Adding Kinect Support to MINDs
Technical Report – MINES ParisTech A/505/CRI

Laurent DAVERIO

CRI, Maths & Systems, MINES ParisTech, France

October 24, 2012

I Introduction

The "**MINDs**" music therapy application [1] was developed at CRI/MINES ParisTech by Benoît Pin, Samuel Benveniste and Pierre Jouvelot. Its aims at helping Alzheimer's Disease patients exercise their memory by "performing" popular songs of their youth using modern technology tools such as wireless Nintendo ***Wiimote*** [2] game controllers.

By the end of 2010, Microsoft released the **Kinect** [3] sensor, a new type of sensing input device for the *Xbox 360* game console. The Kinect has often been called revolutionary, as user interaction only relies on ***gestures*** and spoken commands, *without the need to hold or touch a physical games controller*.

Figure 1: The Microsoft Kinect

The Kinect essentially consists in a **RGB camera** coupled with an **infrared depth sensor** [4], using a technology developed by Israeli company **PrimeSense** [5]. Supporting software allows for tracking the position and gestures of several users sitting or standing in front of the device. Advanced tracking libraries by Microsoft even attempt to recognize the age, gender and expression (smile, frown, etc.) of the users.

After initially marketing the Kinect as a closed, Xbox-only device (which prompted hackers to successfully try and reverse-engineer the Kinect's communication protocol), Microsoft soon realised what benefits could be gained from having a dynamic developers base, and diametrically reversed their stance. Today, the Kinect can be used on all leading operating systems through the availability of two software development kits ("SDKs"):

- **Microsoft' Kinect SDK [6]**, a proprietary SDK for Windows-based applications only.
- **PrimeSense's OpenNI** ("*Open Natural Interaction*"), an **open source**, **multiplatform** SDK (Windows, Linux, OS X, Android).

Given that supporting software tools and libraries were freely available, attempting the integration of a Kinect sensor into the MINDs application made perfect sense.

In this paper we will outline:

- the ***steps*** *we followed* to get a working prototype
- what ***software stack*** we used,
- what ***results*** were obtained, and what ***experience*** was gained in the process.

II Available Kinect software support

II.1 Overview

Freely available Kinect support software operates at various levels, which are outlined in the table below:

High-level support	
Gesture recognition Detect a certain number of gestures (*e.g. a hand wave*) and react accordingly.	**NITE** http://www.openni.org (Costless, proprietary OpenNI plugin)
Skeleton recognition Detect the presence of persons in front of the sensor, and generate "skeletons" representing the corresponding body parts and joints (*hands, head, arms, elbows, shoulders, legs, knees, etc.)*	**OpenNI** http://www.openni.org (Free software – LGPL license)
RGB (webcam) + depth bitmaps Access the raw data returned by the Kinect. No form of image analysis, share recognition, etc. is performed.	**Freenect** http://openkinect.org (Apache/GPL license)
USB communication Allow user programs to access and control the USB port to which the Kinect is plugged.	**LibUSB** http://www.libusb.org (Free software – LGPL license)
Low-level support	

Figure 2: software overview

For the sake of efficiency, we must try and tap software at the highest-available level. In the process of looking for a solution, two approaches were experimented:

- a *lower-level* approach based on ***Freenect*** libraries, and
- a *higher-level* approach based on ***OpenNI + NITE***.

Note *Due to its proprietary, "Windows-only" nature, the* **Microsoft Kinect SDK** *has been left out in favour of more open alternatives like* **OpenNI**.

II.2 Freenect

The ***Freenect*** libraries allows the developer to access the raw ***RGB*** (webcam) + ***depth*** bitmap information. Figure 3 below illustrates this:

- On the left-hand side, **depth information** is coded with colour gradients (warmer colours represent shorter distances). Depth is encoded in 2048 steps, ranging from 1 m to 3.5 m approximately.
- On the right hand side, the VGA (640x480) **video stream** is similar to what can be obtained using a regular webcam.

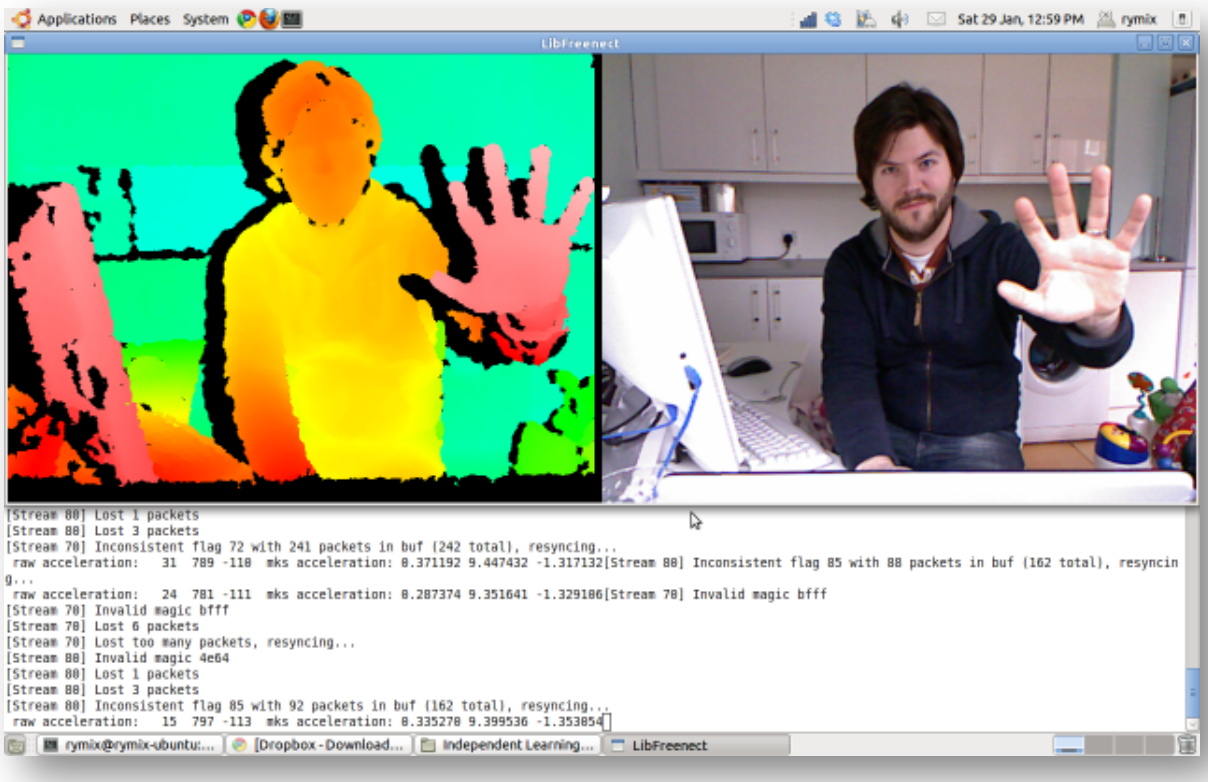

Figure 3: A visualization of Freenect data output [7]

This low-level information can then be manipulated by whatever means you choose. For instance, it's possible to exploit the depth information only, and discard the RGB stream. But discarding part of the returned information is likely to lead to less accurate scene detection.

A very common solution is to use the ***OpenCV*** [8] ("*Open Computer Vision*") libraries, which offer functions related to real-time computer vision: face recognition, gesture recognition, motion tracking, etc.

II.3 OpenNI / NITE

OpenNI is a set of open source (LGPL) "Natural Interaction" libraries written in Java and C++. They are maintained by PrimeSense, the company behind the Kinect hardware technology, and also support other PrimeSense-designed devices such as the *Asus Xtion* [9] sensor family.

Figure 4: Asus Xtion sensor

OpenNI is able to track one or several human figures in a 3D scene, and produce a higher-level output, such as ***real-time hand/finger tracking***, or *skeleton generation* (Figure 5 below).

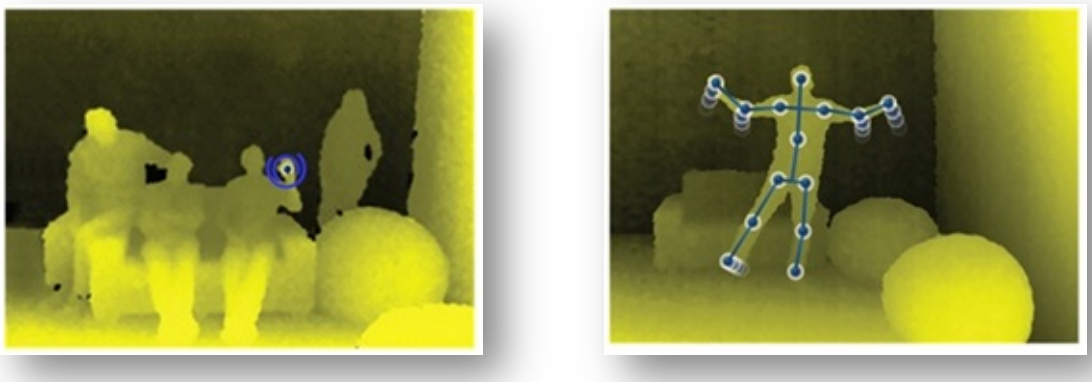

Figure 5: OpenNI higher-level output [10]

In addition to *OpenNI*, PrimeSense also maintains **NITE**, a proprietary *OpenNI* plug-in aimed at ***gesture recognition***. A few gestures are implemented out of the box:

- "**Click**": the user holds her open hand facing the camera, fingers pointing upwards. She then quickly "pushes" her palm towards the camera, and back again.
- "**Wave**": the user waves her open hand facing the camera.
- "**RaiseHand**": the user raiser her hand.

III Integrating Kinect support into MINDs

A thorough web search into related projects was conducted. This led us to elaborating a set of guidelines, and experimenting two approaches, one based on ***Freenect***, and one on ***OpenNI***.

III.1 Guidelines

1. Proposed Kinect support must integrate into MINDs taking into account the specificities of that project:

- MINDs is **free software** (released under the *GPLv3* license).
- MINDs is **multi-platform** (supports Windows, Linux and OS X).
- MINDs is written **using the Python language**.

→ the adopted solution must also be ***free***, ***multi-platform***, and ***written in Python***.

2. In the current release of MINDs, *Wiimote* controllers are used as **multi-button mice**.

→ logically, in the first stage of our experimentation, we should try to **emulate a mouse** (pointer movements + button clicks) using the Kinect sensor.

III.2 Using Freenect

The first approach we tried is based on a Python "recipe" we found on the Internet [11]. It is based on the following tools:

- The ***Freenect*** library and its ***Python bindings*** (for handling Kinect's raw data)
- The ***OpenCV*** library and its ***Python bindings*** (for real-time image analysis)
- The Python ***NumPy*** module (for array manipulation)

It was tried only on a ***Linux*** platform. Popular Linux distributions, such as ***Ubuntu Linux***, natively offer all tools and libraries required to run the script, thus simplifying all installation issues.

a. Principle

This recipe implements a simplified version of a Matlab function, *regionprops()* [12], which measures properties of image regions. The result is rather crude, and works only to a certain point.

The algorithm is as follows:

- 3D data points are first ***thresholded by distance***, keeping only the nearest ones, so as to distinguish them from the scene background. The remaining points are then ***sub-divided into groups*** for which we compute contours and convex hulls.
  Ideally, there should only be one such group, representing the hand of the user we want to track. The hand position will be determined as the ***centroid*** of this group.
- In addition, ***a mouse click will be simulated*** when the user ***closes her fist***. Contours of the group will then "look like a small circle".

b. Issues

Once implemented, this method allowed some control over the mouse pointer and click, but numerous issues made it very uncomfortable to use, prompting us to look for a more sophisticated solution:

- The ***distance threshold parameter is very difficult to adjust***. Small values (1 m) allow to detect the hands, but only when the used is very close to the Kinect. This is unusable in a larger room, for instance if several users are to sit in front of the sensor.
- The algorithm tends to detect a lot of "***false positives***", such the other hand of the user, a forearm, etc. This makes movement tracking and click simulation very erratic.

III.3 Using OpenNI

This method was successfully experimented both under ***Linux*** and ***Windows***. As is detailed in the "***Issues***" section below, the increased complexity of the underlying software stack posed no particular problem under Linux, but required much more trial-and-error experimentations to get it to compile under Windows.

We resorted to the following tools:

- The **OpenNI** natural interaction libraries
- The **NITE** gesture tracking plug-in to OpenNI
- The **PyOpenNI [13]** Python bindings to OpenNI

The **PyOpenNI** Python binding is designed and maintained by ***Xavier Mendez***, a young developer from Barcelona, Spain. It is written in C++, which makes its installation more complicated as compared to a "pure Python" module, and platform-dependent. But Xavier was very helpful, and his advice, both through the official OpenNI mailing list, and by e-mail, proved essential to the success of our experiment.

a. Principles

The use of OpenNI / PyOpenNI makes the solution very simple to develop. Basically, OpenNI can provide us with:

- A "***Gesture Generator***": this object will be able to detect NITE-defined gestures such as "*Click*", "*Wave*" or "*RaiseHand*", and execute *callback functions* as a result.
- A "***Hands Generator***": this object will be able to track a designated hand, returning a 3D $\underline{(x,y,z)}$ position in real-time. For the purposes of mouse pointer simulation, only the (x,y) coordinates will be retained.

b. Issues

As we mentioned above, the main issues we encountered were related to compiling and installing the PyOpenNI module under Windows. The whole step-by-step process was documented in detail, and is currently being added to the official PyOpenNI documentation wiki.

A similar installation under OS X should be possible but might require some work, too. Luckily, the experience acquired during our initial unsuccessful attempts under Windows should prove very useful in order to reach the solution faster.

It should be noted, however, that once PyOpenNI has been successfully compiled on a given platform, the resulting shared libraries can be distributed without having to go though the hassle of a recompilation.

IV Results and perspectives

The two approaches exposed in the previous section ("*Freenect*" and "*OpenNI*") were explored and integrated into a new, experimental version of MINDs (although the "*OpenNI*" approach looks much more promising, and will most likely be the one retained in future work). This new version is only a work in progress, a first step towards a more complete solution: at this stage, we were not trying to provide a fully packaged, ready-to-use solution, but rather a proof-of-concept prototype.

In this regard, the experiment is a total success, as the prototypes successfully run under Linux and Windows.

The next step will be to demonstrate and test the prototypes with real users.

Future paths of experimentation might include refining gesture tracking, possibly implementing custom additional gestures, and implement multiple hand tracking to allow for multi-user interaction.

Acknowledgement

We would like to thank Xavier Mendez, who designed the PyOpenNI module, and whose support was invaluable in making our solution work on the Windows platform.

References

[1] **MINDs** — http://minwii.org

[2] **Wiimote** — https://en.wikipedia.org/wiki/Wii_Remote

[3] **Kinect** — https://en.wikipedia.org/wiki/Kinect

[4] The Kinect also offers a microphone array, for spatial sound recognition, a vertical-tilt motor, and a multi-coloured led which can be controlled by software.

[5] **PrimeSense** – http://www.primesense.com

[6] **Kinect for Windows** — https://www.microsoft.com/en-us/kinectforwindows

[7] Source of the picture: http://rymixxx.wordpress.com/2011/01/29/linux-freenect-demos

[8] **OpenCv** — https://en.wikipedia.org/wiki/OpenCV

[9] **Asus Xtion** — http://www.asus.com/Multimedia/Motion_Sensor/Xtion_PRO_LIVE

[10] Source of the pictures: OpenNI official documentation.

[11] **OpenKinect Mouse Control Using Python** — http://code.activestate.com/recipes/578104

[12] **The Matlab regionprops() function** — http://www.mathworks.fr/fr/help/images/ref/regionprops.html

[13] **PyOpenNI** — https://github.com/jmendeth/PyOpenNI

MINDs+Kinect Documentation

Release 2.1a

Laurent DAVERIO

October 24, 2012

Contents

CHAPTER 1

Project Overview

The aim of this experimentation is to identify existing free software stacks available to add Kinect support to MINDs and other Python projects.

More specifically, the first expected outcome is a solution to **emulate a mouse through real-time hand tracking**.

A quick overview of available solutions returns tools operating at various levels:

1. **Low-level**: USB driver for the Kinect device. Raw video and 3D data capture.
2. **High-level**: scene detection, hand tracking, gesture tracking.

At a second stage, the relevant tools should be integrated into MINDs, and tested "in the field".

1.1 Low-level support

The starting point is the **OpenKinect** (http://openkinect.org) website, which provides links and pointers to the available software, as well as build instructions.

- **libusb**: a suite of user-mode routines for controlling data transfer to and from USB devices on Unix-like systems without the need for kernel-mode drivers.

  URL: http://libusb.org/

- **libfreenect** : Drivers and libraries for the Kinect device on Windows, Linux, and OS X.

  URL: https://github.com/OpenKinect/libfreenect

  libfreenect offers **Python bindings**, a prerequisite for our needs.

- **OpenCV** (*Open Computer Vision*) is a library of programming functions for real-time image analysis and display. We will use it to implement a crude hand-tracking system.

  URL: http://code.opencv.org/projects/OpenCV/wiki/WikiStart

  OpenCV offers **Python bindings**, too.

1.2 High-level support

- Microsoft's Kinect for Windows SDK is a proprietary, Windows-only SDK. As we are looking to build a multiplatform solution, this solution must be left aside..
- **OpenNI** (*Open Natural Interaction*) is an open-source framework providing a high-level Java/C++ API for writing applications utilizing **Natural Interaction**. It is developed by *PrimeSense*, the Israeli company having designed the Kinect's hardware.

URL: http://openni.org

A gesture recognition library, **NITE**, is built on top of OpenNI. Although it is a proprietary pluin, it's free of use.

Python bindings for OpenNI/NITE are provided by a third-party module, PyOpenNI. It is developed and maintained by Xavier Mendez, an young developer from Barcelona, Spain.

1.3 Integration with MINDs

The existing version of MINDs uses a specific Python module to emulate a mouse. It offers a binary choice at launch (*WiiMote* / no *Wiimote*). When Wiimote support is not selected, the application falls back to mouse-only operation.

The first steps for integration would then be :

- To replace the binary choice (*WiiMote* / no *Wiimote*) with a more flexible "controller type" selection.
- To locate the Wiimote middleware component in MINDs, and develop a Kinect middleware module using the same interface.

At a later time, a more extensive refactoring could/should be done, eventually leading to a MINDs v3 release.

CHAPTER 2

Ubuntu Linux PyOpenNI setup instructions

This explains how to install the dependencies required to get MINDs working on **Ubuntu Linux 12.04.1** *"Pangolin"* (32/64-bit).

→ It is loosely based on this page [openkinect.org].

2.1 Setup Python virtual environment

```
# Dev environment
sudo apt-get install mc git subversion python-dev python-virtualenv fabric

# Get dev files from git (Note: temporary repository URL)
git clone ssh://vialfre/Users/daverio/projects/git/kinect.git

# Setup virtualenv and Python dependencies (PyGame, etc.)
cd kinect
. bin/activate
pip install -r requirements.txt
```

2.2 Make Kinect devices available to OpenNI

A pair of conflicting kernel modules must be unloaded, because they capture the Kinect device first, thus making it unavailable to *OpenNI*: `gspca_kinect` and `gspca_main`. This output from `dmesg` shows the typical log trace of `gspca`:

```
# Extract of "dmesg" output:
[ 1477.856021] usb 2-6: new high-speed USB device number 2 using ehci_hcd
[ 1477.988629] hub 2-6:1.0: USB hub found
[ 1477.988708] hub 2-6:1.0: 3 ports detected
[ 1478.804089] usb 2-6.2: new full-speed USB device number 3 using ehci_hcd
[ 1480.340090] usb 2-6.1: new high-speed USB device number 4 using ehci_hcd
[ 1481.876089] usb 2-6.3: new high-speed USB device number 5 using ehci_hcd
[ 1482.055668] Linux video capture interface: v2.00
[ 1482.069476] gspca_main: v2.14.0 registered
[ 1482.072636] gspca_main: kinect-2.14.0 probing 045e:02ae
[ 1482.072691] usbcore: registered new interface driver kinect
```

Here is how to unload them:

```
sudo modprobe -r gspca_kinect
sudo modprobe -r gspca_main
```

The best is to blacklist them permanently:

```
# Symlink blacklist into modprobe.d/
sudo ln -s blacklist_gspca.conf /etc/modprobe.d

sudo depmod -a
sudo reboot
```

2.3 Install and test OpenNI

2.3.1 Install OpenNI binaries

→ This section is inspired from this page [igorbarbosa.com].

Make sure the Kinect is unplugged before starting the installation.

1. Download the packages for **OpenNI** and **NITE** from http://www.openni.org/Downloads/OpenNIModules.aspx.

   Note: For Ubuntu 12.04, you need the **unstable** packages:

   - OpenNI UNstable build for Ubuntu 12.04 - v1.5.4
   - PrimeSense NITE UNstable build for Ubuntu 12.04 - v1.5.2.21

   ---

2. In addition, get the avin2 hardware driver for Kinect. It is an improved version of the official PrimeSense Sensor driver.

You can now install the binaries:

```
# Install dependencies
sudo apt-get install g++ default-jdk freeglut3-dev

cd kinect/sources/openni

## OpenNI
# Untar openni-bin-dev-linux-x64-v1.5.4.0.tar.bz2
cd OpenNI-Bin-Dev-Linux-x64-v1.5.4.0/
sudo ./install.sh

## Avin2 Sensor Driver
# Unzip avin2-SensorKinect-v0.93-5.1.2.1-0-g15f1975.zip
cd avin2-SensorKinect-15f1975/Platform/Linux/CreateRedist
./RedistMaker
cd ../Redist/Sensor-Bin-Linux-x64-v5.1.2.1/
sudo ./install.sh

# Untar nite-bin-linux-x64-v1.5.2.21.tar.bz2
cd NITE-Bin-Dev-Linux-x64-v1.5.2.21/
sudo ./install.sh
```

Note: Uninstalling the modules above can be done through the following command:

```
sudo ./install.sh -u
```

2.3.2 Alternative: Install OpenNI from source code

This section is inspired from this page [keyboardmods.com]. It is here for information only: as of today, OpenNI doesn't build *as is*, the `Makefile` seems to be faulty regarding the `install` part.

```
# Install dependencies (C++, JDK, OpenGL headers, etc.)
sudo apt-get install g++ default-jdk freeglut3-dev doxygen

# Checkout OpenNI sources
cd build
git clone https://github.com/OpenNI/OpenNI.git

# Build and install OpenNI
cd OpenNI/Platform/Linux/Build
make && sudo make install
```

Note: (14) Open `Sample-User.xml` and replace the existing License line with the line below:

```
<!-- This is case-sensitive! -->
< License vendor="PrimeSense" key="0KOIk2JeIBYClPWVnMoRKn5cdY4="/>
```

15. Repeat step 14 for `Sample-Scene.xml` and `Sample-Tracking.xml`.

2.3.3 Test OpenNI

Remember to plug the Kinect first. Look for samples inside the `Samples/Bin/X64-Release/` subdirectories of the OpenNI and NITE packages, e.g.:

```
# Go to the OpenNI samples directory
cd kinect/sources/openni
cd OpenNI-Bin-Dev-Linux-x64-v1.5.4.0/Samples/Bin/x64-Release

# Run sample application
./SampleNiUserTracker
```

2.4 Install PyOpenNI, the Python bindings for OpenNI

In addition to OpenNI, PyOpenNI has a dependency on Boost and its Python bindings.

```
# Install CMake, Boost + Boost Python bindings (1.46)
sudo apt-get install cmake libboost-python-dev

# Build PyOpenNI
git clone https://github.com/jmendeth/PyOpenNI
mkdir PyOpenNI-build
cd PyOpenNI-build
cmake ../PyOpenNI
make

# Install library inside the virtual environment
cp lib/openni.so ../../lib/python2.7/site-packages/
```

2.4.1 Test PyOpenNI

Try to run the `hand-tracker.py` script. It should follow the user's hand after a *"Click"* gesture:

```
cd kinect
python hand-tracker.py
```

2.5 Install MINDs

See *Install MINDs under Linux*.

CHAPTER 3

Windows PyOpenNI setup instructions (32-bit)

This explains how to install the dependencies required to get MINDs working on Windows in **32-bit mode** (the preferred mode for **Windows XP**). It was tested on Windows Seven Pro SP1.

Note: For instructions on how to build the 64-bit version, please refer to the "*win64-setup*" page.

As many operations involved in the build process are lengthy ones (setting up Windows, installing system updates, installing Visual Studio Pro, building/installing the Boost library), the whole process typically takes a couple of days to complete.

3.1 Setup Python working environment

1. Install **Cygwin** – from http://www.cygwin.com

   Cygwin provides useful tools such as a SSH server, Subversion, etc.

   - openssh * http://tsengf.blogspot.fr/2011/06/installing-sshd-in-cygwin-on-windows-7.html * Open inbound port 22 in firewall
   - rsync vim mc unzip subversion

2. Open a `Cygwin` Bash terminal.

```
# Get dev files from git (Note: temporary repository URL)
git clone ssh://vialfre/Users/daverio/projects/git/kinect.git
```

3.2 Install and test OpenNI

Make sure the Kinect is unplugged before starting the installation.

1. Download the packages for **OpenNI** and **NITE** from http://www.openni.org/Downloads/OpenNIModules.aspx.

   Note: In order to remain in line with the "*Ubuntu Linux PyOpenNI setup instructions*" page, we choose the **unstable** packages:

   - OpenNI UNstable build for Windows x86 (32-bit) - v1.5.4.0 Development Edition
   - PrimeSense NITE UNstable build for Windows x82 (32-bit) - v1.5.2.21 Development Edition

2. In addition, get the **avin2 hardware driver for Kinect**:

   - SensorKinect093-Bin-Win32-v5.1.2.1.msi

   It is an improved version of the official PrimeSense Sensor driver, which doesn't work.

   Note: For the comple collection of drivers, see the avin2/SensorKinect download page on Github.

   ---

3. Install OpenNI, SensorKinect and Nite, in that order.

3.2.1 Test OpenNI

Remember to plug the Kinect first. Sample applications can be run from Windows' Start Menu, e.g.:

- *PrimeSense → NITE → Samples → Sample-PointViewer*

3.3 Build and Install PyOpenNI, the Python bindings for OpenNI

3.3.1 Requirements for the dev platform

We need to install the following packages:

1. **Python 2.7.3 (32-bit)** – from http://www.python.org

   → We accept the default location, `C:\Python27`.

2. **Git for Windows** – from Google Code.

   → Here, we also accept the default options, i.e. enable `git-shell` for all users.

3. **Visual Studio 2010 Pro** (install DVD plus online updates)

   → Make sure the "VC++ compiler" option is checked. Other languages (Visual Basic, C#, F#, etc.) can be left out.

4. [Optional] **MinGW** (*"Minimal GNU for Windows"*) – from http://www.mingw.org/wiki/Getting_Started

   MinGC provides a gcc compiler, which could probably be used as an alternative to Visual C++.

   → Download *"mingw-get-inst-20120426"* from https://sourceforge.net/projects/mingw/files/Installer/mingw-get-inst/

   → Accept default location for install (*C:\MinGW*). Only check the *"C++ compiler"* option. *"MSYS System"* is not needed, so we'll leave it out.

   Add `C:\MinGW\bin` to the `PATH` environment variables for all users.

5. **CMake** – from http://www.cmake.org

   → Accept the default location. Add to PATH for all users.

6. **Pkg-config and its dependencies, available from** http://ftp.gnome.org/pub/gnome/binaries/win32/ (both 32-bit and 64-bit versions can be used):

   It must be placed in a PATH directory, e.g. `C:\MinGW\bin`.

   - pkg-config 0.26-1

     → Copy `bin/pgk-config.exe` into the `bin` directory of MinGW.

   - glib 2.28.8-1

     → Copy `bin/libglib-2.0-0.dll` directory into the `bin` directory of MinGW.

   - gettext 0.18.1.1-2

     → Copy `bin/intl.dll` into the `bin` directory of MinGW.

7. The **Boost-Python** C++ library – from http://www.boost.org/

   We need to compile boost-python from source. The process is inspired from this page.

   → Download boost_1_51_0.zip. Unzip it and rename the directory as `C:\boost`.

   Inside the `C:\boost` directory, create file `user-config.jam` as follows:

```
import toolset : using ;
using python : 2.7 : "C:/Python27" : "C:/Python27/include" : "C:/Python27/libs" ;
```

   Open a `Cmd` command prompt window.

```
cd C:\boost

:: Build bjam
bootstrap.bat mingw

:: Build boost-python for Visual Studio
bjam toolset=msvc link=shared --with-python --user-config=user-config.jam

:: Alternative: build boost-python for MinGW
bjam toolset=gcc link=shared --with-python --user-config=user-config.jam
```

3.3.2 Get the source code for PyOpenNI

1. Open a `git-shell` window and **get a clone** of the the *PyOpenNI* repository:

```
# Clone repository
git clone https://github.com/jmendeth/PyOpenNI.git
```

   Close the `git-shell` window.

2. *Patch PyOpenNI*

   At the time of writing, compilation of PyOpenNNI under visual Studio produces a series of C2664 and C2373 errors. For the time being, we will patch the source code.

Note:

- For the *C2664s*, add explicit type casts as suggested by the error messages.
- For the *C2373s*, add the `XN_CALLBACK_TYPE` keyword in the relevant header files.

The author of PyOpenNI was notified of the modifications.

```
diff --git a/src/GestureGeneratorWrapper.cpp b/src/GestureGeneratorWrapper.cpp
index 8395fd2..d0f872f 100644
--- a/src/GestureGeneratorWrapper.cpp
+++ b/src/GestureGeneratorWrapper.cpp
@@ -64,7 +64,7 @@ XnCallbackHandle* GestureGenerator_RegisterGestureCallbacks_wrapped(xn::Ges
     cookie[0] = gesture_recognized;
     cookie[1] = gesture_progress;

-    check( self.RegisterGestureCallbacks(&GestureRecognized_callback, &GestureProgress_callb
+    check( self.RegisterGestureCallbacks((xn::GestureGenerator::GestureRecognized)&GestureRe
     return handle;
 }

diff --git a/src/GestureGeneratorWrapper.h b/src/GestureGeneratorWrapper.h
index 50d96ab..55a5048 100644
--- a/src/GestureGeneratorWrapper.h
+++ b/src/GestureGeneratorWrapper.h
@@ -44,7 +44,7 @@ BP::list GestureGenerator_GetAvailableGestures(xn::GestureGenerator& self);
```

```
19
20
21   //Internal callback implementations
22  -void GestureRecognized_callback(xn::GestureGenerator &generator, const XnChar *strGesture, c
23  -void GestureProgress_callback(xn::GestureGenerator &generator, const XnChar *strGesture, con
24  +void XN_CALLBACK_TYPE GestureRecognized_callback(xn::GestureGenerator &generator, const XnCh
25  +void XN_CALLBACK_TYPE GestureProgress_callback(xn::GestureGenerator &generator, const XnChar
26
27   #endif        /* GESTURE_GENERATOR_WRAPPER_H */
28  diff --git a/src/HandsGeneratorWrapper.cpp b/src/HandsGeneratorWrapper.cpp
29  index 3cb88e3..1ff5815 100644
30  --- a/src/HandsGeneratorWrapper.cpp
31  +++ b/src/HandsGeneratorWrapper.cpp
32  @@ -42,7 +42,7 @@ XnCallbackHandle HandsGenerator_RegisterHandCallbacks_wrapped(xn::HandsGene
33       cookie[1] = update;
34       cookie[2] = destroy;
35
36  -    check( self.RegisterHandCallbacks(&Create_callback, &Update_callback, &Destroy_callback,
37  +    check( self.RegisterHandCallbacks((xn::HandsGenerator::HandCreate)&Create_callback, (xn:
38
39       return handle;
40   }
41  @@ -71,4 +71,4 @@ void XN_CALLBACK_TYPE Destroy_callback(xn::HandsGenerator& src, XnUserID us
42
43       //Call the function
44       func(src, user, fTime);
45  -}
46  \ No newline at end of file
47  +}
48  diff --git a/src/HandsGeneratorWrapper.h b/src/HandsGeneratorWrapper.h
49  index d53b109..fbe5825 100644
50  --- a/src/HandsGeneratorWrapper.h
51  +++ b/src/HandsGeneratorWrapper.h
52  @@ -33,8 +33,8 @@ XnCallbackHandle HandsGenerator_RegisterHandCallbacks_wrapped(xn::HandsGene
53   void HandsGenerator_StartTracking_wrapped(xn::HandsGenerator& self, BP::list point);
54
55   /** Internal callback implementations **/
56  -void Create_callback(xn::HandsGenerator& src, XnUserID user, const XnPoint3D *pPosition, XnF
57  -void Update_callback(xn::HandsGenerator& src, XnUserID user, const XnPoint3D *pPosition, XnF
58  -void Destroy_callback(xn::HandsGenerator& src, XnUserID user, XnFloat fTime, void* cookie);
59  +void XN_CALLBACK_TYPE Create_callback(xn::HandsGenerator& src, XnUserID user, const XnPoint3
60  +void XN_CALLBACK_TYPE Update_callback(xn::HandsGenerator& src, XnUserID user, const XnPoint3
61  +void XN_CALLBACK_TYPE Destroy_callback(xn::HandsGenerator& src, XnUserID user, XnFloat fTime
62
63   #endif        /* HANDS_GENERATOR_WRAPPER_H */
64  diff --git a/src/PoseDetectionCapabilityWrapper.cpp b/src/PoseDetectionCapabilityWrapper.cpp
65  index 8a82c7e..e2703f9 100644
66  --- a/src/PoseDetectionCapabilityWrapper.cpp
67  +++ b/src/PoseDetectionCapabilityWrapper.cpp
68  @@ -40,7 +40,7 @@ XnCallbackHandle PoseDetectionCapability_RegisterPoseDetectedCallback(xn::P
69       BP::object* cookie = new BP::object;
70       *cookie = callback;
71
72  -    check( self.RegisterToPoseDetected(&PoseDetectionCapability_PoseDetection_cb, cookie, ha
73  +    check( self.RegisterToPoseDetected((xn::PoseDetectionCapability::PoseDetection)&PoseDete
74
75       return handle;
76   }
77  @@ -54,7 +54,7 @@ XnCallbackHandle PoseDetectionCapability_RegisterOutOfPoseCallback(xn::Pose
78       BP::object* cookie = new BP::object;
79       *cookie = callback;
80
81  -    check( self.RegisterToOutOfPose(&PoseDetectionCapability_PoseDetection_cb, cookie, handl
```

```
+    check( self.RegisterToOutOfPose((xn::PoseDetectionCapability::PoseDetection)&PoseDetecti

     return handle;
 }
diff --git a/src/PoseDetectionCapabilityWrapper.h b/src/PoseDetectionCapabilityWrapper.h
index 8ead590..68ce95c 100644
--- a/src/PoseDetectionCapabilityWrapper.h
+++ b/src/PoseDetectionCapabilityWrapper.h
@@ -38,6 +38,6 @@ void PoseDetectionCapability_UnregisterOutOfPoseCallback(xn::PoseDetectionC


 /** Internal callback implementations **/
-void PoseDetectionCapability_PoseDetection_cb(xn::PoseDetectionCapability& src, const XnChar
+void XN_CALLBACK_TYPE PoseDetectionCapability_PoseDetection_cb(xn::PoseDetectionCapability&

 #endif        /* POSE_DETECTION_CAPABILITY_WRAPPER_H */
diff --git a/src/SkeletonCapabilityWrapper.cpp b/src/SkeletonCapabilityWrapper.cpp
index ce2a22a..881839f 100644
--- a/src/SkeletonCapabilityWrapper.cpp
+++ b/src/SkeletonCapabilityWrapper.cpp
@@ -81,7 +81,7 @@ XnCallbackHandle SkeletonCapability_RegisterCalibrationStart(xn::SkeletonCa
     BP::object* cookie = new BP::object;
     *cookie = callback;

-    check( self.RegisterToCalibrationStart(&SkeletonCapability_CalibrationStart_cb, cookie,
+    check( self.RegisterToCalibrationStart((xn::SkeletonCapability::CalibrationStart)&Skelet

     return handle;
 }
@@ -95,7 +95,7 @@ XnCallbackHandle SkeletonCapability_RegisterCalibrationComplete(xn::Skeletc
     BP::object* cookie = new BP::object;
     *cookie = callback;

-    check( self.RegisterToCalibrationComplete(&SkeletonCapability_CalibrationStatus_cb, cook
+    check( self.RegisterToCalibrationComplete((xn::SkeletonCapability::CalibrationComplete)&

     return handle;
 }
@@ -109,7 +109,7 @@ XnCallbackHandle SkeletonCapability_RegisterCalibrationInProgress(xn::Ske
     BP::object* cookie = new BP::object;
     *cookie = callback;

-    check( self.RegisterToCalibrationInProgress(&SkeletonCapability_CalibrationStatus_cb, co
+    check( self.RegisterToCalibrationInProgress((xn::SkeletonCapability::CalibrationInProgre

     return handle;
 }
diff --git a/src/UserGeneratorWrapper.cpp b/src/UserGeneratorWrapper.cpp
index 8943b70..892e3e9 100644
--- a/src/UserGeneratorWrapper.cpp
+++ b/src/UserGeneratorWrapper.cpp
@@ -106,7 +106,7 @@ XnCallbackHandle UserGenerator_RegisterUserCallbacks_wrapped(xn::UserGene
     cookie[0] = newUser;
     cookie[1] = lostUser;

-    check( self.RegisterUserCallbacks(&NewUser_callback, &LostUser_callback, cookie, handle)
+    check( self.RegisterUserCallbacks((xn::UserGenerator::UserHandler)&NewUser_callback, (xn

     return handle;
 }
diff --git a/src/UserGeneratorWrapper.h b/src/UserGeneratorWrapper.h
index 82d7430..83c77f0 100644
--- a/src/UserGeneratorWrapper.h
```

```
+++ b/src/UserGeneratorWrapper.h
@@ -41,7 +41,7 @@ XnCallbackHandle UserGenerator_RegisterUserCallbacks_wrapped(xn::UserGenera
 void UserGenerator_UnregisterUserCallbacks_wrapped(xn::UserGenerator& self, XnCallbackHandle

 /** Internal callback implementations **/
-void NewUser_callback(xn::UserGenerator& src, XnUserID user, void* cookie);
-void LostUser_callback(xn::UserGenerator& src, XnUserID user, void* cookie);
+void XN_CALLBACK_TYPE NewUser_callback(xn::UserGenerator& src, XnUserID user, void* cookie);
+void XN_CALLBACK_TYPE LostUser_callback(xn::UserGenerator& src, XnUserID user, void* cookie)

 #endif        /* USER_GENERATOR_WRAPPER_H */
diff --git a/src/wrapper.cpp b/src/wrapper.cpp
index bbb7410..cec60b4 100644
--- a/src/wrapper.cpp
+++ b/src/wrapper.cpp
@@ -233,4 +233,4 @@ BOOST_PYTHON_MODULE(openni) {
     scope().attr("CAPABILITY_LOW_LIGHT_COMPENSATION") = XN_CAPABILITY_LOW_LIGHT_COMPENSATION
     scope().attr("CAPABILITY_ANTI_FLICKER") = XN_CAPABILITY_ANTI_FLICKER;
     scope().attr("CAPABILITY_HAND_TOUCHING_FOV_EDGE") = XN_CAPABILITY_HAND_TOUCHING_FOV_EDGE
-    scope().attr("CAPABILITY_ANTI_FILCKER") = XN_CAPABILITY_ANTI_FILCKER;
+    //scope().attr("CAPABILITY_ANTI_FILCKER") = XN_CAPABILITY_ANTI_FILCKER;
```

(pending...)

3. Open a `Cmd` command prompt window.

```
:: Prepare a build directory
md PyOpenNI-build
cd PyOpenNI-build
```

3.3.3 Build PyOpenNi using Visual Studio

1. Open a `Cmd` command prompt window.

```
:: Run CMake. Ignore messages about compiler flags
cmake -G "Visual Studio 10" ..\PyOpenNI
```

2. From the Explorer, double-click on `PyOpenNI.sln`. Visual Studio will then be launched.
3. In the "Solution Configuration" dropbox, select `Release`
4. Generate the solution by pressing the **F7** key

3.3.4 Alternative : build PyOpenNI using MinGW

1. Open a `Cmd` command prompt window.

```
:: Run CMake. Ignore messages about compiler flags
cmake -G "MinGW Makefiles" ..\PyOpenNI

:: Build PyOpenNI using MinGW
mingw32-make
```

→ For the time being, compilation fails because OpenNI headers require a Microsoft compiler on the `win32` platform.

3.3.5 Install PyOpenNI

Locate the generated `openni.dll` file, and copy it into Python's `site-packages` subdirectory with the **pyd** extension:

```
:: copy Boost-python
cp C:\boost\stage\lib\boost_python-vc100-mt-1_51.dll C:\Python27\Lib\site-packages

:: copy and rename openni.dll
cp bin\Release\openni.dll C:\Python27\Lib\site-packages\openny.pyd
```

3.3.6 Test PyOpenNI

1. Open Python shell (IDLE) and type:

   ```
   import openni
   ```

   No error message should be displayed.

2. In the Explorer, locate `kinect\hand-tracker.py` and double-click on it.

   → The programme should follow the user's hand after a *"Click"* gesture.

3.4 Install MINDs

See *Install MINDs under Windows (32-bit)*.

CHAPTER 4

MINDs installation

4.1 Install MINDs under Linux

1. **Prerequisites:**
   - For **OpenNI**-related libraries, see *Ubuntu Linux PyOpenNI setup instructions*.
   - Ubuntu Linux provides **PyGame** in its system repositories.

     ```
     sudo apt-get install python-pygame

     # Symlink PyGame into the virtual environment
     ln -s /usr/lib/python2.7/dist-packages/pygame* lib/python2.7/site-packages
     ```

     Note: Ubuntu Linux uses nonstandard locations for its Python modules. For more details, see *Install and test libfreenect Python wrappers*.

   - Ubuntu Linux also provides **Fluidsynth** in its system repositories.

     ```
     # Install Fluidsynth
     sudo apt-get install fluidsynth
     ```

2. **Checkout MINDs** from SVN

   ```
   # Activate virtual environment
   cd kinect
   . bin/activate

   # Checkout from SVN (here, the "ld-kinect" branch)
   svn co https://svn.cri.ensmp.fr/svn/minwii/branches/ld-kinect/src minds

   # Checkout full trunk in another directory. This will give access to the songs
   svn co https://svn.cri.ensmp.fr/svn/minwii/trunk sources/minds

   # Develop
   cd minds
   python setup.py develop

   # Shortcut link to the songs directory
   ln -s ../sources/minds/chansons
   ```

3. **Run MINDs**

```
# Run MINDs with the OpenNI Kinect support
runminds -t kinect2
```

4.2 Install MINDs under Windows (32-bit)

Note: This sections describes the manual installation of MINDs. A binary installer is available from the MINDs website.

1. **Prerequisites:**
   - For **OpenNI**-related libraries, see *Windows PyOpenNI setup instructions (32-bit)*.
   - Python **Setuptools** – from http://pypi.python.org/pypi/setuptools

     Download and install setuptools-0.6c11.win32-py2.7.exe
   - **PyGame** – install it from http://www.pygame.org/download.shtml

     Download and install pygame-1.9.1.win32-py2.7.msi

     Note: If default option *"Python from Registry"* fails, select the *"Python from another location"* option instead.
   - **FluidSynth** – from the Qsynth website

     (Qsynth includes a patched version of FluidSynth which can be used under Windows.)

     Download and install qsynth-0.3.6-setup.exe.

     In the install directory, locate the three DLLs:
     - `libfluidsynth.dll`
     - `libgthread-2.0-0.dll`
     - `libsndfile-1.dll`

     and copy them to a PATH directory, e.g. `C:\MinGW\bin` or `C:\Windows\System32`.

     (This is mandated by the ctypes.utils.find_library function used by the `pyFluidSynth` module).
2. **Checkout MINDs** from SVN

   Open a `Cygwin` Bash terminal.

   ```
   cd kinect

   # Checkout from SVN (here, the "ld-kinect" branch)
   svn co https://svn.cri.ensmp.fr/svn/minwii/branches/ld-kinect/src minds

   # Checkout full trunk in another directory. This will give access to the songs
   svn co https://svn.cri.ensmp.fr/svn/minwii/trunk sources/minds
   ```

   Close the terminal.

   From the Explorer, copy the `sources\minds\chansons` directory into the `minds` directory. This will make the songs easier to choose.
3. **Install the MINDs python modules (minwii, ...)**

   Open a `Cmd` command prompt window.

   ```
   :: MINDs source code
   cd kinect\minds

   :: Install MINDs and its dependencies (``PyFluidSynth``, etc.)
   C:\Python27\Python setup.py develop
   ```

4. **Run MINDs**

```
# Run MINDs with the OpenNI Kinect support
runminds -t kinect
```

CHAPTER 5

Low-level Kinect support under Ubuntu Linux

Note: This is an optional part, not required to run MINDs. For standard setup, please refer to *Ubuntu Linux PyOpenNI setup instructions*.

An lower-level alternative to the use of *OpenNI/NITE* is provided by the **LibFreenect** libraries. They have specific dependencies such as *LibUsb* and *OpenCv*.

5.1 Install and test the libfreenect library

Starting with version 11.10 (*"Ocelot"*) or Ubuntu Linux, *LibFreenect* is part of the official repositories, so installation shouldn't pose any particular problem.

```
sudo apt-get install freenect
```

At this stage, you should be able to run the Freenect demos. But prior to this, a few conflicting kernel modules must be unloaded, because the capture the Kinect device, thus making it unavailable to libfreenect. See *Make Kinect devices available to OpenNI* for more details.

```
# Unload gspca_* modules
sudo modprobe -r gspca_kinect
sudo modprobe -r gspca_main

# Run demo
freenect-glview
```

5.1.1 Troubleshooting

Should the software demos fail to connect with the Kinect, the following should also be checked:

1. Check that the Kinect's devices are seen by the PC by typing `lsusb`. The Kinect should appear as **three distinct Microsoft devices**, e.g.:

   ```
   # Extract of "lsusb" output:
   Bus 002 Device 003: ID 045e:02b0 Microsoft Corp. Xbox NUI Motor
   Bus 002 Device 004: ID 045e:02ad Microsoft Corp. Xbox NUI Audio
   Bus 002 Device 005: ID 045e:02ae Microsoft Corp. Xbox NUI Camera
   ```

2. The user must belong to the `plugdev` group. If it's not the case, please run the following command:

```
# Add $USER to the 'plugdev' group
sudo adduser $USER plugdev
```

then log out and in again, and disconnect + reconnect the Kinect.

5.2 Install and test libfreenect Python wrappers

Note: The virtual environment should already have been set up. If this is not the case, please refer to *Setup Python virtual environment*.

The next step is to install the Python bindings for `Freenect` and `OpenCV`. They cannot be installed into the virtual environment using `pip install`, so the easiest is to install them at system level, and then manually symlink them into the virtual environment.

```
sudo apt-get install python-freenect python-opencv

# Freenect
ln -s /usr/lib/pyshared/python2.7/freenect.so lib/python2.7/site-packages

# OpenCV (cv, cv2)
ln -s /usr/share/pyshared/cv.py          lib/python2.7/site-packages
ln -s /usr/lib/pyshared/python2.7/cv2.so lib/python2.7/site-packages

# Xlib (no longer used)
#ln -s /usr/share/pyshared/Xlib lib/python2.7/site-packages
```

Note: Ubuntu Linux uses nonstandard locations for its Python modules, instead of the more standard `/usr/lib/python2.7/site-packages`, namely:

- `/usr/lib/pyshared/python2.7`
- `/usr/lib/python2.7/dist-packages`
- `/usr/share/pyshared`
- ...

The `dpkg -L` command returns the location used for a given module:

```
# Output of "dpkg -L python-opencv":
/.
/usr
/usr/share
[...]
/usr/share/pyshared
/usr/share/pyshared/cv.py
[...]
/usr/lib/pyshared/python2.7
/usr/lib/pyshared/python2.7/cv2.so
```

At this stage, the `hand_tracking.py` script can be tested. This rather crude script works well in certain conditions, typically at short distances from the Kinect device.

```
cd kinect
. bin/activate

./hand_tracking.py
```

(Source: http://code.activestate.com/recipes/578104-openkinect-mouse-control-using-python/)