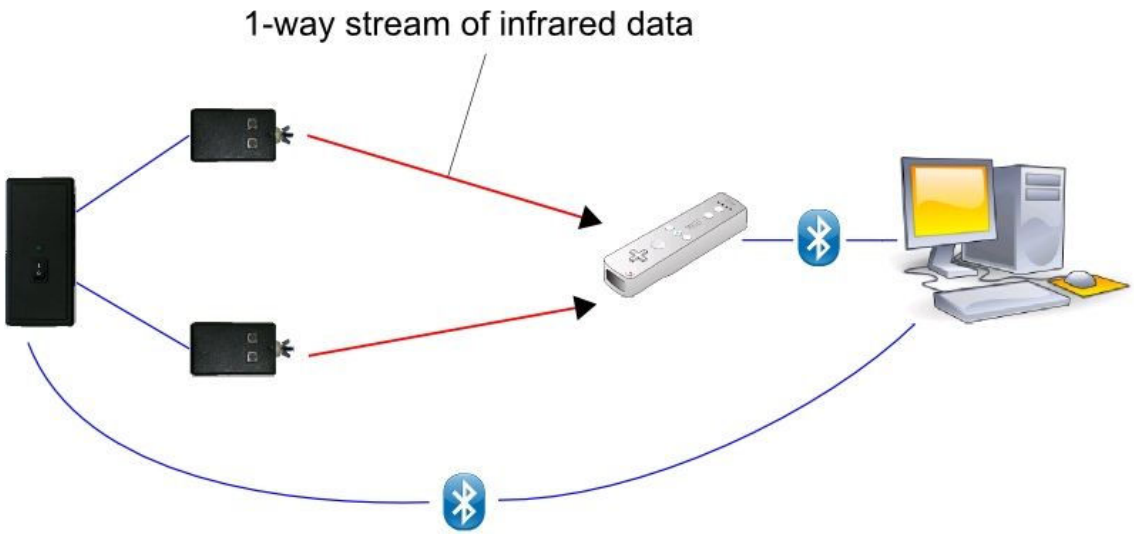

Figure 3.8 - Modules of the second version of the hardware

The main difference in the concept is that now we have a Bluetooth connection between the central device and the computer. In this way, when a button is pressed or released, that information is sent directly to the computer via Bluetooth. Having 2-way communication between the computer and the central device also allows the flexibility to provide the user with different types of feedback. This was something impossible to do in the first version, where there was no way to send messages to the hardware.

The way to identify the points is also completely different. Instead of having each controller constantly emitting a unique pattern, the solution used is more dynamic. The computer keeps tracking of all the points in the Wiimote field of view. All the active controllers are emitting all

the time, unless they receive a request for a flash. When this happens, they flash their LEDs for a few milliseconds.

When the computer needs to identify a controller, it just sends a message requesting that controller to flash, and then if in the next milliseconds one of the points flashes, it will assume that the points matches the right controller.

Considering a possible situation: If there are 2 active controllers, and also some interference (may be caused by a light, a reflection, a wireless infrared emitter), so the computer is tracking 3 points moving. This is how the identification algorithm works:

The most important variables are:

numberPoints: maximum number of points being tracked

activeControllers: number of active controllers at the moment

confidence_level[numberPoints, activeControllers]: a matrix that represents the confidence the system has about each point matching each controller.

The matrix **confidence_level** is initialized with the value "0", because the system has no information regarding what points to trust. So it starts requesting the points to flash, in a round-robin queue. When the controller C is requested to flash, and a few milliseconds after that the system perceives the point P flashing, the matrix is updated, increasing the confidence level for the position [P,C]. After a few flashes, the confidence level for some points goes over a certain threshold, and it can safely be assumed that those points match the active controllers, and it is possible to identify them. The confidence level also decreases with time, and that makes the

system request more flashes, whenever the confidence level goes below the threshold, ensuring it is still tracking the right points.

The idea behind the algorithm is very simple, but the implementation is more complex than it seems, because of all the possible situations that may happen, such as the Wiimote not capturing the flash, the controller not being in line of sight when flashing, or having some interference causing more flashing points to appear. The implementation of this algorithm considers all these possibilities, and tries to deal with them all.

As mentioned before, the controllers used are the same in both versions of the hardware; however the way the central device works internally is completely different, as shown in the following image.

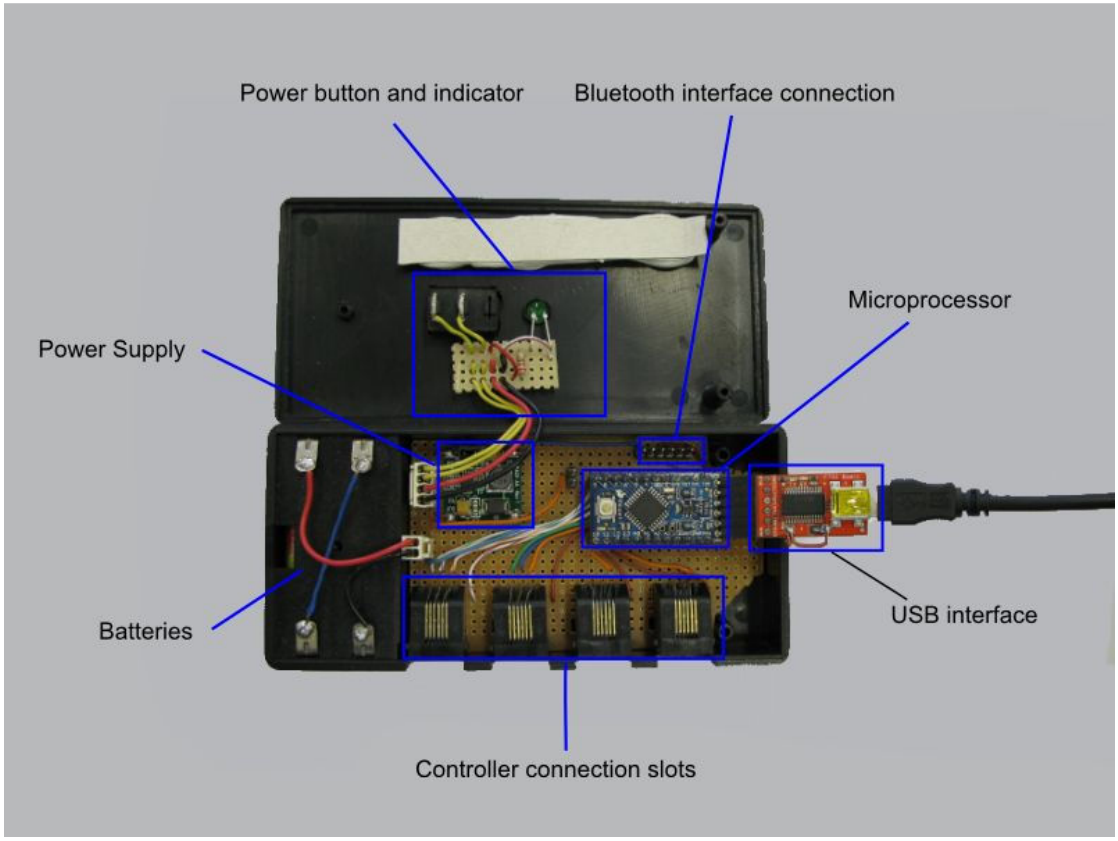

Figure 3.9 - Internal view of the central module V2.0

The external interface is the same as the first version, it supports up to 4 different controllers, connected via a RJ25 connector. The core of this module is a 8-bit microprocessor, the ATmega168. The microprocessor was programmed using the Arduino programming language (a language very similar to C). To communicate with the computer, we can either use an USB or Bluetooth interface (in the figure we can see the USB version). On either case, the data is transmitted via a virtual COM port, using the RS-232, standard, which simplifies the testing of different setups without the need to implement different code for each type of communication.

3.2.1. Problems encountered

This second prototype had much better results than the previous one. The problem with the narrow beam angle of the LEDs still remains, but all the others were solved. With the controller in line of sight, identifying the points works much faster than the previous approach, taking usually around 200ms to identify each one. This is enough time for the point to flash 4 times, drastically reducing the possibility of a random interference flash at the same time and be misinterpreted as the right one. The recognition could be even faster, by using a smaller number of flashes, however would increase the probability of interferences. The delay of 2000ms is almost perceived by the user as an instant, and there was no need to make it faster. It is important to notice that after the points are correctly identified, they are tracked constantly, so this identification process is only necessary when the system is initializing, and when the controllers move away from the line of sight of the Wiimote.

Once the controllers are identified, the navigation is very smooth, as long as the user is aware that she cannot rotate the wrists more than a certain angle. Each point is updated around 94 times per second (the Wiimote sends 100 updates per second, during each second each controller flashes on average 3 times, each flash lasting 2 updates (20ms), so 100-2x3 = 94 (940ms). This

number of updates is more than enough for a smooth image, even in applications where a fast response time is needed.

3.3. Hardware V3.0

The third version of the hardware was created just to make the whole system more portable. The algorithms used to process the data and identify the points are the same as in the previous version, the only difference being the fact that the data must be sent and received via different data streams (one for each controller).

Instead of having a central module, and all the controllers connected to it, each controller is completely independent, communicating directly with the computer. As we can see in Figure 3.10, each controller now has a combination of the technologies used in both the controller and central module of the previous version.

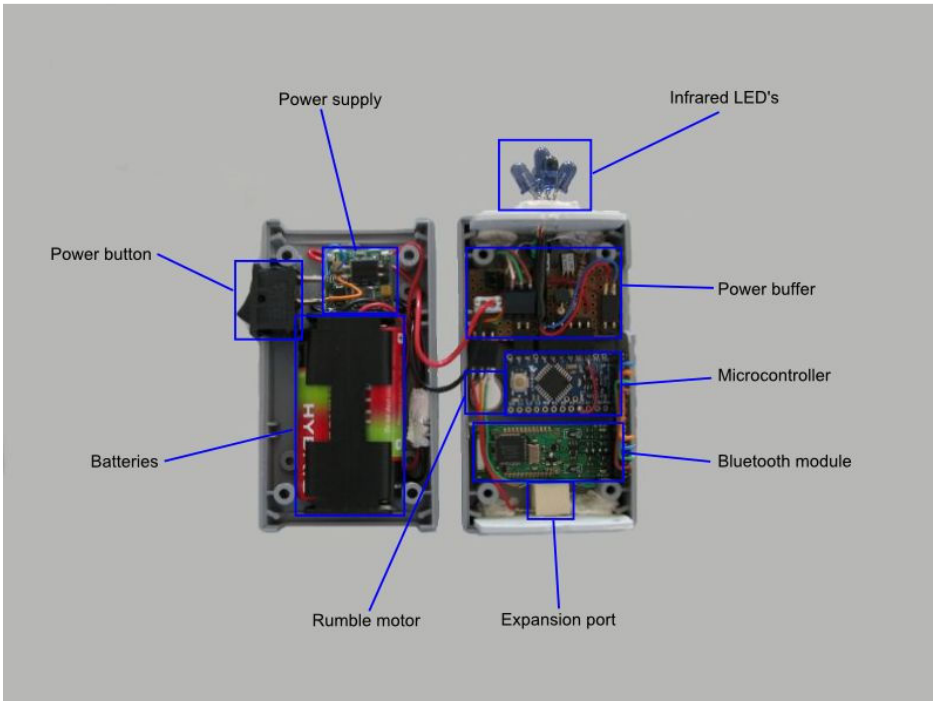

Figure 3.10 - Internal view of a controller V3.0

This version of the controller is bigger and heavier when compared to the previous ones, but it still fits comfortably in the user hand. As in the previous ones, the design is totally ambidextrous, so it can easily be used by everyone, as shown in Figure 3.11.

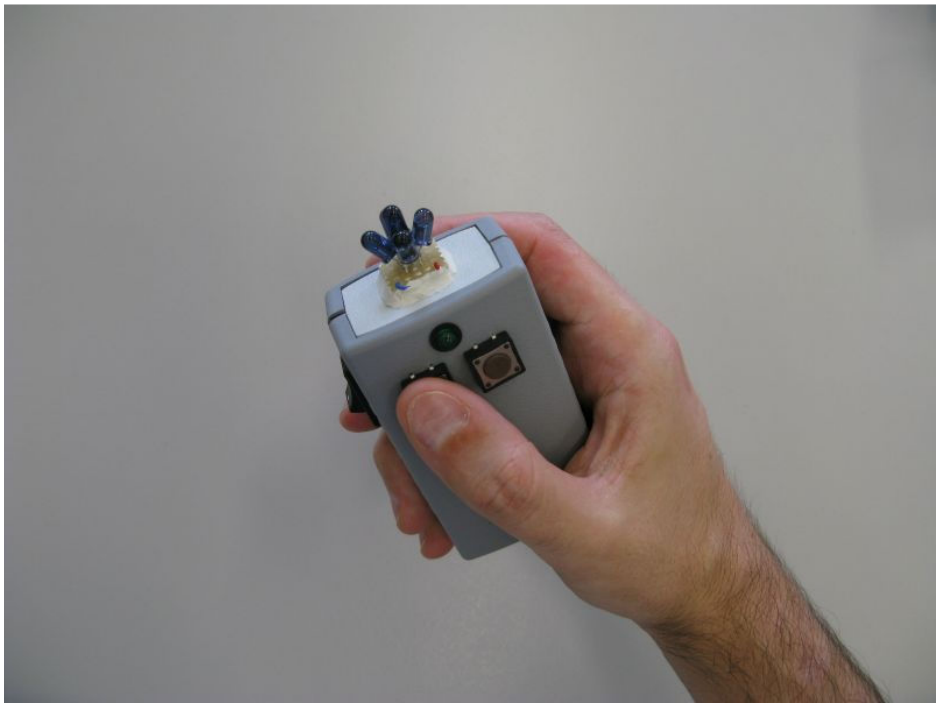

Figure 3.11 - Hand holding the controller V3.0

The algorithms used to process the data and identify the points are the same as in the previous version, the only difference being the fact that the data must be sent and received via different data streams (one for each controller), instead of only one.

While the controller by itself does not offer any extra features when compared to the previous ones, it does have an expansion port, so that some extra sensors can be connected. This port was used to connect a pair of special glasses, allowing to track the user head. These are normal

glasses, with no lenses, with one infrared point on each side, so it is possible to use those two coordinates to triangulate the user position.

Figure 3.12 - 3D glasses

These glasses were built from a pair of polarized 3D glasses, commonly used in cinema theaters. They are big enough so it is possible to use them while using a pair of normal glasses.

It would be possible to use some other object instead of the glasses, such as a hat or a headband. Given that it is important that the infrared points are always visible and facing the camera, a pair of glasses works better, since the user will always be looking at the screen (where the camera also will be).

4. Software

The software consists in an API to handle all the communication with the hardware, and some sample applications. In the following pages the main features of this API will be described, as well as the applications.

4.1. Creating the API

The developed API provides an interface that programmers can use do develop applications without having to worry about the hardware. This API was developed in parallel with the three versions of the hardware, processing all the sensor data and button state changes, providing the higher-level programmer with simple methods to get information about the controller speed, velocity and button states. There were many internal changes made in the internal coding, however the interface was always very similar.

All the example applications used were developed using the OpenFrameworks environment, so an extension of the OpenFramework main class was created, to simplify the process of interfacing with the developed controllers.

To better understand the following pages, a few concepts must be clear:

- The system tracks all the infrared points in line of sight of the Wiimote.
- One controller has one or more infrared points (a simple hand controller will have one point, a head tracker will have two points, and a combination of the two will have three points).
- There may be more visible points than the ones from our system, due to interferences.

Here is a list of the main methods implemented:

- `void initDecoder();`

This method must be called before any of the others, to initialize the communication. It tries to connect to an available Wiimote, and initializes a thread that processes all incoming data, to track and identify all active controllers. In the beginning there aren't any active controllers.

- `void initController(char slot, char[] com_port, char type);`

When initializing a controller, the system needs to know which COM port to communicate with it, and also what kind of controller it is. The slot number is simply an ID to identify the controller, chosen by the user (in version 2.0 of the hardware, this number would have to match the number of the slot where the controller was plugged into the central module). A thread is also created, to handle all the incoming data from that controller.

The available controller types are:

  - HAND_CONTROLLER (1 infrared point)
  - HEAD_CONTROLLER (2 infrared points)
  - HAND_HEAD_CONTROLLER (3 infrared points)

Depending on the type of controller chosen, the system will have to assign one unique point ID to each one of the necessary points, so that they can all be tracked and identified. Once the IDs are assigned, they must be tracked, so the thread created by the `ofInitDecoder()` method will start sending flash requests to all of them, as explained in 3.2.

- `void buttonChanged(char slot, char button, char state)`

This is a callback method that if invoked whenever a button state change occurs. It has three variables, matching the slot where the change happened, the button involved and the new state of the button.

- `char getButtonState(int slot, int button)`

A simple method that informs the state of a button, given the controller it belongs to.

- `RealCoordinate getHandCoordinates(char index)`

As soon as a controller is identified, at any given time it is possible to discover the position, using this method. The result is a point in normalized coordinates. If by any chance the point is not visible or not identified at the moment, if returns the last known coordinate.

- `RealCoordinate getHeadCoordinates(char index)`

Similar to the previous one, returning the head position instead of the hand.

- `float getHeadDistance(char index)`

Returns a value that can be used to measure the distance between the head and the Wiimote. This value is calculated by triangulating the position, by measuring the distances between the two points being used to track the head.

- `RealCoordinate WiiIRDecoder::getHandSpeed(char index )`

This method doesn't actually measure the speed of the controller, instead it uses a combination of the speed and acceleration. It provides smoother values when compared to the calculations of the instant speed, and it is important to deal with the flashes. When a point flashes, it stops being visible for a few milliseconds. At that moment, by calculating the instant speed, that value would

suddenly drop to zero, making the programmers of the higher-level applications having to worry about this. The implemented solution deals with these problems directly.

4.2. Applications

The objective was to create an interface with a fast response time, so it should be tested in applications where that was necessary. A total of four sample applications were developed, to illustrate possible usages of this new interface:

- Virtual Juggling
- Image Viewer
- Hammer Hero
- 3d Navigation

These applications allowed the testing of all the features of the interface, the evaluation of the performance, and provided a better idea of how the users reacted to it, and the major issues that need improvement.

4.2.1. Virtual Juggling

This was the first application to be implemented. It allows the user to perform virtual juggling, by moving her hands the same as if she was actually juggling real objects, as shown in Figure 4.1.

Figure 4.1 – Virtual Juggling

In this game the user has one controller on each hand, and to move them she only needs to move her own hands the same way. The number of oranges on the screen can be controlled by two buttons in one of the controllers (whenever one orange drops down the screen, a new one is thrown up in a random direction). The physics of the game can also be controlled in the two buttons of the other controller, used to increase or decrease the acceleration of gravity, making the game faster or slower.

To grab an orange, the user simply moves the hand underneath the orange when it is falling, and the hand automatically grabs it. Each hand can only hold one orange at a time. To throw an orange, the hand needs to do a fast movement upwards, and then stop or move down again. When the hand stops moving up, the orange is released, moving towards the direction the hand was moving before it stopped, and keeping the same acceleration.

While being simple, this game was perfect to perform the first tests with this interface, since it needs to use most of its features:

- Requires fast response times regarding the hands movement
- Uses all of the data related to the hands (position and velocity)
- Uses button clicks for configurations
- Needs to identify each hand

It was easy to see that the performance of the first version of the hardware was poor, when compared to what was initially expected. When working at high frequencies, the motion was very fluid, however the identification of the points and processing of the buttons was not accurate. The system could only be trusted by decreasing the working frequency of the hardware to significantly lower values. In this case, the response of the system was not at all fluid; the coordinates would be updated less than ten times per second, making it difficult to catch the oranges.

When the prototype of the second version of the hardware was ready for testing, the results were much better. The only limitation is the rotation of the wrists, as mentioned in the previous chapter. The response time of the virtual hands is almost instantaneous, and delays are rare.

This application also has a debug mode that was useful when testing the calibration parameters for the hardware, in order to get a better performance.

4.2.2. Image manipulation

The second application is a typical image manipulation tool, where the user has a group of images, that can be moved, stretched and rotated. The application itself is not a novelty, however by using it with this interface, it is possible to have features that cannot be implemented when using a normal multi-touch device (where there is no way to distinguish the points).

This application also seemed interesting to implement because manipulating images requires different characteristics from the interface, when compared to the previous application; the fast response time is not required, instead in some situations it is necessary to have high precision. This allowed the testing of the interface in a different scenario.

This application is also very simple to use. It can be used by a single user or several users at the same time, each using one or two controllers. Each controller is represented in the application by a dot of a different color. The users can move one image by clicking on it and dragging the dot in the desired direction, by moving the controller. To rotate or stretch the images, two controllers must be used. By clicking one button on each controller, while having the matching dots floating over the same image, the application locks the exact positions of the image to the coordinates of the dots, and then by moving both controllers, the image responds accordingly.

Usually when demonstrating the usability of multi-touch surfaces, similar applications are used. However, there is usually no way to identify the users, or distinguish the fingers from both hands. Most times this not even necessary; usually multi-touch surfaces are small, and used by a single person at each time. When working on a larger scale, when the applications are meant to be used by more than one person, it becomes interesting to be able to distinguish the users. It is

possible to have then cooperating, competing against each other, or simply doing their own things, without interfering with each other.

Figure 4.2 - Image viewer

In this application, there is the option of having all the user manipulating all the images, or assigning sets of images for different users, with different permissions for each one. The images can re-position themselves closer to a specific user. As another option, different sets of images can be used, changing according by who is using the system at each moment.

4.2.3. Hammer Hero

The Hammer Hero game was the first application to be publically exhibited to a large audience, using this new type of controller. The objective was to develop a game easy to explain and fun to watch and play, in order to motivate people to use it and give their feedback.

The result was a game for two players, where they compete against each other. The idea behind the game is simple: in the beginning, the game captures an image with the face of each player. During the game multiple copies of each face keep appearing randomly throughout the screen, and each player must hit the opponent, while avoiding hitting himself.

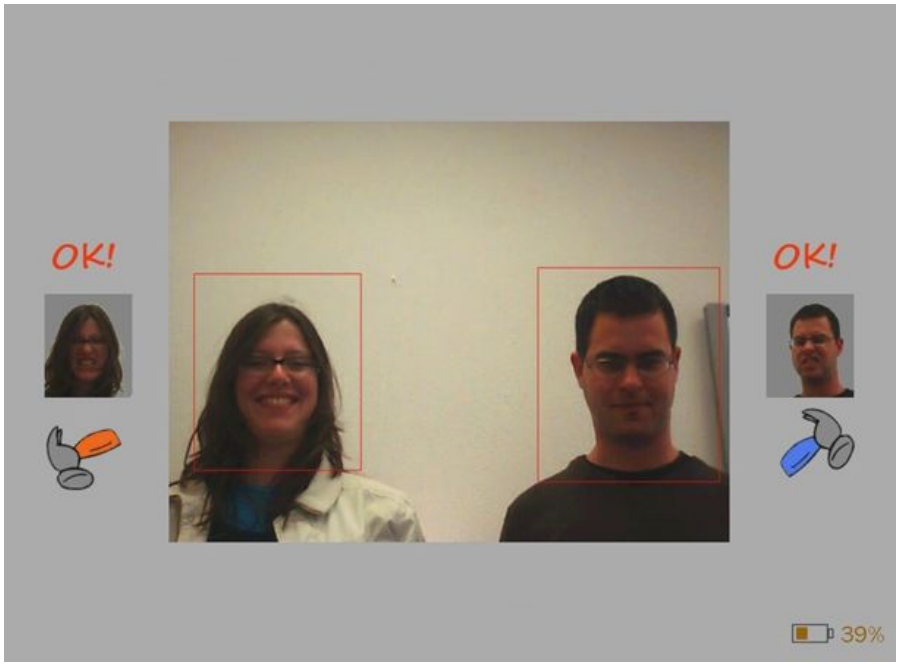

Figure 4.3 - Hammer Hero during face selection

As soon as the game begins, both players start by placing themselves in front of a camera, to capture the faces they want to use in the game. They will see a red square around their face, indicating the area that will be captured. At this moment they can pose, and capture images of their own faces, using one of the buttons on the controller. They can capture as many images as they want, with every new image overwriting the old one. Once the final image is chosen, the

other button of the controller is used to advance to the next stage of the game (only after both players have chosen their images).

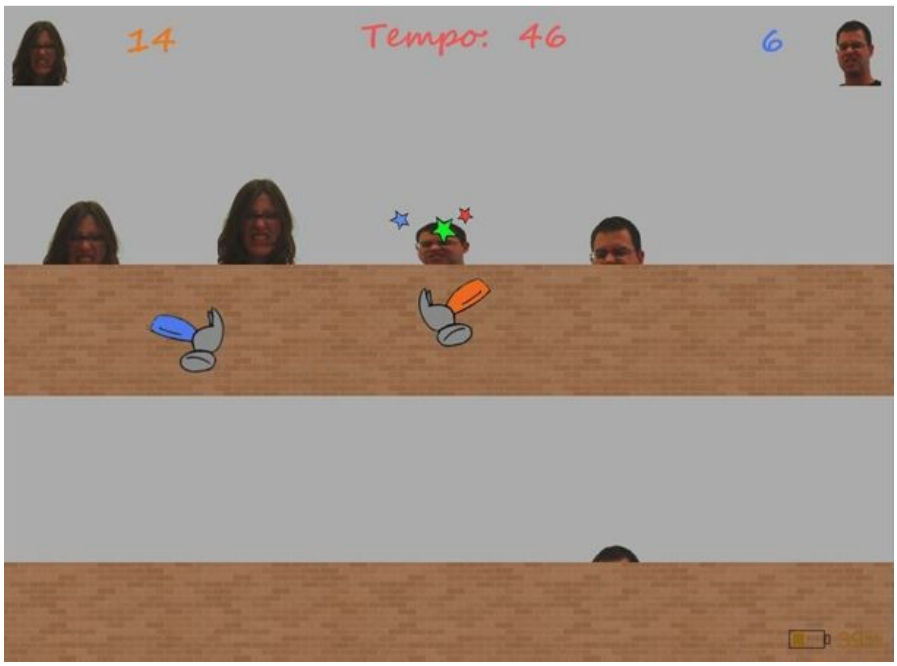

Figure 4.4 - Hammer Hero during the actual game

As shown in the previous figure, there are two walls, and images of both players start appearing on random places all around the screen, first hiding behind the walls, and then moving up and down again after a short moment. Each player uses his controller to move a hammer in the screen. The hammers have different colors, so it is easier for each player to identify his own.
To hit the heads, it is necessary to use the controller in the same was as using a real hammer, by moving it up, and then quickly moving it down

4.2.4. 3D Navigation

The final application developed makes use of all the hardware. The glasses are used to track the user in a 3D area, while the hand controller is user for rotations and some translation movements. In this application, the user can navigate through a 3D scene, as shown in Figure 4.5.

Figure 4.5 - 3D Navigation

This application is more realistic when it is projected on a large area. The user can move in front of the projected area, and the viewpoint will change. If she steps left or right, the viewpoint will change accordingly. By moving forward or backward, the application also changes the viewpoint of the scene. All of this creates the illusion of actually looking into a real 3D world. It is possible to calibrate the tracking of the movement so that it matches the changes in the view on the screen, making it more realistic.

Dealing with rotations poses a bigger challenge. When rotating, the user wants to look at the world from a different angle, however the screen does not move. Alternatives to solve this

problem it may involve using small LCD screens embedded in glasses, or having a room with projections in every wall, for total immersion [9].

Since the objectives of this thesis did not involve the implementation of new visualization systems, a simple projection was used. To control the rotation, different approaches were tested:

- Rotating the head in the desired direction
- Tilting the head in the desired direction
- Moving one hand controller in the desired direction

Rotating the head did not work very well on two levels: it is harder and less accurate to detect the head rotation, and it is confusing for the user having to look away from the image, so this approach was discarded.
Tilting the head has a simpler implementation and very accurate results; however the movement still does not feel natural. On top of that, it is not a comfortable movement, because the user needs to tilt the head more than a certain angle, which can be bad for the neck.
The best option found, while using the current hardware, was a hand gesture combined with a button click. By clicking on a controller button, and moving the arm sideways, the user can rotate the view on the virtual scene. By moving the arm up and down, she can move forwards and backwards. As mentioned before, it is also possible to move forwards and backwards simply by moving in the real work, but this only works in a small area. If the user is moving in a large virtual area, it is not possible to explore it all by physically moving. This would mean the physical space would have to be as large as the virtual space.

5. User tests

The main objectives of these tests were to gather user opinions related to the interface ease of use, adaptability, importance of haptic feedback, as well as the overall system response. The test results were grouped by users with different gaming habits, gender and age. By combining the results of the inquiry and our own observations of the tests, it was possible to better understand the main flaws of the system, and try to uproot them prior to building any other applications.

5.1. Participants and Methodology

The user tests were conducted during the university open day, where students from high-schools can visit the departments and laboratories. So the participants in these tests are mostly high-school and university students, with ages ranging from 13 to 26 years old, with a mean value of 17.4. None of the users had contact with the interface before, and they all had only a short time to adjust to it. The sessions and the questionnaires had to be short, due to the high number of candidates involved.

There were two investigators assuming the roles of facilitator and observer. One session would start with the facilitator asking two players to hold the controllers, and then he would explain how to use the buttons to capture the images. During the practice stage, he would also explain that each player objective was to hit his opponent, mimicking the use of a real hammer. Some users would make their movements too wide, exiting the range of the sensors, and in that case they would be told what they did wrong, and would try to adapt to the range. During the actual game, the facilitator would not say anything, unless he saw the player doing the same mistakes.

Between capturing the images and the beginning of the actual game, a practice stage was introduced, where the players could train moving the hammer, and test the hammering

movement. It was important to add this stage, since the sessions were very short, and none of the users had ever had tried the game or even the interface before.

Each session was around 3 minutes. There would be a small 30 seconds introduction, then image capture and training stages would last around 30 seconds each, and the game itself would take exactly 1 minute. After each session, the players would give a short verbal opinion, and then would be asked to answer a short questionnaire. During all the sessions the observer was taking notes about the first contact of the players with the interface, how fast would they adjust to it, as well as their reactions and comments.

5.2 Questionnaire

Each questionnaire had some personal questions, so the results could be divided according to gender, age and previous gaming experiences of the participants. Then the users would be presented with 5 statements related to their gaming experience.

Sweetser, P. et al described in [22] a model to evaluate player enjoyment in a game. As both the tests and the questionnaires had to be very short, it was not possible to evaluate all of the aspects of the game. Since the main objective was to evaluate the opinions about the interface more than the actual game, the questions were focused on the sense of control and feedback the user has over her actions.

Statements
This controller is adequate to this type of game.
This interface brings more realism to the game.
The rumble feature brings more realism to the game.
It is easy to learn how to use this type of controller.
Not being able to totally rotate the wrists hurts the game experience.

Table 5.1 - Statements presented on the questionnaire

Users would indicate their level of agreement by choosing a value on a 5-point Likert-type scale, with a response of 1 (one) meaning "total disagreement" and a response of 5 (five) meaning "total agreement". The objective was to get some feedback on how the players felt, by playing the game using this type of interface, and how much of a problem were the constraints that the interface presented.

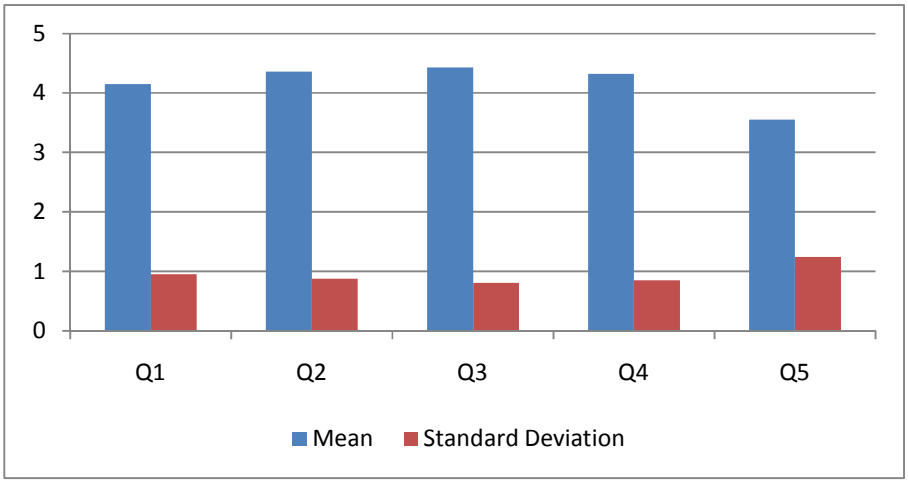

Figure 5.1 – Results of the questionnaires

The results shown in Figure 5.1 represent the answers of all 100 participants. However, it is important to notice that the data was also analyzed considering only specific subjects, and all the results were very similar. The subjects were divided according to gaming habits, gender and age range, and all the answers on these groups had very similar mean values and standard deviation.

The first 4 questions had very positive and similar results. Most subjects agreed that the controller was adequate to this type of game, that the game felt more real using it instead of a standard controller. Most of them also liked and agreed that the rumble feature made the game more real. During the tests, some subjects had some initial problems adjusting to the controller, especially while doing wide movements where they would rotate the wrists more than they should. Once the facilitator reminded them of that restraint, they would adapt easily. Q4 and Q5 show just that, most people found it was easy to learn how to play the game using this controller, and many agreed that not being able to fully rotate the wrists is sometimes a problem.

There was also a section where the participants could write their own opinion, and most of the ones that did wrote comments reinforcing the same points. They would either agree that the game was interesting and this type of interface was a really good idea, or say that not being able to rotate the wrists was sometimes a problem in the middle of the game, and this should be improved in order to make the game more enjoyable.

6. Conclusions and future work

Looking back to the initial objectives of this thesis, the results were very positive. A survey was done on the related work, and the objective was to create a new type interface with characteristics that were not found in similar projects. A study of the available technologies was made, and three prototypes of hardware were developed, to improve quality and overcome problems encountered in the previous versions.

Most of the objectives were achieved, and the final version of the hardware offers a good performance. Some compromises had to be made, limiting the movements more than was desired in the initial vision. Even with such limitations, in the end the result is still a new type of interface with some unique characteristics, offering good performance and flexibility.

Tests were made with a large number of users in order evaluate the response to this new type of interface. By analyzing the reactions of the users when using the applications, specially the Hammer Hero game, and the answers in the questionnaires, it is visible that the users were pleased with the experience. Apart from the restriction of not being able to fully rotate the wrists, most of the users enjoyed using the interface, agreeing that it was simple, natural and adequate to this type of applications.

As future work on the interface, in terms of improving what was already done, the most relevant thing would be allowing the user to fully rotate the wrists. This is mostly a hardware problem, and it can be solved by experimenting with different combinations of infrared emitters, cameras and filters.

Regarding new possible features and applications, one thing to consider is using accelerometers in order to detect the inclination of the controllers. Another possibility is adapting the controllers to use with interactive tables. By combining an interactive table (or a wall) with the kind of hardware developed in this thesis, it would be possible to create a type of pen to identify the users, provide haptic feedback and allow the combination of gestures with button clicks.

6. Bibliography

1. Jacob, O.W., A.M. Brad, and A. Htet Htet, *Writing with a joystick: a comparison of date stamp, selection keyboard, and EdgeWrite*, in *Proceedings of Graphics Interface 2004*. 2004, Canadian Human-Computer Communications Society: London, Ontario, Canada.
2. Shuo, W., et al., *Face-tracking as an augmented input in video games: enhancing presence, role-playing and control*, in *Proceedings of the SIGCHI conference on Human Factors in computing systems*. 2006, ACM: Montr\&\#233;al, Qu\&\#233;bec, Canada.
3. *Wii Yourself.* [cited 2009 11 July]; Wiimote C++ library to handle bluetooth communication]. Available from: http://wiiyourself.gl.tter.org/.
4. Lee, J. *Wiimote Project*. [cited 2009 11 July]; Available from: http://www.wiimoteproject.com/.
5. Daisuke, N., M. Koji, and N. Ryohei, *Virtual marionette theater using hand recognition*, in *Proceedings of the 2nd international conference on Digital interactive media in entertainment and arts*. 2007, ACM: Perth, Australia.
6. Kazuhiro, M., et al., *Statistical segmentation and recognition of fingertip trajectories for a gesture interface*, in *Proceedings of the 9th international conference on Multimodal interfaces*. 2007, ACM: Nagoya, Aichi, Japan.
7. Louis-Philippe, M. and D. Trevor, *Head gesture recognition in intelligent interfaces: the role of context in improving recognition*, in *Proceedings of the 11th international conference on Intelligent user interfaces*. 2006, ACM: Sydney, Australia.
8. Yunde, J., L. Shanqing, and L. Yang, *Tracking pointing gesture in 3D space for wearable visual interfaces*, in *Proceedings of the international workshop on Human-centered multimedia*. 2007, ACM: Augsburg, Bavaria, Germany.
9. Marcio, C.C., H.M. Carlos, and K.Z. Marcelo, *On the usability of gesture interfaces in virtual reality environments*, in *Proceedings of the 2005 Latin American conference on Human-computer interaction*. 2005, ACM: Cuernavaca, Mexico.
10. *Nintendo Wii Console Home Page*. 2006 [cited 2009 11 July]; Available from: http://www.nintendo.com/wii.

11. Kai, N. and S. Rainer, *Pointing gesture recognition based on 3D-tracking of face, hands and head orientation*, in *Proceedings of the 5th international conference on Multimodal interfaces*. 2003, ACM: Vancouver, British Columbia, Canada.
12. *Sony EyeToy Homepage*. [cited 2009 11 July]; Available from: http://www.eyetoy.com/.
13. *3DV Systems Homepage*. 2009 [cited 11 July]; Available from: http://www.3dvsystems.com/.
14. Thomas, S., et al., *Gesture recognition with a Wii controller*, in *Proceedings of the 2nd international conference on Tangible and embedded interaction*. 2008, ACM: Bonn, Germany.
15. Torben, S. and J.G. Henry, *A Wii remote, a game engine, five sensor bars and a virtual reality theatre*, in *Proceedings of the 2007 conference of the computer-human interaction special interest group (CHISIG) of Australia on Computer-human interaction: design: activities, artifacts and environments*. 2007, ACM: Adelaide, Australia.
16. Hyun-Jean, L., et al., *WiiArts: creating collaborative art experience with WiiRemote interaction*, in *Proceedings of the 2nd international conference on Tangible and embedded interaction*. 2008, ACM: Bonn, Germany.
17. Sreeram, S., S.Z. Edmund, and P. Beryl, *3D input for 3D worlds*, in *Proceedings of the 2007 conference of the computer-human interaction special interest group (CHISIG) of Australia on Computer-human interaction: design: activities, artifacts and environments*. 2007, ACM: Adelaide, Australia.
18. *Vicon Motion Systems Homepage*. [cited 2009 11 July]; Available from: http://www.vicon.com/products/t160.html.
19. Hideaki, N., et al., *Illumination sensitive dynamic virtual sets*, in *ACM SIGGRAPH 2007 emerging technologies*. 2007, ACM: San Diego, California.
20. Michele, S., et al., *3dID: a low-power, low-cost hand motion capture device*, in *Proceedings of the conference on Design, automation and test in Europe: Designers' forum*. 2006, European Design and Automation Association: Munich, Germany.
21. Tiago, M., et al., *Towards an interface for untethered ubiquitous gaming*, in *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*. 2008, ACM: Yokohama, Japan.
22. Penelope, S. and W. Peta, *GameFlow: a model for evaluating player enjoyment in games.* Comput. Entertain., 2005. **3**(3): p. 3-3.