School of Technology

OXFORD
BROOKES
UNIVERSITY

MASTER OF SCIENCE DISSERTATION

Title: ____________________

____

____

Surname: ____________________

First Name: ____________________

Supervisor: ____________________

Student Number: __________ Date Submitted: __________

Module Number: __________

Course Title: ____________________

OXFORD
BROOKES
UNIVERSITY

Statement of originality

Except for those parts in which it is explicitly stated to the contrary, this project is my own work. It has not been submitted for any degree at this or any other academic or professional institution.

__________________ __________________

Signature of Author Date

Regulations Governing the Deposit and Use of Master of Science Dissertations for the Department of Computing and Electronics, Oxford Brookes University.

1. The 'top' copies of projects submitted in fulfilment of Master of Science course requirements shall normally be kept by the Department.
2. The author shall sign a declaration agreeing that, at the supervisor's discretion, the dissertation may be submitted in electronic form to any plagiarism checking service or tool.
3. The author shall sign a declaration agreeing that the dissertation be available for reading and copying in any form at the discretion of either the project supervisor or in their absence the Head of Postgraduate Programmes, in accordance with 5 below.
4. The project supervisor shall safeguard the interests of the author by requiring persons who consult the dissertation to sign a declaration acknowledging the author's copyright.
5. Permission for anyone other than the author to reproduce in any form or photocopy any part of the dissertation must be obtained from the project supervisor, or in their absence the Head of Postgraduate Programmes, who will give his/her permission for such reproduction only to the extent which he/she considers to be fair and reasonable.

I agree that this dissertation may be submitted in electronic form to any plagiarism checking service or tool at the discretion of my project supervisor in accordance with regulation 2 above.

I agree that this dissertation may be available for reading and photocopying at the discretion of my project supervisor or the Head of Postgraduate Programmes in accordance with regulation 5 above.

__________________________ __________________

Signature of Author Date

Finger Tracking Desktop Experience

Sylvain BLOT
07086826

MSc in Computer Science
Oxford Brookes University

August 2008

Responsible	**Supervisor**
Jon Rihan Computer Vision Group	Doctor Philip Torr Computer Vision Leader

Finger Tracking Desktop Experience

MSc Computer Science Dissertation Proposal

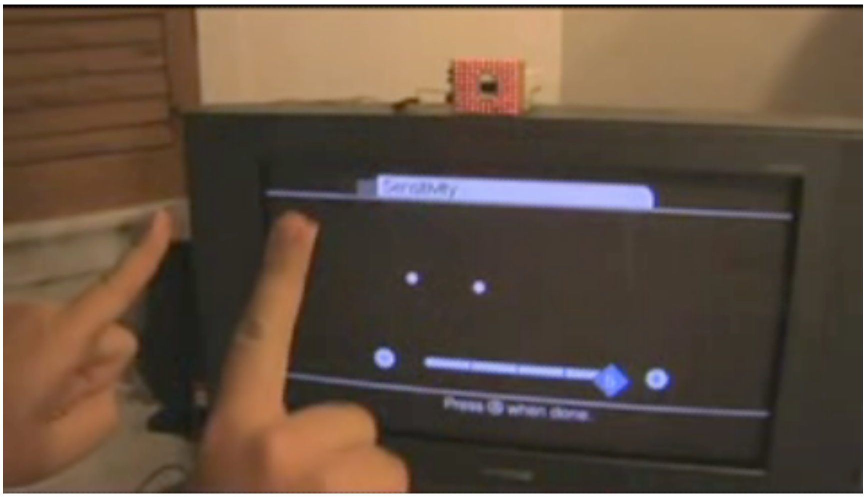
Sylvain BLOT
07086926

Table of Content

1.Introduction

If one where to compare the first personal computer, that appeared in 70's, to the computers used today, one would notice many similarities; such as, the screen, the keyboard, and the mouse. People have interact with the same devices for almost 40 years.

In 40 years a lot of new technologies have been produced to offer new ways to interact with a computer including body sensors and cameras. However, they are not currently used to interact with the desktop.



This failure, in progress, in computer interaction with humans, may retain in the fact that other methods are hard to setup, expensive, or designed to only offer a pointer or a keyboard.

My dissertation will focus on offering a cheap easy way to use "in the air" mouse and keyboard solutions by creating an IR source on fingers and trap them with an IR camera.

The key issues of this dissertation will be :
✦ Create a driver for the IR camera
✦ Modify the GnomeToolKit to enable virtual keyboard on screen on input boxes...
✦ Create a daemon to transform finger gesture to X11 calls
✦ Provide finger gesture customization

The final work will be a study about defining a computer/human interaction, split in two parts:
✦ The technical : A working multi-finger "in the air" desktop environment
✦ The theoretical : How to define a human/computer interaction scheme

2.Rational for choice of project

This project will relate the AI module, by creating a gesture recognition system. For example, the user will be able to create movement shortcuts to launch applications. This might be done by a neuronal network.

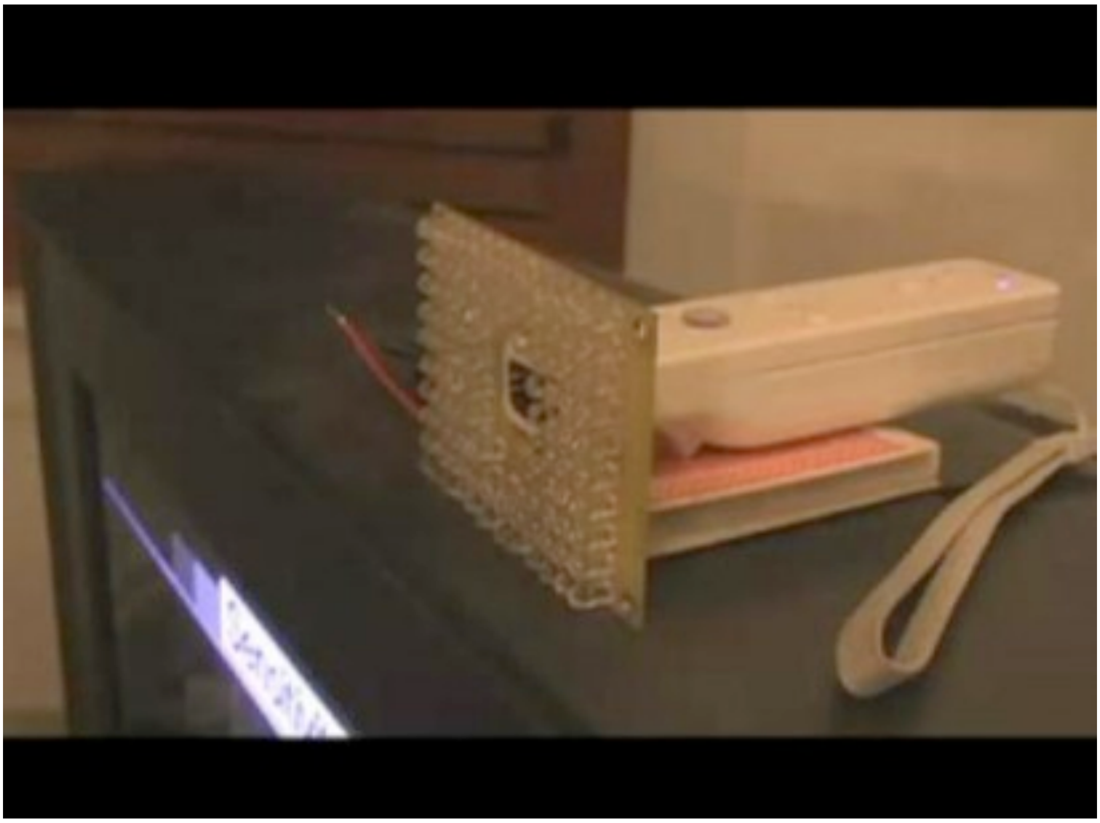
The work will be based on the Linux operating system which i know very well. I have 2 years background as linux trainer that will help me to concentrate only on the functional aspect of the project and not the OS understanding (to make the drivers for example).

Choosing Finger tracking as subject gives me the opportunity to prove I can imagine and realize projects on Linux. This will help me in order to find a job as consultant or developer in a linux company to make linux more user friendly.

3.Objectives

The main purpose of the dissertation is to create a multi-pointer device drivers based on a wiimote hack and to implement gesture recognition.

The input device is an original idea of PhD Johnny Chung Lee [1] and looks like :



An array of LED send InfraReds to the user on front of the sceen who reflects the IR rays with reflecting paper on fingers to the wiimote IR camera. The first task will be to build the LED array.

To develop the software interaction with the wiimote we could make a call to the library wiiuse. The second task will be to write a program to display finger positions on screen, to do this we will need to write a calibration program (screen size, distance from the screen, etc...).

Then when we will have a good approach of the wiiuse library, the gesture will be implemented, the main task will be to convert moves into data able to be saved and known by the recognition system This task should take the longest and the most complicated as there is lot of math and a AI to develop.

Then all the graphical work should be done,
To do :
• Trap interaction with elements of the graphical Toolkit : text boxes, date chooser, calendar etc... Also to enable interaction with our multi-touch input device or for example to enable on screen keyboard.

- Create a program that takes advantage of our pointing device like an album photo where photos are in stack on the screen and can me manipulated with fingers, like in real life on a table.
- Possible adapt controls of a game that works with fingers like a 2 player pong...

To develop all of this features we will need an open system to be able to develop either a drivers or hack a graphical library. So Linux is a perfect choice for this reason, note that the wiiuse library is available for Linux and Windows, and soon for mac.

4.Method

To accomplish my objective, I will need to build the LED array as soon as possible to begin the driver development.
Look at the linux kernel input device API.
Implement the driver.
Create a neuronal network.
Do all the GUI programming

To test and verify the system, I hope you will only have to seat down in front of screen, setup and calibrate the wiimote. Then you will have to move your finger interact with the desktop.

5.Resources

To build the IR finger tracking system we need :

A Wiimote	already owned
20 IR LEDS	8P
a circuit	3P
some resistances	1P
9V batteries	5P
Reflecting paper	1P
Total	18P

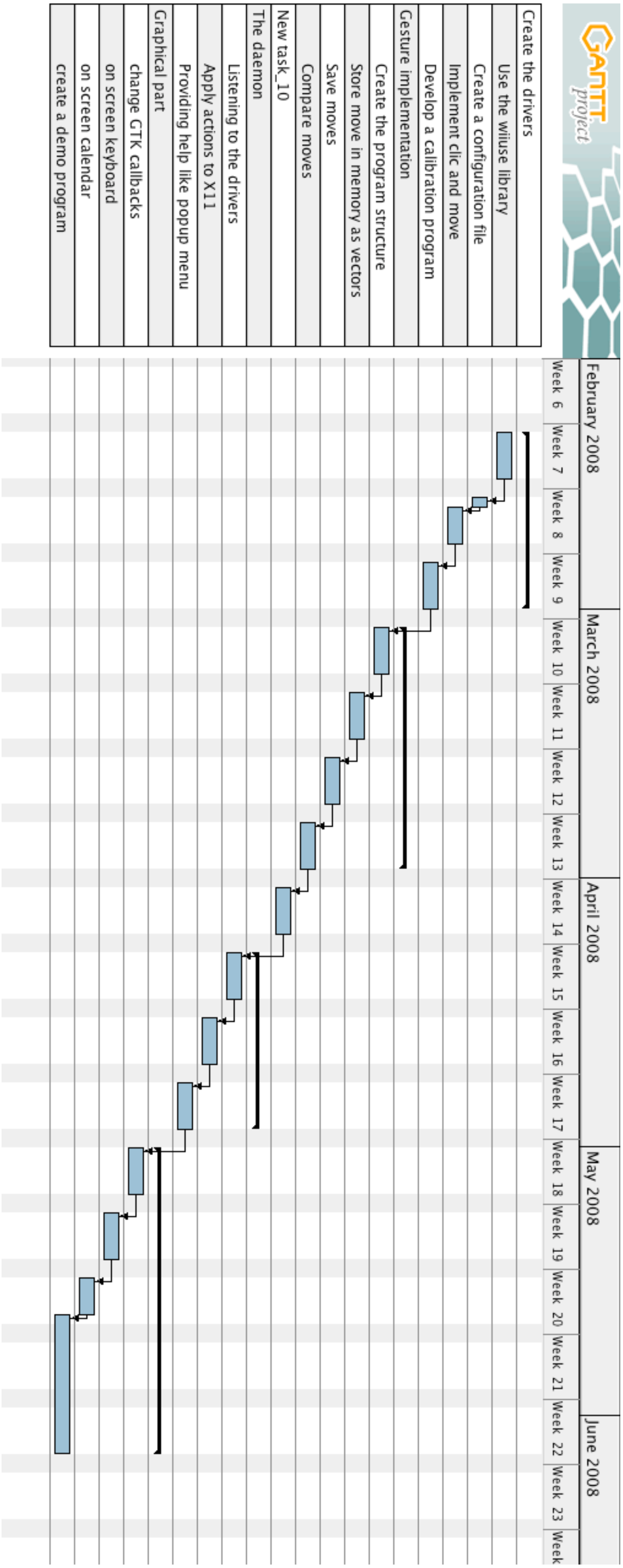
The total budget for this dissertation is 18P.

The development will be done on my laptop which support bluetooth to communicate with the wiimote.

If we enccounter any problems with the wiimote we can count on the online community http://www.wiimoteproject.com/.

For the rest of the dissertation I have enough skill on python and GTK to do it by myself. The hardest part will be the AI, I am in the AI lectures to know more about this.

6.Schedule



Create the drivers
Use the wiiuse library
Create a configuration file
Implement clic and move
Develop a calibration program
Gesture implementation
Create the program structure
Store move in memory as vectors
Save moves
Compare moves
New task_10
The daemon
Listening to the drivers
Apply actions to X11
Providing help like popup menu
Graphical part
change GTK callbacks
on screen keyboard
on screen calendar
create a demo program

7.Reference and Bibliography

Wiiuse, the library to interact with the wiimote : www.wiiuse.net
The full wiimote specifications http://wiibrew.org/index.php?title=Wiimote
Johnny Chung Lee Website: http://www.cs.cmu.edu/~johnny/
Wiimote community http://www.wiimoteproject.com/
Neural network gesture recognition http://www.dcs.gla.ac.uk/~jhw/nn.html

Abstract

This report presents a method of developing a finger tracking system in order to point and click on screen. It will ascertain the correct integration of software and hardware in order to make system as natural and as user friendly as possible. This led to the development of a gesture recognition system designed to enable quick shortcuts to programs and mimic keystroke functions. To obtain said objective, it was necessary to develop a driver to retrieve the user's fingertips from hardware, and to maintain the mapping between the points captured and their human counterparts, ie, the index finger that determines where the cursor is pointing.

The second part consists of specifying the functioning of the recognition software that will make use of artificial intelligence. This will be accomplished by defining training examples and then repeating them to train the neural network. This choice implies setting up the neural network with a finite number of gestures, consequently limiting the system's flexibility. This also sets a limit on the programs potential for adaptability.

All the work previously undertaken will be integrated into the Gnome desktop environment. This includes the development of the drawing screen designed for gesture recognition, as well as an icon status for the wireless connection to the Wiimote and the configuration of the mapping of human movement and the program triggered or keystroke simulated.

The report concludes with a test scenario of the final solution by four impartial subjects. The findings are thereupon discussed and analyzed, proving that the platform works to a certain degree of success.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Computers have evolved by improving most of their components, for example, in following more or less the Moore law, processor speeds have been increased by a factor of 100 000 since the nineteen seventies. Other components of the mother board follow a constant progression as well as the memory, the storage capacity, the main output device, the screen size/resolution, etc.

This paper will look in particular at the state of input devices: the keyboard and mouse pair. Mice have not changed very much: having begun as a mechanical operation they now function on an optical and wireless system, however, they still require a flat surface in order to track movement. The proposed evolution is to liberate the mouse from any grounded surface and give it the capacity to compute its own position in free space.

Many solutions can be used to achieve this new degree of freedom, such as a webcam associated to software able to track fingers or eyes, but the question is how to point on screen without completely changing the user-computer input paradigm?

The purpose of this paper is to find and develop a solution to point on screen simply by moving your hands in the air. To achieve this, it involved the development of a mouse driver able to point on the screen via user posture recognition through a hardware device, and a gesture recognition system designed to simplify action triggering, such as launching a program or simulating a keystroke (control+s to save for example).

The expected result is a significant improvement in the user-computer relation paradigm, exploring in depth the way users interact with the interface, point and click, explore menus, etc.

This subject is challenging and exciting because it demands different user skills. In the first place, a device to point on screen will have to be specified and created, then the programming language must be found to develop a working system. Finally, recognizing movements will present an ideal opportunity to greater explore the artificial intelligence lecture followed this year.

This paper will be divided into two parts; the first will be related to all research made to find a suitable hardware and software combination to point on screen, beginning by describing different technologies and studies. It will then focus on defining the specification of a realistically feasible system.

The second part will focus on the implementation of such a system by outlining all the important steps of the development process, finishing by testing the software with the help of some beta testers to reflect the obtained degree of achievement.

Chapter 2

Research

2.1 Approach

Tracking fingers in order to point on the screen can be done in many ways, as will be illustrated further on in this paper. Techniques are different, but the steps involved in the tracking process remain the same, even if the task can be accomplished using hardware or software:

1. Capture the environment
2. Find fingers
3. Distinguish fingers
4. Understand moves
5. Apply recognized action

The first step, **capturing the environment** can be done by any kind of video camera; the device will capture a defined number of frames per second. This number is very important because it will define the on-screen pointer refresh rate. To make a parallel with mice, a mouse sensitivity is defined by two criterions : the polling rate (in Hz) and the precision (in DPI). The polling rate corresponds to the number of frames per second captured by the camera and the precision by its resolution. The sensitivity is important in how it influences your chances of landing in the pixel you wanted. The key point in finding a suitable device is a good tradeoff between the resolution and the updating frequency.

Once the frame is captured, the next step will be to **find fingers** inside it; this task can be proceeded either by software or hardware. If it is done by software a tracking algorithm will be used, watching over contours or blobs after applying a filter to the input.

Distinguishing finger is a task consisting of creating a software representation of hands that will be mimicked every frame or whenever the number of found fingers changes. Caching could be useful in reducing the elapsed time it takes to realize this task as it could involve too high a level of computation time . It is imperative that every task embarked upon take the shortest time as possible in order for the driver to be faster than the capturing rate, thus avoiding (for example) delays in the mouse path.

The **understanding move** step will then try to figure out, in conjunction with knowledge of previous posture, the desired action being gestured. It can be simple move to point at an area or click (left or right) or even to zoom or trigger a programmed action.

Apply recognized action will lie in controlling the pointer on screen for moving, clicking, scrolling or launching the configured action that was triggered.

Now that the process is specified, we will try to find the best hardware and software balance to satisfy these prerequisites.

2.2 Choosing the hardware

As previously highlighted, the choice of hardware will be dictated by the best trade off between the video capturing resolution and the number of frames per second. In addition, other capacities can be taken into consideration like embedded tracking facilities. . .

2.2.1 Webcam Based Video tracking

Webcams are now embedded on top of almost every screen pointing at the user constantly. It is natural that using them as a pointing device could be a great improvement for human/computer interaction without adding extra hardware to the actual computer.

In August 2008, most of the webcams available on sale have a frequency of 30 frames per second (fps), with less costly models from 15fps. The highest resolution webcam found has a 1.3 megapixel capacity, able to produce an image with a resolution of 1280*760 by interpolating from sensors captured data sized 640*480.

Webcams are sensible to lightning conditions, making them unusable in the dark. In addition to this, detecting finger tips efficiently requires that the user stay close to the sensor. If the user steps away too far from the sensor, the fingers will then be too small to detect individual finger movement.

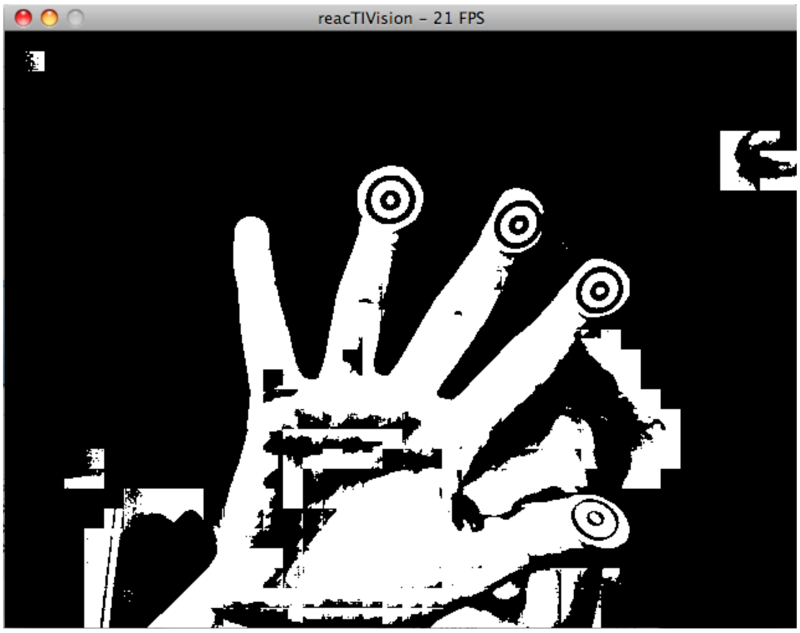
Track Fingers

In order to track fingers from camera, it is necessary to gain access to the device, to apply a filter to the video and to then process each frame to detect first the hand and secondly the fingers. All the steps of this process can be implemented using the OpenCV library described by Sigurdsson & Wong (2008). To detect the hand it is obligatory to subtract the background so that the hand will be the largest connected component of the image. The fingers must thereafter be distinguished using an estimation though Kalman filtering. Moreover, a strict method of using the system will have to be applied in order to distinguish finger movement properly, avoiding using a 3D representation of the hand that would increase the code complexity and could not be implemented in 2 months.

All the algorithms are part of OpenCV Library which is natively available for the C++ programming language but also interfaced with ch, ipp, mathlab and python.

Track Fiducial

Figure 2.1: Fiducial markers tracking using the Reactivison framework



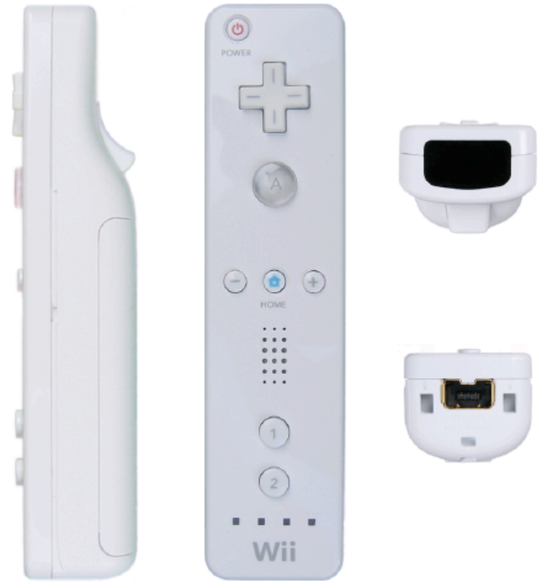
Rather than tracking fingers which is not as accurate as expected, another approach could be tracking fiducial markers stuck to the phalanx. This will avoid tracking the hand and concentrate the work on finding the fiducial markers, moreover, individual fiducial markers can be used on each finger. Distinguishing between fingers will thus be simplified.

This approach is used by Jorda (2005) for the Reactable project: "The reactable is a collaborative electronic music instrument with a tabletop tangible multi-touch interface." To track fiducial markers a library named the reactivision framework was created (presented by Kaltenbrunner & R. (2007)), and can be setup to employ user defined fiducial markers: their coordinates can then be retrieved and used to point on screen.

This solution offers fast user detection and sends back the coordinates through the TUIO protocol Kaltenbrunner et al. (2005). Unfortunately at the moment the protocols do not distinguish between fingers, although this feature will be available in the future version 2 release (information obtained after email exchange with Ph.D. Martin Kaltenbrunner, responsible for the protocol development).

2.2.2 The Nintendo Wiimote

Figure 2.2: The Nintendo Wiimote

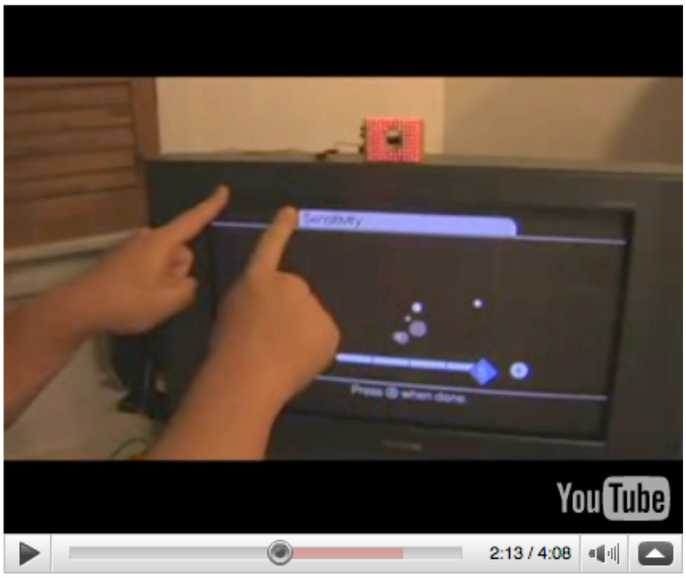


The Nintendo Wiimote is the game controller designed to work in conjunction with the Nintendo Wii game engine. It features embedded buttons, a motion sensor, an IR sensor and communicates with the Wii through a bluetooth link. The motion sensor measures acceleration force and gives relative positioning and movement of the Wiimote in space. To have a better idea of where the Wiimote is in space, the IR sensor tracks a device called the sensor bar. About 20cm in length, it features 2 IR leds in both extremities; these 2 fixed points in front of the player permit the Wii to know where exactly the Wiimote is, thus enabling on screen pointing and motion analysis.

This concentrated technology for less than $30 became quickly a most wanted device for many hackers of any kind of project, like scientists measuring force using the 3-axis linear accelerometer.

To handle the communication with computers, drivers have been developed and are available in almost any language. The driver implementation functions using the following scheme: asking for a connection to the Wiimote (this requires putting the Wiimote into the discovery mode by pressing simultaneously the buttons 1 and 2, and then asking for a report of the states of all the components retrievable at maximum frequency of 100 times per second).

Figure 2.3: Finger tracking using an IR led array and reflective tape on fingers



The most important component of the controller for this paper is the IR camera which, thanks to the PixArt System-on-a-Chip, processes the image to extract a maximum of 4 points and maintain their position with a resolution of 1024*768 pixels.

Ph.D. Johnny Chung Lee idea

Ph.D. Johnny Chung Lee (2007) had the brilliant idea of surrounding the Wiimote with IR leds pointing in the same direction as the camera. By placing the Wiimote in the direction of the user, IR light emitted by the diodes can be reflected back to the IR sensor. He proved that it worked with merely his fingers, however it is somewhat too diluted as can be seen in the figure below. To improve the signal quality, he suggested adding reflective tape on whichever fingers are expected to be tracked. The result is a convenient solution to track up to 4 fingers in space.

Another solution could potentially be used: rather than reflecting light, simply emit light from your fingers, which can be done with customized gloves featuring added IR leds at the extremities of four chosen fingers.

2.2.3 Comparing

In order to find which of the previously presented technologies best fit the need to create a cheap, accurate finger solution, here is a summary of their features:

The best tradeoff involving the least effort regarding development is the Nintendo Wiimote. It will be required before using it to take care of unwanted sunshine, to

Table 2.1: Compare capturing solution properties

	Resolution	Update Rate	Max. trackable fingers	Lightning condition sensitivity
Webcam finger tracking	640*480	30fps	10	Not if using IR light in dark and skin color adaptive filter
Webcam fiducial markers tracking	640*480	30fps	10	As long as fiducial can be seen
The Nintendo Wiimote	1024*768	100fps	4	Sensible to sunshine

avoid rays pointing in the direction of the camera.

2.3 Gesture

Now that the solution for finger tracking has been chosen, it becomes necessary to look at a gesture recognition system capable of considerably enhancing user interaction, simplifying complex keyboard actions with an easy movement. Keystroke combinations used for launching applications or defined actions within applications can be replaced with simple gestures.

Some approaches may be considered:

- A Hidden Markov Model
- A Neural Network
- Pattern Matching

The chosen approach will be to have a defined number of gestures, and to classify input data using a neural network. This will be a quicker solution as long as the neural network is correctly trained to achieve a good degree of generalization.

Another complicated problem lies in defining a way to figure out at what moment a gesture starts and stops to record the neural network inputs.

2.4 Technical choices

Most of the time, choosing a programming language is dictated by efficiency, portability or the availability of libraries, but for this project the major decisive factor is productivity as the programming tasks will be carried out in a month. This is the reason why an interpreted object oriented programming language will perfectly fit the needs : Python.

Python offers fast prototyping, runs on Windows, Linux and MacOS, and offers bindings to most of the major C library (pthread...) as well as to graphical interfaces library.

Moreover, a Wiimote driver is available under the GPL license developed by Stéphane Duschesneau (n.d.) for the Wii Whiteboard project.

All the programming cycles will be done under GNU/Linux running the Gnome desktop environment. One of the goals will be to fit the program properly inside the user interface by using the integrated graphical toolkit: GTK. Through the use of PyGTK, the python binding to GTK will be made possible.

The last required component will be the neural network library. Like all the technical choices previously decided, the chosen solution will be open-source and accessible in python : FANN, for Fast Artificial Neural Network (Nissen 2003).

Before investigating any further some time will be spent on learning the python programming language by reading Tarek Ziadé (2006) book.

Chapter 3

Method

3.1 Design methodology

The entire development process can be divided into multiple independent single tasks, for example the pointer driver can be developed without requiring the graphical user interface.

The method used divides the work into smaller simple tasks. The first one will be to create the glove and the infrared array to make trackable the fingers by the Wiimote. Then a standalone driver will be implemented to point on screen. This task will also be divided into small steps; distinguish fingers, then implement clicking and finish by zooming.

Then, a graphical user interface will be developed as a skeleton for the program. The GUI will control everything, so first it should bind the driver to start the connection to the Wiimote. Next a threading model will have to be implemented to allow the GUI to operate when the driver is running. After that the window to input gesture will be developed and attached to the driver to retrieve the points to draw.

At this point everything will be ready to start implementing the recognition system that will retrieve the input from the driver. The first step will be to create the input example set for the training neural network. The neural network is then trained with data and saves. The next step will be to create the configuration of gesture binding to actions.

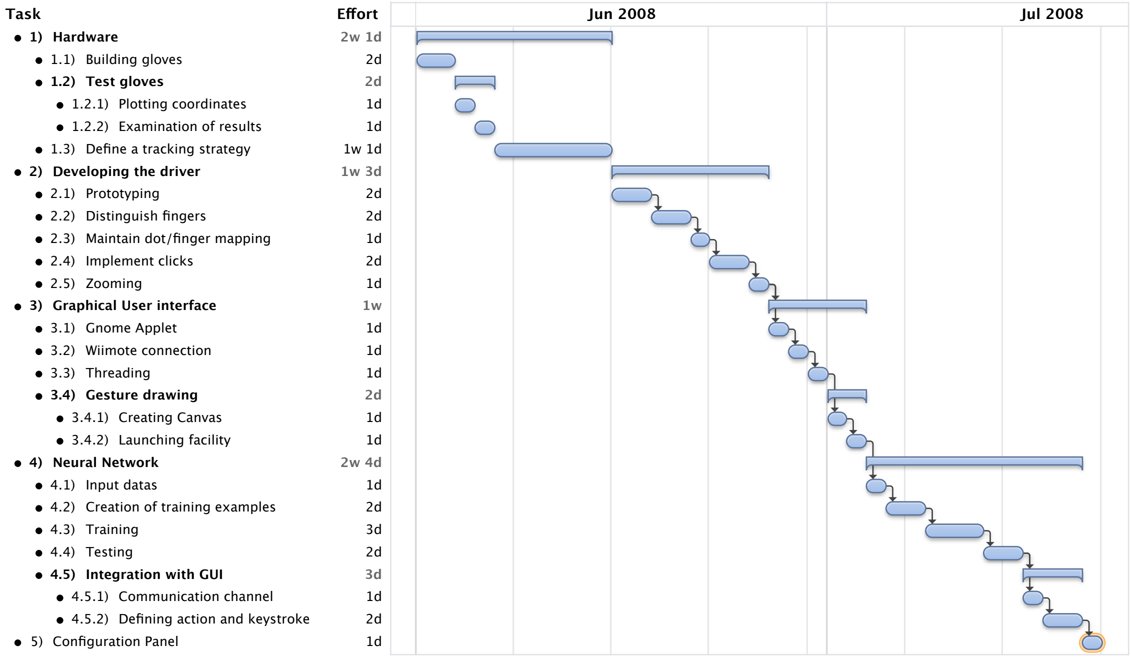
The last step will be to permit the graphic configuration in order to let the user set up the program.

This design ensures a development process that separates tasks by profiting from the oriented object programming facilities.

The program will be called and referenced as **Fiingers**, to make a contraction of Wii and fingers.

3.2 Project Management

Figure 3.1: Development Gantt Chart



Task	Effort
• **1) Hardware**	2w 1d
• 1.1) Building gloves	2d
• **1.2) Test gloves**	2d
• 1.2.1) Plotting coordinates	1d
• 1.2.2) Examination of results	1d
• 1.3) Define a tracking strategy	1w 1d
• **2) Developing the driver**	1w 3d
• 2.1) Prototyping	2d
• 2.2) Distinguish fingers	2d
• 2.3) Maintain dot/finger mapping	1d
• 2.4) Implement clicks	2d
• 2.5) Zooming	1d
• **3) Graphical User interface**	1w
• 3.1) Gnome Applet	1d
• 3.2) Wiimote connection	1d
• 3.3) Threading	1d
• **3.4) Gesture drawing**	2d
• 3.4.1) Creating Canvas	1d
• 3.4.2) Launching facility	1d
• **4) Neural Network**	2w 4d
• 4.1) Input datas	1d
• 4.2) Creation of training examples	2d
• 4.3) Training	3d
• 4.4) Testing	2d
• **4.5) Integration with GUI**	3d
• 4.5.1) Communication channel	1d
• 4.5.2) Defining action and keystroke	2d
• 5) Configuration Panel	1d

3.3 Hardware: Glove design

3.3.1 Positioning LEDs

Before building the gloves, it was necessary to define how the 4 leds would be placed and used in order to point, click and zoom on the desktop. After some discussion and a survey completed by potential users, it was decided to use two gloves, one with three leds (the "pointing hand") and the other with the remaining last led (the "zooming hand").

The pointing hand will be tracked in order to point on screen and click: the natural choice for the pointing finger is the index used to indicate a way to follow in real life or to point at something or someone. The two other leds will be placed in the thumb and the middle finger, to simulate more or less the use of a mouse in the air; the thumb will trigger a left click and the middle finger a right click. With this layout, moving the hand in the air to point will be close to moving a mouse on a surface and pointing will be more natural.

The left hand, aka "zooming hand", will have a led positioned on the index finger. When this light is detected the driver will switch into the zooming mode, whereupon the distance between the two index fingers will become the zoom scale. The presence of this hand is completely unnecessary for a classic use of the system.

3.3.2 LED choice

As described before, the use of a Wiimote on a computer has been reversed engineered, so there is no information on the IR camera sensitive to IR light. The wavelength of Ir light oscillates from 780 nm to 1 000 000 nm, making the choice of the appropriate led not particularly easy. In addition to this, their are led models with a viewing angle from 5° to 160°, a voltage from 1,3V to 13V, etc.

After 3 tries the appropriate model was found, the model TSAL6100 from www.vishay.com which has been selected:

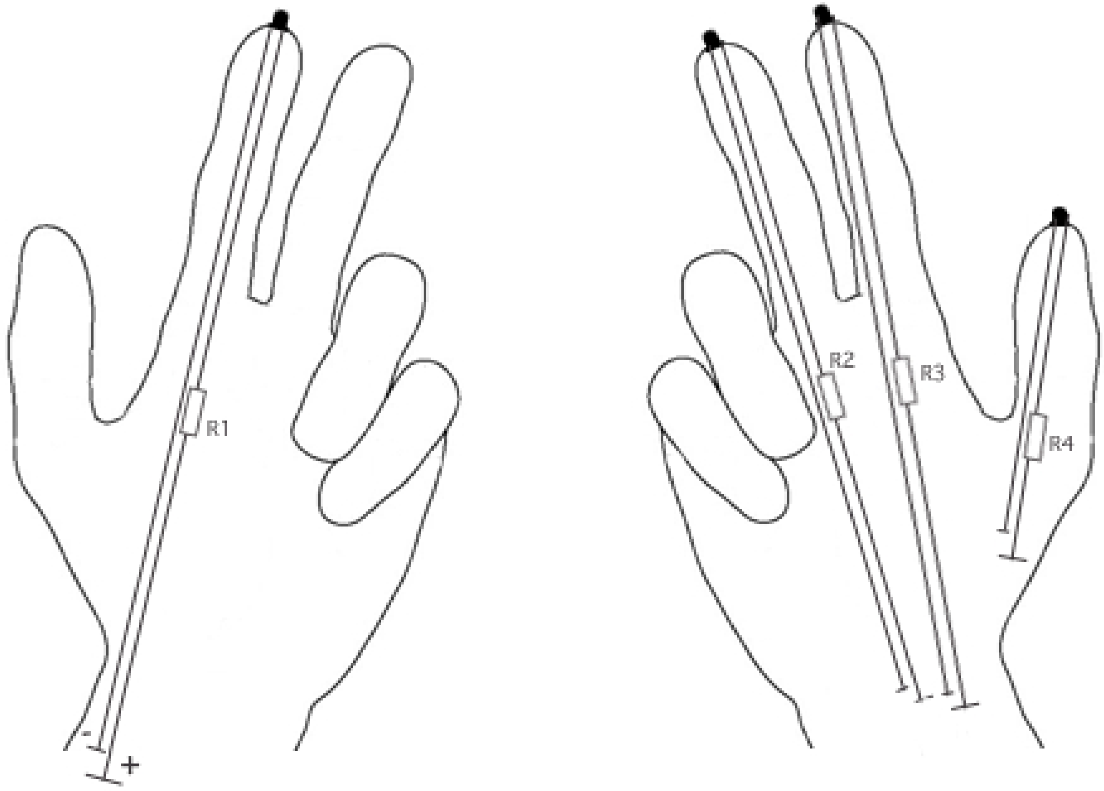
Reverse voltage :	5 V
Forward current :	100 mA
Peak wavelength :	940 nm
Viewing Angle :	20 °

Most of the problems encountered during previous tests were due to a too low forward current. The result was a point not trackable at more than 30cm away from the Wiimote. Now with TSAL6100 leds points are trackable up to 3 meters away.

3.3.3 Building Gloves

A simple circuit is used to power led, with each led being completely independent and having its own resistance, and finally a circuit that is in parallel. For the development, the power supply comes from the wall socket, meaning that there are two large cables coming out of the glove. This can be replaced by individual batteries inside each glove. The following figure shows the circuit inside the glove; evidently it would entail a considerable amount of positive and negative wire connected together to make two cables connected to the power supply.

Figure 3.2: Transparent view of the IR gloves



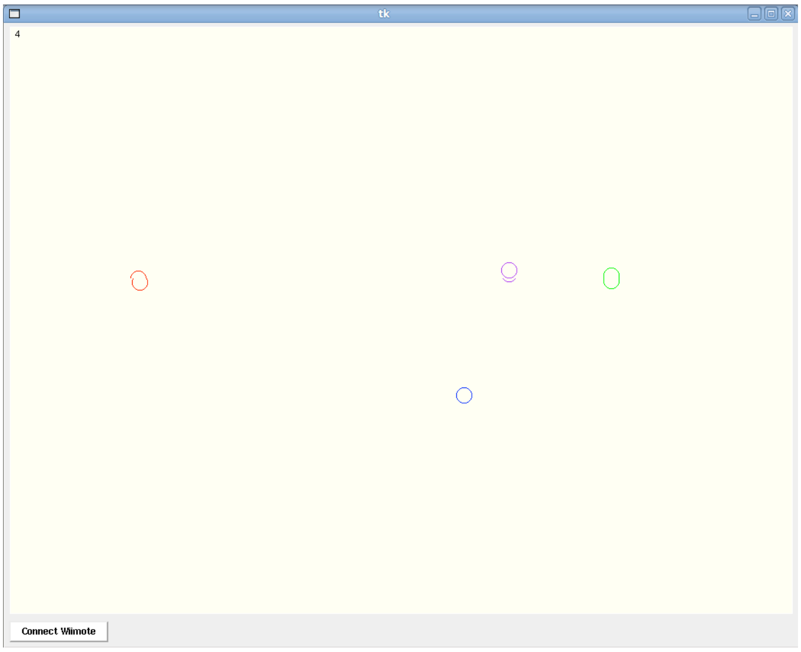
3.3.4 Testing gloves

To test the gloves and have a better idea of how the Wiimote keeps track of leds, a small application was developed using the TK graphical interface library:

This GUI helped to figure out how leds are tracked and some assumptions can be made:

- If the sunshine point directly to the Wiimote, points appear and disappear in an unmanageable way, so it's really important to use this software before using fiingers to place the wiimote correctly to be sure that no sunshine are captured, switch the gloves off and look at the software window if any points appear on screen. Beware that reflected sunshine coming from white surface can also affect the reception.

Figure 3.3: Finger drawn on a window



- If the led are close to each other the two points become one unique point. This is true for a spacing lesser than 1 cm at 50 cm away from the camera. Distinguishing index and middle finger is not possible when the glove are more than 3 meters away from the sensor
- When 2 points became too close from each other they can swap.
- The Wiimote best track point when 4 led are detected
- The sensor capture angle is 45 °

All this information will be helpful in developing the pointing driver which has to take into consideration all these assumptions.

3.4 Pointing driver

3.4.1 Infinite Loop

First of all, the driver will be launched only from the command line, the main function instantiate the MouseDriver class, then called the connectWiimote method and the run method. This last call will be threaded in the future.

The method used to constantly get new coordinates and turn them into actions will be developed using an infinite loop, following this algorithm :

```
while running
        get the points coordinates
        if the number of points is greater than 2
                if the number of points found have changed
                        maintain points
                        caching maintained state
                else
                        apply caching mask
                map points to hands
                look for click
                        left
                        right
                look for zoom if 4 points detected
                warp pointer
```

The code has been optimized to work at the same rate as the Wiimote driver, so it also loops at 100Hz.

3.4.2 Distinguish point

At the beginning of the loop all the visible points are stored inside an array including their x and y value and their Wiimote discoverer index.

```
class Point ():
    def __init__(self, pos, $x$, $y$):
        self.pos ← pos
        self.$x$ ← $x$
        self.$y$ ← $y$
```

To distinguish fingers the array is sorted using a custom algorithm passed in argument to the array *sort* method: *points.sort(self.maintain)* where *self.maintain* is the sorting algorithm.

```
def maintain (self, el1, el2):
    deltay ← el1.$y$ − el2.$y$
    if abs(deltay) $< 30$:
        if el1.$x >$ el2.$x$:
            return $-1$
        else:
            return $1$
    else:
        if deltay $> 0$:
            return $1$
        else:
            return $-1$
```

This function will be called to sort the 3 items long array, including the thumb, the index and middle finger. This algorithm sort points two by two (*el1* and *el2*) and is called as many time as required to have proper sorted data. Returning **−1** mean that *el1* will be at the left of *el2* in the array and returning **1** that *el1* will be at the right of *el2*.

The algorithm consists of a simple comparison of coordinates in two dimensional space; if *deltay*, which represents the difference in ordinates between the two points, is upper than 30 pixels it means that the thumb is one of the two points: the point which has the lowest ordinate value will represent the thumb; if *deltay* is lesser than 30 pixels it means that the two points present are the index and the middle finger. The lowest abscissa value will represent the index and the the greatest the middle finger.

To work properly, it is important that the pointing hand is in a normal state (not clicking). The consequence is that if points need to be sorted while clicking it may pause the action to maintain the finger again.

3.4.3 Caching

Even if the sorting algorithm is as simple as possible, it costs time to execute it, which is why when a consistent state is found, the sorted order is kept inside an array to map direct points to finger tips.

This is done by a single line of code:

```
caching[:] ← [elem.pos for elem ∈ points]
```

This line feeds the caching array with the previously sorted Wiimote index.

3.4.4 Clicks

Clicking while moving is not a required feature as far as the application is designed to use the desktop and not play games where reflexes are important. This distinction made, to click it is required to stop moving. Every time the distance travelled between two iterations is lesser than 1.5 pixels a counter is started and increases by one. When the counter reaches 400 clicking will be possible and the coordinate of the thumb and the middle finger are saved to detects clicks.

Left click

The left click is done by the thumb and consists of moving it to the right. The total distance traveled by the thumb since the index stopped moving is computed to know the state of the click and also the relative angle to get the direction of the move (ascending or descending phase).

Right Click

The same principle is used for right clicking, the only difference being the angle test because the move is not left to right and then right to left but down and up.

3.4.5 Zooming

Zooming is only possible when both hands are present. To work properly, the user must work with one hand and then introduce the second. Zooming is not a default feature of the X server and is available through the use of *compiz* a compositing window manager which will replace the default window decorator: *metacity.*

To be able to zoom, the requirements are:

- A fully accelerated graphic card
- Compiz
- Setup Compiz to zoom in with following binding
  - Zoom In : Super + Scroll Up
  - Zoom Out : Super + Scroll Down

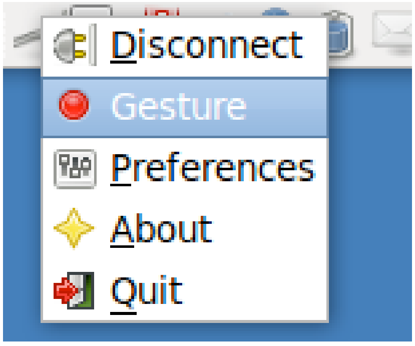
3.5 Graphical Interface

The graphical interface is the part that will put everything together into a single software. It will manage the connection and the disconnection with the Wiimote, start the pointing driver and the recognition system, and also provide a configuration panel to map gestures to actions.

The graphical toolkit will be PyGTK, because it best fits with the Gnome Desktop environment. Moreover, as GTK is available on Windows and MacOS as well, porting the software will not be restricted by the graphical layer.

3.5.1 The point of control

Figure 3.4: The Gnome Status icon Menu

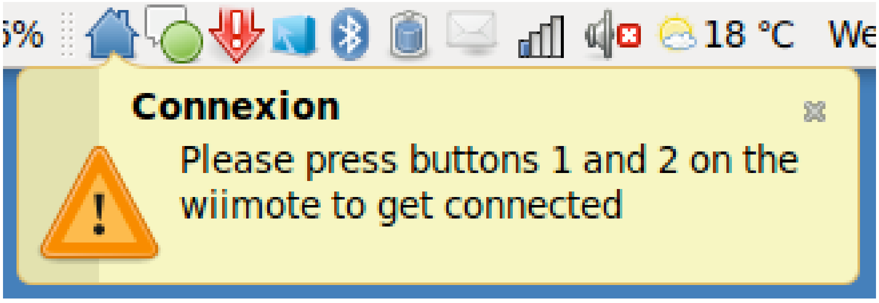


The program will be as integrated as possible, so i will be a simple status icon located inside a notification area on the Panel. When clicking on it, it will create a popup menu to choose the desired actions:

- **Connect:** This will create an instance of the pointing driver, and call its *connectWiimote* method. The communication with the user will be provided by notification popups, the first one to ask the user to press the button 1 and 2 on the Wiimote to make is discoverable and the second one to print the result of the request. If the connection is established, the driver will be started and this item will not be displayed anymore, to be instead replaced by a disconnect button.
- **Disconnect:** activating this item will stop the bluetooth connexion, only available when previously connected.
- **Gesture:** This item is only shown if the connection with the Wiimote is established, and will switch the driver into the gesture mode and display a fullscreen windows to draw a gesture on screen.
- **Configuration:** Used to map finger gesture with actions.

- **About:** Draw a simple box, describing the functionalities.
- **Quit:** stop the bluetooth connection if necessary and quit the program.

Figure 3.5: Status icon popup example



Connexion

Please press buttons 1 and 2 on the wiimote to get connected

3.5.2 Communication

All the communication between modules is implemented using signals, for example, the driver in gesture mode emits a signal to the gesture window to draw the finger. To define the signal, the *gobject* class is subclassed to use its properties, their type, how to access them and how to set them.

Following a simplified version of the class defining signals:

```
import gobject

class Signal (gobject.GObject):
    __gproperties__ ← {
        'gesture': (gobject.TYPE_BOOLEAN, 'To␣Activate␣Gesture␣Mode',
            'To␣Activate␣Gesture␣Mode',
            False, gobject.PARAM_READWRITE)
        }

    def __init__ (self):
        gobject.GObject.__init__ (self)
        self.gesture ← False

    def do_get_property (self, property):
        if property.name = 'gesture':
            return self.gesture
        else:
            raise AttributeError, 'unknown␣property␣%s' % property.name
    def do_set_property (self, property, value):
        if property.name = 'gesture':
            self.gesture ← value
        else:
            raise AttributeError, 'unknown␣property␣%s' % property.name

gobject.type_register (DriverSignal)
```

The private variable *gpropertie* defines a signal called **gesture**, a type of boolean, and can be set to True of False using the *do_set_property* method and retrieved by the *do_get_property*.

Now signals are defined it is necessary to connect them to a callback function.

```
self.signals.connect ('notify::gesture', self.gesture_cb)
```

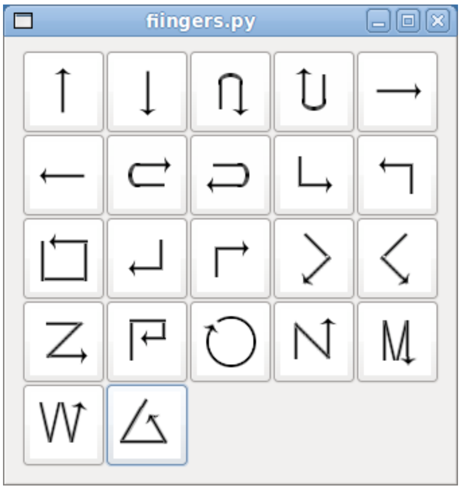
This will launch the *self.gesture_cb* method everytime the **gesture** signal value is set.

This simplifies the communication between processes and the separation between the processing and the GUI.

This concept is used by the driver to draw gestures on screen and to ask the recognizer class to process a gesture, and also by the configuration window to refresh the configuration for all the program.

3.5.3 Configuraton

Figure 3.6: The configuration window to setup gestures



The configuration consist of binding gestures to a program or keytroke; this is stored using an array of instances of the Action class:

```
class Action ():
    def __init__ (self, name):
        self.name ← name
        self.action ← 'key'# key or prog
        self.keystroke ← ''
        self.program ← ''
        self.activated ← False
```

Name will be the unique name of the gesture, action could be set to *key* to stipulate that a keystroke is bound to the gesture or *prog* for a command to launch.

To store a configuration, the configured object is serialized and saved inside a file using the *cPickle* module.

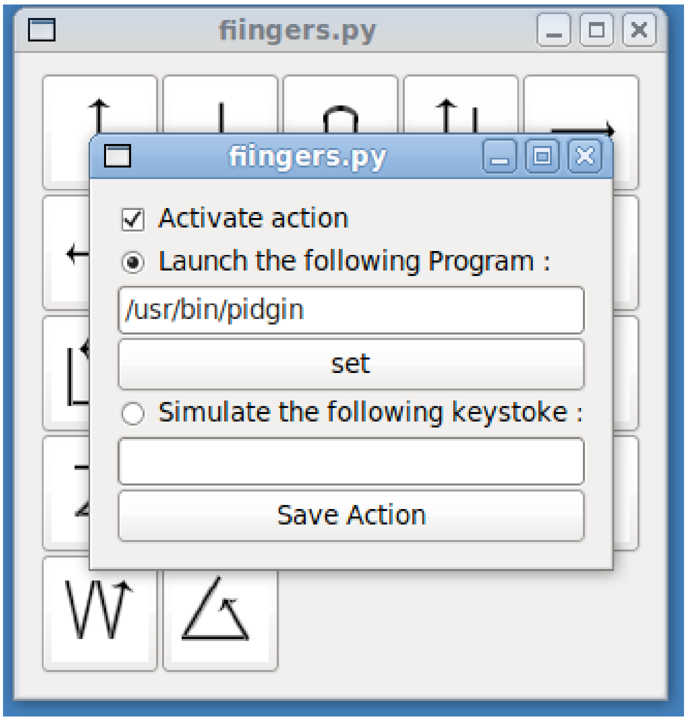
!/usr/bin/env python

```
from cPickle import Pickler, Unpickler

class Action ():
    def __init__ (self, name):
        self.name ← name
        self.action ← 'key'# key or prog
        self.keystroke ← ''
        self.program ← ''
        self.activated ← False

class Actions ():
    def __init__ (self, actions):
        self.actions ← actions

class Conf ():
    def __init__ (self):
        try:
            self.actions ← self.loader ()
        except :
            self.actions ← self.default_build ()

    def backup (self):
        backup ← Pickler (open ('gestures.pickle', 'w'))
        backup.dump (self.actions)

    def loader (self):
        datas ← Unpickler (open ('gestures.pickle', 'r'))
        actions ← datas.load ()
        return actions
    def default_build (self):
        list ← [ ]
        gesture_names ← ["Finger␣UP", "Finger␣Down", "Finger␣Up-Down",
    "Finger␣Down-Up", "Finger␣Right", "Finger␣Left",
    "Finger␣Left-Right", "Finger␣Right-Left",
    "Finger␣Down-Right", "Finger␣Up-Left", "Finger␣Rectangle",
    "Finger␣Down-Left", "Finger␣Up-Right",
    "Finger␣Left␣Arrow", "Finger␣Right␣Arrow",
    "Finger␣ZigZag", "Finger␣Flag", "Finger␣Circle",
    "Finger␣N␣Letter", "Finger␣M␣Letter", "Finger␣W",
    "Finger␣Triangle"]

        for name ∈ gesture_names:
            action ← Action (name)
            list.append (action)

        return Actions (list)
```

The *init* method will particularly take care that if it is impossible to load the configuration for any reasons, like file corruption, the configuration will be setup again by calling the *default_build* method.

Every time the configuration window will modify the setup, the object will be modified, marshaled and saved.

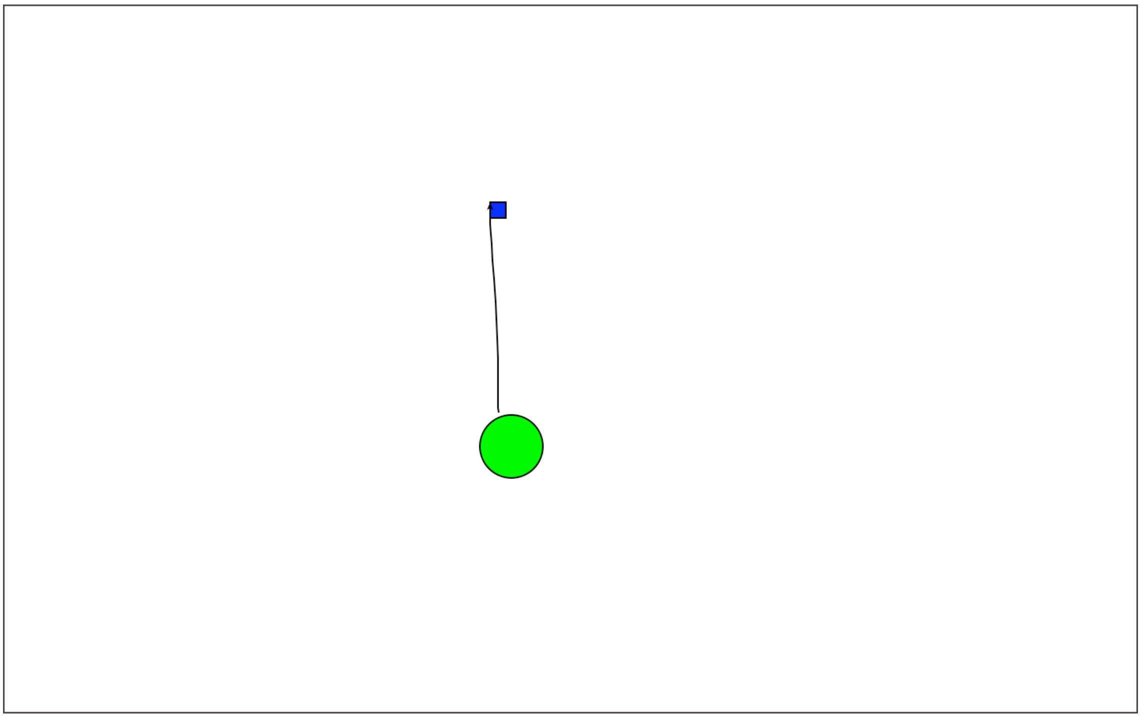
Figure 3.7: Window to bind a gesture to an action



☑ Activate action
● Launch the following Program :
/usr/bin/pidgin
set
○ Simulate the following keystoke :
Save Action

3.5.4 Gestures Window

Figure 3.8: The window to draw gesture input on screen



The gesture window is displayed in the gesture mode, which stops the pointing driver and runs its gesture tracking mode which only tracks one finger to communicate its coordinates to the window. Drawing gestures are done inside a fullscreen window, which waits for signal sent by the driver. It draws the finger with a small blue square, the starting area using a big green square and the gesture with an arrow.

At first, the GTK default canvas *(gtk.DrawingArea)* was used to draw the the gesture, but it takes too much time to redraw. Redrawing could be sped up using masks to exclude the part that are not required to be redrawn but the output was still delayed and pixelized.

To speed this up and have a better drawing, the project uses a *goocanvas* object instead. This negates any need to use a mask anymore and drawing is done by Cairo using vector graphics and taking advantage of display hardware acceleration when available, rather than simple pixel drawing which takes much more time.

The window responds to three signals; one for the position of the finger, another for the starting area and the last one to be closed once a gesture has been recognized.

The following code defined the blue square representing the finger on screen:

```
self.c ← goocanvas.Canvas ()
self.finger1 ← goocanvas.Rect (x ← 0, y ← 0, width ← 20, height ← 20, fill_color ←
        "blue")
```

```
self.c.get_root_item().add_child(self.finger1)
```

On the other side the driver will move the square representing the finger, to do it *gobject* signals are used:

```
'printpoint': (gobject.TYPE_PYOBJECT, '', '', gobject.PARAM_READWRITE)
```

gobject.TYPE_PYOBJECT means that the *printpoint* object will be a python object, used to define what the signal will contain : a queue using the *deque* python module.

The *do_set_property* will append a new point to the queue and the *do_get_property* will pop out the last entered value, permitting to do FIFO synchronous calls. The last step is to spawn a method to redraw the finger everytime the 'printpoint' object is set, this is done when instantiating the gesture window by:

```
self.handlerPrint ← self.signals.connect('notify::printpoint', self.gestureApp.printpoint_cb)
```

Finally, the method executed to redraw the finger :

```
1 def printpoint_cb(self, obj, property
2     if ¬self.closed:
3         point ← self.sig.get_property('printpoint')
4         if point ≢ None:
5             self.finger1.set_simple_transform(point.x, point.y, 1, 0)
6         else:
7             self.finger1.set_simple_transform(1, 1, 1, 0)

9         self.finger1.request_update()
```

If the retrieved value at line 3 is none the point simply disappears, and if the value represents a point the finger position is translated to the new coordinates. This can appear obscure but the point coordinates are maintained inside a thread, making it none too obvious to retrieve the coordinates from it. *Gobject* offers a simple thread-safe way to communicate with the graphical user interface.

The same method is used to print the starting area and the gesture stroke.

The driver operates that way to record a gesture:

```
Retrieving point from the
If one and only one point is detected
        if the distance elapsed is small
                if the timer is set
                        increase timer
                else
                        start the timer
        if the timer reach the threshold
                start recording coordinate
                if 17 points are recorded
                        send the recognize signal to the GUI
```

3.6 Gesturing recognition system

Artificial Neural networks can simulate a function by learning from the example of this function. So to create the neural network able to classify input gesture, it will be required to specify inputs and define gestures to train it.

3.6.1 Define input

The input data is a finger path that is retrieved by the driver. It is composed of a 17 item long array of tuple representing coordinates (x,y).

```
input=[(426, 196), (459, 191), (489, 189), (513, 188), (524, 188),
(529, 188), (531, 189), (533, 191), (532, 193), (531, 194), (531, 196),
(530, 198), (529, 199), (529, 200), (527, 208), (525, 224), (521, 250)]
```

Using this as input of the neural network is not appropriate, because it takes into consideration the location of the mouse path on the screen. To avoid that this data will be converted to angle vectors. Those angles will then be transformed into cosines and sines.

```
gesture ← [ ]
for $i \in$ range (len (input) − 1):
    angle ← math.atan2 (input[$i+1$][1] − input[$i$][1], input[$i+1$][0] − input[$i$][0])
    gesture.append (math.cos (angle))
    gesture.append (math.sin (angle))
```

Using this loop will normalize the input data into a 32 items array. Here is the result on input:

```
[0.98871550422476662, -0.14980537942799493, 0.99778515785660893,
-0.06651901052377393, 0.99913307309235189, -0.041630544712181326,
1.0, 0.0, 1.0, 0.0, 0.89442719099991586, 0.44721359549995793,
0.70710678118654757, 0.70710678118654746, -0.44721359549995793,...]
```

The inputs data are now in range of -1 to 1, making them usable with a symmetric activation function.

3.6.2 Define Gestures

Gesture will be define by a list of angle in degrees, and stored inside an array such as this:

```
gestures ← [ ]
```

```
#    # Gesture 0: Finger Up
gestures.append ([90.0, 90.0, 90.0, 90.0, 90.0,
90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0,
90.0, 90.0, 90.0])

#    # Gesture 1: Finger Down
gestures.append ([270.0, 270.0, 270.0, 270.0, 270.0,
270.0, 270.0, 270.0, 270.0, 270.0, 270.0, 270.0,
270.0, 270.0, 270.0, 270.0])

-- output truncated --

#    # Gesture 20: Finger W Letter
gestures.append ([300.0, 300.0, 300.0, 300.0, 300.0,
60.0, 60.0, 60.0, 300.0, 300.0, 300.0, 60.0, 60.0,
60.0, 60.0, 60.0])

#    # Gesture 21: Finger N Triangle
gestures.append ([233, 233.0, 233.0, 233.0, 233.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 127.0, 127.0, 127.0,
127.0, 127.0])
```

The array is then parsed to compute sines and cosine of the angles to produce a correct input for the network.

3.6.3 Neural Network design

The neural network will have 3 layers:

- **The input layer**, composed of 32 synapses.
- **A hidded layer**, composed of 32 neurons
- **The output layer**, composed of 22 axons, one per gesture

A graphical representation of the neural network is available in appendix B.1, and also how it was drawn using the dot language.

fully connected layers transfer function : log-sigmoid incremental training algorithm, standard back-propagation method momentum, variable learning rate (slowly reduced) input noise

3.6.4 Training

The training example will be defined inside a file, which will be loaded to create the neural network. The training example file should follow this syntax: one line to describe the format of data followed by one line to contain the input value and another for the output value.

```
number_of_training_pattern number_of_input number_of_output
training_pattern_1
result_1
...
training_pattern_n
result_n
```

The output of the neural network is an array made of -1 value for all fields despite the index of the recognized gesture set to 1.

[1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1] stand for gesture 0 recognized : Finger UP.

To generate the training set, the gestures are defined and transformed to the desired format by a file *data.py*. But after some tests, the gestures themselves are not sufficient to train the network properly to achieve a good level of generalization. A greater number of examples will be necessary. To generate more examples from gesture, random jitter will be introduce into vectors angle. This noise will avoid the neural network overflowing the training data set and will somewhat simulate the deviation made by the user when drawing the path on screen.

This is introduced by this function:

```
def noise(angle):
    if (random.random() > 0.2):
        rnd ← random.random()
        if random.random() < 0.5:
            rnd ← −rnd
            angle+ ← 30 ∗ rnd
            if (angle > 360): angle− ← 360
                if (angle < 0): angle+ ← 360
        return angle
```

The file will then be created with 220 examples, 10 for each gesture. The program *data.py* will generate the training set file and the output will look like this:

```
220 32 22

-0.068927289 0.997621686 0.000000000 1 0.000000000 1 0.000000000
1 0.010054706 0.999949450 0.000000000 1 -0.084210095 0.996448022
0.088466672 0.996079137 0.029599985 0.999561824 0.024203364
0.999707056 0.000000000 1 0.000000000 1 0.000000000 1 0.000000000
1 0.000000000 1 0.000000000 1

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

-0.095671646 -0.995412948 0.051845893 -0.998655097 0.059815166
 -0.998209470 0.030439473 -0.999536612 -0.000000000 -1.000000000
0.002580730 -0.999996670 0.097220446 -0.995262872 -0.000000000
-1.000000000 -0.000000000 -1.000000000 -0.000000000 -1.000000000
-0.083757094 -0.996486201 -0.000000000 -1.000000000 -0.041067556
-0.999156372 -0.054225433 -0.998528719 -0.061454112 -0.998109910
-0.000000000 -1.000000000

0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
output truncated
```

The chosen neural network library is FANN and should be installed before going any further.

```
cvs -d:pserver:anonymous@fann.cvs.sourceforge.net:/cvsroot/fann login
cvs -z3 -d:pserver:anonymous@fann.cvs.sourceforge.net:/cvsroot/fann co -P fann
cd fann
./configure
make
sudo make install
cd python
python setup.py install
```

Now that the training is created and the library installed, the neural network can be setup and trained using the file *train.py*. This is done by setting up an instance of *libfann.neural_net* and then starting the training.

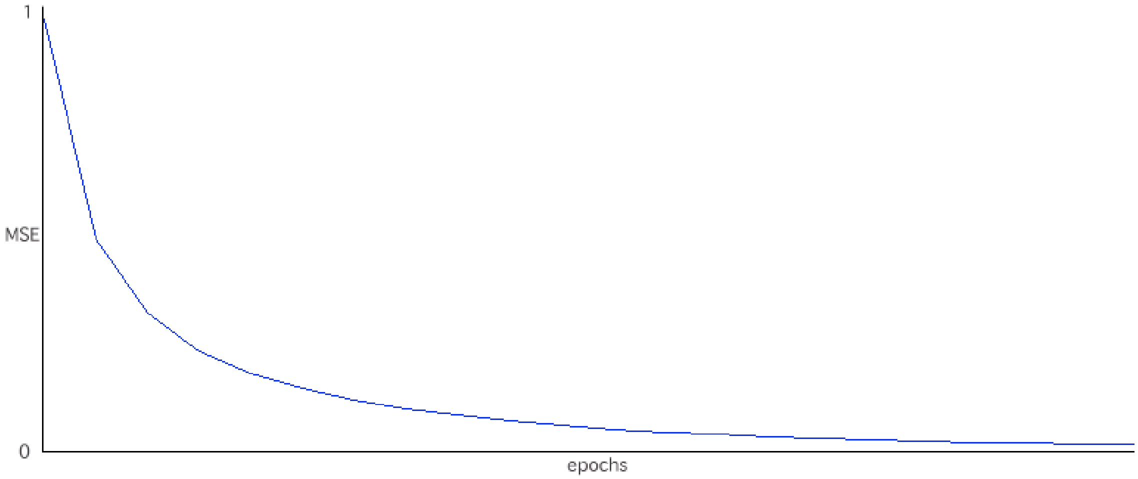
!/usr/bin/python

```
from pyfann import libfann

connection_rate ← 1
learning_rate ← 0.7
num_input ← 32
num_neurons_hidden ← 32
num_output ← 22
```

```
desired_error ← 0.000001
max_iterations ← 100000

ann ← libfann.neural_net ()

arg ← [num_input, num_neurons_hidden, num_output]
ann.create_standard_array (arg)

ann.set_learning_rate (learning_rate)
ann.set_training_algorithm (libfann.TRAIN_RPROP)

ann.set_activation_function_hidden (libfann.SIGMOID_SYMMETRIC)
ann.set_activation_function_output (libfann.SIGMOID_SYMMETRIC)

ann.set_rprop_increase_factor (1.2)
ann.set_rprop_decrease_factor (0.5)
ann.set_rprop_delta_min (0.0)
ann.set_rprop_delta_max (50.0)
ann.print_parameters ()

ann.train_on_file ("training_set.input", max_iterations,
iterations_between_reports, desired_error)

ann.save ("neural.net")
```

The training will stop once the desired mean squared error is reached. Here is a representation os the MSE reduction along the training:

Figure 3.9: Training graph



3.6.5 Recognition

The recognition of an input gesture is done by the recognizer class.

```
class Recognizer ():
    def __init__ (self):
        self.ann $\leftarrow$ libfann.neural_net ()
        self.ann.create_from_file ("nn/neural.net")

    def recognize (self, input):
        gesture $\leftarrow$ [ ]
        for $i \in$ range (len (input) $- 1$):
            angle $\leftarrow$ math.atan2 (input[$i+1$][1] $-$ input[$i$][1],
                input[$i+1$][0] $-$ input[$i$][0])
            gesture.append (math.cos (angle))
            gesture.append (math.sin (angle))

        print gesture
        print len (gesture)
        self.ann.reset_MSE ()

        calc_out $\leftarrow$ self.ann.run (gesture)

        print "calcout", calc_out
        return calc_out.index (max (calc_out))
```

This class when instantiated loads the neural network from a file and offers a *recognize* method which takes a gesture input as retrieved by the driver: a list point coordinates. The first step is to convert this data into angle vector sines and cosines to submit them as input of the neural network to obtain a classification using *self.ann.run(gesture)*. The output is a 22 item long list of weight, the maximum value index is then sent back, and should represent the gesture recognized.

The GUI will then find the associated action or keystroke, and execute it.

3.7 Testing

Testing the software cannot be automatized as the two main features of the program are supposed to be done by a human, pointing on screen and gesture recognition. So it is not possible to test the software with a automated test suite. Instead a usability testing was planned and findings from the subjects will help to verify the specific usability goals will be satisfied.

3.7.1 Goals

The purpose of this test is try to prove that all the implemented functionalities are working properly for a selection of subjects. This means that the user can point everywhere on the screen, left or right click, double click and zoom. Secondly, we will verify that users can input all the gestures correctly and be recognized. Last but not least, it will test the usability of the graphical user interface to connect and disconnect the Nintendo Wiimote and also setup the action binding to gesture.

3.7.2 Define the tests

The tests were conducted by myself in a quiet room and involved observant subjects while they completed the five scenarios of real case usage. Before beginning the test suite, pre-test activities are done to help the user to get used to the glove. This consists of simple advice on hand posture and how to perform clicks.

After all the results are collected, observations made by the subject will be noted down and discussed.

Pointing

Mouse usability scenario will be defined and played one time per subject, whereafter the elapsed time to realize each scenario will be collected and analyzed.

- **Scenario 1:** Launch the software and move the cursor on screen
  - Task A: Launch the software
  - Task B: Move the cursor on screen
  - Time limit: 10 minutes
- **Scenario 2:** Open a PDF file on desktop
  - Task A: Move to the file
  - Task B: Double click on it
- **Scenario 3:** Launch the chess games
  - Task A: Navigate through menu Application ¿ Games

  - Task B: Click on Chess menu Item

- **Scenario 4:** Change Desktop Background
  - Task A: Right Click on Desktop and select ”Change Destop Background” in the contextual menu
  - Task B: Select a new desktop, close the window

- **Scenario 5:** Input text inside Gedit
  - Task A Use Onboard to write ”I'm using fiingers”

- **Scenario 6:** Zoom on the applet
  - Introduce the second hand
  - Move to the desired place

Gesture recognition

To test the gesture recognition system, each gesture will be performed five times by each subject. This will give a recognition rate for each of them and help to prove that input data is correctly selected and permit a good enough level of generalization for the neural network.

The subjects

It has been proven in the industry that a population of four to six subject is enough to spot eighty percent of any usability problems. In our case it is sufficient to have merely an idea about the completed achievement of the software.

The subject population is volunteer based, with each individual having a different approach to and knowledge of computers.

- **Anna**, is a 22 year old student, who is studying psychology in France. She uses her laptop everyday and runs Ubuntu linux into it. She is right-handed.

- **Romain**, is a 24 year old student, studying Computer Science. He uses his laptop everyday to develop programs, using both Linxu and Windows. He is right-handed.

- **Ehab**, is a 23 year old student. He studies Finance and uses his laptop everyday to listen to music and write reports, and who has no experience of Linux at all. He is left-handed.

- **Jacques**, is a 70 year old retired man. He uses his Apple computer two or three time per week to check his bank account on the internet, manage photos and place video calls. He is ambidextrous.

3.7.3 Test results

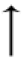
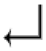
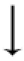
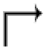
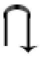
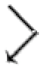
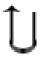
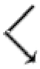
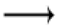
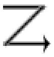
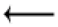
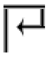
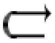
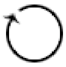
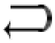
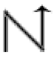
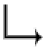
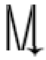
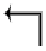
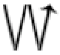
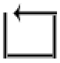
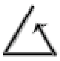
Pointing scenarios

Table 3.1: Pointing test scenarios results

	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5	Scenario 6
Subject 1	1m32	0m02	0m06	0m12	1m02	0m04
Subject 2	1m28	0m05	0m12	0m10	0m25	0m03
Subject 3	1m57	0m04	0m07	0m18	0m35	0m06
Subject 4	2m55	0m06	0m20	0m29	0m50	0m15

Gesture recognition rate

Table 3.2: Gesturing test suite results

Gesture	Anna	Romain	Ehab	Jacques	Gesture	Anna	Romain	Ehab	Jacques
	100%	100%	100%	100%		80%	60%	100%	80%
	100%	100%	100%	80%		100%	80%	80%	60%
	100%	100%	100%	80%		80%	100%	80%	80%
	100%	100%	100%	100%		100%	100%	100%	80%
	100%	100%	100%	100%		60%	80%	60%	20%
	100%	100%	100%	100%		60%	60%	80%	40%
	100%	100%	80%	100%		40%	60%	80%	60%
	100%	100%	80%	100%		80%	80%	100%	80%
	80%	80%	100%	60%		40%	20%	60%	20%
	80%	60%	100%	60%		20%	20%	40%	40%
	80%	60%	80%	60%		60%	80%	100%	80%

3.7.4 User feedback

Anna: It's not as easy to move one finger at once to click, but after practicing it's become much more natural. My small hands force me to separate my index and my middle finger too much which becomes wearying after half an hour or intense use.

Romain: The system is usable and offers a good level of precision. It could be interesting to add more action to bind when a gesture is recognized, like increasing the volume or pausing your music.

Ehab: The gesturing system is good but it doesn't give you a great deal of freedom. The gestures have to be drawn following a certain path and don't really recognize what's drawn on screen.

Jacques: On top of the fact that the gloves are too small, I needed some time to split my finger moves. This is quite good fun but the mouse is more convenient for me.

3.8 Discussion

The purpose of this paper was to develop a new input device method to interact with the desktop. As the test reveals more than the theory, the degree of achievement will be discussed by commenting on the test result.

3.8.1 Wiimote and gloves: pointing

The subjects have followed six scenarios to test the usability of the pointing system. The first task was to launch the software and move the cursor on screen. All the subjects have finished this task in a time that does not exceed 3 minutes. This process involve to get connected to the Wiimote and work properly. Other scenario prove as well that left and right clicking works properly, as going through menu, ect.

The system works well be could be enhanced by adding new feature, such as rotating. This could be sent via the TUIO protocol to applications that support it. Also there is only one solution proposed, put this could have be done in many other ways.

3.8.2 Gesturing

Most of the gestures have a recognition average greater than 80%, which means that all the gestures are recognizable and certifies that the system is usable. But even if most of the gestures are recognized all the time, some of them have a critical recognition rate $> 50\%$. Why did the neural network fail so much with the W and M finger gesture? It is probably due to gesture drawing complexity, but as Anna got a good recognition rate, we could consider that this was not the system that failed but the fact that people draw in different ways. As this problem cannot be solved in that state of the system, the best alternative was to replace this gesture by another which is less complex.

To use the system properly the user will have to learn how it works and practice getting a 100% recognition rate. The system could not adapt the recognition to the user, it is the user that must adapt his use. This is the biggest weakness of the solution, but could be avoided with an adaptive system.

Romain formulated an interesting idea; more action could be added such as increasing the volume, pausing the video player, etc. This would be feasible as long as the software that should receive the action implements an IPC system. One is available and widely used on linux graphical interface, called DBus and developped by Freedesktop. It works with a system of messages that could be discovered by asking the *dbus* daemon which registers them.

In fact, this feature was initially planned to be added, although it did not make the final design of the project, however, a simple python script would work admirably and be called by fiinger gesture recognition system.

Implement a Dbus message sending script

- First identify the program:

  In this example it is **Rhythmbox**, the music player; be sure that it offers a DBus interface by using the *dbus-send* command:

```
dbus-send --print-reply --session --dest="org.freedesktop.DBus"
/org/freedesktop/DBus org.freedesktop.DBus.ListNames
```

  It will print on screen all the program identifiers available on the Dbus system, note down the program identifier printed on screen:

```
string "org.gnome.Rhythmbox"
```

- Now, we have his identifier, we are going to find the message that permit to change the system volume by using the service introspection method:

```
dbus-send --session --type=method_call --print-reply \\
--dest=org.gnome.Rhythmbox /org/gnome/Rhythmbox \\
 org.freedesktop.DBus.Introspectable.Introspect

method return sender=:1.31 -> dest=:1.63 reply_serial=2
   string "<!DOCTYPE node PUBLIC "-//freedesktop//DTD D-BUS
Object Introspection 1.0//EN"
"http://www.freedesktop.org/standards/dbus/1.0/introspect.dtd">
<node>
  <node name="Player"/>
  <node name="PlaylistManager"/>
  <node name="Shell"/>
  <node name="Visualizer"/>
</node>
"
```

  What we want to control is the *Player*, so introspect it :

```
dbus-send --session --type=method_call --print-reply \\
--dest=org.gnome.Rhythmbox /org/gnome/Rhythmbox/Player\\
 org.freedesktop.DBus.Introspectable.Introspect

method return sender=:1.31 -> dest=:1.64 reply_serial=2
string "<!DOCTYPE node PUBLIC "-//freedesktop//DTD D-BUS Object
Introspection 1.0//EN"
"http://www.freedesktop.org/standards/dbus/1.0/introspect.dtd">
<node>
        [ output truncated]
    <method name="setVolume">
```

```
      <arg name="volume" type="d" direction="in"/>
    </method>
        [ output truncated]
</node>
"
```

The method *setVolume* takes one argument called volume which is a double.

- Finally, to transform it as a script, 2 solutions are possible, the first one is simply make a bash script:

```
#!/bin/bash
dbus-send --session --type=method_call --print-reply \\
--dest=org.gnome.Rhythmbox /org/gnome/Rhythmbox/Player \\
org.gnome.Rhythmbox.Player.setVolume double:1.0
```

Then make it in a file and make it executable by using a *chmod +x* on it. Now you will juste have to setup a gesture to trigger this script and this will do the job.

The other solution is to implement it in python, this offer the oportinity to ensure that rhythmbox is running before sending the command:

!/usr/bin/python

```
import dbus

DBUS_START_REPLY_SUCCESS ← 1
DBUS_START_REPLY_ALREADY_RUNNING ← 2

Connect to the currectsession bus

bus ← dbus.SessionBus ()

Force Rhythmbox to start is not running

(success, status) ← bus.start_service_by_name ('org.gnome.Rhythmbox')
force_visible ← (status = DBUS_START_REPLY_SUCCESS)

Open the Rhythmbox Player object and get its list of services

rbshellobj ← bus.get_object ('org.gnome.Rhythmbox', '/org/gnome/Rhythmbox/Player')
rbprops ← dbus.Interface (rbshellobj, 'org.gnome.Rhythmbox.Player')
```

```
Set the volume to 80
rbprops.setVolume (0.8)
```

It's also require to make it executable.

This little hack proves that even if the system does not have a functionality built inside, it remains possible to extend it, the only limit being the user creativity.

Chapter 4

Conclusion

Over the past chapter, we have used a variety of different software and technologies to produce the final system. The goal was to enhance the classic user input method, by tracking the user's fingers in the air. What we can observe is that fiinger responds to this demand by providing a pointing solution and a gesture recognition system, and can now offer a new degree of freedom for a relatively small price.

"Life is short, you need Python", Bruce Eckel. This sentence refers to the key point in the development process to success which helped me test several alternatives for each algorithm implementation in a short development time. But also it was not as easy to optimize to obtain good performances as seen during the driver development.

Concerning the gesture recognition, the choice of a neural network described and implemented offers a good performance but in the meantime could be improved by offering a training mode which will record user own gestures and train the the neural network with them in accordance with the user's pointing manners. The future evolution of the system could explore said idea to offer a better user experience.

The next step in the enhancement of the program could be to make it available to anyone by porting it to Windows and MacOS. Issues regarding possible issues have been taken into consideration during the development to reduce the porting effort as all the components are open-source and the code documented.

This project will not be commercialized and widely adapted, but it's a working modest attempt to enhance the way to interact with a computer without completely change the current paradigm. This will be the key point of future research on this subject.

Bibliography

Duschesneau, S. (n.d.), ‘Gtk wii whiteboard’.
URL: *http://www.stepd.ca/gtkwhiteboard/*

Freedesktop (n.d.), ‘Dbus’.
URL: *http://www.freedesktop.org/wiki/Software/dbus*

Jorda, S. (2005), The reactable, *in* ‘Proceedings of the International Computer Music Conference (ICMC 2005)’, Barcelona, Spain.

Kaltenbrunner, M., Bovermann, T., Bencina, R. & Costanza, E. (2005), Tuio - a protocol for table based tangible user interfaces, *in* ‘Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation (GW 2005)’, Vannes, France.

Kaltenbrunner, M. & R., B. (2007), reactivision: A computer-vision framework for table-based tangible interaction, *in* ‘Proceedings of the first international conference on ”Tangible and Embedded Interaction” (TEI07). Baton Rouge, Louisiana.’.

Lee, J. C. (2007), ‘Wiimote projects’.
URL: *http://www.cs.cmu.edu/ johnny/projects/wii/*

Nissen, S. (2003), Implementation of a fast artificial neural network library (fann), Technical report, Department of Computer Science University of Copenhagen (DIKU). http://fann.sf.net.

Sigurdsson & Wong (2008), *Finger Tracking for Rapid Cropping Applications.*

Ziadé, T. (2006), *Programmation Python*, Eyrolles.

Appendix A

Fiingers User Doc

A.1 The Software

A.1.1 Prerequisite

The program require the following library :

- virtkey
- fann
- python-xlib

This package should be available through your distribution package manager. For example Ubuntu user should type:

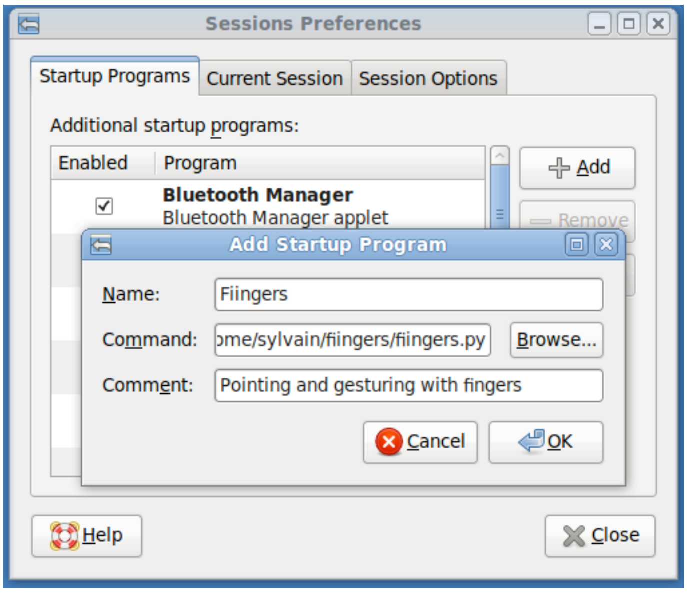
```
sudo aptitude install virtkey fann python-xlib
```

A.1.2 Add the software to session

To add fiingers to start with the session, you should go to the *System menu*, then select *Configuration* and click on *Session*. Select the add button and setup the popup window with the following informations:

Then simply logout, and login in again to see a new icon appearing in the notification area representing a blue house.

Figure A.1: Start fiinger with the session



Sessions Preferences

Additional startup programs:

Enabled	Program
☑	**Bluetooth Manager** Bluetooth Manager applet

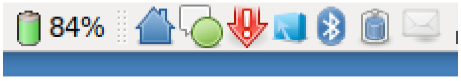
Add Startup Program

Name: Fiingers

Command: ome/sylvain/fiingers/fiingers.py

Comment: Pointing and gesturing with fingers

Figure A.2: Fiinger status icon



A.2 Pointing on screen

Pointing on screen could be done by two ways:

The first one consist ofusing gloves with 3 leds on the thumb, index and middle finger of the right hand and one led on the finger of the left hand.

The second possibility is to use an IR array surrounding the Wiimote and pointing to the user, then use reflective tape on finger tips to point on screen.

A.3 Gestures

A.3.1 Train the Neural Network

This step is not required but could interest people who wants to customize the software. The training set is generate by the file *data.py* under the *nn* directory. This file can be modified to change some gestures. Once you want to generate the training set simply execute the script :

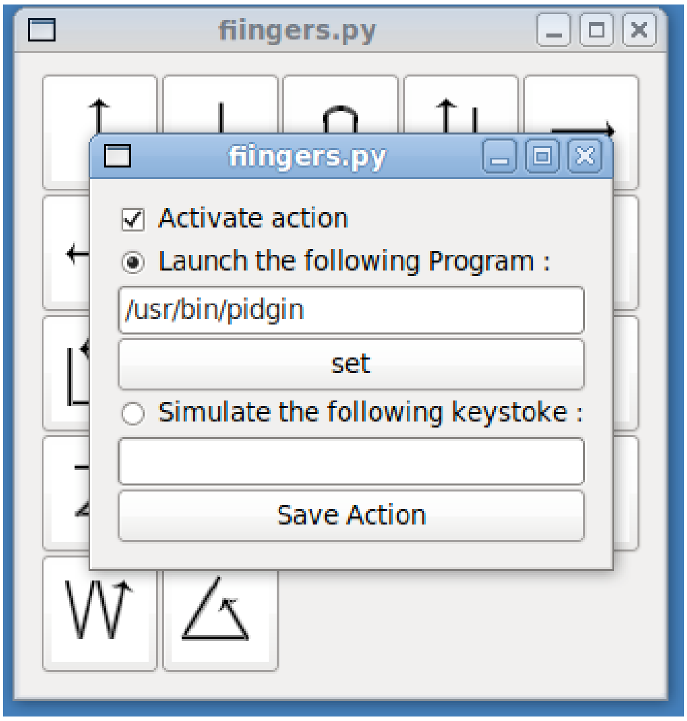
```
python data.py
```

The next step is to train the network, all his properties can be tuned by editing the file *train.py*: like the learning rate, the momentum, the number of neurons and layers. Finally execute this script to generate the file *neural.net* which is the saved trained network.

A.3.2 Setup Gestures/Action mapping

The configuration of actions is done by selecting configuration in the status icon popup menu. The Windows that will appear show the list of gesture as button. Selecting one will popup a window to setup the gesture. Clicking on the keystroke input box will wait for keystoke and selecting set program will launch a file chooser popup to select the desired program.

Figure A.3: Gesture setup window



fiingers.py

fiingers.py

☑ Activate action

● Launch the following Program :

/usr/bin/pidgin

set

○ Simulate the following keystoke :

Save Action

A.3.3 Input Gesture

Select the gesture item from the status icon popup menu. Place on finger on the direction to the camera, wait half a second until a green circle surround the pointer, then leaving the green area will start to record the gesture. No action are required the record stop automatically.

Appendix B

Additional production

B.1 Graphical representation of the Neural Network

B.1.1 Generate the graph

The graphical representation of the neural network is realized using the the pydot python library which wrap the dot language designed to draw graph.

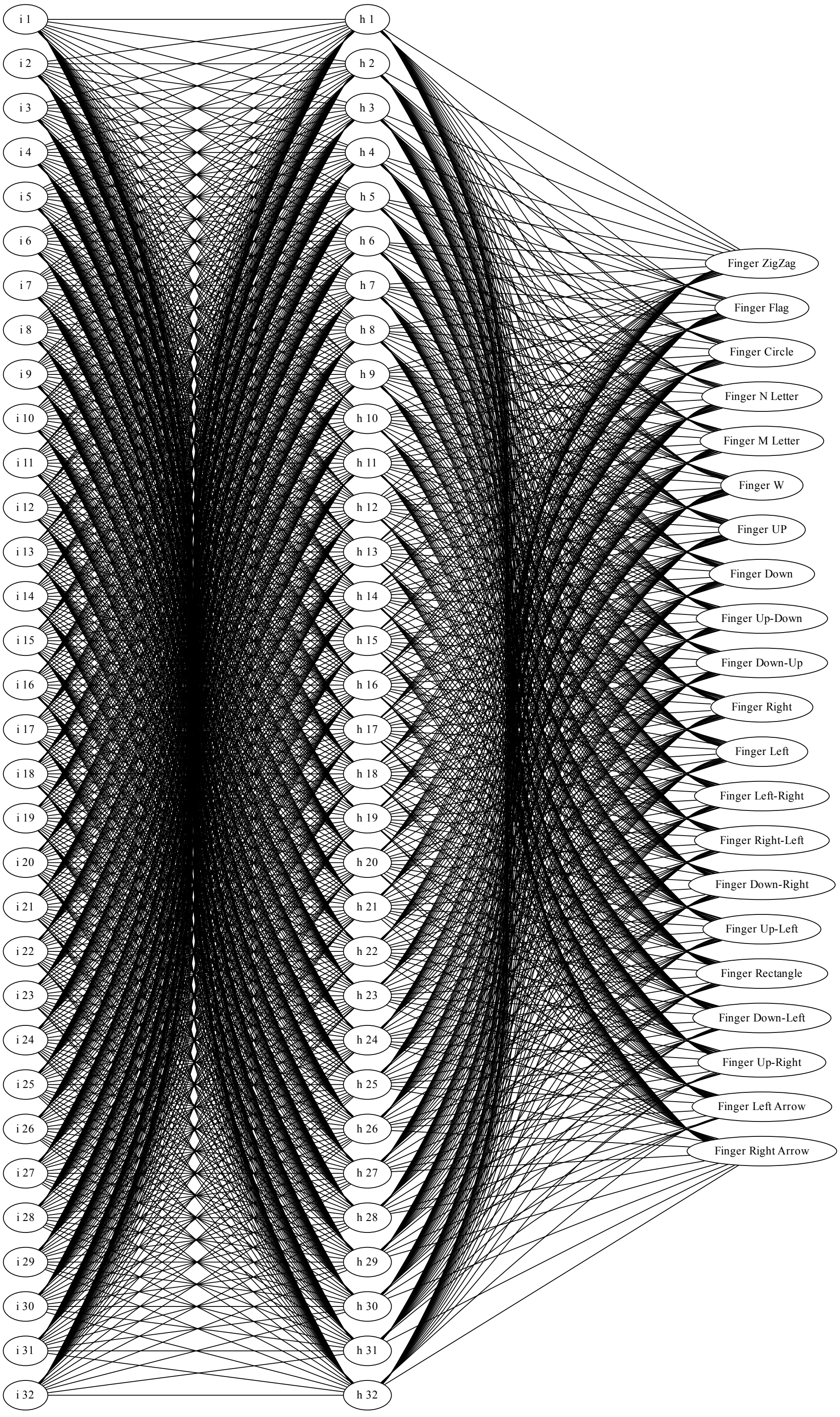
Here is the written code to obtain the graph:

!/usr/bin/env python encoding: utf-8

```
import sys
import os
import pydot

def main ():
    gesture_names ← ["Finger␣UP", "Finger␣Down", "Finger␣Up-Down",
    "Finger␣Down-Up", "Finger␣Right", "Finger␣Left", "Finger␣Left-Right",
    "Finger␣Right-Left", "Finger␣Down-Right", "Finger␣Up-Left",
    "Finger␣Rectangle", "Finger␣Down-Left", "Finger␣Up-Right",
    "Finger␣Left␣Arrow", "Finger␣Right␣Arrow", "Finger␣ZigZag",
    "Finger␣Flag", "Finger␣Circle", "Finger␣N␣Letter", "Finger␣M␣Letter",
    "Finger␣W", "Finger␣Triangle"]

    dot ← pydot.Dot ('Gesture␣Recognition␣Neural␣Network␣Design',
    graph_type ← 'graph', rankdir ← 'LR', rank ← 'source', ranksep ←
          "5", ordering ← "in")
    for $i \in$ range $(32)$:
        input ← 'i␣%i' % $(i + 1)$
        for $j \in$ range $(32)$:
            hidden ← 'h␣%i' % $(j + 1)$
            dot.add_edge (pydot.Edge (input, hidden))

    for $i \in$ range $(32)$:
        input ← 'h␣%i' % $(i + 1)$
        for $j \in$ range $(21)$:
            output ← gesture_names$[j]$
            dot.add_edge (pydot.Edge (input, pydot.Node (output, shape ←
                    'plaintext')))

    dot.write ('pydot_neuralnet.pdf', format ← 'pdf')

    pass

if __name__ = '__main__':
    main ()
```

B.1.2 The Neural Network graph

Figure B.1: The implemented neural network



Appendix C

Fiingers Source Code

C.1 Main Program

!/usr/bin/env python

```
from interface import gui

interface ← gui.FiingersGUI (queue)
interface.main ()
```

C.2 Driver

import WiimoteLib

```
import Xlib.display
import Xlib.ext.xtest
import time
import threading
import math
import mousecontrol

from linuxWiimoteLib3 import Wiimote
import sys

from collections import deque
import virtkey
```

```
class Point ():
    """Define␣a␣point␣by␣his␣coordinates␣and␣wiimote␣recognized␣position"""
    def __init__ (self, pos, $x$, $y$):
        self.pos ← pos
        self.$x$ ← $x$
        self.$y$ ← $y$
    def update (self):
        pass

class Points ():
    """Map␣points␣to␣real␣fingers"""
    def __init__ (self, nb, points):
        self.nb ← nb
        if nb $= 3$:
            self.lindex ← None
            self.thumb ← points[0]
            self.index ← points[1]
            self.middle ← points[2]
        elif nb $= 4$:
            self.lindex ← points[0]
            self.thumb ← points[1]
            self.index ← points[2]
            self.middle ← points[3]

class MouseDriver (threading.Thread):
    """The␣main␣driver␣class"""

    def runner (self):
        """switch␣betwenn␣pointin␣and␣gesturing␣mode"""
        while self.$w$:
            self.run ()
            self.run2 ()

    def connectWiimote (self):
        """Request␣the␣connexion␣to␣the␣wiimote"""
        if ¬self.$w$.Connect ():
            self.isConnected ← False
            print "wiimote␣not␣connected"
            return False
        else:
            print 'connect␣finish'
            self.w.SetLEDs (False, True, False, True)
            self.w.activate_IR ()
            self.isConnected ← True
            self.running ← True
            self.solid ← False
            print "battery␣level␣=␣%f" % (self.w.WiimoteState.Battery ∗ 100/200)
            return True

    def maintain (self, el1, el2):
        "Use␣to␣map␣points␣to␣finger␣tip"
        distance ← self.distance (el1.x, el1.y, el2.x, el2.y)
        angle ← math.atan2 ((el2.y − el1.y), (el2.x − el1.x))
        angle ← math.degrees (angle)
        if angle < 0:
            angle+ ← 360

        tmp ← el1.y − el2.y
        print "coord␣%i␣%i␣%i␣%i" % (el1.x, el1.y, el2.x, el2.y)
        ‖ print ”tmp=
        if abs (tmp) < 30:
            if el1.x > el2.x:
                print '-1'
                return −1
            else:
                print '1'
                return 1
        else:
            if tmp > 0:
                print '1'
                return 1
            else:
                print '-1'
                return −1

    def run2 (self):
        """The␣gesture␣recognition␣mode␣
-␣Maintain␣one␣point
-␣Detect␣gesture␣begin␣end
-␣Send␣messages␣to␣the␣Goo␣Canvas␣to␣draw␣the␣gesture
-␣Submit␣the␣gesture␣to␣recognition"""
```

```
print "enterring␣gesture␣mode"
points ← [ ]
self.solid ← False
self.finger1 ← Point (1, 70, 70)
self.finger2 ← Point (2, 40, 40)
self.running2 ← True
ppoints ← [ ]
ppoints.append (Point (3, 0, 0))
ppoints.append (Point (3, 0, 0))
capturing ← False
recording ← False
moving ← True
capturing1 ← [ ]
startpoint ← None
inside_start ← False
past_nbfound ← 0
timer ← 0

while self.running2:
    nb_found ← self.w.WiimoteState.IRState.nbFound
    points ← [ ]
    || request points coordinates
    if self.w.WiimoteState.IRState.Found1:
        points.append (Point (1, 1024 −
                self.w.WiimoteState.IRState.RawX1, 768 −
                self.w.WiimoteState.IRState.RawY1))
    if self.w.WiimoteState.IRState.Found2:
        points.append (Point (2, 1024 −
                self.w.WiimoteState.IRState.RawX2, 768 −
                self.w.WiimoteState.IRState.RawY2))
    if self.w.WiimoteState.IRState.Found3:
        points.append (Point (3, 1024 −
                self.w.WiimoteState.IRState.RawX3, 768 −
                self.w.WiimoteState.IRState.RawY3))
    if self.w.WiimoteState.IRState.Found4:
        points.append (Point (4, 1024 −
                self.w.WiimoteState.IRState.RawX4, 768 −
                self.w.WiimoteState.IRState.RawY4))

    if len (points) ≠ nb_found:
        print "sync␣missed"
        continue

    nb_found ← len (points)

    past_nbfound ← nb_found
```

```
if nb_found = 1:
    point ← points[0]
    ppoint ← ppoints[0]
        || Look for gesture to begin
    if ¬capturing:
        if moving:
            d1 ← self.distance (point.x, point.y, ppoint.x, ppoint.y)
            if d1 < 3:
                moving ← False
                startpoint ← point
        else:
            d1 ← self.distance (point.x, point.y, startpoint.x, startpoint.y)
            if d1 < 10:
                timer+ ← 1
            else:
                moving ← True
                timer ← 0

            if timer > 2000:
                timer ← 0
                moving ← True
                capturing ← True
                inside_start ← True
                self.parent.signals.set_property ('startingpoints', startpoint)

    else:
        || Capturing a gesture
        if inside_start:
            d1 ← self.distance (point.x, point.y, startpoint.x, startpoint.y)
            if d1 > 40:
                recording ← True
                inside_start ← False
                last_captured ← None
                captured ← None
                capturing1 ← []

        if recording:
            timer ← timer + 1

            if timer % 1000 = 0:
                capturing1.append ((point.x, point.y))
                print 'captured'
```

```
                        if len(capturing1) = 17:
                            self.parent.signals.set_property('gesturepoints', capturing1)
                            self.parent.signals.set_property('recognize', capturing1)
                            print 'timer', timer
                            capturing ← False
                            recording ← False
                            timer ← 0
                            print capturing1
                            capturing1 ← [ ]

                self.parent.signals.set_property('printpoints', point)
                ppoints ← points
            else:  # 0 or 3 or 4 points found
                timer ← 0
                self.parent.signals.set_property('printpoints', None)
                capturing ← False
                recording ← False
                moving ← True
        print "exit␣gesture␣mode"

    def run(self):
        "the␣loop␣function␣to␣move␣cursor"
        print "beginning␣mouse␣mode"
        self.running ← True

        self.mouse ← mousecontrol.MouseControl()
        past_nbfound ← 0
        nb_found ← 0
        ppoints ← [ ]
        caching ← [ ]
        fix ← 0
        ptv ← deque()
        self.hand ← Points(3, [Point(0, 0, 0), Point(0, 0, 0), Point(0, 0, 0)])
        ptv.append(self.hand)

        ppoint3 ← None
        startleftclick ← False
        startrightclick ← False
        startdrag ← False
        startscroll ← False
        startrotate ← False

        maintained ← False
        missfix ← False
        move_cursor ← False
```

```
self.moving ← True
moving_t ← 0
self.c ← 0

thumbd ← 0
thumba ← 0

zoomDp ← 10000
zooming ← False
zoomer ← 0

keypress ← virtkey.virtkey ()

while self.running:
    self.c+ ← 1

    if self.isConnected:
        past_nbfound ← nb_found
        nb_found ← self.w.WiimoteState.IRState.nbFound
        points ← [ ]
            ‖ retrieve points coordinates
        if self.w.WiimoteState.IRState.Found1:
            points.append (Point (1, 1024 −
                  self.w.WiimoteState.IRState.RawX1, 768 −
                  self.w.WiimoteState.IRState.RawY1))
        if self.w.WiimoteState.IRState.Found2:
            points.append (Point (2, 1024 −
                  self.w.WiimoteState.IRState.RawX2, 768 −
                  self.w.WiimoteState.IRState.RawY2))
        if self.w.WiimoteState.IRState.Found3:
            points.append (Point (3, 1024 −
                  self.w.WiimoteState.IRState.RawX3, 768 −
                  self.w.WiimoteState.IRState.RawY3))
        if self.w.WiimoteState.IRState.Found4:
            points.append (Point (4, 1024 −
                  self.w.WiimoteState.IRState.RawX4, 768 −
                  self.w.WiimoteState.IRState.RawY4))

        if len (points) ≠ nb_found:
            print "sync␣missed"
            continue

        if nb_found < 2:
            past_nbfound ← 0
            continue
```

```
          || zooming implentation
if past_nbfound $= 4 \land$ nb_found $< 4$:
    keypress.*release_keysym* $(116)$
    ppoint3 $\leftarrow$ None
    zooming $\leftarrow$ False
    zoomer $\leftarrow 0$

if nb_found $= 4 \land$ past_nbfound $= 3 \land$ maintained:

    if $\neg$zooming:
        ppoint3 $\leftarrow$ ppoints
        zooming $\leftarrow$ True
    zoomer$+ \leftarrow 1$

    if zoomer $> 100$:

        keypress.*press_keysym* $(116)$
        tmp $\leftarrow$ points
        points2 $\leftarrow [\,]$
        for $i \in$ caching:
            points2.*append* (tmp.*pop* $(i)$)
        points $\leftarrow [\,]$
        points.*append* (tmp.*pop* ())
        points.*extend* (points2)

if nb_found $\neq$ past_nbfound $\land$ nd nb_found $\neq 4$:

    if past_nbfound $\leq 3 \land$ nb_found $> 2$:
        maintained $\leftarrow$ False
    elif nb_found $> 1 \land$ past_nbfound $> 2$:
        maintained $\leftarrow$ True
        missfix $\leftarrow$ True
    else:
        print "unknown␣case"
        print nb_found
        print past_nbfound
        past_nbfound $\leftarrow 0$
        continue

    if $\neg$maintained:
        || sort points and create cache
        startleftclick $\leftarrow$ False
        points.*sort* (self.maintain)
        points.*reverse* ()
        caching[: ] $\leftarrow$ [elem.pos for elem $\in$ points]
        maintained $\leftarrow$ True
```

```
                **elif** nb_found $> 2$:   $\#$   $\#$ previous state was solid
                    ptmp $\leftarrow [\,]$
                    **for** $i \in$ caching:
                        **for** $p \in$ points:
                            **if** $i = p$.pos:
                                ptmp.*append* $(p)$

                    **if** *len* (ptmp) $<$ nb_found:
                        missfix $\leftarrow$ True

                    points $\leftarrow$ ptmp

                **if** nb_found $= 2$:
                    **continue**

                **if** missfix:
                    past_nbfound $\leftarrow 0$
                    missfix $\leftarrow$ False
                    **continue**

                missfix $\leftarrow$ False
                maintained $\leftarrow$ True

                $\#$   $\#$ look for Clicks
                $\#$   $\#$

                    || Map point to finger tips
                self.hand $\leftarrow$ *Points* (nb_found, points)

                    || Zooming scale
                **if** nb_found $= 4$:
                    zoomD $\leftarrow$ self.*distance* (self.hand.index.$x$,
self.hand.index.$y$, self.hand.lindex.$x$, self.hand.lindex.$y$)
    **if** zoomD $>$ zoomDp:
        self.mouse.*mouse_click* $(4)$
    **else**:
        self.mouse.*mouse_click* $(5)$

    zoomDp $\leftarrow$ zoomDp
```

```
        || Try to detect when the user will click
    indexD ← self.distance (self.hand.index.x,
self.hand.index.y, ptv[0].index.x, ptv[0].index.y)
    if indexD < 3:
        moving_t+ ← 1
        if moving_t > 1000 ∧ self.moving:
            print "block"
            self.moving ← False
            snap_thumb ← self.hand.thumb
            snap_middle ← self.hand.middle
    else:
        self.moving ← True
        moving_t ← 0

    if ¬self.moving:
        || Look for Left click
        if self.hand.thumb ≢ None:
            thumbd ← self.distance (self.hand.thumb.x,
self.hand.thumb.y, ptv[0].thumb.x, ptv[0].thumb.y)
    thumbtotal ← self.distance (self.hand.thumb.x,
self.hand.thumb.y, snap_thumb.x, snap_thumb.y)
    if thumbd > 3:
        thumba ← math.atan2 ((self.hand.thumb.y−
ptv[0].thumb.y), (self.hand.thumb.x − ptv[0].thumb.x))
    if abs (thumba) < math.pi % 2:
        print "ascendent␣mvmnt"
        || if thumbtotal ¿ 40:
        if thumbtotal > 20:
            if ¬startleftclick:
                self.mouse.mouse_click (1)
                startleftclick ← True
    else:
        print "descendent␣mvt"
        startleftclick ← False
    print "rel=␣%f␣total=␣%f" % (thumbd, thumbtotal)
```

```
        || Look for Right click
    if self.hand.middle ≢ None:
        middled ← self.distance (self.hand.middle.x,
self.hand.middle.y, ptv[0].middle.x, ptv[0].middle.y)
    middletotal ← self.distance (self.hand.middle.x,
self.hand.middle.y, snap_middle.x, snap_middle.y)
    if middled > 3:
        middlea ← math.atan2 ((self.hand.middle.y−
ptv[0].middle.y), (self.hand.middle.x − ptv[0].middle.x))
    if 0 < middlea < math.pi:
        print "ascendent␣mvmnt"
        if middletotal > 20:
            if ¬startrightclick:
                self.mouse.mouse_click (3)
                startrightclick ← True
    else:
        print "descendent␣mvt"
        startrightclick ← False
    print "rel=␣%f␣total=␣%f" % (middled, middletotal)
    pass

    || Transform index finger coordinates to a cursor position It takes the screen
    || resolution into consideration

    if self.moving:

        if self.hand.index.x < 112:
            self.indexx ← 112
        elif self.hand.index.x > 912:
            self.indexx ← 912
        else:
            self.indexx ← self.hand.index.x

        self.indexx ← (self.indexx − 112) ∗ self.swidth/800

        if self.hand.index.y < 50:
            self.indexy ← 50
        elif self.hand.index.y > 650:
            self.indexy ← 650
        else:
            self.indexy ← self.hand.index.y

        self.indexy ← (self.indexy − 50) ∗ self.sheight/600

            || Move the cursor to his new position
        self.mouse.mouse_warp (int (self.indexx), int (self.indexy))
```

```
ptv.appendleft (self.hand)
if len (ptv) > 20: ptv.pop ()

del self.mouse

print "exit␣mouse␣mode"
return

def distance (self, x1, y1, x2, y2):
    """return␣the␣distance␣between␣2␣points"""
        return math.sqrt (((x1 − x2) ∗ (x1 − x2)) + ((y2 − y1) ∗ (y2 − y1)))

def abort (self):
    """Stop␣the␣driver␣and␣stop␣the␣connection␣with␣the␣wiimote"""
    self.running ← False
    self.running2 ← False
    time.sleep (0.3)
    self.w.Dispose ()
    del self.w
    return

def sigterm (self, sn, stack):
    ‖ Trap Ctrl+C from terminal
    self.abort ()

def __init__ (self, parent ← None):

    threading.Thread.__init__ (self)

    if parent ≢ None:
        self.parent ← parent

    self.isConnected ← False
    ‖ self.mouse = mousecontrol.MouseControl()
    self.w ← Wiimote ()
    self.display ← Xlib.display.Display ()
    self.screen ← self.display.screen ()
    self.swidth ← self.screen['width_in_pixels']
    self.sheight ← self.screen['height_in_pixels']
    self.root ← self.screen.root
    self.past_nbfound ← 0
    self.table ← [ ]
    self.running ← False
    self.running2 ← False
```

C.3 Neural Network

C.3.1 Generate data

!/usr/bin/env python

```
import math
import time
import random

def noise (angle):
    """Function␣to␣introduce␣noise␣into␣gesture"""
    if (random.random () > 0.2):
        rnd ← random.random ()
        if random.random () < 0.5:
            rnd ← rnd
        angle+ ← 30rnd
        if (angle > 360): angle− ← 360
        if (angle < 0): angle+ ← 360
    return angle
```

Define the gestures

```
gestures ← [ ]

#   # Gesture 0: Finger Up
gestures.append ([90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0,
90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0])

#   # Gesture 1: Finger Down
gestures.append ([270.0, 270.0, 270.0, 270.0, 270.0, 270.0, 270.0, 270.0,
270.0, 270.0, 270.0, 270.0, 270.0, 270.0, 270.0, 270.0])

#   # Gesture 2: Finger Up-Down
gestures.append ([90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 270.0,
270.0, 270.0, 270.0, 270.0, 270.0, 270.0, 270.0])

#   # Gesture 3: Finger Down-Up
gestures.append ([270.0, 270.0, 270.0, 270.0, 270.0, 270.0, 270.0, 270.0,
90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0])

#   # Gesture 4: Finger Right
gestures.append ([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0])
```

```
#    # Gesture 5: Finger Left
gestures.append ([180.0, 180.0, 180.0, 180.0, 180.0, 180.0, 180.0, 180.0,
180.0, 180.0, 180.0, 180.0, 180.0, 180.0, 180.0, 180.0])

#    # Gesture 6: Finger Right-Left
gestures.append ([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 180.0, 180.0, 180.0, 180.0,
180.0, 180.0, 180.0, 180.0])

#    # Gesture 7: Finger Right-Left
gestures.append ([180.0, 180.0, 180.0, 180.0, 180.0, 180.0, 180.0, 180.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0])

#    # Gesture 8: Finger Down-Right
gestures.append ([270.0, 270.0, 270.0, 270.0, 270.0, 270.0, 270.0, 270.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0])

#    # Gesture 9: Finger Up-Left
gestures.append ([90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 180.0, 180.0, 180.0,
180.0, 180.0, 180.0, 180.0, 180.0])

#    # Gesture 10: Finger Rectangle
gestures.append ([270.0, 270.0, 270.0, 270.0, 0.0, 0.0, 0.0, 0.0, 90.0, 90.0, 90.0, 90.0,
180.0, 180.0, 180.0, 180.0])

#    # Gesture 11: Finger Down-Left
gestures.append ([270.0, 270.0, 270.0, 270.0, 270.0, 270.0, 270.0, 270.0, 180.0, 180.0,
180.0, 180.0, 180.0, 180.0, 180.0, 180.0])

#    # Gesture 12: Finger Up-Right
gestures.append ([90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0])

#    # Gesture 13: Finger Left Arrow
gestures.append ([210.0, 210.0, 210.0, 210.0, 210.0, 210.0, 210.0, 210.0, 330.0, 330.0,
330.0, 330.0, 330.0, 330.0, 330.0, 330.0])

#    # Gesture 14: Finger Right Arraw
gestures.append ([330.0, 330.0, 330.0, 330.0, 330.0, 330.0, 330.0, 330.0, 210.0, 210.0,
210.0, 210.0, 210.0, 210.0, 210.0, 210.0])

#    # Gesture 15: Finger Zigzag
gestures.append ([0.0, 0.0, 0.0, 0.0, 0.0, 220.0, 220.0, 220.0, 220.0, 220.0, 220.0, 0.0,
0.0, 0.0, 0.0, 0.0])
```

```
#    # Gesture 16: Finger Flag
gestures.append ([90, 90.0, 90.0, 90.0, 90.0, 90.0, 0.0, 0.0, 0.0, 270.0, 270.0, 270.0,
270.0, 180.0, 180.0, 180.0])

#    # Gesture 17: Finger Circle
gestures.append ([348.75, 326.25, 303.75, 281.25, 258.75, 236.25, 213.75, 191.25, 168.75,
146.25, 123.75, 101.25, 78.75, 56.25, 33.75, 11.25])

#    # Gesture 18: Finger N Letter
gestures.append ([90, 90.0, 90.0, 90.0, 90.0, 310.0, 310.0, 310.0, 310.0, 310.0, 310.0,
90.0, 90.0, 90.0, 90.0, 90.0])

#    # Gesture 19: Finger M Letter
gestures.append ([90, 90.0, 90.0, 90.0, 315.0, 315.0, 315.0, 315.0, 45.0, 45.0, 45.0,
45.0, 270.0, 270.0, 270.0, 270.0])

#    # Gesture 20: Finger W Letter
gestures.append ([300.0, 300.0, 300.0, 300.0, 300.0, 60.0, 60.0, 60.0, 300.0, 300.0,
300.0, 60.0, 60.0, 60.0, 60.0, 60.0])

#    # Gesture 21: Finger N Triangle
gestures.append ([233, 233.0, 233.0, 233.0, 233.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 127.0,
127.0, 127.0, 127.0, 127.0])

good ← [ ]
new_gesture ← [ ]
```

Transform gestures into training examples

```
for x ∈ range (10):
    for gesture ∈ gestures:
        i ← 0
        good ← [ ]
        for angle ∈ gesture:
            i+ ← 1
            angle ← angle − 90
            if angle < 0:
                angle ← 360 + angle
            angle ← noise (angle)
            rad ← math.radians (angle)
            cos ← math.cos (rad)
            sin ← math.sin (rad)
            good.append (cos)
            good.append (sin)
        ‖ print i
        new_gesture.append (good)
```

```
file ← open('training_set.input', 'w')
file.write('220␣32␣22\n\n')
```

Save the training set into a file

```
ii ← 0
for i, gesture ∈ enumerate(new_gesture):

    for data ∈ gesture:
        if data = 0.0:
            w ← '0␣'
        elif data = 1.0:
            w ← '1␣'
        else:
            w ← '%1.9f␣' % (data)
        file.write(w)
    file.write('\n\n')

    if ii % 22 = 0:
        ii ← 0

    for j ∈ range(22):
        if j ≠ ii:
            file.write('-1')
        else:
            file.write('1')
        file.write('␣')
    || file.write('
    file.write('\n\n')

    ii+ ← 1
file.close()
```

C.3.2 Train Network

!/usr/bin/python

```
from pyfann import libfann

connection_rate ← 1
learning_rate ← 0.7
num_input ← 32
num_neurons_hidden ← 32
num_output ← 22
```

```
desired_error ← 0.000001
max_iterations ← 100000

ann ← libfann.neural_net ()

arg ← [num_input, num_neurons_hidden, num_output]
ann.create_standard_array (arg)

ann.set_learning_rate (learning_rate)
ann.set_training_algorithm (libfann.TRAIN_RPROP)

ann.set_activation_function_hidden (libfann.SIGMOID_SYMMETRIC)
ann.set_activation_function_output (libfann.SIGMOID_SYMMETRIC)

ann.set_rprop_increase_factor (1.2)
ann.set_rprop_decrease_factor (0.5)
ann.set_rprop_delta_min (0.0)
ann.set_rprop_delta_max (50.0)
ann.print_parameters ()

ann.train_on_file ("training_set.input", max_iterations, iterations_between_reports, desired_error)

ann.save ("neural.net")
```

C.3.3 Recognition class

```
!/usr/bin/python
```

```
from pyfann import libfann
import math

class Recognizer ():
    """Class␣to␣recognize␣a␣gesture␣with␣the␣neural␣network"""
    def __init__ (self):
        self.ann ← libfann.neural_net ()
        self.ann.create_from_file ("nn/neural.net")

    def recognize (self, input):
        """Transform␣the␣gesture␣into␣correct␣input
and␣submit␣it␣to␣the␣neural␣network"""
        gesture ← [ ]
        for i ∈ range (len (input) − 1):
            angle ← math.atan2 (input[i+1][1]−input[i][1], input[i+1][0]−input[i][0])
            gesture.append (math.cos (angle))
            gesture.append (math.sin (angle))

        self.ann.reset_MSE ()

        calc_out ← self.ann.run (gesture)
            ‖ return the founded gesture
        return calc_out.index (max (calc_out))
```

C.3.4 Configuration class

```
!/usr/bin/env python
```

```
from cPickle import Pickler, Unpickler

class Action ():
    """Define␣an␣action"""
    def __init__ (self, name):
        self.name ← name
        self.action ← None
        self.activated ← False

class Actions ():
    def __init__ (self, actions):
        self.actions ← actions

class Conf ():
    """Class␣that␣backup␣and␣load␣the␣configuration"""
    def __init__ (self):
        self.list ← [ ]

        try:
            self.actions ← self.loader ()
        except :
            self.actions ← self.default_build ()

    def backup (self):
        """Save␣the␣configuration␣into␣a␣file"""
        backup ← Pickler (open ('gestures.pickle', 'w'))
        backup.dump (self.actions)
    def loader (self):
        """Load␣the␣configuration"""
        datas ← Unpickler (open ('gestures.pickle', 'r'))
        actions ← datas.load ()
        return actions

    def default_builf (self):
        list ← [ ]
        gesture_names ← ["Finger␣UP", "Finger␣Down", "Finger␣Up-Down",
"Finger␣Down-Up", "Finger␣Right", "Finger␣Left", "Finger␣Left-Right",
"Finger␣Right-Left", "Finger␣Down-Right", "Finger␣Up-Left",
"Finger␣Rectangle", "Finger␣Down-Left", "Finger␣Up-Right",
"Finger␣Left␣Arrow", "Finger␣Right␣Arrow", "Finger␣ZigZag",
"Finger␣Flag", "Finger␣Circle", "Finger␣N␣Letter", "Finger␣M␣Letter",
    "Finger␣W", "Finger␣N"]

        for name ∈ gesture_names:
            action ← Action (name)
            list.append (action)

        return Actions (list)
```

C.4 GUI

C.4.1 Main program

!/usr/bin/env python

```
import gtk
import time
import pygtk
import gobject
import threading
from gobj_signal import DriverSignal
from gesture_windowgoo import GestureWindow
from config import ConfigWindow
import pynotify
import re
import sys, traceback
pygtk.require ('2.0')

from pointer import FingersDriver
from nn import recognizer, config
```

```
import virtkey
import subprocess

class FiingersGUI ():
    def simulate_keys (self, keys):
        """␣simulate␣the␣keys␣using␣python-virtkey
:param␣k:␣(modifiers,␣keysym);␣returned␣by␣keystroke_to_x11
Function␣copied␣from␣Gestikk␣http://gestikk.reichbier.de/gestikk
Under␣the␣terms␣of␣the␣Gnu␣GPL␣v2␣"""
        modifiers, key ← keys
        ∥ Debugger.debug('Simulating keystroke ...')
        v ← virtkey.virtkey ()
        if modifiers:
            v.lock_mod (modifiers)
        try:
            v.press_keysym (key)
            v.release_keysym (key)
        except Exception, e:
            print 'KEY␣SIMULATOR␣ERROR:', e
        finally:
            if modifiers:
                v.unlock_mod (modifiers)

    def keystroke_to_x11 (self, keystroke):
        """␣convert␣"<Control><Super>t"
:param␣keystroke:␣The␣keystroke␣string.
-␣can␣handle␣at␣least␣one␣'real'␣key
-␣only␣ctrl,␣shift,␣super␣and␣alt␣supported␣yet␣(case-insensitive)
:returns:␣tuple:␣(modifiers,␣keysym)
:see:␣http://ubuntuforums.org/showthread.php?p=4441207&postcount=5

Function␣copied␣from␣Gestikk␣http://gestikk.reichbier.de/gestikk
Under␣the␣terms␣of␣the␣Gnu␣GPL␣v2␣"""

        modifiers ← 0
        key ← ""
        splitted ← re.findall ('<[^>]+>', keystroke)    # gets ¡Control¿ and ¡Super¿
        ordinary ← re.findall ('(>|^)([^<]+)', keystroke)[0][1]    # gets 't'.
        for stroke ∈ splitted:
            lstroke ← stroke.lower ()
            if lstroke.startswith ('<') ∧ lstroke ≠ '<':
                lstroke ← lstroke[1: −1]
            if lstroke = "control":
                modifiers| ← gtk.gdk.CONTROL_MASK
            elif lstroke = "shift":
                modifiers| ← gtk.gdk.SHIFT_MASK
            elif lstroke = "alt":
                modifiers| ← gtk.gdk.MOD1_MASK
            elif lstroke = "mod2":
                modifiers| ← gtk.gdk.MOD2_MASK
            elif lstroke = "mod3":
                modifiers| ← gtk.gdk.MOD3_MASK
            elif lstroke = "super" ∨ lstroke = "mod4":
                modifiers| ← gtk.gdk.MOD4_MASK
            elif lstroke = "mod5":
                modifiers| ← gtk.gdk.MOD5_MASK
            else:
                raise Exception ('Unknown␣Modifier:␣%s' % lstroke)
        key ← gtk.gdk.keyval_from_name (ordinary)
        return (modifiers, key)

    def quit_cb (self, widget, data ← None):
        """Function␣called␣to␣quit␣the␣program"""
        if data:
            data.set_visible (False)
        try:
            self.w.abort ()
            self.w.running2 ← False
        except :
            print 'no␣wiimote␣connected␣at␣all'
        gtk.main_quit ()

    def gesture_trigger (self, widget, data ← None):
        """Launch␣gesture␣mode"""
        self.signals.set_property ('gesture', True)

    def gesture (self):
        """Create␣the␣gesture␣window␣and␣assign␣callbacks"""
        self.gestureApp ← GestureWindow (self.signals)
        self.handlerPrint ← self.signals.connect ('notify::printpoints',
self.gestureApp.printpoints_cb)
    self.handlerStarting ← self.signals.connect ('notify::startingpoints',
self.gestureApp.startingpoints_cb)
    self.handlerGesture ← self.signals.connect ('notify::gesturepoints',
self.gestureApp.gesturepoints_cb)
    self.signals.connect ('notify::recognize', self.recognize_cb)

    def reloadaction_cb (self, obj, property):
        """Called␣to␣saved␣the␣configuration"""
        action ← obj.get_property ('saveaction')
        self.configuration ← action
        self.config.actions.actions ← action
        self.config.backup ()
    def gesture_cb (self, obj, property):
        """Activate/Desactivate␣the␣gesture␣Mode"""
        if obj.get_property ('gesture') = True:
            print "Switching␣to␣Gesture␣Mode"
            self.w.running ← False
            self.gestureApp.w.show ()
            self.gestureApp.running ← True
        else:
            print "Switching␣to␣Mouse␣Mode"
            self.w.running2 ← False
            self.gestureApp.running ← False

    def recognize_cb (self, obj, property):
        """Called␣by␣the␣driver␣when␣a␣gesture␣is␣recognized
It␣execute␣the␣binded␣action"""
        gesture ← self.signals.get_property ('recognize')
        print "gesture", gesture
        try:
            result ← self.recognizer.recognize (gesture)
            print 'passed'
            self.gestureApp.close ()
            self.w.running2 ← False
        except :
            print '-' ∗ 60
            traceback.print_exc (file ← sys.stdout)
            print '-' ∗ 60
        if result ≠ None:
            || find relevant Action and launch it
            print 'action␣recognized␣=␣%i' % (result)
            message ← "Gesture␣%s␣recognized" %
                   (self.configuration[result].name)
            print message

            self.signals.set_property ('gesture', False)

            if self.configuration[result].activated:
                if self.configuration[result].action = 'key':
                    || fake keystroke
                    key ← self.keystroke_to_x11 (self.configuration[result].keystroke)
                    self.simulate_keys (key)
                    pass
                else:
                    || laucnh program
                    try:
                        subprocess.Popen (self.configuration[result].program)
                    except :
                        print "couldn’t␣lauch␣programme␣%s" %
                              (self.configuration[result].program)
                    pass

        else:
            print ’not␣foud’

        n ← pynotify.Notification ("Connexion", "Please␣press␣buttons␣1␣and␣2
on the wiimote to get connected",␣"dialog − warning")

    n.set_urgency (pynotify.URGENCY_CRITICAL)
    n.set_timeout (1000000)
    n.attach_to_status_icon (self.statusIcon)
    n.show ()

    def popup_menu_cb (self, widget, button, time, data ← None):
        """Manage␣the␣popup␣menu"""
        data ← self.build_menu ()
        if data:
            data.show_all ()
            data.popup (None, None, None, 3, time)
            pass

    def activate_icon_about (self, widget, data ← None):
        """Print␣a␣the␣classic␣About␣Box"""
        msgBox ← gtk.MessageDialog (parent ← None, buttons ←
              gtk.BUTTONS_OK,
message_format ← "fiigers␣is␣a␣finger␣tracking␣mouse␣driver␣and␣gesture
recognition software.It works with a Nintendo Wiimote ∧ gloves.")
    msgBox.run ()
    msgBox.destroy ()

    def prefPanel (self, widget, data ← None):
        """Create␣the␣configuration␣Window"""
        ConfigWindow (self.configuration, self.signals)
    def connectWiimote (self, widget, data ← None):
        """Wiimote␣connexion"""
        print "Press␣(1)␣and␣(2)␣on␣the␣Wiimote"
        n ← pynotify.Notification ("Connexion", "Please␣press␣buttons␣1␣and␣2
on the wiimote to get connected",␣"dialog − warning")
    n.set_urgency (pynotify.URGENCY_NORMAL)
    n.set_timeout (5000)
    n.attach_to_status_icon (self.statusIcon)
    n.show ()
    self.w ← FingersDriver.MouseDriver (self)
    connexion ← self.w.connectWiimote ()
    if connexion:
        self.w.running ← True
        self.w.running2 ← False
        self.gesture ()
        threading.Thread (target ← self.w.runner).start ()
        print "Success:␣thread␣supposed␣to␣run"
        self.statusIcon.set_from_stock (gtk.STOCK_CONNECT)
        m ← pynotify.Notification ("Connexion", "You␣are␣now␣connected", "dialog-info")
        m.set_urgency (pynotify.URGENCY_NORMAL)
        m.set_timeout (5000)
        m.attach_to_status_icon (self.statusIcon)
        m.show ()
        return True
    else:
        print "Failed"
        del self.w
        m ← pynotify.Notification ("Connexion", "Connection␣failed,
please check battery level ∧ tryagain.",␣"dialog − error")
    m.set_urgency (pynotify.URGENCY_NORMAL)
    m.set_timeout (5000)
    m.attach_to_status_icon (self.statusIcon)
    m.show ()
    self.statusIcon.set_from_stock (gtk.STOCK_DISCONNECT)
    return False

    def disconnectWiimote (self, widget, data ← None):
        self.w.abort ()
        self.statusIcon.set_from_stock (gtk.STOCK_DISCONNECT)
        del self.w

    def delete_event (self, widget, event, data ← None):
        return False

    def destroy (self, widget, data ← None):
        gtk.main_quit ()
    def __init__ (self, queue):
        """Create␣the␣GUI␣bases"""
        gtk.gdk.threads_init ()
        self.queue ← queue

        self.config ← config.Conf ()
        self.configuration ← self.config.actions.actions

        pynotify.init ('fiingers')
        self.signals ← DriverSignal ()
        self.signals.connect ('notify::gesture', self.gesture_cb)
        self.signals.connect ('notify::saveaction', self.reloadaction_cb)

        self.statusIcon ← gtk.StatusIcon ()

        self.statusIcon.set_from_stock (gtk.STOCK_HOME)
        self.statusIcon.set_tooltip ("fiingers")

        self.statusIcon.connect ('activate', self.popup_menu_cb, None, 0)
        self.statusIcon.connect ('popup-menu', self.popup_menu_cb)
        self.statusIcon.set_visible (True)

        self.recognizer ← recognizer.Recognizer ()

    def main (self):
        gtk.main ()

    def build_menu (self):
        """Create␣the␣Status␣Icon␣popup␣menu␣content"""
        menu ← gtk.Menu ()
        menu.set_title ("Fiingers")

        #    # Connexion
        try:
            if self.w:
                menuItem ← gtk.ImageMenuItem (gtk.STOCK_DISCONNECT)
                menuItem.connect ('activate', self.disconnectWiimote)
                menu.append (menuItem)

                menuItem ← gtk.ImageMenuItem ('Gesture')
                img ← gtk.image_new_from_stock (gtk.STOCK_MEDIA_RECORD,
gtk.ICON_SIZE_MENU)

    menuItem.set_image (img)
    menuItem.connect ('activate', self.gesture_trigger)
    menu.append (menuItem)
    except :
        menuItem ← gtk.ImageMenuItem (gtk.STOCK_CONNECT)
        menuItem.connect ('activate', self.connectWiimote)
        menu.append (menuItem)

    #   # Setup Tool
    menuItem ← gtk.ImageMenuItem (gtk.STOCK_PREFERENCES)
    menuItem.connect ('activate', self.prefPanel)
    menu.append (menuItem)

    #   # About Box
    menuItem ← gtk.ImageMenuItem (gtk.STOCK_ABOUT)
    menuItem.connect ('activate', self.activate_icon_about)
    menu.append (menuItem)

    #   # Quit App
    menuItem ← gtk.ImageMenuItem (gtk.STOCK_QUIT)
    menuItem.connect ('activate', self.quit_cb, self.statusIcon)
    menu.append (menuItem)
    return menu

if __name__ = '__main__':
    interface ← FiingersGUI ()
    interface.connectWiimote ()
    interface.main ()
```

C.4.2 Signals

```
import pygtk
pygtk.require ('2.0')

import gobject
from collections import deque

class DriverSignal (gobject.GObject):
    """The␣class␣implenting␣communication␣between␣process"""
    __gproperties__ ← {
        'gesture': (gobject.TYPE_BOOLEAN, 'To␣(des)activate␣Gesture␣Mode',
'', False, gobject.PARAM_READWRITE),
    'printpoints': (gobject.TYPE_PYOBJECT, 'Call␣to␣draw␣the␣finger␣position',
'', gobject.PARAM_READWRITE),
    'startingpoints': (gobject.TYPE_PYOBJECT, 'Call␣to␣draw␣points',
'', gobject.PARAM_READWRITE),
    'gesturepoints': (gobject.TYPE_PYOBJECT, 'Call␣to␣draw␣the␣gesture␣stroke',
'', gobject.PARAM_READWRITE),
    'recognize': (gobject.TYPE_PYOBJECT, 'Call␣to␣recognize␣a␣gesture',
'', gobject.PARAM_READWRITE),
    'saveaction': (gobject.TYPE_PYOBJECT, 'Call␣to␣save␣gesture␣mapping␣configuration
'', gobject.PARAM_READWRITE)
    }

    def __init__ (self):
        gobject.GObject.__init__ (self)
        self.gesture ← False
        self.startingpoints ← None
        self.queue ← deque ()
        self.gpoints ← None
        self.recognize ← None
        self.action ← None

    def do_get_property (self, property):
        """To␣retrieve␣the␣value␣of␣a␣property"""
        if property.name = 'gesture':
            return self.gesture
        elif property.name = 'printpoints':
            return self.queue.popleft ()
        elif property.name = 'startingpoints':
            return self.startingpoints
        elif property.name = 'gesturepoints':
            return self.gpoints
        elif property.name = 'recognize':
            return self.recognize
        elif property.name = 'saveaction':
            return self.action
        else:
            raise AttributeError, 'unknown␣property␣%s' % property.name

    def do_set_property (self, property, value):
        """To␣set␣the␣value␣of␣property,␣this␣will␣trigger
the␣registered␣callback␣function"""
```

```
        if property.name = 'gesture':
            self.gesture ← value
        elif property.name = 'printpoints':
            self.queue.append (value)
        elif property.name = 'startingpoints':
            self.startingpoints ← value
        elif property.name = 'gesturepoints':
            self.gpoints ← value
        elif property.name = 'recognize':
            self.recognize ← value
        elif property.name = 'saveaction':
            self.action ← value
        else:
            raise AttributeError, 'unknown␣property␣%s' % property.name

gobject.type_register (DriverSignal)
```

C.4.3 Configuration

!/usr/bin/env python

```
import pygtk
pygtk.require ('2.0')
import gtk

class KeyRecognizer (gtk.Entry):
    """␣simple␣widget␣for␣recognizing␣shortcuts␣(copied␣from␣gestikk␣source␣code)
\t\tUnder␣the␣terms␣of␣the␣Gnu␣GPL␣v2␣"""

    def __init__ (self):
        gtk.Entry.__init__ (self)
        self.set_property ('editable', False)
        self.connect ('key-press-event', self.sig_keypress)

    def sig_keypress (self, w, event):
        """␣signal:␣key␣press␣event!␣"""
        if event.state & gtk.gdk.MOD2_MASK = gtk.gdk.MOD2_MASK:
            || remove numlock
            event.state^ ← gtk.gdk.MOD2_MASK
        if event.state & gtk.gdk.MOD4_MASK = gtk.gdk.MOD4_MASK ∧
              event.state & gtk.gdk.SUPER_MASK = gtk.gdk.SUPER_MASK:
            event.state^ ← gtk.gdk.MOD4_MASK    # only SUPER, not MOD4
        self.set_text (gtk.accelerator_name (event.keyval, event.state))
```

```
class FileSelection ():
    """This␣widget␣permit␣to␣select␣a␣program␣to␣bind␣to␣a␣gesture"""

    def file_ok_sel (self, w):
        """Executed␣when␣a␣file␣is␣selected"""
        self.parent.program_name.set_text (self.filew.get_filename ())
        self.filew.destroy ()

    def __init__ (self, parent):
        self.parent ← parent

        || Create a new file selection widget
        self.filew ← gtk.FileSelection ("File␣selection")
        self.filew.ok_button.connect ("clicked", self.file_ok_sel)
        || Connect the cancel button to destroy the widget
        self.filew.cancel_button.connect ("clicked", lambda w: self.filew.destroy ())
        self.filew.show ()

class ActionWindow ():
"""This␣is␣the␣window␣to␣setup␣a␣gesture"""

    def backup (self, widget):
    """Save␣the␣configuration␣of␣a␣gesture"""
        if self.activated.get_active ():
            self.config[self.number].activated ← True
            else:
                self.config[self.number].activated ← False

            if self.keystrokeR.get_active ():
                self.config[self.number].action ← 'key'
                self.config[self.number].keystroke ← self.keystroke_name.get_text ()
            else:
                self.config[self.number].action ← 'prog'
                self.config[self.number].program ← self.program_name.get_text ()

            self.signal.set_property ('saveaction', self.config)

    def close_application (self, widget, event, data ← None):
        return False

    def select_executable (self, widget, data ← None):
        FileSelection (self)
    def __init__(self, number, signal, config):
        """The␣draw␣the␣geture␣configuration␣Window"""
        self.number ← number
        self.signal ← signal
        self.config ← config

        window ← gtk.Window(gtk.WINDOW_TOPLEVEL)
        window.connect("delete_event", self.close_application)
        window.set_border_width(10)
        window.show()

        box ← gtk.VBox()

        self.programR ← gtk.RadioButton(None, label ←
              "Launch␣the␣following␣Program␣:")
        self.keystrokeR ← gtk.RadioButton(self.programR, label ←
              "Simulate␣the␣following␣keystoke␣:")
        self.programR.show()

        self.activated ← gtk.CheckButton(label ← "Activate␣action")
        box.pack_start(self.activated)
        self.activated.show()

        box.pack_start(self.programR)
        self.program_name ← gtk.Entry(max ← 0)

        box.pack_start(self.program_name)
        self.program_name.show()

        button ← gtk.Button(label ← "set")
        box.pack_start(button)
        button.show()

        button.connect("clicked", self.select_executable)

        box.pack_start(self.keystrokeR)
        self.keystrokeR.show()

        self.keystroke_name ← KeyRecognizer()
        box.pack_start(self.keystroke_name)
        self.keystroke_name.show()

        button2 ← gtk.Button(label ← "Save␣Action")
        box.pack_start(button2)
        button2.show()
        button2.connect ('clicked', self.backup)

        # setting up the window
        if self.config[self.number].activated:
            self.activated.set_active (True)

        if self.config[self.number].action = 'key':
            self.keystrokeR.set_active (True)
            self.keystroke_name.set_text (self.config[self.number].keystroke)
        elif self.config[self.number].action = 'prog':
            self.programR.set_active (True)
            self.program_name.set_text (self.config[self.number].program)

        box.show ()
        window.add (box)

class ConfigWindow:
    """This␣is␣the␣main␣configuration␣window"""

    def close_application (self, widget, event, data ← None):
        return False

    def save_action (self, widget, data ← None):
        pass

    def custom_action (self, widget, data ← None):
        ActionWindow (data, self.signal, self.config)

    def __init__ (self, config, signal):
        ‖ create the main window and attach delete event signal to terminating the
        ‖ application
        self.config ← config
        self.signal ← signal

        window ← gtk.Window (gtk.WINDOW_TOPLEVEL)
        window.connect ("delete_event", self.close_application)
        window.set_border_width (10)
        window.show ()

            ‖ The table to pack all the gesture buttons
        table ← gtk.Table (5, 5, True)
        window.add (table)

        row ← 0
        column ← 0
        for i, actions ∈ enumerate (self.config):
            if column = 5:
                print "reset␣column"
                column ← 0
                row+ ← 1

            ‖ buttons
            image ← gtk.Image ()
            string ← "images/%i.png" % (i)
            image.set_from_file (string)
            image.show ()
            ‖ a button to contain the image widget
            button ← gtk.Button ()
            button.add (image)
            table.attach (button, column, column + 1, row, row + 1)
            button.show ()
            button.connect ("clicked", self.custom_action, i)

            column ← column + 1

        table.show ()
```

C.4.4 Gesture drawing

!/usr/bin/env python coding=utf-8

```
import time

import gobject
gobject.threads_init ()

import pygtk
pygtk.require ("2.0")

import gtk
import goocanvas

class GestureWindow ():

    def close (self, widget ← None, data ← None):
        """tell␣the␣GUI␣to␣close␣gesture␣window"""
        self.sig.set_property ('gesture', False)
        self.running ← False
    def __init__ (self, sig):
        ‖ To send signals
        self.sig ← sig
        self.gestureP ← [ ]
        self.running ← False
            ‖ Create the Canvas
        self.c ← goocanvas.Canvas ()
        self.w ← gtk.Window ()
            ‖ Attach it to a window
        self.w.add (self.c)
        self.w.show_all ()
        self.w.connect ("key_press_event", self.close)
        self.w.show ()
        self.w.fullscreen ()
        self.w.hide ()

            ‖ The starting point green circle
        self.startingp1 ← goocanvas.Ellipse (center_x ← 0, center_y ← 0, radius_x ←
                40, radius_y ← 40, fill_color ← "green")

            ‖ The square representing the finger
        self.finger1 ← goocanvas.Rect (x ← 0, y ← 0, width ← 20, height ←
                20, fill_color ← "blue")

        self.c.get_root_item ().add_child (self.startingp1)
        self.c.get_root_item ().add_child (self.finger1)

    def gesturepoints_cb (self, obj, property):
        """Called␣by␣the␣driver␣to␣draw␣the␣stroke"""
        point ← self.sig.get_property ('gesturepoints')
        if point ≢ None:
            p_points ← goocanvas.Points (point)
            self.polyline ← goocanvas.Polyline (points ← p_points, end_arrow ←
                    True)
            self.c.get_root_item ().add_child (self.polyline)

    def printpoints_cb (self, obj, property):
        """Called␣by␣the␣driver␣to␣move␣the␣finger␣projection␣on␣screen"""
        point ← self.sig.get_property ('printpoints')
        if point ≢ None:
            self.finger1.set_simple_transform (point.x, point.y, 1, 0)
        else:
            self.finger1.set_simple_transform (1, 1, 1, 0)

        self.finger1.request_update ()
    def startingpoints_cb (self, obj, property):
        """Called␣by␣the␣driver␣to␣move␣the␣starting␣area␣of␣a␣gesture"""
        point ← self.sig.get_property ('startingpoints')
        if point ≢ None:
            self.gestureP.append ((point.x, point.y))
            self.startingp1.set_simple_transform (point.x, point.y, 1, 0)
        else:
            self.startingp1.set_simple_transform (0, 0, 1, 0)

        self.startingp1.request_update ()
```

C.5 TkPlotting Utility

!/usr/bin/env python coding=utf-8

```
from Tkinter import *
from pointer import linuxWiimoteLib3

def circle (x, y, r, coul ← 'black'):
    """Trace␣a␣circle␣from␣(x,y)␣center␣with␣a␣radius␣r"""
    can.create_oval (x − r, y − r, x + r, y + r, outline ← coul)

def draw_points ():
    "Retrieve␣and␣draw␣the␣detected␣points␣on␣screen"
    w ← linuxWiimoteLib3.Wiimote ()   # instanciate the wiimote driver
    w.Connect ();   # ask to get connect the wiimote
    w.activate_IR ()
    while 1:
        detected ← 0
        can.update_idletasks ()
        can.delete (ALL)   # clean the canvas

        if w.WiimoteState.IRState.Found1:
            circle (1024 − w.WiimoteState.IRState.RawX1, 768 −
                    w.WiimoteState.IRState.RawY1, 10, 'red')
            detected+ ← 1

        if w.WiimoteState.IRState.Found2:
            circle (1024 − w.WiimoteState.IRState.RawX2, 768 −
                    w.WiimoteState.IRState.RawY2, 10, 'blue')
            detected+ ← 1
        if w.WiimoteState.IRState.Found3:
            circle (1024 − w.WiimoteState.IRState.RawX3, 768 − w.WiimoteState.IRState.RawY3, 10, 'purple')
            detected+ ← 1

        if w.WiimoteState.IRState.Found4:
            circle (1024 − w.WiimoteState.IRState.RawX4, 768 − w.WiimoteState.IRState.RawY4, 10, 'green')
            detected+ ← 1

        can.create_text (10, 10, text ← detected)

#    #    # Main Program :

fen ← Tk ()
can ← Canvas (fen, width ← 1024, height ← 768, bg ← 'ivory')    # create the drawing area
can.pack (side ← TOP, padx ← 5, pady ← 5)

b1 ← Button (fen, text ← 'Connect␣Wiimote', command ← draw_points)    # when the button is pressed the program start
b1.pack (side ← LEFT, padx ← 3, pady ← 3)

fen.mainloop ()    # the main loop
```