Università di Pisa
Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea Specialistica in Tecnologie Informatiche

Tesi di Laurea

Design of a 3D mouse using accelerometers

Laureando:
Lucio Davide Spano

Relatore:
Dott. A. Cisternino

Anno Accademico 2008/2009

Ai miei genitori

To my parents

Abstract

In later years, the number of devices equipped with accelerometers has highly increased, due to their employment in mobile devices for screen orientation and in games for gesture recognition. This thesis debates their advantages and limitations for the creation of a three-dimensional mouse prototype, using a game controller equipped with these sensors. After describing their functioning and highlighting which kind of applications they already support, the work focuses on the design and the implementation of a library for managing a three-dimensional pointer abstraction. In order to address the position drift problem, due to the fact that an accelerometer cannot distinguish between the gravity and input acceleration, two motion-tracking algorithms are proposed: the first one is based only on a three-axial accelerometer and is able to recognize either linear motion on three axes or rotation about two axes. The second one, combining the input of an accelerometer and a gyroscope, can recognize linear motion and rotation on three axes at the same time. The abstraction is tested in a three dimensional environment where the user can move and rotate the pointer, register and analyse movement data. In conclusion are discussed the possible application of the results in windows systems and for future works.

Riassunto analitico

Negli ultimi anni, il numero di dispositivi equipaggiati con accelerometri è notevoltemte aumentato, a causa del loro impiego nei dispositivi mobili per l'orientamento dello schermo e nei giochi per il riconoscimento di gesti. Questa tesi discute i loro vantaggi e limitazioni per la creazione di un prototipo di mouse tridimensionale, utilizzando un controller per videogames equipaggiato con questi sensori. Dopo la descrizione del loro funzionamento e delle applicazioni che attualmente supportano, il lavoro si concentra nella progettazione ed implementazione di una libreria per la gestione dell'astrazione di un puntatore tridimensionale. Per risolvere il problema della deriva della posizione, dovuta al fatto che un accelerometro non può distinguere fra l'accererazione di gravità e quella di input, sono proposti due algoritmi: il primo è basato solo su un accelerometro triassiale ed è in grado di riconoscere in modo esclusivo il movimento lineare su tre assi o la rotazione su due. Il secondo, combinando l'input di un accelerometro e di un giroscopio, può riconoscere i movimenti lineari e le rotazioni su tre assi allo stesso tempo. L'astrazione è stata provata in un ambiente tridimensionale dove l'utente può muovere e ruotare il puntatore, registrando ed analizzando i dati del movimento. In conclusione sono discusse le possibili applicazioni in un sistema a finestre e gli sviluppi futuri.

Contents

List of Figures

Introduction and motivation

Introduction

This thesis discusses the development of a three-dimensional mouse library, using accelerometers as main input sensors. It is also considered the introduction of a gyroscope in order to enhance their motion tracking capabilities.

A mono-axial accelerometer is a device able to recognize the acceleration on one axis. Due to the fact that it is internally built using a damped mass attached to an elastic component (i.e. a spring), the acceleration is proportional to the difference between the current and the rest position of the mass. As a result, the accelerometer senses the gravity when it is motionless.

A gyroscope exploits the conservation of the angular momentum for sensing the rate of the rotations about one or two axis.

Many MEMS *(Micro Electro-Mechanical Systems)* implementations of these sensors are available on the market, and they found many applications in different fields. However the currently increasing usage of accelerometers is on mobile devices: many smartphones and *PDA*s (*Personal Data Assistant*) are equipped with them.

Due to this fact, many studies have been carried out in gesture recognition algorithms using these devices, mainly based on statistical models rather than on the physical meaning of sensor output. The motivation is the integration process drift that affects the resulting position and the inability in distinguish the gravity from the input acceleration when the device is moving and rotating.

A mouse 3D prototype has been developed using the *Wiimote*[1] controller for the *Nintendo Wii* game console, which has the advantage of being equipped with a three-axis accelerometer and has also the possibility to be expanded with a gyroscope (the *Wii Motion Plus*[2]). Furthermore this device is really cheap compared to *PDA*, and smartphones and is designed specifically to be used as a gesture recognition controller.

The library for supporting the 3D mouse abstraction has been designed in order to be device independent, introducing an intermediate abstraction layer which allows having a homogeneous representation of the input data for many appliances.

The 3D pointer abstraction contains information about the button pressed, the current position as a three-dimensional point and the rotation angle about the three axes.

[1]http://www.nintendo.it/
[2]http://www.nintendo.it/

Two motion tracking algorithm are introduced, which propose a novel approach in order to exploit the physical meaning of the sensed data. The first one uses only the accelerometer and is able to recognize the motion on three or the rotation about two axes. Such movements cannot be recognized at the same time, due to the fact that the tilt recognition is performed exploiting the gravity sensing, which can be considered only if the device is motionless. The criterion for separating the two motion types represents the novelty with respect to previous works. It exploits the acceleration module: if its value is close to the mean gravity acceleration, the device is considered motionless.

The second algorithm uses also gyroscopes in order to recognize the device rotations. Due to the fact that the data for each type of movement comes from a different sensor, it is possible to recognize rotations and linear motions at the same time.

The test environment, developed using the *Windows Presentation Foundation* technology, is able to show the movements of an arrow in a three dimensional space, which represents the mouse pointer: it changes its position according to the linear motions and its orientation according to the rotation. Graphs for data visualization are displayed together with together with the current direction of the gravity acceleration, in order to have feedbacks on the algorithm results. The program is also able to save the sensed input in order to perform various tests on the same data.

The thesis is structured as follows: chapter 1 describes in details the sensors and their applications in related works, chapter 2 is about the selected hardware and its properties, chapter 3 highlights the library design principles, chapter 4 shows how the concepts are lowered into the implementation and discusses the details about the two gesture recognition algorithms, chapter 5 demonstrates the test environment, finally chapter 6 is about conclusions and future works.

Motivation

Through the years of my studies in Computer Science, I gradually found out that the thing I like most in this discipline is trying to create programs which can simplify the way people works. Especially in later years I concentrated on *Human Computer Interaction* field, and I was so excited by all the studies about novel communication channels between a human being and a machine.

Such enhancements in interaction are represented by novels *tangible user interfaces*, where is it possible to interact with real objects (cubes, spheres etc.) in order to provide input to the applications, or the *multi-touch* interaction which is now available with the Apple *iPhone*[3] and through wider screens, as in Microsoft *Surface*[4].

A real revolution on the interaction in later years has been the introduction of the *Wiimote*[5] controller for the Nintendo *Wii* game console. Using a combination of accelerometers and an infra-red camera, the game interaction is much more natural, because the player controls his avatar performing

[3]http://store.apple.com/

[4]http://www.microsoft.com/surface/

[5]http://www.nintendo.it/

body motions rather than using a regular controller. This opened a new great market for video games: also people who never considered amusing staying in front of a screen pushing buttons, tried this modality and found it really intuitive.

This change has not yet arrived for desktop computers, though that different attempts have been made in order to overcome the standard interaction techniques.

With my work I tried to deal with the problems in creating an interaction abstraction for developers: the success of an input device is really tied to the simplicity of both development libraries and interaction capabilities. The main focus of my thesis, and I hope of my future work, is trying to define such models in order to have more friendly interfaces.

1 Inertial Measurement Sensors: technologies and applications

This chapter discusses the manufacturing and the properties of the sensors exploited for the creation of the three-dimensional mouse library. After that an overview of their application and of the related work in gesture recognition is provided.

1.1 Structure and properties

This section will introduce the structure and the functioning of two types of *Inertial Measurement Sensors (IMU)*: the mono-axial accelerometer and the gyroscope.

1.1.1 Mono-axial accelerometers

A mono-axial accelerometer is a sensor for the measurement or the recording of accelerations along one axis. Almost all sensors on the market have the same functioning principle: the behaviour of mass attached to a spring (or another elastic component), forced to move only along its extension direction. The mass is also damped in order to avoid oscillations.

When the inertial body senses an external force (i.e. gravity), the spring lengthens or shortens according to its direction, until the elastic force balances the external one. The sensor output is proportional to the spring displacement. As shown in figure 1.1, the mass is constrained to linear motions, so one accelerometer is needed for each dimension considered (i.e. for measuring a three dimensional acceleration, three accelerometers are needed).

If the linear motion direction is not perpendicular to gravity, the accelerometer senses as input not only the acceleration to be measured, but also the component of gravity which is parallel to its motion. If this component is constant or it is possible to establish its variations during the measurement, it must be subtracted from the output. Otherwise it is impossible to distinguish it from the real input.

The mathematical model can be defined combining Hooke's law and Newton's second law of motion[1]:

$$mc = -kx - b\frac{dx}{dt}, \tag{1.1}$$

where c is the acceleration sensed by the mass, x the displacement measured

[1] for more information, please see [SJ05] p. 190 and 116

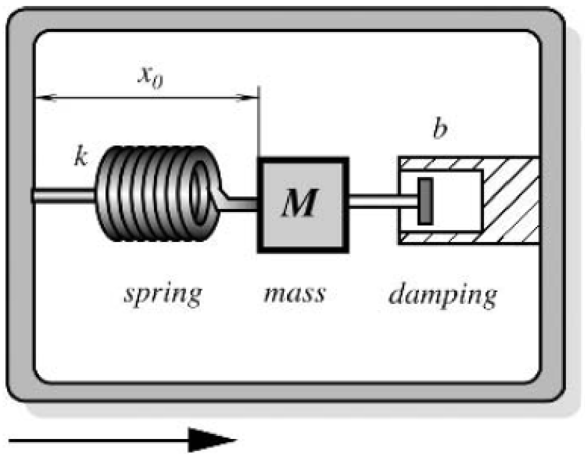

Figure 1.1: Model of an accelerometer

on the spring, m the hanging mass, k the elastic constant of the spring and b the damping coefficient. The acceleration c can be expressed using two components: the spring reaction and the input of the sensor:

$$c = \frac{d^2x}{dt^2} - \frac{d^2y}{dt^2}, \tag{1.2}$$

and substituting it in 1.1 gives

$$m\frac{d^2x}{dt^2} + b\frac{dx}{dt} + kx = m\frac{d^2y}{dt^2}. \tag{1.3}$$

The differential equation 1.3 represents the mathematical model for the accelerometer. It is of a second order, which means that the selection of a damping coefficient which yields to a critically damped state is important in order to avoid oscillations.

Many MEMS *(Micro Electro-Mechanical Systems)* implementations of the mono-axial accelerometer exist: the detection of the mass displacement with respect to the accelerometer case can be done with many transducers capable to sense microscopic movement or linear accelerations. The following is a short summary of the main types, for a more detailed description see [Fra04]:

- *Capacitive Accelerometers* contain a plate connected to the case (which is stationary) and another one attached to the inertial mass, which is free to move inside the housing. The two plates form a capacitor modulated by their distance. A circuit creates a signal proportional to its capacitance.

- *Piezoresistive Accelerometers* incorporate strain gauges for measuring the strain of the mass supporting spring, which can be correlated with its displacement.

- *Piezoeletric Accelerometers* exploit the ability of some crystals and certain ceramics to produce an electric potential when compressed. In these models, the piezoeletric crystal is both the sensor and the elastic element. When acceleration occurs, the mass compresses the crystal which generates directly the output signal.

1.1.2 Gyroscopes

A gyroscope is a rotating device which can keep constant the direction of its spin axis, exploiting the fundamental principle of the conservation of angular momentum, as defined in [SJ05]:

the total angular momentum of a system is conserved if the net external torque acting on the system is zero[2].

The mechanical version of a gyroscope is composed by a massive toroidal rotor (see figure 1.2), wrapped with a framework free to rotate about one or two axis.

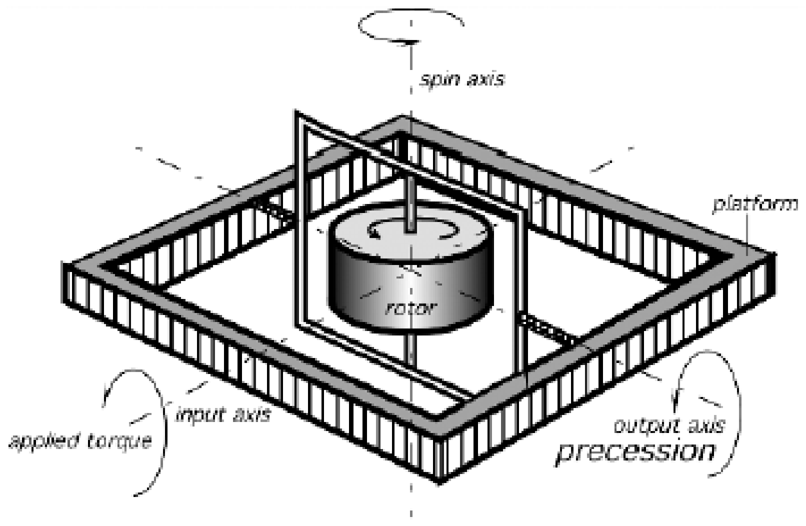

Figure 1.2: Single degree of freedom mechanical gyroscope

Such device has two important properties that can be exploited for measuring the torque[3]:

1. without the action of external forces, the spin axis of a gyroscope will remain fixed with respect to the space;

[2][SJ05] SERWAY R.A., JEWETT J.W. Principles of physics: a calculus-based text *Thomson Learning* 2005 316.

[3]for more information, please see [SJ05] p. 350.

2. if the gyroscope platform rotates about a plane perpendicular to its spin axis (the *input* axis in figure 1.2), it will turn the spin axis around a third axis, perpendicular with respect to the plane defined by the others (the *output* axis is figure 1.2). This phenomenon is called *precession*.

Those two properties and the conservation of angular momentum state that the rate of rotation of the spin axis about the output axis is proportional to the applied torque:

$$T = I\omega\Omega, \tag{1.4}$$

where T is the input torque, I the gyroscope inertia, ω the angular speed of the rotor (supposed to be fixed) and Ω is the angular rate about the output axis.

These type of gyroscopes are not suitable for building low cost devices: rotors, motors, support bearing, etc. limit the possibility to reduce the dimension of the device and the construction is very difficult. Thus other *MEMS* technologies have been developed and the most common implementations are based on a vibrating element, which replaces the rotor.

The functioning principle of such devices relies on the Coriolis acceleration, an apparent acceleration caused by the change of radial position of an object in a rotating reference frame[4]. The device is equipped with a suspended mass moving with a simple linear harmonic motion. When the support of the mass is rotated around a direction perpendicular to the harmonic motion, the sensor detects the movement of the mass caused by the

[4]for more information, please see [SJ05] p. 159

Coriolis acceleration, which is proportional to the rate of turn[5]

The three main types of *MEMS* vibrating gyroscopes are described in [TW04] and summarized by figure 1.3:

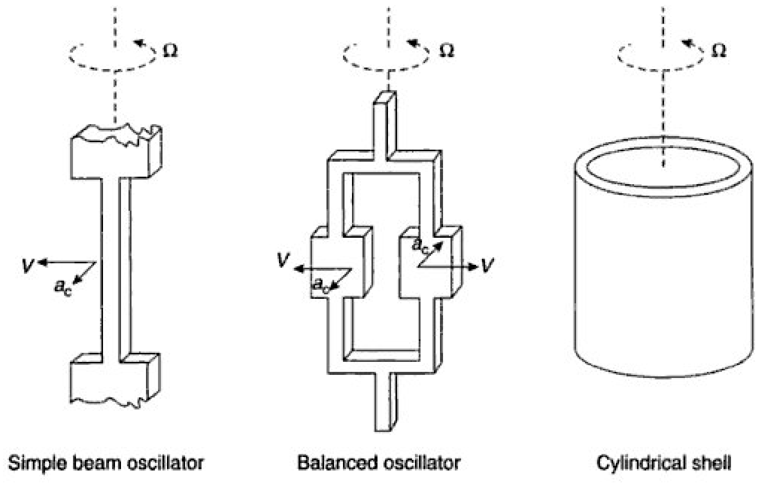

Figure 1.3: MEMS gyroscope categories

1. *Simple oscillators.* They are oscillators that can be modelled as a single vibrating mass. This type of gyroscopes has problem of mechanical asymmetry which makes the sensor sensitive to external vibrations.

2. *Balanced oscillators.* A pair of test masses is driven to resonate and the displacement from the plane of oscillation is measured to produce the rotation rate. Such devices overcome the problems of simple oscillators.

3. *Shell resonators.* The mass has a form of cylinder, ring or hemisphere which is symmetric about the axis of rotation.

[5]for more information, please see [Fra04] p.316

1.2 Applications

Accelerometers and gyroscopes have a plenty of applications, a good summary can be found at [AA.c].

1.2.1 Accelerometers

In engineering, the accelerometers are used for sensing the vibration on cars, machine health surveillance[6] (rotating equipment, gear failure etc.) and building monitoring[7](in particular for safety, i.e. seismic activity). The inclusion of accelerometers in notebooks allowed the creation of a *Quake-Catcher Network*[8]: a distributed and low cost network for earthquakes studying and monitoring.

In Biological Sciences high frequency acceleration recording is used in order to study the animals when they are out of sight, collecting information about the expending energy in the wild[9] [10].

Such devices found application also in medicine, for example in sport watches for runners in order to determine their speed and evaluate the dis-

[6][Boy03] BOYES W. Instrumentation Reference Book *Elseiver Science* 2003

[7][Saa06] SAAR O. S. Dynamics in the Practice of Structural Design *WIT Press* 2006.

[8]http://qcn.stanford.edu/

[9] [TTR$^+$05] SUGA et al. Method for underwater measurement of the auditory brainstem responce of fish *Fisher Science* 71 (5), 2005, 1115-1119.

[10] [WWQ$^+$06] WILSON et al. Moving towards acceleration for estimates of activity-specific metabolic rate in free-living animals: the case of the cormorant *Journal of Animal Ecology* 5 (75), 2006, 1081–1090.

tance covered(produced by Nike[11], Polar[12], ect.), or as steps counter to encourage people to walk, for instance in Belgium.

In *Inertial Navigation Systems* accelerometers are used as helper sensors (i.e. used when no other system is available): also if used in combination with gyroscopes, they have proved to be unsuitable for this purpose because of the error growth in the integration process through the time (see [TP05]).

Certainly the raise of interest for such devices is derived by the incorporation into personal appliances: smartphones and PDAs (*Personal Data Assistant*) includes accelerometers for different purposes, such as switch between portrait and landscape screen mode, and for creating fancy interaction in video-games (tilt or shake). The following is a brief list of manufactures which includes accelerometers in their devices:

- Apple (iPhone, iPod touch and iPod nano 4G)[13]
- HTC (Dream, Hero, Touch Pro, Touch Pro 2, Touch Diamond, Touch Diamond 2 etc.)[14]
- Nokia (5500, N95 and N82)[15]
- Samsung (Omnia, Omnia HD and GT-I7110)[16]

[11]http://www.nike.com/
[12]http://www.polaritalia.it/it/
[13]http://store.apple.com/
[14]http://www.htc.com
[15]http://www.nokia.com/
[16]http://www.samsung.com

- Sony-Ericsson (G705, W595, W760, W910, K850i etc.)[17]

Apple and Lenovo[18] laptops include accelerometers to detect drops in order to avoid hard disk data loss.

Another field where the accelerometers are recently used is the entertainment. Modern consoles have controllers equipped with accelerometers for gesture-recognition, like Nintendo's *Wiimote*[19] (for Nintendo Wii) or the Sony *DualShock 3* (for PlayStation 3)[20]. The *Wiimote* controller features will be discussed more in detail in section 2.2.

1.2.2 Gyroscopes

Gyroscope sensors have been used in spacecraft orientation. For example the Cassini-Huygens in 1997 has been equipped with small hemispherical resonator gyroscopes, as described in [LG05].

In automotive systems, the first micro-machined yaw rate sensor was introduced by [LGG+97] in 1997, for the *Electronic Stability Programs* (ESP). It evolved through three generations of sensors, which allow vehicle stabilization *(Electronic Active Steering)*, roll-over mitigation *(ROM)*, suspension control (*Active Suspension Control*) and also the first tries to remove as many mechanical components as possible from the steering mechanism of cars *(Steer by Wire)*. A further discussion on the structure of the sensors

[17]http://www.sonyericsson.com/
[18]http://www.lenovo.com/
[19]http://www.nintendo.it/
[20]http://it.playstation.com/

employed can be found in [NGK+07].

Gyroscopes are beginning to be used also in entertainment: Nintendo created an expansion for its Wiimote control, called WiimotionPlus[21] , which uses *MEMS* gyroscopes for enhancing its motion sensing capabilities. This device will be analysed more in detail in section 2.3.

Such devices are applied also in image stabilization systems for video or photo cameras, and also for managing the tail rotor of radio-controlled helicopters.

1.2.3 Gesture recognition

The spread of mobile devices equipped with *MEMS* accelerometers caused the raise of studies on gesture recognition in *Human Computer Interaction* research area. The possibility to have such wireless equipments opens new possibilities for interaction in many applications: for musical input, in home environments, in augmented reality, in palmtop interfaces and also for the classic interaction with computers.

Many experiments have been carried out, especially with only a three axial accelerometer (which is the most common sensor included in mobile devices). The following is a short summary of the state of the art in this area.

In [SH97] is presented a *MIDI (Musical Instrument Digital Interface)* controller based on a set of ten gestures. The input was collected using an acce-

[21]http://www.nintendo.it/

lerometer based system. A similar application can be found in [MP97], where the baton for the orchestra in *Brain Opera*'s installations (where untrained people interacted intuitively with the music) is implemented through inertial components, still for managing *MIDI* controllers. The baton was composed by five pressure-sensitive resistors condensed into the stick, a three axial accelerometer in the base and an infra-red LED on the top. The resistors measured the hand pressure, the accelerometers sensed drag movements and beats, while the infra-red emission was received by a photo-diode sensitive only to the signal radiated by the baton, which determines its horizontal and vertical position. The data was sent to an host computer for processing.

Early works in palmtop interfaces started with the design of *graspable displays.* In [SI97] was proposed a prototype of a virtual newspaper, able to slide down the text if the display was tilted around its base. In [Bar00] was introduced the Itsy research prototype from Compaq, a small PDA equipped with a three axial accelerometer. The author designed the *Rock 'n' scroll* user interface and tested it with a photo album, controlled by device movements. Tilt was associated to scrolling functions, fanning gesture were employed for viewing the next or the previous photo, while holding the device on a position changed the screen orientation from landscape to portrait and vice versa. Event thought that the approach leads to great possibilities for PDA interfaces, some problems rose in separating the old interaction techniques from the new ones: for instance during a test with a video game, it was impossible to distinguish movements of the device caused by an enthusiastic button press from the tilt gesture.

The design of a new device able to perform inertial measurement is demonstrated in [BP01]. The instrument is a 3.18 centimetres side large cube, equipped with three single axis gyroscopes and two two-axial accelerometers, able to recognize basic gestures (such as line motion an twist) on six axes. The authors made the choice of having a recognition algorithm implemented on board, which consists of two steps. The first searches for peaks in a reasonable timing window, in order to identify the portion of the signal where a motion may be present, while the second determines if the gesture is present and which type can be identified in each window. They created also an abstraction of the device capabilities for developers, in order to create applications on top of it.

Another experiment in dynamic gesture recognition for mobile devices can be found in [MMST00]. Here is presented an approach for the recognition of static (display up or down, phone on left or right ear) and dynamic acceleration (watch the caller number when alarming, answer a call etc.). For static phone gesture recognition is suggested the exploitation of *self-organizing maps of Kohonen*[22] , an artificial neural network able to form spatially organized feature maps from a n-dimensional input, with unsupervised learning (the machine receives only the input, without a given categorization of the target outputs). Such networks are usually exploited for feature extraction and pattern recognition. It is possible to extract how the mobile is used training them with normal usage acceleration signals. For re-

[22]for further information please see [Gun97] p. 123

cognizing dynamic accelerations, the application of a *Hidden Markov Model*[23] is proposed, a statistical representation of the random evolution of a system, supposed to be memoryless. The adjective hidden is due to the fact that the state of the model is not directly observable (in this case the machine cannot directly observe the gesture), but it is visible only the output dependent on it (the accelerometer data). The training process allows building the probability set which connects the observable states with the unobservable ones. With those techniques, the recognition accuracy for static acceleration was between 89.4 and 100 percent, while for dynamic acceleration was between 88 and 100 percent. The dynamic motion recognition had problems tied to the shape of the phone and the physical traits of users. In other research works by the same group, the *Hidden Markow Model* training was refined using gesture duplicates perturbed by Gaussian noise, obtaining a recognition accuracy of 98 percent for a DVD player gesturing interface in [MKKK04] and for a video recorder in [KKM$^+$06].

A similar application of *Hidden Markow Models* was also employed by [SPHB08], using a Nintendo *Wiimote*[24] controller as input device. The framework developed in this work allows the user to train the system with any kind of gesture and, after that, the software can distinguish among the gestures recorded. They performed also an evaluation on five gesture types (square, circle, roll, Z, tennis smash) and obtained an averaged recognition rate of 90 percent.

[23]for further information please see [Alp04] p. 305
[24]http://www.nintendo.it/

Late works tried to reach a higher recognition rate, going beyond the Hidden Markow Models. In [WPZ$^+$09] is described a novel method for accelerometer-based gesture recognition called *FDSVM (Frame Based and multi-class Support Vector Machine)* . This approach firstly collects the gesture data and represents it using a frame-based descriptor, and then a multi-class gesture classifier based on a supervised learning method called *Support Vector Machine*[25] is used in order to recognize gesture features. The algorithm was claimed to have a higher recognition rate compared with other approaches.

[25]for further information please see [Alp04] p. 218

2 Prototype Hardware

In order to develop the three dimensional mouse library, a prototype of the input device is needed. This chapter describes the hardware employed for its realization and the selection criteria.

2.1 Selection criteria

In late years, various devices equipped with accelerometers were introduced on the market, as already discussed in section 1.2. Most of them are *PDA*s or smartphones from different manufacturers, with different shapes and capabilities. For this reason the choice was to select an existing device rather than build a new one.

The following requirements were considered for its selection:

R.2.1 *Availability of a three-axial accelerometer sensor.*

The device has to be equipped with sensor able to recognize the acceleration on three dimensions.

R.2.2 *Availability of a three-axial gyroscope sensor.*

The device has to be equipped or expanded with a sensor able to recognize the angular rate on three dimensions.

R.2.3 *Availability of wireless communication capabilities.*

The device has to be able to send the sensed data to a desktop computer without a wired connection. This requirement has two purposes: the first is ensuring the possibility to establish a communication between the device and the PC, the latter is having an instrument not limited in its movement by a connection wire.

R.2.4 *Availability of code libraries for reading the output of sensors.*

The sensors output reading has to be possible, either with software running on the device or on the connected computer.

R.2.5 *Cost.*

The device should be cheap in order to be affordable to as many people as possible.

R.2.6 *Ergonomics*

The device should be comfortable in use.

*PDA*s and smartphones allow the development of software which runs on the device. This leads the possibility to define a standard data format for the output of different sensors. However this is not possible for equipments designed only as input devices, such as the Nintendo *Wiimote*[1], and it is

[1]http://www.nintendo.it/

unlikely to suppose to have this feature in all devices.

The *Wiimote* is the device closest to the idea of three dimensional mouse. Reviewing all the requirements in detail, its usage has no drawbacks for the prototype creation:

R.2.1 It is equipped with a three-axial accelerometer (see section 2.2).

R.2.2 It is possible to expand it with a three-axial gyroscope (see section 2.3)

R.2.3 It is possible to connect it to a personal computer via a *Bluetooth*[2] connection.

R.2.4 The data format of the accelerometer and gyroscope output has been reverse-engineered. Code-libraries for reading it are available.

R.2.5 The Wiimote is cheaper than every *PDA* or a smartphone: it costs about 30€, while the WiimotionPlus costs about 20€.

R.2.6 It is the only device designed for sensing hand motions, so it is really comfortable to use.

The Nintendo's controller has also other hardware for gesture recognition (i.e. the infra-red camera) but, in order to maintain the compatibility with the other devices, it is not exploited in the prototype development. Thus, the Nintendo solution for motion tracking, based on the combination of a triangulation on the infra-red camera output and the accelerometer data,

[2] *Bluetooth* is an industry standard for *Wireless Personal Area Network*s *(WPAN)*. For more information please see [AA.a]

cannot be used. The following sections of this chapter describe more in detail the *Wiimote* and *Wii Motion Plus* features.

2.2 Wiimote description

The *Wii Remote*, unofficially called *Wiimote*, is the main controller of the Nintendo *Wii* game console[3]. Its particular design can be seen in figure 2.1: it has the shape of a remote controller, usable with one hand, rather than the classic two-handed form of a game controller.

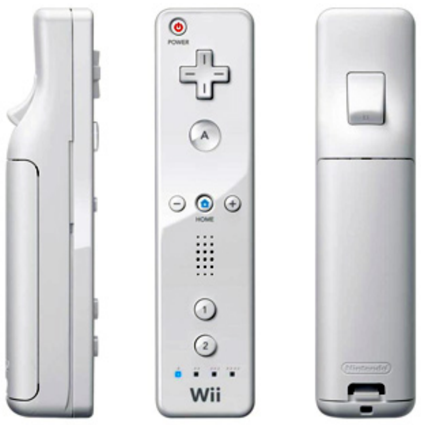

Figure 2.1: *Wiimote* controller

The communication with the console is based on a *Broadcom BCM2042*[4] Bluetooth chip. It works also with a personal computer, because it relies on the *Bluetooth Human Interface Devices* standard from [AA.a], a protocol for

[3]http://www.nintendo.it/

[4]http://www.broadcom.com/

the support of devices such as joysticks, mouses and keyboards, maintained by [AA.a]. It has a wide support in operating systems, due to its derivation from the *USB (Universal Serial Bus) Human Interface Device* protocol. Its maximum report frequency is of $100Hz$.

The controller contains an *Analog Devices ADXL330*[5] three-axial capacitive accelerometer, located on the top surface of the main circuit, near the A button (see figure 2.1). This sensor can measure accelerations in a $\pm 3g$ range[6], with a sensitivity of 10%. The orientation of the axis with respect to the device is shown in figure 2.2.

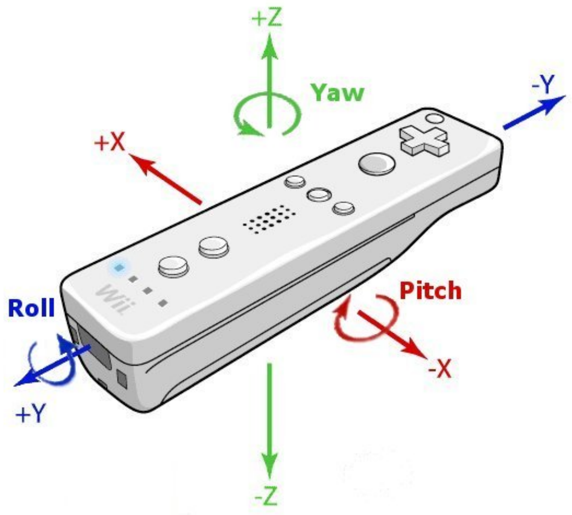

Figure 2.2: *Wiimote* accelerometer axis orientation

The *Wiimote* is equipped with a 128×96 monochrome camera, which has

[5]http://www.analog.com/

[6]g is the standard symbol for the gravity acceleration, approximately $9.8ms^{-2}$

built-in image processing capabilities. Looking through the infra-red filter in the remote casing top, it can track up to four moving objects. This camera, used together with the *Sensor Bar* (a crosspiece with two infra-red clusters at each end), is exploited in *Wii* games for pointing objects displayed on the screen (buttons, menus etc.) via a triangulation algorithm.

The device has a proprietary port on its bottom, where various controller extensions *(Nunchuck, Classic Controller, Guitar Hero Guitar, Guitar Hero Drums, Balance Board, Wii Motion Plus)* can be plugged.

This and other technical information on the *Wiimote* can be found at [AA.b].

2.3 Wiimotion Plus description

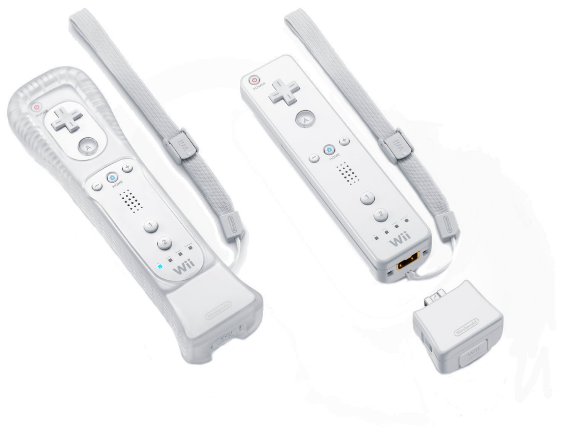

Figure 2.3: *Wiimote* with the *Wii Motion Plus* extension

The *Wii Motion Plus* is an extension for the *Wiimote* controller (see figure 2.3), equipped with two vibrating gyroscopes: the dual-axis *InvenSense IDG-600*[7] and the single-axis *EPSON TOYOCOM X3500W*[8]. The first one senses the angular rate about the X and Y axis (*pitch* and *roll*), while the second sensor recognizes it about the Z axis (*yaw*), as shown in figure 2.2. Their sensitivity is on the order of 1%.

[7]http://www.invensense.com/

[8]http://www.epsontoyocom.co.jp/

3 Mouse 3D library design

This chapter discusses the design of the *Mouse 3D* library. The first part describes the requirements elicitation and formalization process, while the second and the third deeply analyse the conceptual modelling of the classes within the library.

3.1 Requirements

3.1.1 Motivation

The project objective is the creation of an abstraction able to represent a three-dimensional pointer over the space. The definition of a set of requirements is needed in order to define success criteria for the software library development.

The requirements elicitation process has started from the observation of interaction through the ordinary two-dimensional mouse, and from the analysis of the common window tool kit development *API*s *(Application Programming Interfaces)*. The identification of mouse distinctive properties has been performed relying on the author's usage experience. The most important aspects can be summarized as follows:

- A mouse has usually three buttons and a scroll wheel. The buttons are exploited for selecting objects which currently contains the pointer.
- The physical device motion is not directly mapped to the pointer, but it is scaled in order to increase or decrease its sensitivity.
- When the device is lifted and its position changes, the movement are not sensed by the window system. This fact means that is possible to "turn off" the mouse motion-tracking process when needed.
- The screen pointer responds immediately to user inputs, without waiting for the end of a gesture. Some interaction techniques (i.e. drag and drop) exploit this feature for direct object manipulation.

In order to compare the approaches for mouse input management, an analysis of various windows tool kit as been performed as well. The considered frameworks were *Microsoft Windows Forms*[1] and *Windows Presentation Foundation*[2], *Java Swing*[3] and the *HTML 4.0 DOM*[4] The *API*s are quite similar to each other and they offers at least the following data:

[1]for more information, please see [Mac05] p. 67
[2]for more information, please see [Mac08] p. 171
[3]for more information, please see [Zuk05] p.23
[4]for more information, please see [Fla06] p.389

- A two dimensional point which corresponds to the mouse pointer relative position with respect to the target widget reference system (usually it is the distance between the top left corner of the widget and the pointer position).

- Information about the button clicked and the number of clicks.

- Information about wheel scroll amount.

In all considered tool kits, the occurrence of a mouse input is notified using the *observer* pattern from [GHJV95]. The common modelled events are:

- *Mouse move.* It occurs in case of a mouse motion. Usually the notification contains the information about the current position.

- *Mouse down.* It occurs when a mouse button has been pressed. The notification contains information in order to identify the button.

- *Mouse up.* It occurs when a button has been released. The notification contains information in order to identify the button.

- *Mouse over.* It occurs when the mouse pointer is over the widget area (sometimes it is called also *Mouse enter*).

- *Mouse out.* It occurs when the mouse pointer has just leaved the widget area (sometimes it is called also *Mouse exit*).

In order to support such notifications, it is sufficient to represent the pointer abstraction with its absolute position in a three dimensional space, reporting also button-related information. The calculation of the widget-relative mouse position, the mouse-over or mouse-out notifications etc. is performed by the window system and do not belong to the pointer representation.

Another trivial requirement, which is also the main hardware selection criteria as described in section 2.1, is that the user input has to be sensed using accelerometers and gyroscopes. The prototype should be exploited in order to find out the possibilities and limitations in employing accelerometers for creating a three dimensional mouse, with or without coupling them with gyroscopes.

3.1.2 Definition

The following is a formalization of the requirements for the 3D mouse library.

R.3.1 *Definition of gestures using a 3D accelerometer.*

The library realization must have a set of recognizable gestures, exploiting only the data coming from a three-axial accelerometer.

R.3.2 *Definition of gestures using a combination of a 3D accelerometer and a 3D gyroscope.*

The library realization must have a set of recognizable gestures exploiting a combination of a 3D accelerometer and a 3D gyroscope.

R.3.3 *Definition of an abstract 3D mouse pointer.*

The library realization must define a consistent abstraction of a 3D mouse pointer. It must include at least its position in space and information about pressed or released buttons.

R.3.4 *Sensitivity control.*

It must be possible to modify the device sensitivity with respect to its motion.

R.3.5 *Device independence.*

Even though that the prototype will be created using a specific device, the library must be designed in order to easily support the integration of other similar appliances.

R.3.6 *Status change notification.*

The library must have an interface in order to allow the notification of the pointer abstraction status changes.

R.3.7 *Real-time response.*

The user must not notice delays on gestures elaboration. It should be possible for an application built on top of the 3D mouse abstraction to give feedback also during the gesture, according to the current pointer status.

R.3.8 *Stop tracking.*

It must be possible to stop the gesture tracking in order to move the device without changing the pointer state.

3.2 Components

After the requirements elicitation process, the library design continued with the definition of its components and their relations.

The structure of the identified components is shown in figure 3.1. The analysis of the accelerometer data and the gesture recognition is done by the *Mouse3D* component, which contains all the classes for the mouse state management. It is responsible for the gesture recognition and for the definition of all details regarding the pointer abstraction provided: the state representation, the possible input gestures, its sensitivity and its change notification mechanism. This component is the core of the library and all the requirements will be evaluated against it.

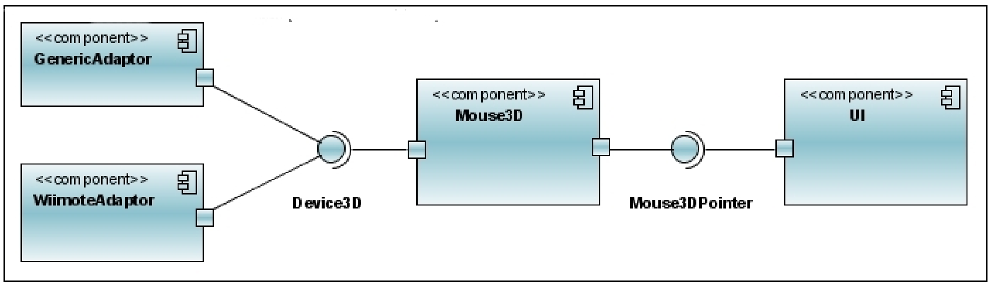

Figure 3.1: Mouse 3D library components

The *Mouse3D* component relies on an interface, the *Device3D*, as its generic input provider. If one specific device has to be supported, the data must be exchanged through this interface definition, in order to let the component to be agnostic of the data reading details, according to the requirement **R.3.5**.

All components which provide the *Device3D* interface are responsible to deal with the specific data representation of a particular device, separating the concerns of data reading and analysis: the generic adaptor component decodes the data from the specific device-dependent format and encodes it according to the *Device3D* interface, while the *Mouse3D* component analyses it in order to produce changes on the pointer status.

In Figure 3.1 is shown the *WiimoteAdaptor*, which is tied to the *Wiimote* data reading. The *GenericAdaptor* component represents all other possible adaptor for similar appliances.

The *Mouse3D* component provides also the *Mouse3DPointer* interface, exploited by a generic user interface (represented by the *UI* component in figure 3.1) in order to receive notifications about pointer state changes.

3.3 Packages

3.3.1 Overview

The next step was the refinement of the component definition in software packages and classes.

The package structure reflects the components organization. The prototype consists of three packages: *WiimoteAdaptor*, *Mouse3D* and a *UI* (a test user interface for the library). This chapter does not focus on the last package, which is discussed later in chapter 5.

First of all, it was decided to include all the interfaces for inter-component data exchange into the *Mouse3D* package, in order to gather all the classes shared by the components in a single place. However, a *Mouse3D* cannot be used without a concrete device adaptor. This means that a working set of classes which correspond to the component interface cannot be completely defined in this package. For this reason the *Device3D* class must be abstract (it cannot be directly instantiated).

The *Mouse3DPointer* can be a concrete class indeed, since it does not depend on package-external inputs.

The following sections discuss in detail the class design for all packages.

3.3.2 ***WiimoteAdaptor*** package

The *WiimoteAdaptor* package contains the classes responsible to read data from the *Wiimote* and adapt it to the format expected from the *Mouse3D* component, hiding device dependent characteristics, in order to meet the requirement **R.3.5**. The involved classes for solving this problem are shown in figure 3.2.

The *Mouse3D* package contains the abstract *Device3D* class, which represents the generic device adaptor. It has operations for attaching and detaching *Device3DListener*s and an operation for notifying the occurrence of device state change. The latter is represented by the *DeviceState* class and contains the generic information provided by an three-axial accelerometer and gyroscope: the acceleration (*AccX, AccY* and *AccZ* attributes) and the angular rate (*Yaw, Pitch* and *Roll*) on three dimensions.

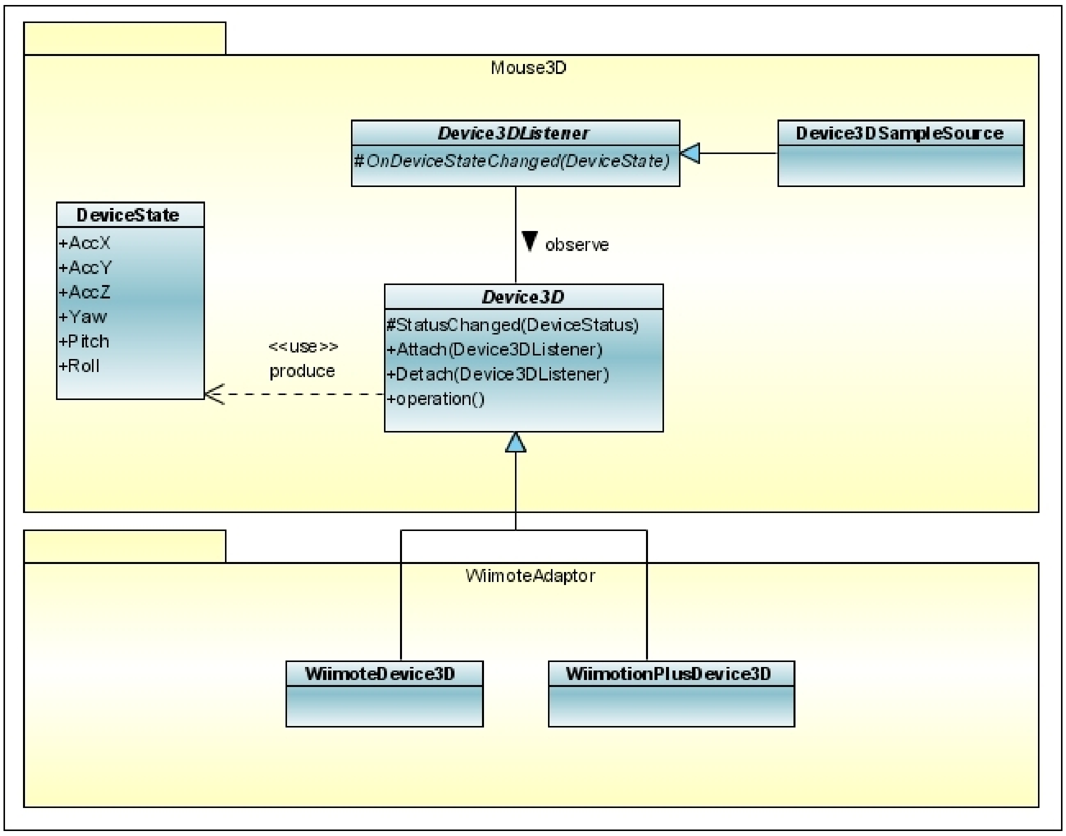

Figure 3.2: Class diagram of the *WiimoteAdaptor* component

When a change of state in the generic device occurs, the *Device3D* executes the *StatusChanged* operation. It contains the invocation of the *OnDeviceStateChanged* operation for all the attached *Device3DListener*s, which are the abstract observer of a *Device3D*.

The *WimoteAdaptor* package contains two concrete *Device3D*: the *WiimoteDevice3D* and the *WiimotionPlusDevice3D*. Both read data from the same device, but the first one adapts only the accelerometer related data, while the second one extract also the angular rate from the *Wii Motion Plus* extension.

3.3.3 *Mouse3D* package

The three dimensional pointer abstraction is contained in the *Mouse3D* package. Its structure is outlined by figure 3.3. The entire class design relies on polymorphism and on the observer pattern from [GHJV95], both for data reading and providing. The *Device3DSampleSource* class is a subtype of *De-*

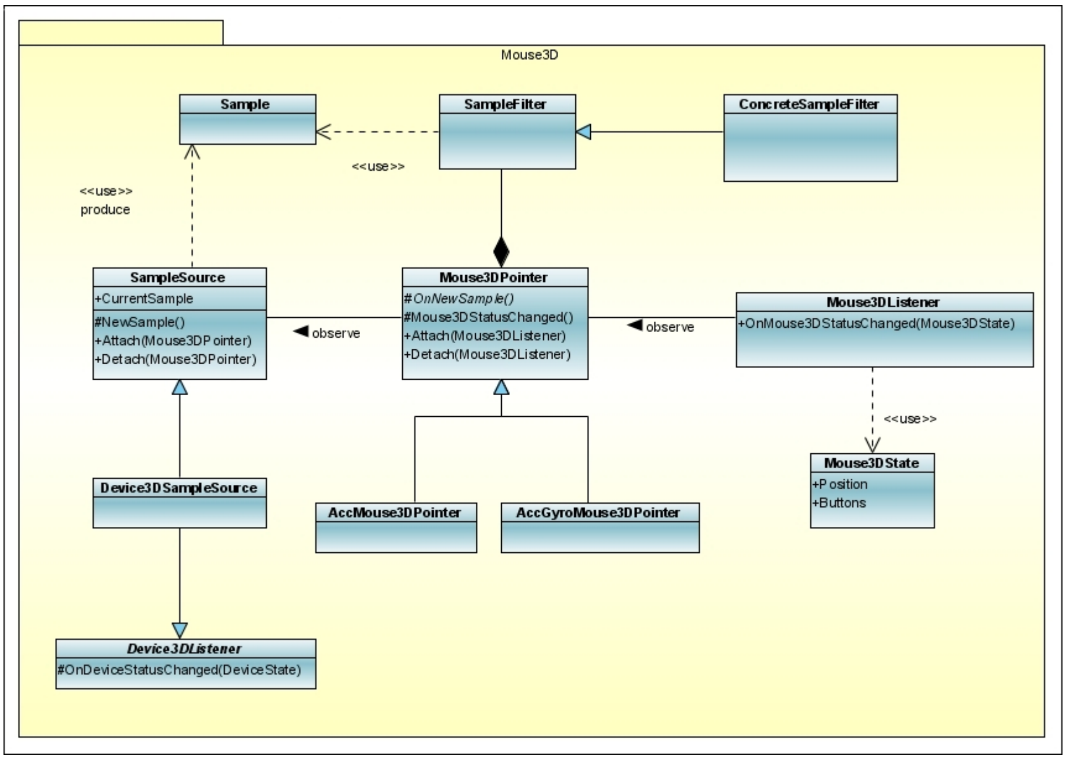

Figure 3.3: Class diagram of the *Mouse3D* component

vice3DListener, which is the abstract observer of a concrete *Device3D* (see section 3.3.2), the super-class of all accelerometer-enabled appliances. The *Device3DSampleSource* is also a subtype of the *SampleSource* abstract class, the generic data provider for the pointer abstraction. Such modelling distinguishes the pointer data providing concept from the generic device management, opening the possibility to attach data sources which are not directly tied to a device. This feature has a practical rather than theoretic motivation: the possibility to repeat different test on previous recorded data, emulating the real device.

The pointer is represented by the *Mouse3DPointer* abstract class, which is an observer of the generic data source *SampleSource*. The latter class contains the regular operations of the subject in the observer pattern. It can *Attach* or *Detach* a *Mouse3DPointer* and, through the *NewSample* operation, notify the arrival of a new *Sample*, which is the generic input of a *Mouse3DPointer*.

The pointer class is also a data source, observed by a generic abstract *Mouse3DListener*, which is the super-type of all classes interested on *Mouse-3DPointer* status changes (i.e. the user interfaces). The notification of such variations is performed by the *Mouse3DStatusChanged* operation of the *Mouse3DPointer* class, which consist of the invocation of the *OnMouse3D-StatusChanged* operation for all attached *Mouse3DListener*s (another observer pattern application).

The pointer state is represented by the *Mouse3DState* class which at design stage includes, according to the requirement **R.3.3**, at least the *Position* and the *Buttons* information.

It is worth pointing out that the data flow from the *Device3D* to the *Mouse3DListener* is a pipeline (it can be seen in figure 3.2 and 3.2 following the *observe* relations), where each subsequent stage may buffer some data. Thus, in order to meet the **R.3.7**, it is sufficient to make the condition that the time duration corresponding to buffed data must be lower than the gesture extent, at least of an order of magnitude, though that a movement least about one second.

The concrete classes which refine *Mouse3DPointer* are *AccMouse3DPointer* and *AccGyroMouse3DPointer*. The first is designed for recognizing gestures relying only on accelerometer data, while the second exploits the combination of both accelerometers and gyroscopes. Internally they contain a workflow based on concrete subtypes of *SampleFilter*. This class is a generalization of all basic analysis that can be done on *Sample* attributes, represented as a generic function, which has a *Sample* as parameter and as output. Their output may depend either only on a single *Sample* or on a certain number of samples received before. The modelling of this generic class allows defining reusable operations such as means, integration etc. and combining them quickly, in order to define the behaviour of the pointer abstraction. All the *SampleFilter*s implemented to create the prototype are described in section 4.3.

4 Mouse 3D library implementation

This chapter describes the implementation of the Mouse 3D library, developed for the *Microsoft .Net* framework in *C#*[1] for *Windows*.

4.1 Wiimote data reading

The *Wiimote*, as already explained in section 2.2, can communicate with a personal computer through a *Bluetooth* connection. First of all, the pairing[2] process must be completed. This operation must be performed using the operating system interface. After that, the *Wiimote* is listed as a *Human Interaction Device* and it is possible to open a handle for the device using the *HID and Device Management Win32 APIs*. Once this handle has been opened, the computer can start to send and receive *reports* (the communica-

[1]for more information, please see [Tro07]

[2]in the *Bluetooth* protocol, the pairing is the process of a relationship between two devices encrypting data using a shared secret. For more information, please see [AA.a]

tion unit in *HID*) from the *Wiimote*.

The report data format has been reversed engineered by [AA.b] and the *WiimoteLib*, a *.Net* library for the communication management is available at [Pea].

This library allows the developer to:

- Send a command to the device in order to specify which data is needed from the application. For instance, the developer can use only the accelerometer data, without caring about the infra-red camera output.
- Receive notifications (events) about the current output of the desired sensors and buttons.
- Receive notifications on the hardware plugged into the expansion port.

At the time of writing, the reading of the accelerometer data, the infra-red camera output and the notifications about the state of buttons work smoothly. Due to the fact that the *Wii Motion Plus* is a relatively new device, the support is not yet complete (some device behaviours have not been reversed engineered yet). However the gyroscope data is accessible and, also if the library is not stable, the prototype development has not been invalidated.

The *WiimoteAdaptor* package implementation is represented in figure 4.1.

The *WiimoteLib* provides the `Wiimote` class, which contains methods for initializing the device and its state. The state change notification mechanism is quite simple, since the *.Net* framework has a built-in support for the observer pattern: the `event`[3] keyword specifies a multicast `delegate`[4] which can be called only within the defining class (the event notification) and natively supports attaching and detaching operations. A `delegate` is an object representing a set of methods with the same signature. The same mechanism is used by the `Device3D` class in order to notify the state change.

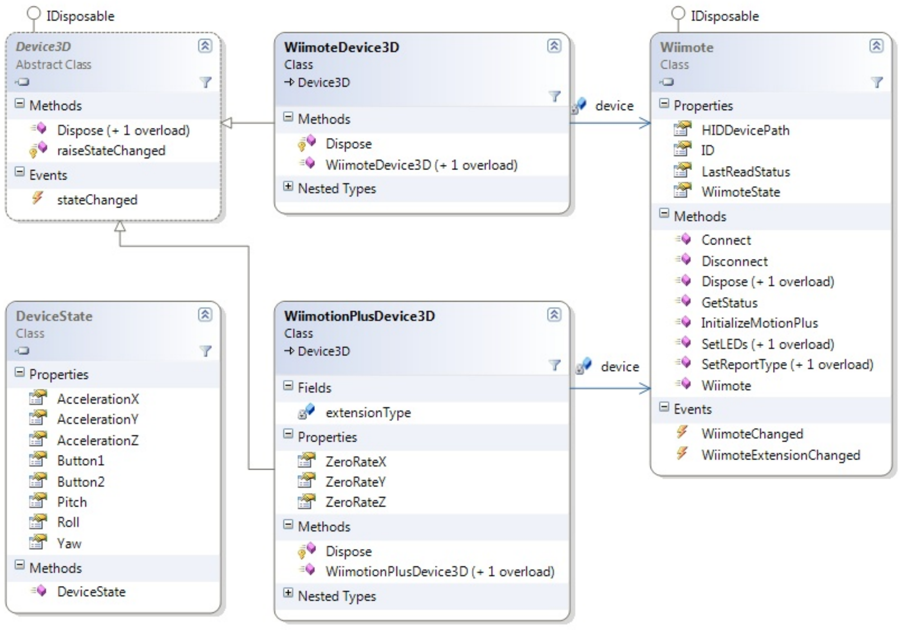

Figure 4.1: Implementation of the *WiimoteAdaptor* package

The `WiimoteDevice3D` receives the accelerometer raw data for each axis from the `Wiimote` class, represented as a byte (the value is between 0 and 255). In order to have more precise acceleration values, the class maintains calibration data such as the zero value (which is near to 127) and the factor

[3]for more information, please see [Tro07] p. 362
[4]for more information, please see [Tro07] p. 341

used to convert the sensor output into an acceleration in meters per second squared.

The device zero for one axis can be found positioning the device in a way that gravity is perpendicular to the axis.

The conversion factor can be indeed calculated simply meaning an adequate number of samples, when the device senses the gravity only along the considered axis. The mean has to be repeated two times, one for each direction. The result will be two mean values, m_{min} and m_{max}, where $m_{max} > m_{min}$. The conversion factor is

$$c = \frac{m_{max} - m_{min}}{2g}. \tag{4.1}$$

Once the raw acceleration is converted, a `DeviceState` object is filled with the conversion results, the buttons state and the time interval, calculated as a difference between the arrival time of the current and of the previous sample. After that, the object is used for the notification.

The `WiimotionPlusDevice3D` applies the same conversion for the accelerometer value, but it reads also the *Wii Motion Plus* extension output.

First of all, the gyroscope has to be plugged and the class must receive a new extension notification. After that, it can be initialized (sending a command in order to inform the *Wiimote* that the gyroscope data is needed), and finally the application can start reading the sensor output.

The angular rate on one axis is represented as an integer. Depending on the value of a “fast” flag for each axis, the output is different:

- *Slow mode* (fast flag set to zero). Measuring 20 on the raw data means turning about one degree per second.

- *Fast mode* (fast flag set to one). Measuring 20 on the raw data means turning about five degrees per second.

In order to convert the raw data to an angular rate in degrees per second, it is sufficient to divide by 20 in slow and by 4 in fast mode.

This sensor should be calibrated to find the zero rate value for each axis. It is enough meaning the gyroscope output values when it is motionless.

4.2 Mouse abstraction

This section discusses the implementation of the pointer abstraction. The details about the algorithms used for extracting the motion data are described in section 4.3.

The implementation structure for the package *Mouse3D* is shown in figure 4.2 and reflects the one discussed in section 3.3.3.

The C# `event`[5] property type, already presented in the previous section, allows having an implicit representation of the generic listener class for all observer patterns used in the model. The `Sample` class contains information about the acceleration and the torque sensed by the device. It maintains also information on the raw data measured by the `Device3D` class, in order

[5]for more information, please see [Tro07] p. 362

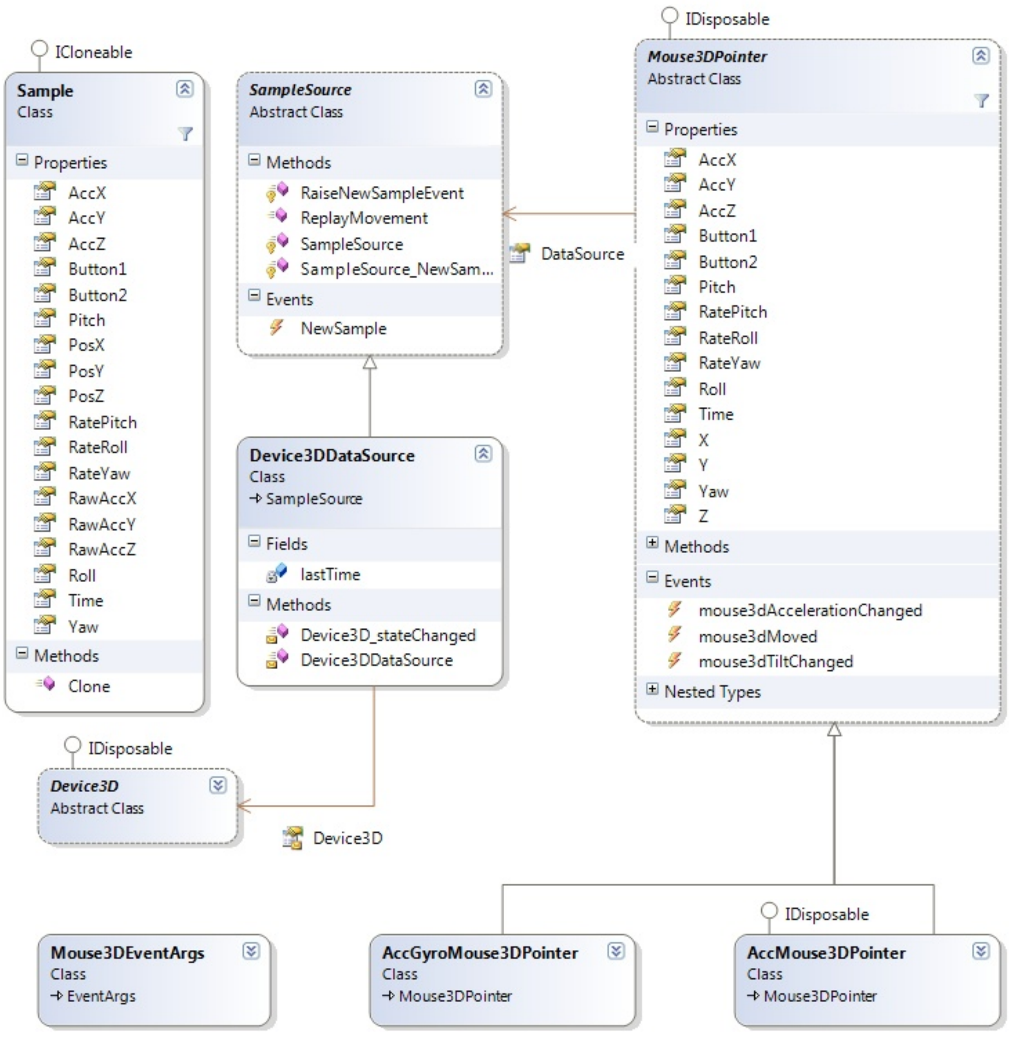

Figure 4.2: Implementation of the *Mouse3D* package

to allow repeating tests on the same input (see section 5.3). Such data should be removed when the prototyping phase is over.

The `Mouse3DPointer` abstract class properties describe the current state of the pointer abstraction: the acceleration, the rotation angle about the three axes and the pointer position. The acceleration is not really needed, but it is maintained for debugging purposes.

The state change notification is performed by the following events:

- `mouse3DMoved`. This event occurs when the pointer position changes.
- `mouse3DAccelerationChanged`. This event occurs when the pointer acceleration changes.
- `mouse3DTiltChanged`. This event occurs when the pointer has been rotated along one or more axes
- `mouse3DButtonPressed`. This event occurs when a button is pressed.
- `mouse3DButtonReleased` This event occurs when a button is released.

Each event is parametric with respect to the `Mouse3DEventArgs` class, which is a snapshot of pointer abstraction status at notification time.

The set of possible notifications has been selected according to the gesture recognition algorithm, discussed in section 4.3

4.3 Motion tracking algorithm

4.3.1 Interpreting data

Before entering into the proposed motion tracking algorithm details, it is worth pointing out which kind of information can be extracted from the acceleration data analysis.

First of all, for the implementation of the tracking algorithm it is not possible to use a buffer with the complete gesture data. This is due to the need of a real-time user feedback, as stated in the requirement **R.3.5**. For this reason the algorithm cannot wait until the gesture ends for recognizing it, which means that pattern recognition cannot be applied on the movement as a whole.

The proposed approach is more “physical” rather than “probabilistic”: it tries to extract in real time the physical meaning of the received data and consequently updates the pointer abstraction.

Usually the direct manipulation of the physically sensed acceleration is not considerable in motion tracking, for two main reasons: the first is the error growth in the double integration process through the time, while the second is the inability to separate the input and the gravity acceleration from the sensed one, when the device is translated and rotated at the same time.

However these points have a lower influence than usual on the creation of a mouse pointer abstraction. The motion duration is very short and the integration error should remain low. The proposed approach indeed, assumes that the user indicates when the gesture tracking starts, pulling the trigger-like button (see figure 2.1), and releasing it when the gesture is finished. This supposition has also another motivation, which is the requirement **R.3.8**: when the trigger-button is not pressed, the device can be moved without changing the pointer state.

Furthermore, it is not really necessary to track the position with a high precision: it will be scaled with a constant factor in order to set the movement sensitivity (requirement **R.3.4**), so it is sufficient that the tracked position changes according to the real motion direction of a "convincing" but not really precise amount.

The gravity measuring at stay has an advantage and a drawback. The advantage is the chance to obtain tilt information on two axes, without having a gyroscope. Assuming that a device should be grasped in a predefined position, it can be defined a rest axis for the gravity acceleration. For example the *Wiimote* in figure 2.2 is at rest position when the Z axis senses all the gravity on its negative direction. The tilt around one of the two not-rest axes can be calculated using a combination of the arccosine function. The rest axis coordinate is exploited to choose between the two angles with the same cosine. In order to have the rotation amount, the rest angle has to be subtracted from the calculated one. The tilt around the rest axis cannot be determined.

For example in order to calculate the tilt around the Y axis for the *Wiimote* (the red angle in figure 4.3), the rest angle can be considered $\frac{3}{2}\pi$. Suppose that the current normalized sensed acceleration is (x, y, z).

The current angle α is

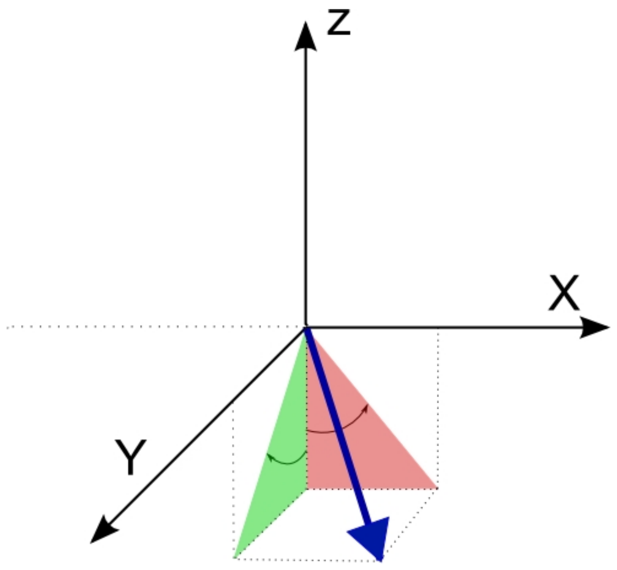

Figure 4.3: Tilt calculation with accelerometers

$$\alpha = \begin{cases} \arccos(x) & \text{if } z \geq 0 \\ 2\pi - \arccos(x) & \text{if } z < 0 \end{cases} \tag{4.2}$$

The sensed tilt around the Y axis (roll) is

$$roll = \alpha - \frac{3}{2}\pi. \tag{4.3}$$

The disadvantage of sensing the gravity acceleration is that, when the rotation and the linear movement are composed at the same time, it is impossible to distinguish it from the input acceleration when the gravity orientation changes with respect to the device.

An example of this situation can be seen in figure 4.4: at the beginning of the movement (1), the gravity is sensed along the negative direction of

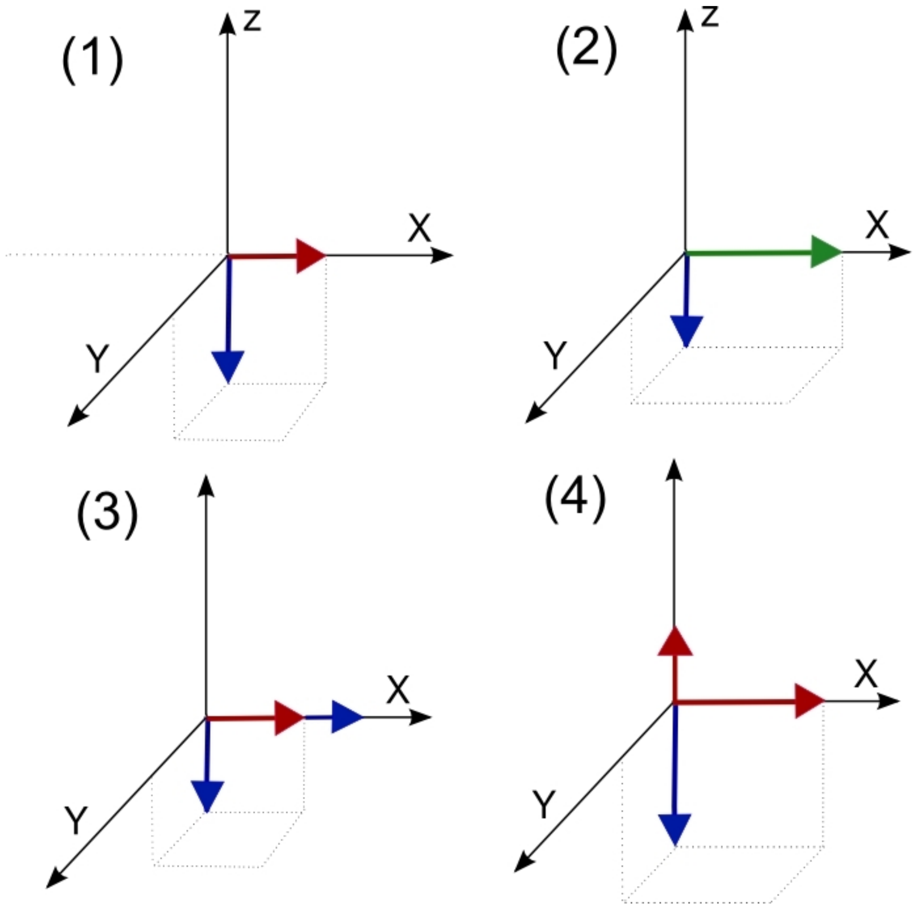

Figure 4.4: Change of gravity orientation with respect to the device during motion

the Z axis. If, during the motion, the output of the sensor lowers along the Z axis (in module) and raise on the X axis, the acceleration measurement is like the one represented in (2). This change can be the result of two different motions. The first one is a rotation around the Y axis, which distributes part of the gravity acceleration along the Z axis and the reminder along the X axis, represented in (3). Supposing that the user has not changed the applied acceleration direction, also a part of the input is sensed along the Z axis, thought that in figure 4.4 it is not shown for the sake of clarity (it is small with respect to redistributed part of gravity). The second one (4) is due to a change on the user input acceleration: at a certain time in the motion, s/he applies a greater acceleration along the X axis and starts to move also in the positive direction of the Z axis. The result is that on the Z axis is sensed the difference between gravity and the input acceleration, while the value on X axis raises. The two motions are really different, but the accelerometer measures them with the same values.

To summarize the discussion, using only accelerometers either linear motions on three axes or tilt on two axes can be tracked, while using a combination of accelerometers and gyroscopes, both motion and rotation can be detected.

4.3.2 Three axis accelerometer

Before beginning any computation, the data coming from the device must be filtered in order to reduce the noise that affects the samples. For this purpose, two representations of the same sample are maintained.

The first one is calculated using a low pass filter: its output is obtained adding the 90% the previous value and the 10% of the current. This leads having a representation with low influences by sudden changes, good for representing gravity, which is nearly constant when the device is motionless.

The second has the opposite purpose: it is the output of a high pass filter, able to highlight sudden changes on data. This is suitable for representing the motion part of the input.

As discussed in the previous section, the gravity can be used for roll and pitch rotations recognition (yaw is about the Z axis and cannot be sensed), while the motion component can be exploited in order to find linear motions. However is not possible to recognize the two types of movements at the same time.

The proposed criteria for separating the two types of motion, which is a novelty with respect to previous works, is a test on the module of the low passed acceleration vector: if it corresponds (with an adequate range of tolerance) to g[6], then the acceleration sensed is only gravity and the device must be stationary[7]. In order to increase the effectiveness of the criteria, the condition should hold on a certain number of samples in order to declare the device motionless.

[6] g is the standard symbol for the gravity acceleration, approximately $9.8ms^{-2}$

[7] This sentence assumes that it is not possible for a user to give an acceleration that could be mistaken for gravity: the occurrence of such situation means that the user has accelerated the device exactly $2g$ along the current gravity direction

If the module tolerance range is adequate, the rotation motion should not drift the module of the acceleration for too long, and its change will not be recognized as a linear motion.

The values for the tolerance and the number of samples for declaring the device stationary were chosen trying to use the device with the test environment (see section 5). A tolerance of $0.49ms^{-2}$ and 20 samples worked fine.

Figure 4.5 represents the motion tracking algorithm, supposing that the trigger-like button is pressed. Once a sample is received, it is filtered using by a `LowPassFilter`, a subclass of the generic `SampleFilter` (see section 3.3.3). After that the module m of the filter output is calculated: if $|m - g| \leq \Delta$, where Δ is the tolerance, the device is moving and the sample counter `state` is set to `NSAMPLE`. This parameter is a heuristic evaluation of the time to wait for assuming that the device is motionless. Otherwise it is supposed that the device is moving: indeed, if the `state` counter is greater than zero, the module value may be due to noise during a motion tracking. In this case the counter is decreased.

After the `state` counter update, two cases are possible:

1. `state` > 0. The device is moving, so the original sample passes in a sequence of two filters: the `HighPassFilter` and the `PositionIntegrationFilter`. The first one is an implementation of the previously discussed high-pass filter, used for extracting the sudden changes in the acceleration due to its motion, while the second performs a double integration on the high-passed value, in order to calculate the new position

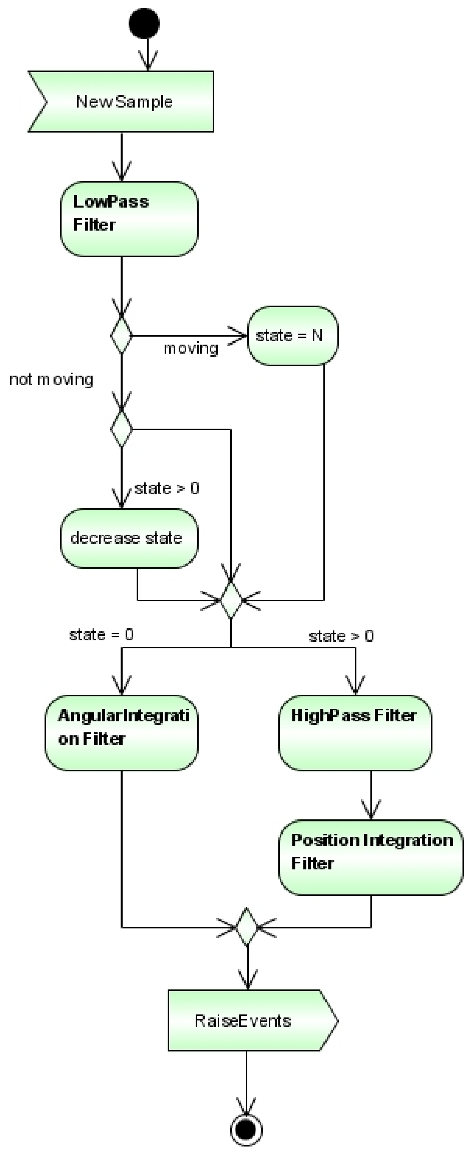

Figure 4.5: Accelerometer motion tracking algorithm

of the pointer abstraction.

2. `state` = 0. The device is stationary. The low-passed sample is filtered by a `TiltFilter` which performs the roll and the pitch calculations, as already explained in section 4.3.1.

After that the notification of the events related to the pointer state change is performed (if needed).

The module testing technique solves the position drift problem when the device is motionless. However the gravity may still bias the position measurement: that situation happens when, during a motion, the device is rotated and the gravity changes its direction with respect to the device.

The problem is that, from the algorithm point of view, the two scenarios discussed section 4.3.1 and represented in figure 4.4 cannot be distinguished. The position changes always assuming the (4) situation in figure 4.4: the pointer changes its X coordinate moving right and its Z coordinate moving up.

4.3.3 Combination of accelerometers and gyroscopes

The prototype equipment with a gyroscope leads the possibility to separate the rotation about three axes from the motion sensing, due to the fact that the values are provided by different sensors.

In the same way as accelerometers, also the gyroscope output is affected by noise. A low pass filter is again exploited in order to narrow its impact on the sensed data.

The algorithm is shown on figure 4.6. After having received the current sample, the computation of the low passed value is performed by the `LowPassFilter` class, which filters not only the accelerometer data, but also the gyroscope output. The acceleration module value is used again in order to establish if the device is moving, with the same tolerance and number of samples as in section 4.3.2.

If the device is moving, the instantaneous motion component is extracted using the `HighPassFilter`. The position is then integrated through the `PositionIntegrationFilter`.

Unlike the previous algorithm, the fact that the device is moving does not affects the tilt recognition, performed using directly the integration of the angular rate about the three axes, performed by the `AngularIntegrationFilter`.

Thus the motion tracking procedure has no more need for performing a motion separation and the gyroscope sensor output can replace the `TiltFilter` discussed in 4.3.2. This algorithm can recognize also yaw rotations, completing the six degrees of freedom for the movement.

It is worth pointing that the gravity orientation change problem during the motion, as discussed in section 4.3.1, affects also this version of the algorithm.

4.3.4 Gesture samples

This section completes the presentation of the motion tracking algorithm showing four movement sample data:

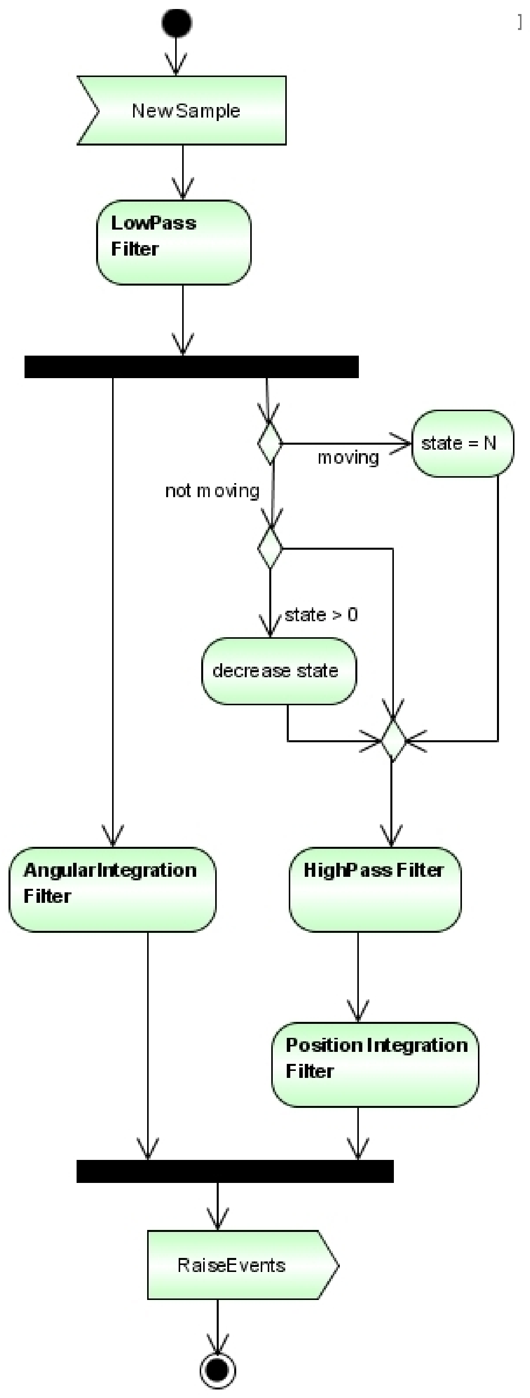

Figure 4.6: Accelerometer and gyroscope motion tracking algorithm

1. Linear motion on one axis
2. Linear motion on two axes
3. Roll rotation sensed using only accelerometers
4. Yaw rotation sensed using the gyroscope.

For linear motions is reported a graphical representation of the device output, the low passed data and the changes of the position over time. For the rotation sensed with the accelerometers, the position is replaced by the rotation angle calculated.

The gyroscope sensed rotation is displayed through the sensor raw output, the low passed data and the resulting rotation angle.

In figures 4.7 and 4.8 the low passed acceleration is complemented by the module change over time, in order to show the point where the algorithm begins to sense the motion: where it starts to increase or decrease the start of a movement is recognized.

The opposite is shown in figure 4.9, the module is more or less constant, but the accelerations along the X and Z axis respectively decreases and increases approximately of the same amount. This is recognized as a rotation about the Y axis (roll).

Figure 4.10 represents the recognition of a rotation about the Z axis (yaw), which is not possible using only accelerometers. The low-passed data is integrated in order to obtain the angle covered.

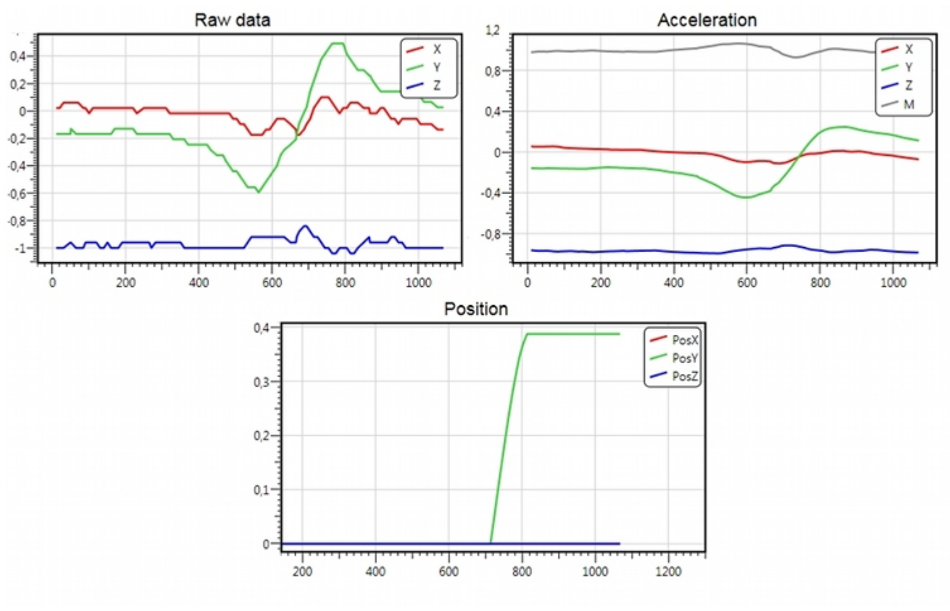

Figure 4.7: Linear motion recognition on Y axis

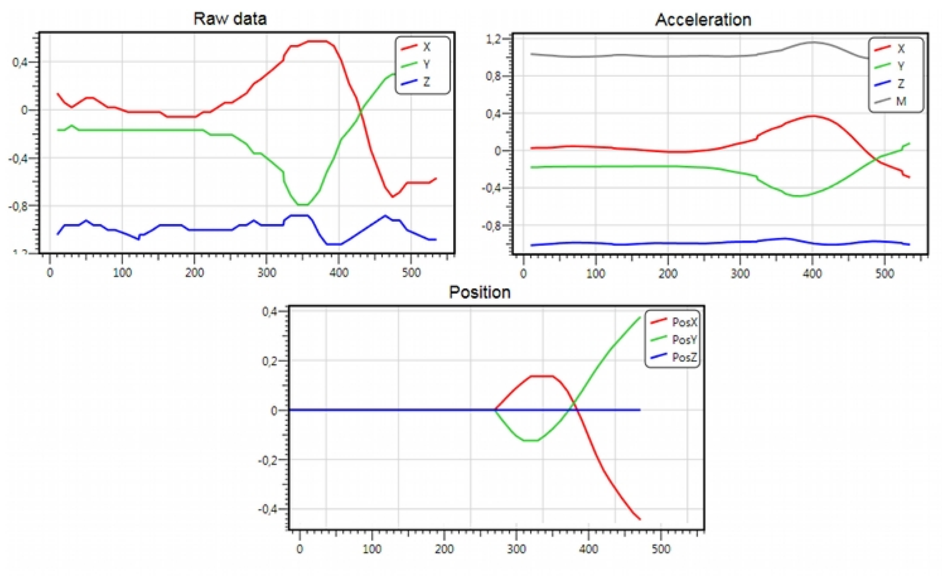

Figure 4.8: Linear motion recognition on X and Y axis

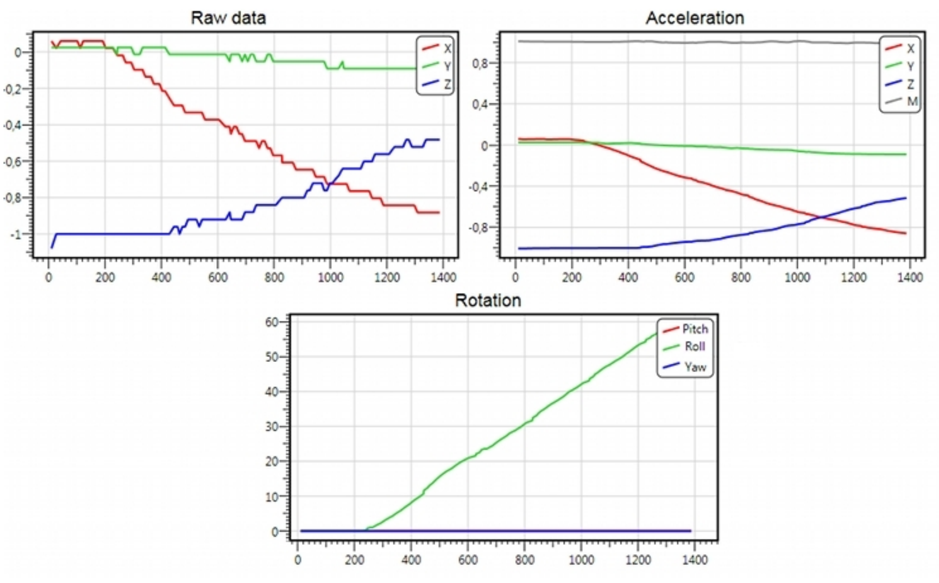

Figure 4.9: Pitch rotation recognition using an accelerometer

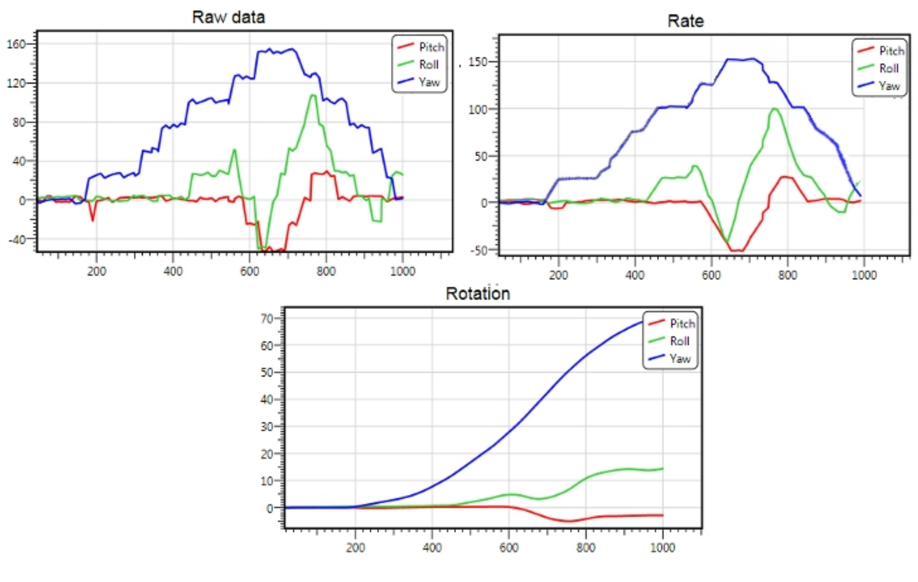

Figure 4.10: Yaw rotation recognition using a gyroscope

5 Test environment

This chapter describes the test environment developed for the *Mouse3D* library. It consists mainly in a monitoring part where charts for sensed and calculated values are plotted, together with a 3D world where it is possible to move an arrow, representing the mouse pointer.

5.1 Windows Presentation Foundation

Before discussing the test application features, the technology used to realize the application deserves to be discussed.

The *Windows Presentation Foundation* (WPF) is a revolutionary graphical display system for *Windows*. The core features of WPF can be summarized as follows:

- *Declarative user interface definition.* According to the web pages design experience, the definition of the layout for a user interface is no longer based on pixel coordinates, but it can be performed in a declarative manner using the Extensible Application Markup Language (*XAML*, pronounced “zammel”) and has a hierarchical tree representation. The behaviour of the user interface can be defined procedurally in a code-behind file. The XAML document content is not directly interpreted by the graphic system, but it is compiled in a binary format which, together with the code-behind methods, will be re-compiled as a *.Net* application.

- *DirectX painting.* *WPF* renders the interface controls using *Microsoft DirectX*[1] API, a fast graphical engine which exploits the capabilities of most video cards. This library is usually adopted for games and 3D graphics. This allows having a native support for transparency, gradients, 3D rendering and many other graphical capabilities.
- *Vector Graphics.* The drawing model is no more the control *paint* procedure, where the developer specifies an algorithm for drawing it from scratch. In WPF the developer creates graphical primitives (as lines, rectangles etc.) and adds them inside the user interface component. The graphical system will render them until they will be removed. Such mechanism is called *retention* and allows having a resolution independent painting process in WPF.
- *Observable properties.* Dependencies for controls can be expressed using the dependency properties, which enables the notification of their state

[1]http://www.microsoft.com/

change, in order to update the rendering. For example the background can be defined as a dependency property and, when it is changed, the drawing is updated without invoking any refresh procedure.

- *Control and containers.* The tree representation of the user interface allows using standards controls (i.e. buttons) as containers of other widgets, allowing an easy layout configuration: if a button with an image and a text label is needed, it not necessary to define a subclass of the `Button` class. It is sufficient to define a stack panel within the button definition containing an image and a label.

- *Animations.* WPF has a native support for animations, which can be defined declaratively using frames and a time line, without the need of a timer control.

- *Styles and templates.* The user interface appearance can be defined using styles, which allow easy changes to layout, separating it from its structure.

- *Commands.* The application features (such as opening or saving a file, undo, redo, etc.) can be accessed using different controls (a button, a menu, a keyboard short cut etc.). WPF as a native support for defining the command in one place and binding it to different controls.

It is really likely that this graphic system model will become more and more popular. Certainly it fits very well for the usage of a three dimensional mouse: it is really easy to mix 2D and 3D graphics within the same interface, enabling the design of applications which exploits three dimensional movements.

5.2 Environment user interface

In order to validate the library design and the motion tracking algorithms, the test application contains both an analysis and an interactive part.

The analysis part contains different plot areas where the raw sensor data can be compared with the motion tracking algorithm output, in order to have an idea of the underlining process. Such comparison is helpful for debugging the motion tracking procedures.

The interactive part contains a three dimensional space where an arrow, which represents the *Mouse 3D* pointer, can be moved and rotated according to the device motions and rotations.

The environment can be initialized in order to use the *Wiimote*, with or without the *Wii Motion Plus*. If the extension is plugged, then the application creates an `AccGyroMouse3DPointer` instance, otherwise a `AccMouse3DPointer` is exploited in order to sense the movements (see section 4.3.2 and 4.3.3).

The application contains also a calibration wizard. The user is requested to put the *Wiimote* in the same position as the picture displayed and to wait until the next step is reached. The procedure consists of six steps and, at the end of it, the application uses the new calibration values which are the acceleration zero of each axis, the conversion factor in order to obtain meters for second squared and the zero value for the angular rate (if the *Wii Motion Plus* is connected).

Figure 5.1 and 5.2 show the two parts of the user interface: the interactive 3D space and the analysis charts, which will be discussed more in detail in following sections.

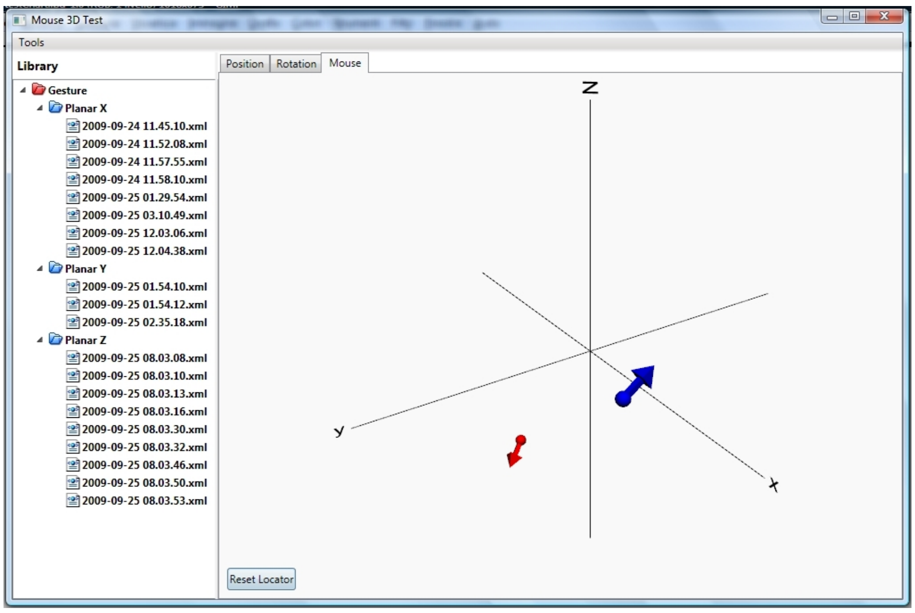

Figure 5.1: The test environment UI for the pointer control

5.2.1 Interactive 3D space

The three dimensional space is the interactive part of the test environment. It simulates a mouse pointer with an arrow that can be moved in three directions, according to the linear motions recognized by the library.

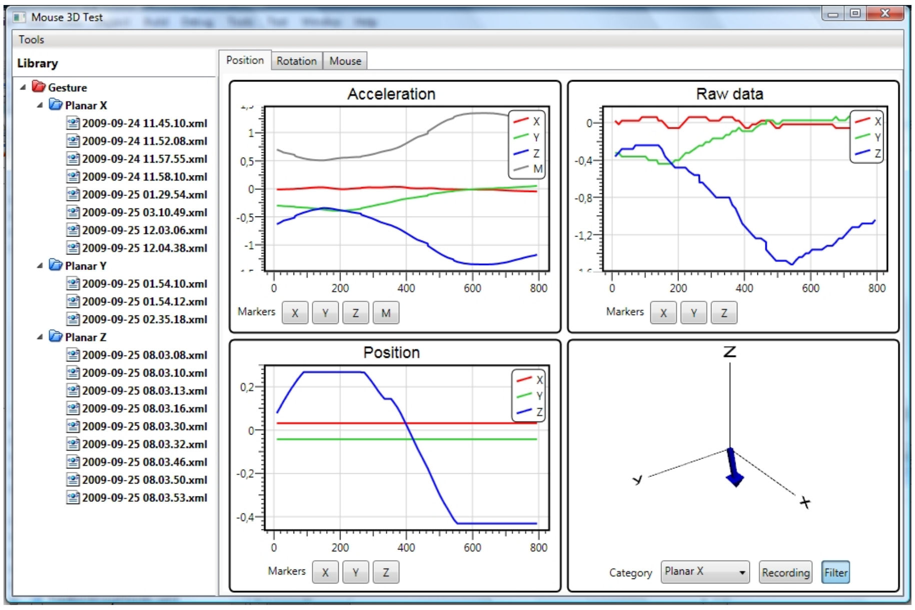

Figure 5.2: The test environment UI for data analysis

Depending on the motion tracking algorithm, the arrow can be rotated about two or three axes (see section 4.3.1), showing the result of the recognition of a rotation gesture.

Figure 5.3 shows the effect of a linear motion along the Z axis. The arrow is motionless (1) until the device is moved upward (2). The arrow starts moving as soon as the algorithm senses the peak on the acceleration and ends the path when the *Wiimote* stops (3).

The effect of a rotation gesture is shown in figure 5.4: from the current position (1), the *Wiimote* rotates around the X axis (2), and arrow rotates around the same axis too, using as centre the sphere at its end (3).

The other motions are variations of these basic movements, changing the axis or combining two or three of them.

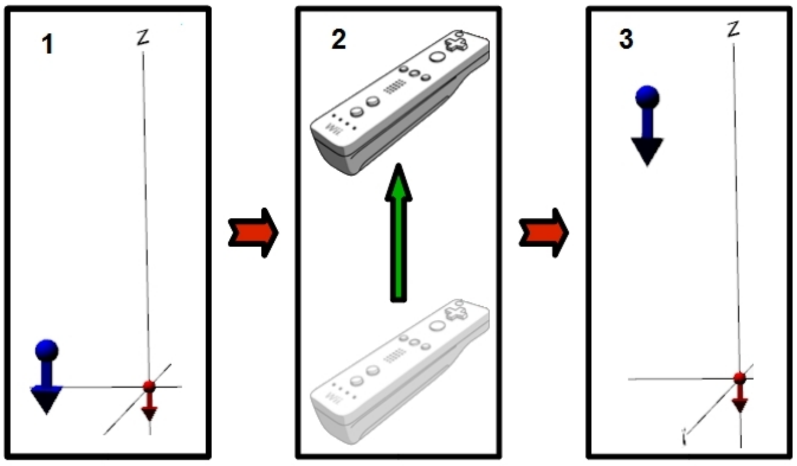

Figure 5.3: Linear motion along the Z axis

5.2.2 Acceleration and rotation monitoring

An accurate analysis of motion data can be performed using five different graphs, split in two tabs (acceleration and rotation). The plotted quantities are:

1. Acceleration
2. Raw accelerometer data
3. Position

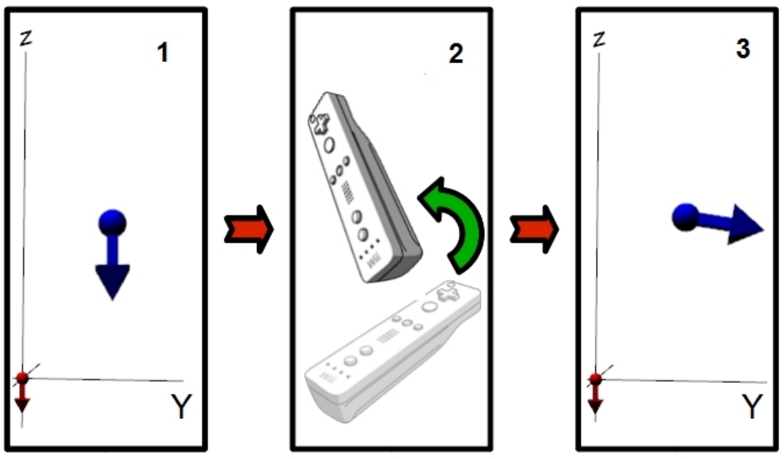

Figure 5.4: Rotation movement around the X axis (pitch)

4. Rotation

5. Raw gyroscope data (not used if the *Wii Motion Plus* is not connected).

The functions are plotted when a gesture ends, displaying the flow of the measured quantity. Is it possible to zoom in and out the graph, adding also circular markers in order to highlight each sample calculated or received by the device. If the mouse pointer is over one of these points is displayed a tool tip containing the X and Y coordinate.

Figure 5.5 shows a plot example. On the left part is displayed the acceleration with markers for the module line graph. The tool tip below the mouse pointer shows the value for the selected sample. On the right part a portion of the raw data can be deeply analysed using the zoom function.

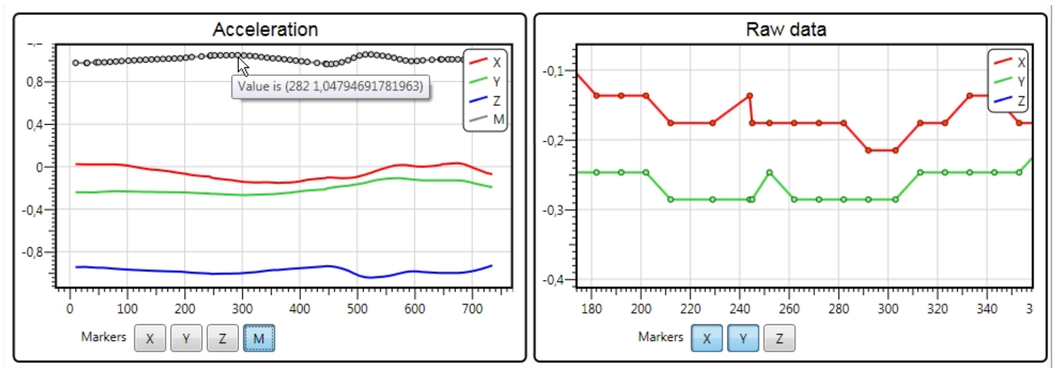

Figure 5.5: Graph plotting example

5.3 Data Recorder

During the development of the library, it was important to repeat tests on the same data especially for the motion tracking algorithm. However is not possible to repeat exactly the same movement and the sensor output will be always different. Another drawback in testing the algorithm directly was the inability of performing a step by step debugging, due to the fact that the time between samples is calculated by the `WiimoteAdaptor` (see section 4.1) as the difference between the arrival time of the current one and the previous one. As a result, the samples cannot be delayed.

To overcome these problems, the test application contains a data recording feature. If the user pushes the record button on the analysis part of the user interface each gesture performed will be saved in an XML file. The gesture can be categorized in different folders, in order to remember the recorded gesture type.

The XML contains the list of samples with the raw acceleration data, the angular rate (if the gyroscope is not connected, the elements are set to zero) and the arrival time. However saving this data is not sufficient in order to have the exact reproduction of the movement, because it depends also on the state of the filters contained within the `AccMouse3DPointer` or `AccGyroMouse3DPointer` class. For instance the low pass filter needs the value calculated for the previous sample (see section 4.3.2) in order to calculate the new one.

For restoring the pointer state, the `Mouse3DPointer` class was provided with two methods for setting and getting its state, represented by an instance of the `PersistPointerState`. It contains the parameters of the low and high pass filter, the value of the `state` variable (see section 4.3.2 and 4.3.3) the current value for the acceleration, position and rotation.

Before the application starts writing the sample values, the value of this object is persisted in the XML file. It is worth pointing out that this operation is not really needed for the pointer abstraction, and it should be internal to the `Mouse3D` package when the prototyping phase will be finished.

The user can replay a previous recorded gesture double clicking one of the XML file listed on the left part of the interface (see figure 5.1). The red arrow in the interactive part moves according to the recorded data, which is plotted in the analysis tabs.

6 Conclusions and future works

6.1 Conclusions

This thesis describes the design and the implementation of a three dimensional mouse library, which uses accelerometers and gyroscopes in order to recognize the device movements.

Due to internal manufacture, the accelerometers sense the gravity, which is impossible to distinguish from the input acceleration if the sensor is translated and rotated. In addition, the double integration process of the acceleration data in order to obtain the position is subject to drift caused by the amplification of measurement errors. The same problem affects also the gyroscope when the angular rate is integrated in order to obtain the rotation.

The use of the physical meaning of the accelerometer data was not considered in previous works due to these problems and statistical models were adopted. This approach was not suitable for the development of the library, because they relies on having the whole gesture data, while the pointer abstraction should give an immediate feedback to the user, also when s/he is still performing the motion.

However, thanks to the a novel approach for establishing when the device is moving, which consists on testing the module of the sensed acceleration vector and comparing it to gravity, the drift problem has a low impact on the proposed motion tracking algorithms.

In order to create the prototype for the library development a Nintendo *Wiimote*[1] controller has been used, because it is equipped with a three-axis accelerometer and can be expanded with a gyroscope. It has also other hardware which can be used for motion tracking, but it has not been used to maintain the library compatibility with other devices (i.e. *PDA* and smartphones) which are equipped only with accelerometers. Due to this fact, the gesture recognition cannot be performed as in Nintendo games.

The result of the work is the creation of the library, which has the following characteristics:

- *Mouse 3D abstraction.* The library make available to developers an abstraction of a 3D pointer, with a set of gestures for controlling it. The abstraction is represented by the current position over the space and the rotation angle around three axes. The user interface components can subscribe to its status change events.

- *Device independence.* The library has an abstraction layer between the device and the pointer abstraction. The support for a new device can

[1]http://www.nintendo.it/

be added simply creating a data reader class with a given interface.

- *Motion tracking algorithms.* A novel approach to the motion tracking is proposed in order to recognize the device movements. Two algorithms have been designed and developed: the first one recognizes either the linear motion along three axes or the rotation about two axes using accelerometers, while the second can sense also the rotation about the third axis coupling accelerometers and gyroscopes.
  The main idea is to establish a criteria for assuming that the device is motionless, which is testing the acceleration vector module. If it corresponds to the gravity acceleration the device is stationary, otherwise the device is moving. Only in the latter case the acceleration is double integrated in order to obtain the position change. Thought that the motion considered lasts about a second, the drift is marginal. The rotation is calculated measuring the angles defined by the gravity vector and the axes for the version which uses only accelerometers, while when also gyroscopes are exploited the angles are calculated integrating the angular rate.
  Due to limitations inherent to accelerometers, if the device is translated and rotated at the same time, changing also the gravity vector orientation with respect to the device, the movement may be incorrectly interpreted.

Together with the library is provided also a test environment which allows the user to move the pointer in a three dimensional space.

In order to spread the usage and to improve the development of the library, it will be distributed as an open source project.

6.1.1 Requirements review

In section 3.1.2, a set of requirements has been defined in order to establish a success criteria for the library development. The following is a review of these requirement in order to validate the objectives.

R.3.1 *Definition of gestures using a 3D accelerometers.*

The library recognizes five gestures in order to control the mouse pointer using an accelerometer equipped device. These gestures are linear motions along X, Y and Z axis, and the rotation about the X and Y axis (pitch and roll). The two gesture groups cannot be combined together: the motion tracking algorithm senses either linear motions or rotations.

R.3.2 *Definition of gestures using a combination of a 3D accelerometer and a 3D gyroscope.*

The library recognizes six gestures in order to control the mouse pointer, using a combination of accelerometers and gyroscopes. These gestures are linear motions and rotations along the X, Y and Z axis. The two gesture groups can be combined together.

R.3.3 *Definition of an abstract 3D mouse pointer.*

The library defines an abstraction of a three dimensional mouse pointer, represented as a point with X, Y and Z coordinates, pitch, roll and yaw rotation angles. It contains also information about the pressed or released buttons.

R.3.4 *Sensitivity control.*

Is it possible to set the scale factor for the calculated position in order to increase or decrease the motion sensitivity.

R.3.5 *Device independence.*

The library relies on an abstraction layer between the position calculation and the data reading. Each supported device has a data reader class which translates the device specific data into the format expected from the library.

R.3.6 *Status change notification.*

The mouse pointer abstraction has a set of events related to its status change. If a user interface is interested in such notification, it can subscribe the event notifications.

R.3.7 *Real-time response.*

The library is able to update the pointer position for each sample received and, in order to recognize the beginning of a motion, at maximum 20 samples are needed. Thought that a gesture contains about 100 samples, the position is updated also during the motion.

R.3.8 *Stop tracking.*

By default the motion is tracked only if the trigger-like button of the *Wiimote* is pressed (in other devices either a software or hardware button can be used). In this way the motion tracking can be stopped simply releasing the button.

6.2 Future works

The thesis work can be considered as the end of my master degree studies or as the starting point of my future works. This section is about the possible further developments of the project.

6.2.1 Algorithm improvements

The motion tracking algorithm exploits the acceleration data in a novel manner, establishing a criteria to consider the device motionless in order to defeat the drift deriving from the gravity sensing. This approach has been demonstrated to work well for linear motions.

However if the device is translated and rotated at the same time, the motion can be detected and the rotation too (if a gyroscope is used), but with the current implementation is not possible to distinguish the input acceleration from the gravity which, during the rotation, can have changed its direction with respect to the device. This leads to recognition of a motion different from the real one.

A possibility to overcome this problem is the exploitation of the gyroscope data in order to maintain a representation of the device orientation.

A quaternion[2] representing the device orientation transform for the gra-

[2] an extension of complex numbers which can be represented as a 4x4 matrix

vity can be maintained. If the representation is accurate, it can be used in order to subtract it from the sensed acceleration during the movement. However this needs a very precise measurement of the rotation angles and, thought that the angular rate integration is subject to drift, can result in an output far away from the real motion. However at least for pitch and roll the error can be controlled updating the quaternion each time that the device is motionless (and the module computation here should be still crucial) exploiting the accelerometer data as described in 4.1.

6.2.2 User testing

Though that different tests have been performed by the author and other people at the CVS lab at the Computer Science Department, the library needs a user evaluation, in order to understand if the interaction model can be accepted by common people.

The analysis should propose different application contexts and evaluate if the gestures are suitable for creating usable and intuitive applications, collecting both quantitative ratings on the interaction effectiveness and also qualitative data, such as suggestions or critics about the prototype functioning.

6.2.3 Window manager integration

Another further improvement for the library will be the extension of the operating system window manager in order to support the notification of the

Mouse3D events not only within the current frame, but also to share the same mouse pointer between two or more windows.

In order to realize such application, a good understanding of the user interface manager (also called window manager) of the host operating system is needed. It manages the input and the output devices in order to share them between all the running applications.

For instance in Windows operating systems, the events related to the input devices are notified to the applications using messages. The user interface manager has a queue for each application and is in charge to deliver the messages to the application that registered for listening the events.

It is also possible to define the so called *Registered Windows Message*s, which are user defined notifications (the user in this case is the developer) which are guarantee to be unique in the system.

Using them is it possible to define a Windows application which notifies the status of the mouse 3D abstraction: for each event related to it, a new *Registered Window Message* should be enlisted. Such messages can be for instance the followings:

1. `WM_MOUSE3DMOVE`: the 3D pointer has moved within the window client area (the window viewport).

2. `WM_NCMOUSE3DMOVE`: the 3D pointer has moved within the window non-client area (the window borders).

3. `WM_MOUSE3DENTER`: the 3D pointer entered within the window client area.

4. `WM_NCMOUSE3DENTER`: the 3D pointer entered within the window non-client area.
5. `WM_MOUSE3DLEAVE`: the 3D pointer leaved within the window client area.
6. `WM_NCMOUSE3DLEAVE`: the 3D pointer leaved within the window non-client area.
7. `WM_MOUSE3DROTATED`: the mouse 3D pointer has been rotated.
8. `WM_MOUSE3DBTN1DOWN`: The button one has been pressed.
9. `WM_MOUSE3DBTN2DOWN`: The button two has been pressed.
10. `WM_MOUSE3DBTN1UP`: The button one has been released.
11. `WM_MOUSE3DBTN2UP`: The button two has been released.
12. `WM_MOUSE3DBTN1CLK`: The button one has been clicked.
13. `WM_MOUSE3DBTN2CLK`: The button two has been clicked.
14. `WM_MOUSE3DBTN1DCLK`: The button one has been double clicked.
15. `WM_MOUSE3DBTN2DCLK`: The button two has been double clicked.

The messages should contain a parameter depending on the message type: if it is related to the mouse position, it should contain the position represented as a three-dimensional point expressed in the target window coordinate system, otherwise if it is related to the rotation pitch, roll and yaw should be listed.

The *Mouse3D* manager application is in charge to invoke a broadcasting of the status changes notifications.

Such implementation will allow sharing the pointer resource among different applications.

Acknowledgements

First of all, I would like to thank my parents who, with their care and trust, allowed me to come in Tuscany for attending one of the most valuable Universities in Computer Science. This achieved goal belongs in large part to them. I hope that to see me arriving at the end pays them back for all their efforts.

Special thanks go to Andrea and Emanuele, who always support their "continental" brother from a distance. Your love makes me feel that you're always close to me, though that unfortunately we can meet each other only few times.

At the top of the list of persons who have contributed to my education and that I want to thank, there is the supervisor of this thesis, Antonio Cisternino. His example at CVS lab taught me to have passion for what I am doing, and that with dedication it is possible to finish also very challenging works. I want to mention here all the people from the laboratory and the members of the *Cinque&Cinque* team, which are always ready for collaboration and for offering their useful ideas: Cristian Dittamo, Marco Mura, Stefano Simula, Nicola Andrenucci, Davide Morelli, Davide Carfì, Donato Ferrante, Sara Berardelli.

Many thanks also to the HIIS laboratory at ISTI CNR, where I work since two years. Thanks to the head Fabio Paternò, the researchers Barbara Leporini and Carmen Santoro and all the other people: Claudio Porta, Giuliano Pintori, Mauro Lisai, Ariel Kuri, Giulio Galesi, Antonio Scorcia, Antonello Calabrò, Elia Contini, Giuseppe Ghiani.

I want to thank Nicola Addis, my best childhood friend, and all my gang from Perfugas: Gavinuccio Deiana, Nicoletta Pani, Barbara Solinas, Maura Maiorca, Roberta Damiano. A credit should be given to all members of my "Casa del Popolo" (in English it sounds like "People's Home") in Pisa: Agostino Leoni, Giancarlo Dore, Antonio Virdis, Pier Miscali, Federica Scintu, Marta Scintu, Alessandra Lai.

Dulcis in fundo, many, many thanks to my girlfriend Pinuccia Flore. She helped me, she pampered me, she supported me, she encouraged me, she advised me and she did many other things for me. I am not as good as she is writing words, but I want to say to her that I would never achieved this goal without having her by my side, and I hope that she would like to stand to me for a long time. You are really the most beautiful thing that the experience at the Pisa University gave to me.

Thank you.

Ringraziamenti

Il primo ringraziamento va ai miei genitori, che con il loro impegno e fiducia mi hanno consentito di venire in Toscana per frequentare una delle migliori Università nel campo dell'Informatica. Questo traguardo è in gran parte loro. Spero che vedermi arrivare alla fine del percorso li ripaghi di tutti gli sforzi.

Un ringraziamento speciale lo devo ad Andrea ed Emanuele, che sostengono sempre il loro fratello "continentale". Il vostro affetto mi fa sentire sempre di avervi vicino, anche se purtroppo ci vediamo poco.

In cima alla lista dei ringraziamenti alle persone che hanno collaborato alla mia formazione, c'è sicuramente il relatore di questo lavoro di tesi, Antonio Cisternino. Il suo esempio al CVS lab mi ha insegnato a guardare con passione quello che faccio e che con l'impegno si possono portare a termine lavori che sembrano delle imprese. Voglio ricordare qui tutti i componenti del laboratorio ed i membri del team *Cinque&Cinque*, sempre pronti a collaborare ed offrire le loro preziose idee: Cristian Dittamo, Marco Mura, Stefano

Simula, Nicola Andrenucci, Davide Morelli, Davide Carfì, Donato Ferrante, Sara Berardelli.

Un grosso ringraziamento va anche al laboratorio HIIS all'ISTI CNR, dove lavoro da più di due anni. Grazie al capo Fabio Paternò, alle ricercatrici Barbara Leporini e Carmen Santoro e tutti gli altri componenti: Claudio Porta, Giuliano Pintori, Mauro Lisai, Ariel Kuri, Giulio Galesi, Antonio Scorcia, Antonello Calabrò, Elia Contini, Giuseppe Ghiani.

Voglio ringraziare Nicola Addis, il mio più caro amico d'infanzia, e tutta la mia combriccola perfughese: Gavinuccio Deiana, Nicoletta Pani, Barbara Solinas, Maura Maiorca, Roberta Damiano. Una menzione al merito va data anche ai componenti della mia "Casa Del Popolo" a Pisa: Agostino Leoni, Giancarlo Dore, Antonio Virdis, Pier Miscali, Federica Scintu, Marta Scintu, Alessandra Lai.

Dulcis in fundo, un grazie di cuore va alla mia ragazza, Pinuccia Flore. Lei mi ha aiutato, coccolato, sostenuto, incoraggiato, consigliato e chi più ne ha più ne metta. Non sono bravo con le parole come lei, ma voglio dirle che questo traguardo non l'avrei mai raggiunto se non mi fosse stata accanto e spero che scelga di sopportarmi ancora per tanto tempo. Sei veramente la cosa più bella che l'esperienza all'Università di Pisa mi abbia regalato.

Grazie.

Bibliography

[AA.a] AA.VV. Bluetoot special interest group. http://www.bluetooth.org.

[AA.b] AA.VV. A wiki dedicated to homebrew on the nintendo wii. http://wiibrew.org/wiki/Main_Page.

[AA.c] AA.VV. Wikipedia, the free encyclopaedia. http://en.wikipedia.org.

[Alp04] Ethem Alpaidin. *Introduction to machine learning*. MIT Press, 2004.

[Bar00] Joel F. Bartlett. Rock 'n' scroll is here to stay. *IEEE Comput. Graph. Appl.*, 20(3):40–45, 2000. doi:http://dx.doi.org/10.1109/38.844371.

[Boy03] Walt Boyes, editor. *Instrumentation Reference Book*. Elseiver Science, 2003.

[BP01] Ari Y. Benbasat and Joseph A. Paradiso. Compact, configurable inertial gesture recognition. In *CHI '01: CHI '01 extended abstracts on Human factors in computing systems*, pages 183–184, New York, NY, USA, 2001. ACM. doi:http://doi.acm.org/10.1145/634067.634178.

[Fla06] David Flanagan. *Javascript: the definitive guide*. O'Reilly, 2006.

[Fra04] Jacob Fraden. *Handbook of Modern Sensors*. Springer, 2004.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlisside. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[Gun97] Kevin N. Gunrey. *An introduction to neural networks*. CRC Press, 1997.

[KKM$^+$06] Juha Kela, Panu Korpipää;, Jani Mäntyjärvi, Jani, Sanna Kallio, Giuseppe Savino, Luca Jozzo, and Di Marca. Accelerometer-based gesture control for a design environment. *Personal Ubiquitous Comput.*, 10(5):285–299, 2006. doi:http://dx.doi.org/10.1007/s00779-005-0033-8.

[LG05] Edward C. Litty and Lennor L. Gresham. Hemispherical resonator gyro: an iru for cassini. *Proceedings of SPIE*, 2803:299, 2005.

[LGG$^+$97] M. Lutz, W. Golderer, J. Gerstenmeier, J. Marek, B. Maihofer, S. Mahler, H. Munzel, and U. Bischof. A precision yaw rate sensor in silicon micromachining. In *Solid State Sensors and Actuators, 1997. TRANSDUCERS '97 Chicago., 1997 International Conference on*, volume 2, pages 847–850 vol.2, Jun 1997. `doi:10.1109/SENSOR.1997.635234`.

[Mac05] Mattiew MacDonald. *Pro .NET 2.0 Windows Forms and custom controls in C#*. Apress, 2005.

[Mac08] Mattiew MacDonald. *Pro WPF in C# 2008: Windows Presentation Foundation with .Net 3.5*. Apress, 2008.

[MKKK04] Jani Mäntyjärvi, Juha Kela, Panu Korpipää, and Sanna Kallio. Enabling fast and effortless customisation in accelerometer based gesture interaction. In *MUM '04: Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, pages 25–31, New York, NY, USA, 2004. ACM. `doi:http://doi.acm.org/10.1145/1052380.1052385`.

[MMST00] V. M. Mäntylä, J. Mäntyjärvi, T. Seppänen, and E. Tuulari. Hand gesture recognition of a mobile device user. In *Internation IEEE Conference on Multimedia and Expo*, pages 281–284, 2000.

[MP97] T. Marrin and J. Paradiso. The digital baton: a versatile performance instrument. In *International Computer Music Conference*. Computer Music Association, 1997.

[NGK+07] R. Neul, U.-M. Gomez, K. Kehr, W. Bauer, J. Classen, C. Doring, E. Esch, S. Gotz, J. Hauer, B. Kuhlmann, C. Lang, M. Veith, and R. Willig. Micromachined angular rate sensors for automotive applications. *Sensors Journal, IEEE*, 7(2):302–309, Feb. 2007. `doi:10.1109/JSEN.2006.888610`.

[Pea] Brian Peak. Managed library for nintendo's wiimote. `http://wiimotelib.codeplex.com/`.

[Saa06] O. S. Saar. *Dynamics in the Practice of Structural Design*. WIT Press, 2006.

[SH97] H. Sawada and S. Hashimoto. Gesture recognition using an acceleration sensor and its application to musical performance control. *Electronics and Communications in Japan, Part III*, 80(5):9–17, 1997.

[SI97] David Small and Hiroshi Ishii. Design of spatially aware graspable displays. In *CHI '97: CHI '97 extended abstracts on Human factors in computing systems*, pages 367–368, New York, NY, USA, 1997. ACM. `doi:http://doi.acm.org/10.1145/1120212.1120437`.

[SJ05] Raymond A. Serway and John W. Jewett. *Principles of physics: a calculs-based text.* Thomson Learing, fourth edition, 2005.

[SPHB08] Thomas Schlömer, Benjamin Poppinga, Niels Henze, and Susanne Boll. Gesture recognition with a wii controller. In *TEI '08: Proceedings of the 2nd international conference on Tangible and embedded interaction*, pages 11–14, New York, NY, USA, 2008. ACM. `doi:http://doi.acm.org/10.1145/1347390.1347395`.

[TP05] Chin-Woo Tan and Sungsu Park. Design of accelerometer-based inertial navigation systems. *Instrumentation and Measurement, IEEE Transactions on*, 54(6):2520–2530, Dec. 2005. `doi:10.1109/TIM.2005.858129`.

[Tro07] Andrew Troelsen. *Pro C# 2008 and the .NET 3.5 platform.* Apress, 4 edition, 2007.

[TTR+05] Suga T., Akamatsu T., Kawabe R., Hiriashi T., and Yamamoto Y. Method for underwater measurement of the auditory brainstem responce of fish. *Fisher Science*, 5(71):1115–1119, 2005.

[TW04] David H. Titterton and John L. Weston. *Strapdown inertial navigation technology.* Institution of Electrical Engineers, second edition, 2004.

[WPZ+09] Jiahui Wu, Gang Pan, Daqing Zhang, Guande Qi, and Shijian Li. Gesture recognition with a 3-d accelerometer. In *UIC*

'09: Proceedings of the 6th International Conference on Ubiquitous Intelligence and Computing, pages 25–38, Berlin, Heidelberg, 2009. Springer-Verlag. `doi:http://dx.doi.org/10.1007/978-3-642-02830-4_4`.

[WWQ+06] Rory P. Wilson, Craig R. White, Flavio Quintana, Lewis G. Halsey, Nikolai Liebsch, Graham R. Martin, and Patrick J. Butler. Moving towards acceleration for estimates of activity-specific metabolic rate in free-living animals: the case of the cormorant. *Journal of Animal Ecology*, 75(5):1081–1090, September 2006. `http://dx.doi.org/10.1111/j.1365-2656.2006.01127.x`, `doi:10.1111/j.1365-2656.2006.01127.x`.

[Zuk05] John Zukowski. *The definitive guide to Java Swing.* Apress, 2005.