Music Paint Machine: an Interactive Music System for Educational and Artistic Purpose

Riccardo Donadi

12 December 2011

Contents

Abstract

This thesis investigates the porting process of the Music Paint Machine into the Kinect Music Paint Machine (KMPM) and examines the positive and negative aspects that were found using the brand-new device Microsoft Kinect in the artistic and educational field of interactive systems. A brief introduction of the Music Paint Machine, its component and its usage is initially outlined. Afterwards a further introduction about the Kinect is also necessary for understanding the components, and how they work. After a short explanation about drivers installation and driver utilized, the discussion then focuses on the analysis of the MAX/MSP patch created for realizing the Kinect Music Paint Machine. In the conclusion, pro and cons are outlined together with the future applications that the Kinect Music Paint Machine can have.

Chapter 1

Introduction

The purpose of this work, carried out in the Institute for Psychoacoustics and Electronic Music in Ghent was to find a way to use the new brand device, called Microsoft Kinect, and make it work with the Music Paint Machine, an interactive music system, created by Luc Nijs, that provides visual feedback by monitoring a musician's performance (see details in chapter 2).

The theoretical background of the conceptual design of the Music Paint Machine is the Embodied Music Cognition paradigm. This research paradigm acknowledges the embodied nature of the musical mind (Leman, 2007) [1]. What happens in the mind depends on properties of the body and therefore body and body movement have an impact on meaning formation.

Embodied music cognition is a direction within systematic musicology interested in studying the role of the human body in relation to all musical activities [2]. It considers the human body as the natural mediator between mind (focused on musical intentions, meanings, significations) and physical environment (containing musical sound and other types of energy that affords human action). Given the impact of body movement on musical meaning formation and signification, the musical mind is said to be embodied. Embodiment assumes that what happens in the mind is depending on properties of the body, such as kinaesthetic properties. Embodied music cognition tends to see music perception as based on action. For example, many people move when they listen to music. Through movement, it is assumed that people give meaning to music. This type of meaning-formation is corporeal, rather than cerebral because it is understood through the body. This is different from a disembodied approach to music cognition, which sees

musical meaning as being based on a perception-based analysis of musical structure. The embodied grounding of music perception is based on a multimodal encoding of auditory information and on principles that ensure the coupling of perception and action.

During the last decade, research in embodied music cognition has been strongly motivated by a demand for new tools in view of the interactive possibilities offered by digital media technology.

With the advent of powerful computing tools, and in particular real-time interactive music systems, gradually more attention has been devoted to the role of gesture in music. This musical gestures research has been rather influential in that it puts more emphasis on sensorimotor feedback and integration, as well as on the coupling of perception and action. With new sensor technology, gesture-based research has meanwhile become a vast domain of music research, with consequences for the methodological and epistemological foundations of music cognition research.

Computers and interactive systems have become very popular and widespread tools especially in supporting teaching and learning process: Internet, for example, is the prime model of an interactive system where users can be fully immersed in their experiences by viewing material, commenting on it and then actively contributing to it. In contrast with this large scale diffusion, interactive systems in music teaching and in particular in music instrumental learning are not having a great success. The phenomenon is not only due to an inadequate technology level but also to an underlying skepticism from the traditional classical music world and especially from the instrumental music teachers which learning methods have undergone very few modifications. In traditional music education, self-monitoring is primarily acquired through a master-apprentice model. In this model, a "master" or music teacher helps the "apprentice" or student to develop the necessary monitoring skills by giving verbal and gestural feedback on different aspects of their playing (e.g. posture, technique, interpretation) (Ericsson, 1997; Hoppe, Sadakata, & Desain, 2006; Lehmann, 1997) [3] [4] [5]. By receiving repeated feedback from the teacher, students develop awareness about and evaluate one's own performance quality [6].

In this contest, Music Paint Machine rise up like a support tool used by the teacher for improving the self-monitoring in the students during the instrumental lessons. Self-monitoring is an important aspect of music edu-

cation that implies an awareness and evaluation of the quality of one's own performance (Altenmuller & Gruhn, 2002; Goolsby, 1995; Palmer & Drake, 1997; Woody, 2001) [7] [8] [9] [10].

Chapter 2

Music Paint Machine

The Music Paint Machine is an interactive system that provides visual feedback by monitoring a musician's performance (Figure 2.1). The system allows a musician to draw a painting on a computer screen by playing a musical instrument and moving on a colour mat (see for more details section 2.1), [6].

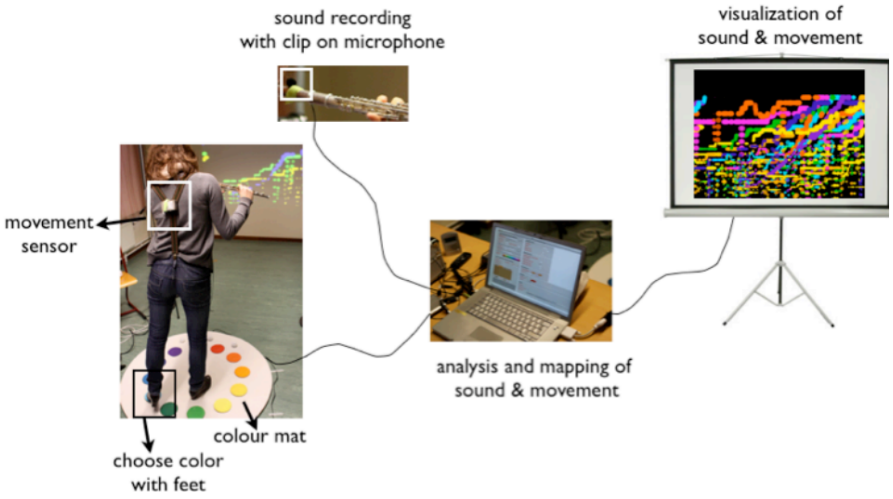

Figure 2.1: System Overview

The aim of the Music Paint Machine with an embedded monitoring and feedback system, is complementing traditional approaches with technology-based application that:

1. makes it possible to monitor different aspects of playing a musical instrument (e.g. breathing, sound characteristics, posture, bow speed/pressure) by using state of the art sensing technologies;

2. stimulates intrinsic feedback mechanisms by providing an immersive experience that engages its users in an interactive feedback loop and stimulates intrinsic motivation;

3. provides off line feedback (paintings, data logs) as pedagogical documentation that can be used for further reflection.

All these goals will be achieved through the method summarized in the picture below (Figure 2.2) and a better explanation of it is given in [6].

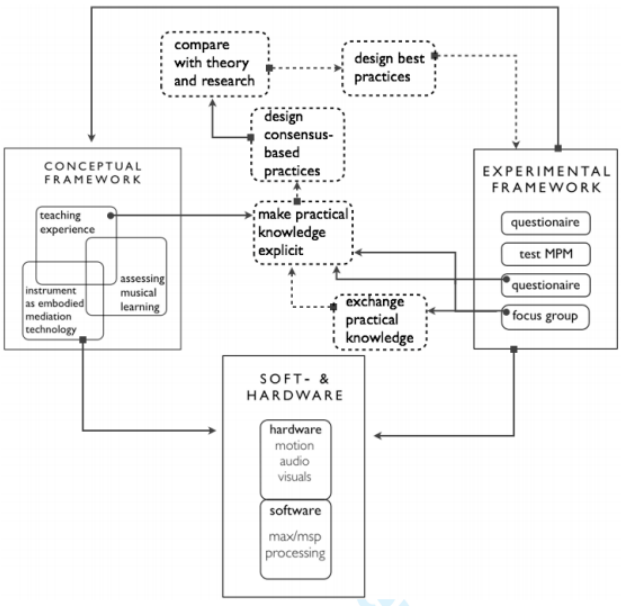

Figure 2.2: Overview of the method

2.1 System Overview

The system consists of a hardware and software part.

2.1.1 Hardware

Color Dance Mat

The multicoloured dance mat (Figure 2.3) consists of 12 pressure sensors (contact switches), 4 extra switches (situated on top of the mat) and a USB interface which are all integrated in the Medium-Density fibreboard (MDF) floor plate [11]. A cover presenting a twelve-colour wheel hides the hardware and makes choosing colours in the game clear. Stepping on a colour activates a pressure sensor underneath. The USB interface is a hacked numeric keypad. The twelve contact switches replace the button switches of the original keyboard matrix.

Figure 2.3: Mat

The circular pressure points have the size of compact disc (Ø 12 cm) and they are connected on an arduino board. The inner circumference, the one on which the circles are aligned, has a diameter of 12 cm, while the diameter of the MDF board is around 120 cm.

Motion Sensor

The movements of the user are tracked using inertial measurement sensors (Figure 2.4):

- an LY530AL (single-axis gyro);
- an LPR530AL (dual-axis gyro);
- an ADXL345 (triple-axis accelerometer);
- an HMC5843 (triple-axis magnetometer).

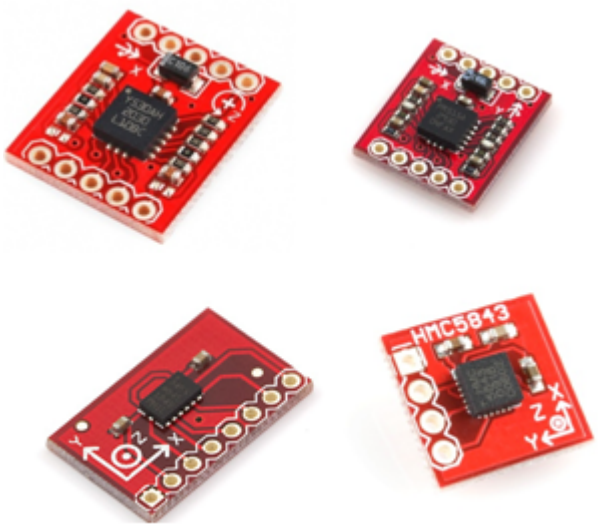

Figure 2.4: Sensors

The output of the sensors, after being processed and transformed in an orientation vector, is then sent to the computer trough a wireless Bluetooth connection.

All these sensors are packed inside two little boxes attached to the torso, with a flexible strap (Figure 2.5) that doesn't hinder breathing. A compass (Figure 2.6) is necessary for a correct calibration.

Figure 2.5: Strap

Figure 2.6: Compass

Audio Devices

The audio devices used, are:

- A Digital Microphone System;
- A Slim-line USB-Midi Controller.

The Digital Microphone System (AKG-DSM 700 V2) consists in (Figure 2.7):

1. A Digital true diversity receiver (DSR 700);
2. A Digital body-back transmitter (DPT 700);
3. A Digital handheld transmitter (DHT 700).

Figure 2.7: AKG-DSM 700v2

The sound is recorded by a body-pack transmitter whose microphone is placed on the instrument. The audio signal is then sent to the nearby receiver which recovers the audio and re-sends it to the laptop.

The Korg USB-Midi controller (Figure 2.8) is used by the musician during the performance for starting or stopping the player, clearing the screen from the painting and recording the melody.

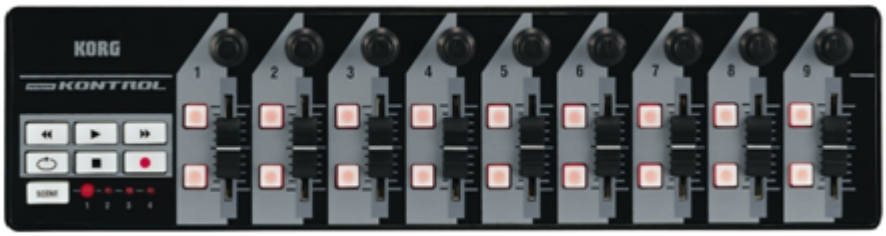

Figure 2.8: Korg USB-Midi controller

2.1.2 Software

Max/Msp/Jitter

Max is the environment you use to create visual programs, called patches, plus a set of building blocks (called objects) used in those programs. MSP is a set of Max objects for audio and signal processing. Jitter is a set of Max objects for video, graphics, and matrix data processing [12].

Processing

Processing is an open source programming language and integrated development environment (IDE) built for the electronic arts and visual design communities with the purpose of teaching the basics of computer programming in a visual context, and to serve as the foundation for electronic sketchbooks. The language builds on the Java programming language, but uses a simplified syntax and graphics programming model [13].

2.1.3 Mapping Body Movement

The canvas that is either portrayed on a display or projected on some surface reflects the player's movement and choices of colour. The pressure sensors, embedded in the dance mat, give the player the opportunity to choose the drawing colour. A set of 12 basic colours is available and this initial colour's saturation can be dynamically controlled by moving the torso either forward (more saturated) or backwards (less saturated). The movement of the torso also determines the X-position of the paintbrush on the screen. The Wii remote, strapped to the chest, is used to capture leaning forward and backward (pitch) and turn left or right (roll) [11].

Mapping Musical Features

All other drawing commands are determined by musical features. The vertical position of the paintbrush on the canvas is determined by pitch. A sustained note produces a horizontal line, while a melody produces a curved line that follows the melodic contour. The thickness of the paintbrush is determined by the loudness of what is played. The louder a user plays, the thicker the brushstroke becomes. Loudness and pitch are currently tracked by an object of the CPS programming environment (Figure 2.9) [6].

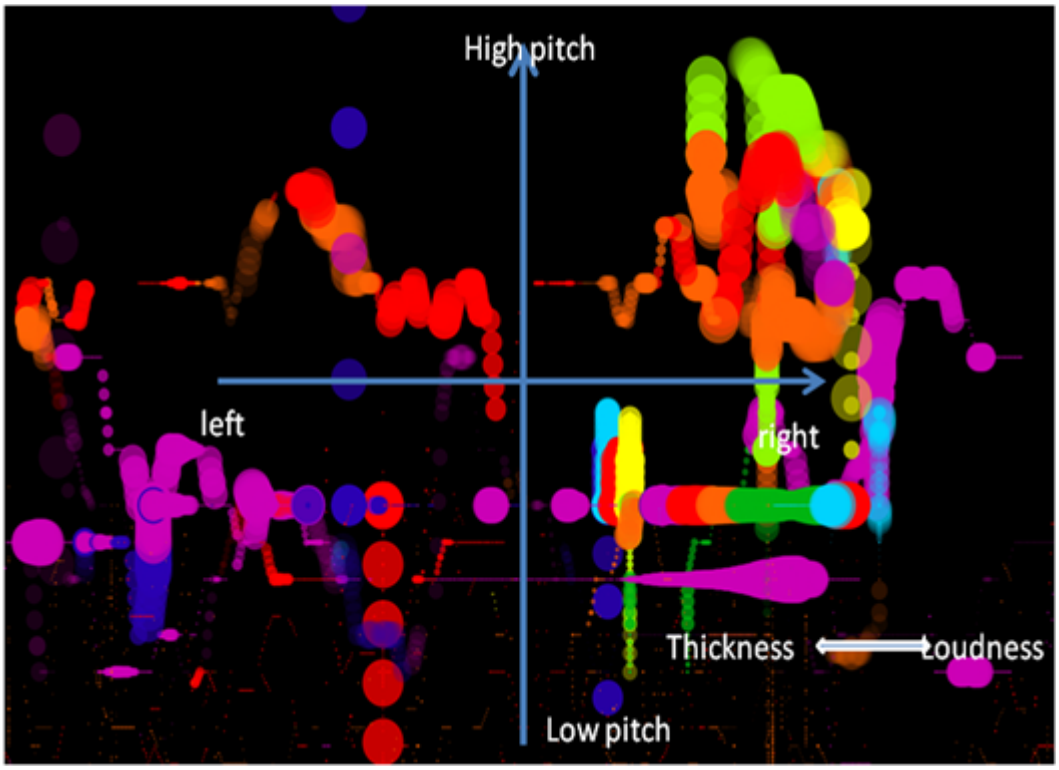

Figure 2.9: Mapping

Chapter 3

Microsoft Kinect

November 2010 marks not only the Kinect's historic anticipated release as a new addition to Microsoft's Xbox 360 product line (Figure 3.1) but it is also like a breath of fresh air for the developer's community. The expectation are very high and everybody is re-thinking a new way of designing human interactive application and supporting tools for human education.

Figure 3.1: Microsoft Kinect

This little extract of a review, taken in a high-tech specialized web-site, gives (with a bit of exaggeration) the perfect idea of the Kinect's community feelings:

"Microsoft Kinect is poised to shake up the video game console experience. Announced and demonstrated as Project Natal in June 2009, Kinect seems almost magical the way it can "see" every movement of your body and reproduce it within the video game you're playing [14]. Plus, it recognizes your face and voice so it can pick you out in the room and know who you are, even if you're playing with a group of friends. Kinect is a highly in-

novative combination of cameras, microphones and software that turns your body into the video game controller. The name Kinect is inspired by the words "kinetic", which means to be in motion, and "connect", which means it "connects you to the friends and entertainment you love" [source: Rule]. It's not just the games that get you moving, either: Kinect turns your Xbox 360 into a voice-activated console with video capturing and facial recognition, applicable for everything from selecting a TV show to creating digital artwork".

3.1 The Kinect Sensor

3.1.1 General Overview

Kinect is based on software technology developed internally by Rare, a subsidiary of Microsoft Game Studios owned by Microsoft, and on range camera technology by Israeli developer PrimeSense, which interprets 3D scene information from a continuously-projected infrared structured light. This 3D scanner system called Light Coding employs a variant of image-based 3D reconstruction [15].
The Kinect sensor is a horizontal bar connected to a small base with a motorized pivot and is designed to be positioned lengthwise above or below the video display. The device features an RGB camera, depth sensor and multi-array microphone running proprietary software, which provide full-body 3D motion capture, facial recognition and voice recognition capabilities (Figure 3.2).

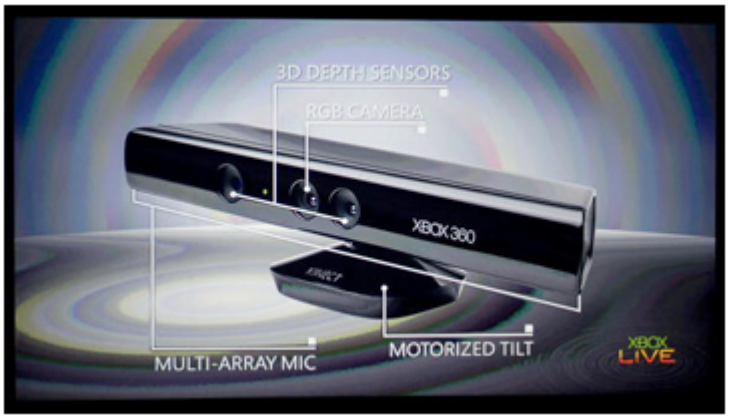

Figure 3.2: Kinect Overview

The Kinect sensor's microphone array enables the Xbox 360 to conduct acoustic source localization and ambient noise suppression, allowing for things such as headset-free party chat over Xbox Live. The depth sensor consists of an infrared laser projector combined with a monochrome CMOS sensor, which captures video data in 3D under any ambient light conditions. The sensing range of the depth sensor is adjustable, and the Kinect software is capable of automatically calibrating the sensor based on gameplay and the player's physical environment, accommodating for the presence of fur-

niture or other obstacles. Through reverse engineering efforts, it has been determined that the Kinect sensor outputs video at a frame rate of 30 Hz. The RGB video stream uses 8-bit VGA resolution (640 × 480 pixels) with a Bayer color filter, while the monochrome depth sensing video stream is in VGA resolution (640 × 480 pixels) with 11-bit depth, which provides 2,048 levels of sensitivity. The Kinect sensor has a practical ranging limit of 1.2–3.5 m (3.9–11 ft) distance when used with the Xbox software. The area required to play Kinect is roughly $6m^2$, although the sensor can maintain tracking through an extended range of approximately 0.7–6 m (2.3–20 ft). The sensor has an angular field of view of 57° horizontally and 43° vertically, while the motorized pivot is capable of tilting the sensor up to 27° either up or down. The horizontal field of the Kinect sensor at the minimum viewing distance of 0.8 m (2.6 ft) is therefore 87 cm (34 in), and the vertical field is 63 cm (25 in), resulting in a resolution of just over 1.3 mm (0.051 in) per pixel. The microphone array features four microphone capsules and operates with each channel processing 16-bit audio at a sampling rate of 16 kHz.

3.1.2 Chip

The key component of the Kinect, built according to the PrimeSensor's Reference Design, is the PrimeSense's PS1080 system on a chip (SoC). The PS1080 SoC houses extremely parallel computational logic, which receives a Light Coding infrared pattern as an input, and produces a VGA-size depth image of the scene. The PS1080 SoC is a multi-sense system (Figure 3.3), providing a synchronized depth image, color image and audio stream [16]. The PS1080 includes a USB 2.0 PHY, whose USB 2.0 interface is used to pass all data to the host.

The PS1080 makes no assumptions about the host device CPU – all depth acquisition algorithms run on the PS1080, with only a minimal USB communication layer running on the host. This feature provides depth acquisition capabilities even to computationally limited host devices.

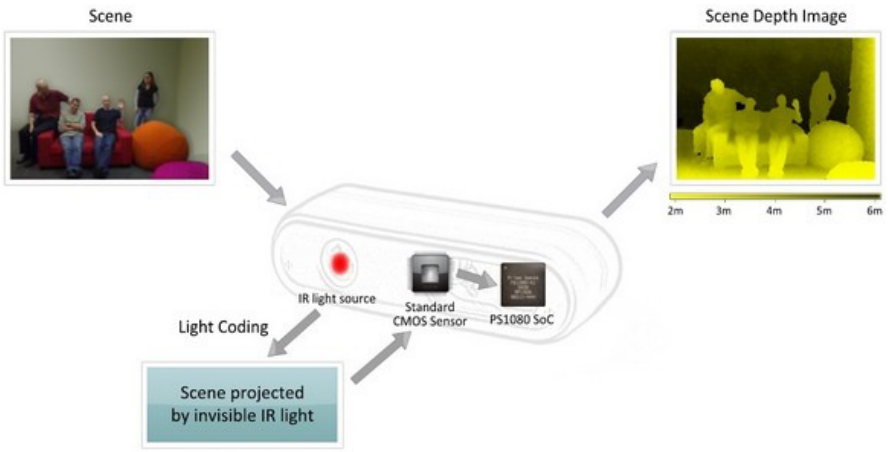

Figure 3.3: PS1080 SoC

3.1.3 Reference Design

The PrimeSensor's Reference Design is an end-to-end solution that enables a computer to perceive the world in three-dimensions and to translate these perceptions into a synchronized depth image, in the same way that humans do [17]. The solution includes a sensor component, which observes the scene (users and their surroundings), and a perception component, or brain, which comprehends the user interaction within these surroundings. The Reference Design is the sensor component of the solution.

Method of operation

PrimeSense's technology (Figure 3.4) for acquiring the depth image is based on Light Coding. Light Coding works by coding the scene volume with near-IR light. The IR Light Coding is invisible to the human eye [17]. The solution then utilizes a standard off-the-shelf CMOS image sensor to read the coded light back from the scene. PrimeSense's SoC chip is connected to the CMOS image sensor, and executes a sophisticated parallel computational algorithm to decipher the received light coding and produce a depth image of the scene. The solution is immune to ambient light. The PS1080 controls the IR light source in order to project the scene with an IR Light Coding image. The IR projector is a Class 1 safe light source, and is compliant with the IEC 60825-1 standard. A standard CMOS image sensor, receives the projected IR light and transfers the IR Light Coding image to the PS1080. The PS1080 processes the IR image and produces an accurate per-frame depth image of the scene. The PrimeSensor includes two optional sensory input capabilities: color (RGB) image and audio (the PrimeSensor has two microphones and an interface to four external digital audio sources). To produce more accurate sensory information, the PrimeSensor Reference Design performs a process called Registration. The Registration process's resulting images are pixel-aligned, which means that every pixel in the color image is aligned to a pixel in the depth image. All sensory information (depth image, color image and audio) is transferred to the host via a USB2.0 interface, with complete timing alignment (Table 1).

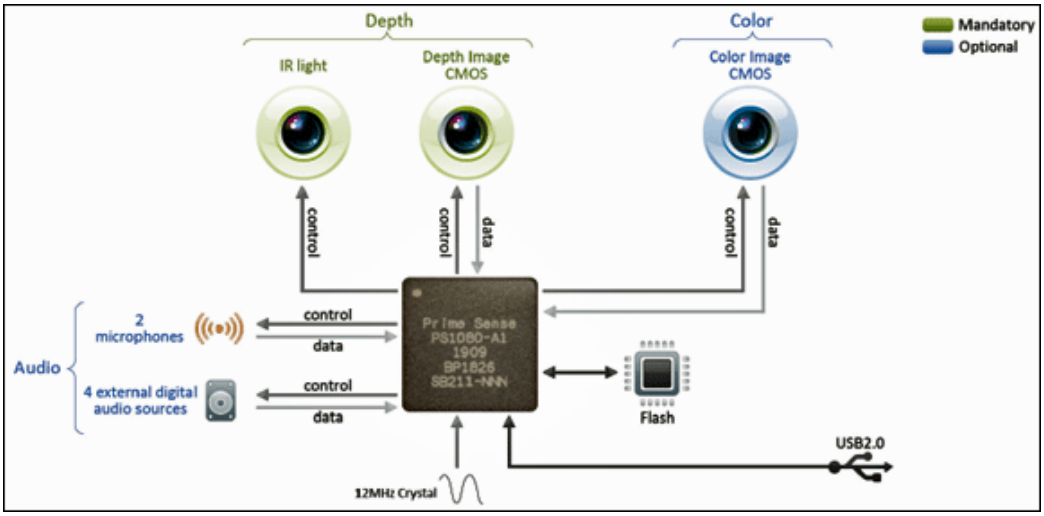

Figure 3.4: Block Diagram

Table 1 - Specification

Property	**Spec**
Field of View(Horizontal, Vertical, Diagonal)	58° H, 45° V, 70° D
Depth image size	VGA(640x480)
Spatial x/y resolution(@ 2m distance from sensor)	3mm
Depth z resolution (@ 2m distance from sensor)	1cm
Maximum image throughput (frame rate)	60fps
Operation range	0.8m - 3.5m
Color image size	UXGA (1600x1200)
Audio: built-in microphones	Two mics
Audio: digital inputs	Four inputs
Data interface	USB 2.0
Power supply	USB 2.0
Power consumption	2.25W
Dimensions (Width x Height x Depth)	14cm x 3.5cm x 5cm
Operation environment (every lighting condition)	Indoor
Operating temperature	0°C - 40°C

Chapter 4

Kinect Music Paint Machine Development

As seen before, the old music paint machine (MPM v2.0) requires the use of many devices, which take a lot of useful time for setting up the environment. The installation can take at least half an hour for a trained person and an indefinitely amount of time for a first approaching user. Considering that everything is set up, calibration can be annoying because you need a compass and usually people don't have it in their pockets. The coloured wood mat is also heavy, difficult to move around a room and it's the only one configuration possible.
These few reasons and the recent release of the new-brand technology Microsoft Kinect were an inspiration to try a different and more user-friendly approach of research.
This work at Ipem consisted in a feasibility study and a successive software implementation of an intermediate Music Paint Machine which uses only the Kinect.

Most of the notions and the solutions found in this explorative task are already used for developing the Music Paint Machine v3.0.
The table below (Table 2) illustrates exactly the position of my research respect the previous and future development of the Music Paint Machine project.

Table 2 - Project Progress

MPM v2.0	**Kinect MPM**	**MPM v3.0**
9DOF inertial sensor	**Kinect**	Kinect
coloured mat with buttons	**Software virtual mat**	coloured mat with pressure sensors
12 colours	**8 colours/ different way choosing colours**	choose colour number
fixed mapping	**dynamic mapping**	dynamic mapping
one screen	one screen	subdivided screen
explorative mode	explorative mode	different levels

4.1 Description of the Kinect Music Paint Machine

The interactive system, as told in the previous chapters, requires a lot of external devices (Figure 4.1).

Figure 4.1: System Devices

All these stuff are now completely replaced by:

1. Microsoft Kinect;
2. A large plastic mat.

The large plastic mat can be spread on the floor or in the ground and has 8 coloured circles placed around a circumference. The dimensions are very similar to the heavy one but the thickness is about 3 millimetre (Figure 4.2).

Figure 4.2: Plastic Mat

4.2 Software Description

The execution of the Kinect Music Paint Machine requires a previous installation of essential software packages (available for Linux, Mac OS X, and Microsoft Windows) that grant the correct functioning of the interactive music system. The machine used is a common laptop running Microsoft Windows 7.
The software packages required are:

1. PrimeSense sensor module (Kinect driver);
2. OpenNi Framework;
3. PrimeSense Nite Middleware;
4. OSCeleton;
5. Max/MSP/Jitter;
6. Processing;
7. Microsoft Kinect Driver (facultative/only Microsoft Windows systems).

4.2.1 PrimeSense sensor module

This software package is necessary for working with the Microsoft Kinect. OpenNI driver supports only the 3D depth sensor, X-tion Pro, which works similarly to Kinect because the two systems share the same PrimeSense technology. For resolving this compatibility problem, it was developed this Kinect mod version that works with the unstable OpenNI release.

4.2.2 OpenNi Framework

OpenNI (Open Natural Interaction) is a multi-language, cross-platform framework that defines APIs for writing applications utilizing Natural Interaction [18]. OpenNI APIs are composed of a set of interfaces for writing NI applications. The main purpose of OpenNI is to form a standard API that enables communication with both:

- Vision and audio sensors (the devices that "see" and "hear" the figures and their surroundings);

- Vision and audio perception middleware (the software components that analyse the audio and visual data that is recorded from the scene, and comprehend it). For example, software that receives visual data, such as an image, returns the location of the palm of a hand detected within the image.

OpenNI supplies a set of APIs to be implemented by the sensor devices, and a set of APIs to be implemented by the middleware components. The benefits of using the OpenNI's API give the possibility of:

1. Porting the applications in any middleware modules ("write once, deploy everywhere");
2. Writing algorithms on top of raw data formats regardless of which sensor device has produced them;
3. Building sensor that power any OpenNI compliant application.
4. Tracking real life (3D) scenes by utilizing data types that are calculated from the input of a sensor (NI application developers).

The OpenNI is an open source API that is publicly available at www.OpenNI.org.

Abstract Layered View

The figure below (Figure 4.3) displays a three-layered view of the OpenNI Concept with each layer representing an integral element:

- Top: Represents the software that implements natural interaction applications on top of OpenNI;
- Middle: Represents OpenNI, providing communication interfaces that interact with both the sensors and the middleware components, which analyze the data from the sensor;
- Bottom: Shows the hardware devices that capture the visual and audio elements of the scene.

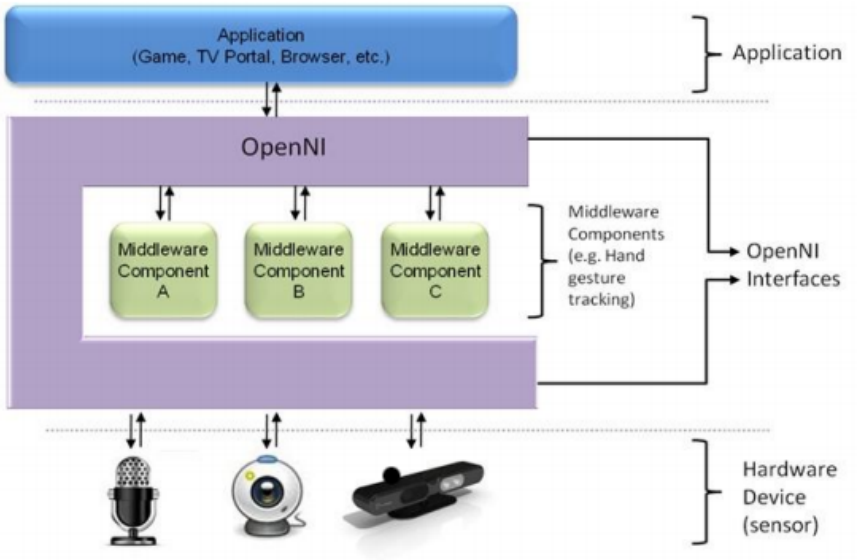

Figure 4.3: OpenNI Layer

4.2.3 PrimeSense Nite Middleware

PrimeSense's Natural Interaction Technology for End-user - NITE - is the middleware that perceives the world in 3D, based on the PrimeSensor depth images, and translates these perceptions into meaningful data in the same way that people do [19]. The PrimeSensor 3D sensor observes the users' environment, while NITE acts as the perception engine that comprehends the user interaction within these surroundings.
NITE middleware includes both computer vision algorithms that enable identifying users and tracking their movements, as well as framework API for implementing Natural Interaction UI controls that are based on user gestures.

NITE Algorithms and NITE Controls

NITE middleware includes:

- A computer vision engine;
- A framework that provides the application with gesture-based UI controls.

This division reflects the two layers that compose NITE:

1. NITE Algorithms - The lower layer that performs the groundwork of processing the stream of raw depth images and utilizes computer vision algorithms to perform:

   a. Scene segmentation – a process in which individual users and objects are separated from the background and tagged accordingly;

   b. Hand point detection and tracking - Users hands are detected and followed;

   c. Full body tracking – Based on the Scene segmentation output, users' bodies are tracked to output the current user pose - a set of locations of body joints.

   NITE Algorithms is an OpenNI compliant module.

2. NITE Controls - A layer of application framework that

   a. Analyzes the hand points generated by NITE Algorithms;

b. Provides tools that enable application writers to design the flow of the application according to the hand movements;

c. Identifies specific gestures, and provides a set of UI controls that are based on these gestures.

Abstract Layered View

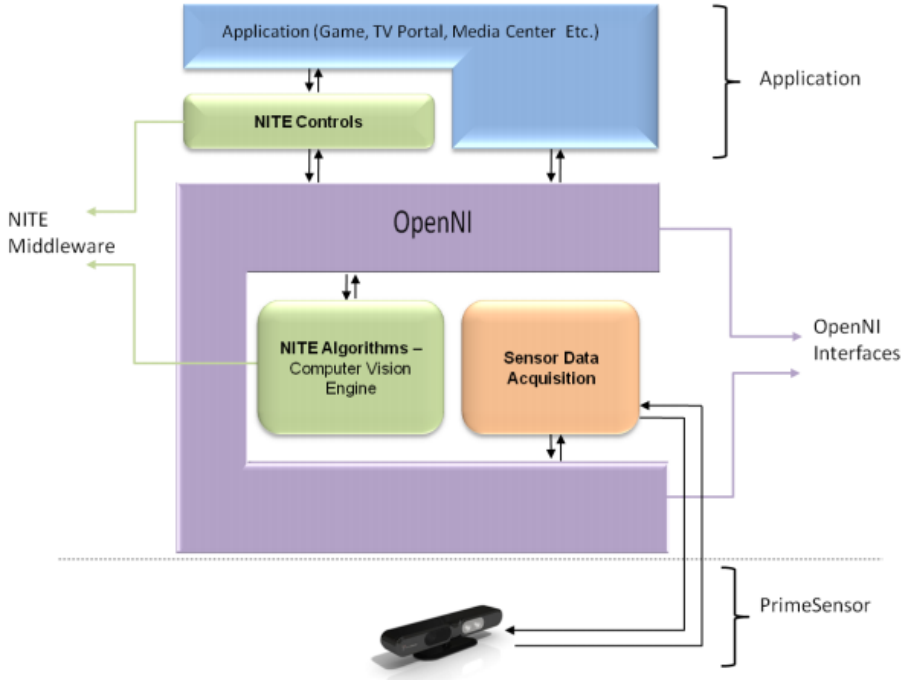

Figure 4.4: Nite Layer

- The lower layer (Figure 4.4) is the PrimeSensor device, which is the physical acquisition layer, resulting in raw sensory data – a stream of depth images;
- The next layer, which is C-shaped (already executed on the host – a PC, a Set Top Box, etc.) represents OpenNI. OpenNI provides communication interfaces that interact with both the sensor's driver and the middleware components, which analyze the data from the sensor;

- The sensor data acquisition is a simple acquisition API, enabling the host to operate the sensor. This module is OpenNI compliant – the interfaces it exposes conform to OpenNI API standard. In other words, this is an OpenNI plug in;
- The NITE Algorithms layer is the computer vision middleware and is also plugged into OpenNI. It processes the depth images produced by the PrimeSensor;
- The NITE Controls layer is an applicative layer that provides application framework for gesture identification and gesture-based UI controls, on top of the data that was processed by NITE Algorithms. NITE Controls communicates with NITE Algorithms via the standard interfaces and data types defined by OpenNI;
- The top-most layer is the natural-interaction-based application. This application can use NITE Controls, and can also approach OpenNI directly in order to access the data generated by NITE algorithms, or even the raw data produced by the sensor.

4.2.4 OSCeleton

OSCeleton is just a small program that takes Kinect skeleton data from the OpenNI framework and spits out the coordinates of the skeleton's joints via OSC messages which can then be used on your language/framework of choice [20].

4.2.5 Microsoft Kinect Driver

On June 16 of 2011, Microsoft released a non commercial Kinect software development kit (SDK) for Windows. The Kinect for Windows SDK beta includes drivers, rich APIs for raw sensor streams and human motion tracking, installation documents, and resource materials. It provides Kinect capabilities to developers who build applications with C++, C#, or Visual Basic by using Microsoft Visual Studio 2010 [21]. The SDK includes the following features:

1. Raw sensor stream;
2. Skeletal Tracking;
3. Advanced audio capabilities;
4. Sample code Documentation;
5. Easy installation.

4.3 Software Installation Guide

Few simple steps are required for working with the Kinect.

Step 0.
Uninstall any previews drivers, such as CLNUI.

Step 1.
Download and install the latest unstable OpenNI Binaries from OpenNI website selecting the *OpenNi Modules* voice in the Downloads tab.
(*OpenNI Unstable Build for Windows x86 (32-bit) v1.4.0.2 Development Edition*).

Step 2.
Download and install the latest unstable OpenNI Compliant Middleware Binaries (NITE) from OpenNI website selecting the *OpenNi Modules* voice in the Downloads tab.
(*PrimeSense NITE Unstable Build for Windows x86 (32-bit) v1.5.0.2 Development Edition*).

Step 3.
Download and install the latest unstable OpenNI Compliant Hardware Binaries from OpenNI website selecting the *OpenNi Modules* voice in the Downloads tab.
(*PrimeSensor Module Unstable Build for Windows x86 (32-bit) v5.0.4.4*).

Step 4.

- Download Kinect Drivers (`https://github.com/avin2/SensorKinect`) and unzip;
- Open the unzipped folder and navigate to Bin folder;
- Run the msi Windows file.

Step 5.

1. Plug in your Kinect device and connect its USB port with your PC.
2. Wait until the driver software is found and applied.
3. Navigate to the Device Manager (Control Panel). You should see something like the following (Figure 4.5):

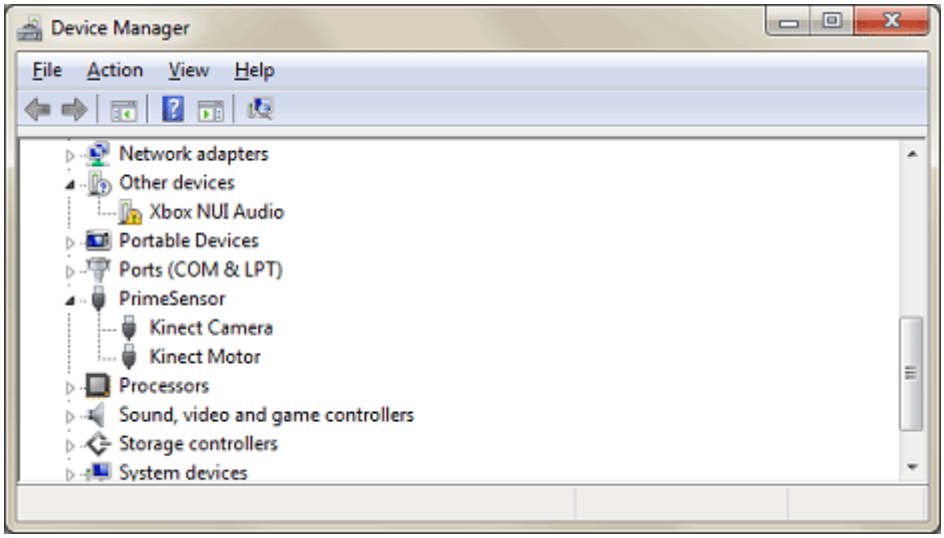

Figure 4.5: Device Manager

Step 6.

Navigate to C:\Program Files\OpenNI\Samples\Bin\ Release (or C:\ Program Files (x86)\OpenNI\Samples\ Bin\Release) and try out the existing demo applications. Try the demos found in C:\Program Files\Prime Sense\ NITE\backslash Samples\ Bin\ Release (or C:\Program Files (x86)\Prime Sense\ NITE\Samples\Bin\Release), too.

4.4 Microsoft Kinect Audio driver and OpenNI/Nite Camera driver

At the time of Kinect SDK release, this project was in its final steps and it was preferred to continue using the PrimeSense OpenNI/NITE drivers. One big con of these drivers is that they don't support audio, making very difficult to take advantage of all the Kinect features. Once you install the Kinect SDK you can't use anymore OSCeleton which is built with OpenNI/NITE API. The top priority was to make the Music Paint Machine simple and try to remove most of the devices that were used. With this goal in mind it was found a very simple way to use OpenNI/NITE motion drivers for the camera devices and the Kinect SDK audio driver for the microphones.
The procedure can be used only in Windows OS and consists in few steps:

1. Click Start and the Run;
2. Type devmgmt.msc in the text box and then hit Enter key or click on the OK button. Device Manager should display right away;

On the Device Manger window, it should appear something like that (Figure 4.6).

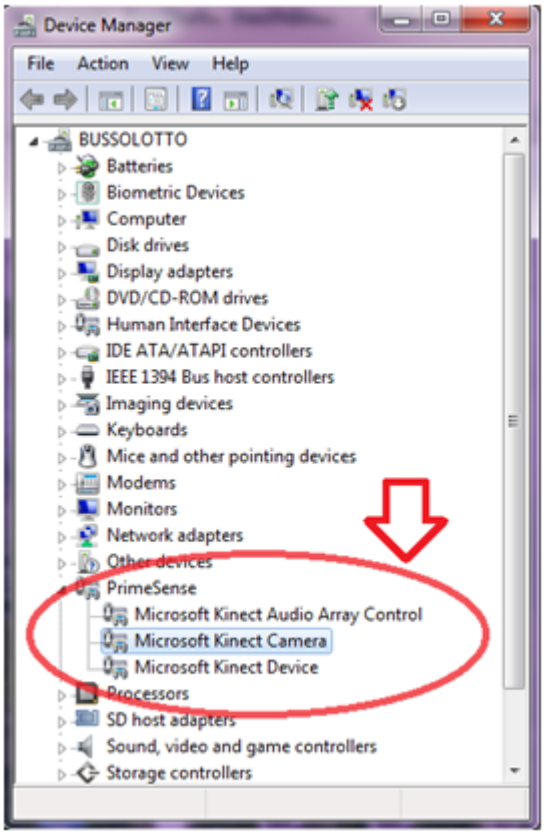

Figure 4.6: Device Manger

3. Right-click on **Microsoft Kinect Camera** and then click on **Update Driver Software**;

4. Click on **Browse my Computer for driver software** and then click on **Let me pick up from a list of devices driver on my computer**.

Update Driver Software window should display right away (Figure 4.7).

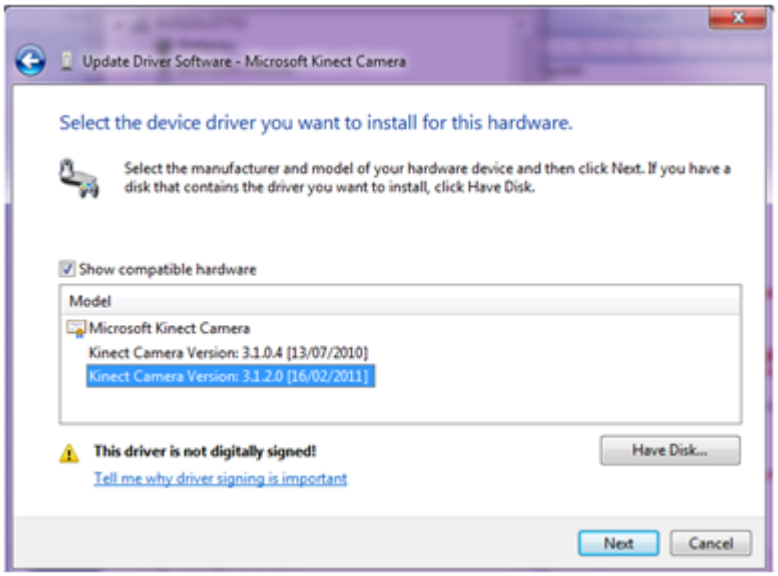

Figure 4.7: Update Driver Software window

5. Double-click on the most recent version of **Kinect Camera**. Kinect Camera should be install in few seconds.

In the Device Manager window (Figure 4.8), it should appear something like that:

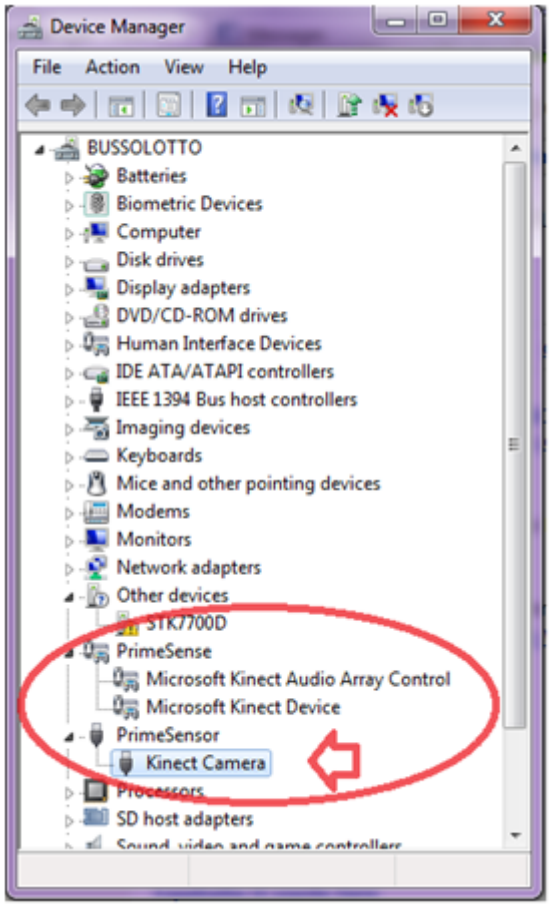

Figure 4.8: Device Manager

4.5 Preliminary Notion

The figure below (Figure 4.9) represents the coordinate system and skeleton representation for a user facing the sensor.

4.5.1 Skeleton Representation

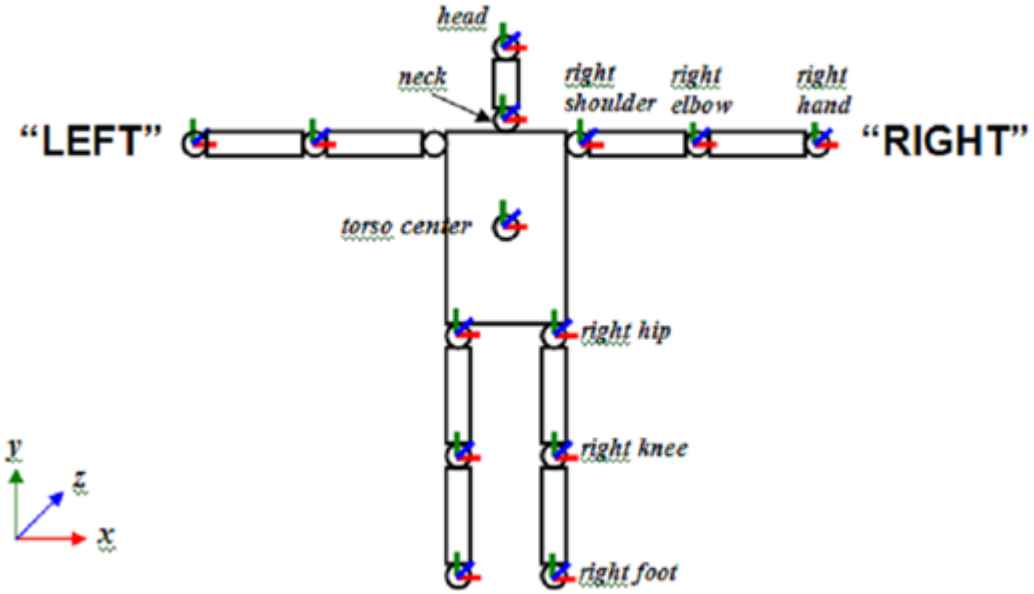

Figure 4.9: Skeleton

Joint positions and orientations are given in the real world coordinate system. The origin of the system is at the sensor. +X points to the right of the, +Y points up, and +Z points in the direction of increasing depth. Joint positions are measured in units of mm.

4.5.2 OSCeleton Joints Transformations

When the coordinates of the joints arrive to OSCeleton they undergo a transformation imposed by the author of the program. The points expressed in millimetres now are expressed in pixel. In the web site it's not mentioned the reason of this conversion but I suppose it was done for having some useful values in case of painting something on the laptop screen. The formulas of the transformations are:

- On the x-axis: $OSCx = off_x + (mult_x * (1280 - X)/2560)$;
- On the y-axis: $OSCy = off_y + (mult_y * (1280 - Y)/2560)$;
- On the z-axis: $OSCz = off_z + (mult_z * Z * 7.8125/10000)$.

The first two formulas normalize the value between 0 and 1 while the last one normalizes the value between 0 and 7.

4.6 Patch Analysis

4.6.1 Direction

The implementation of the direction requires the use of three joints: torso, left and right hips.

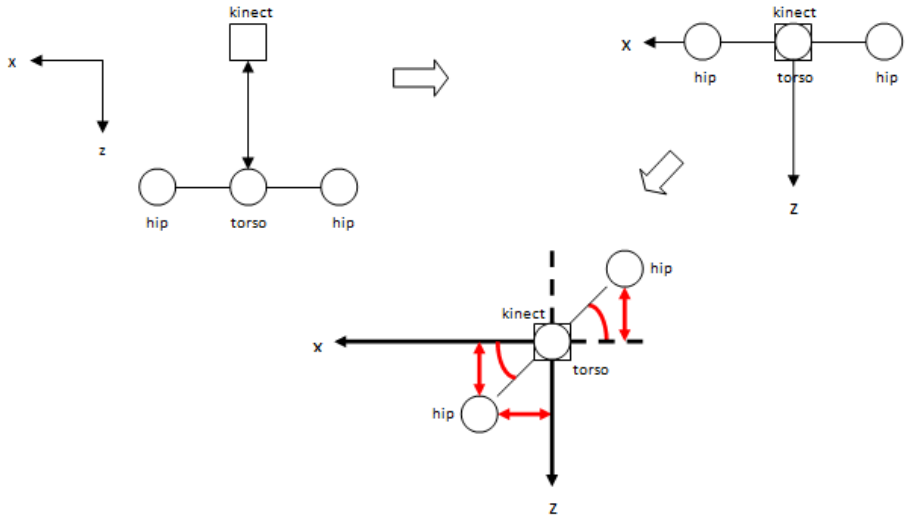

Figure 4.10: Direction

The pictures above (Figure 4.10) represent the steps followed to calculate the direction and they are schematized in the points below:

1. Anti-transformation from the OSCeleton coordinate to real world coordinate system;
2. Translation of the torso in the center of Kinect coordinate system;
3. Translation of the hips in relation with the torso translated coordinates;
4. Transformation of the z-axis coordinates into z-OSCeleton coordinates.

This kind of approach allows the computation of the Music Paint Machine Direction using two methods:

- The distance of the hips in relation to a fix point which is the center of the coordinate system;

- The angle created by one hip as a ratio of edge ($\arctan(\frac{z}{x})$).

The benefits of this technique are:

- Absence of calibration;
- Normalization of the OSCeleton coordinates.

The two hips value of the z-axis, are then sent as inputs in a Max External object (RicClass) written in Java. The object has the function of triggering the direction values (-1, 0, 1) in relation to the maximum distance the hips reach from each other (default is 10cm). The class permits also to modify the threshold, connecting a number box to the inlet 2. The same procedure is also used for the angle in the evaluation of the direction with the difference that the Java class name is Dir and that the default angle is 30°.

Shoulders Vs Hips

In the developing process, a very annoying problem appeared. The direction values (-1, 0, 1) were changing while the player was in the still stand position and with the clarinet near the mouth. After spending few days dealing with the shoulders data, thinking that the problem was caused only by the noise of values, a test experiment showed that moving the arms near the torso, it affected the correctness of the shoulders position (Figure 4.11).

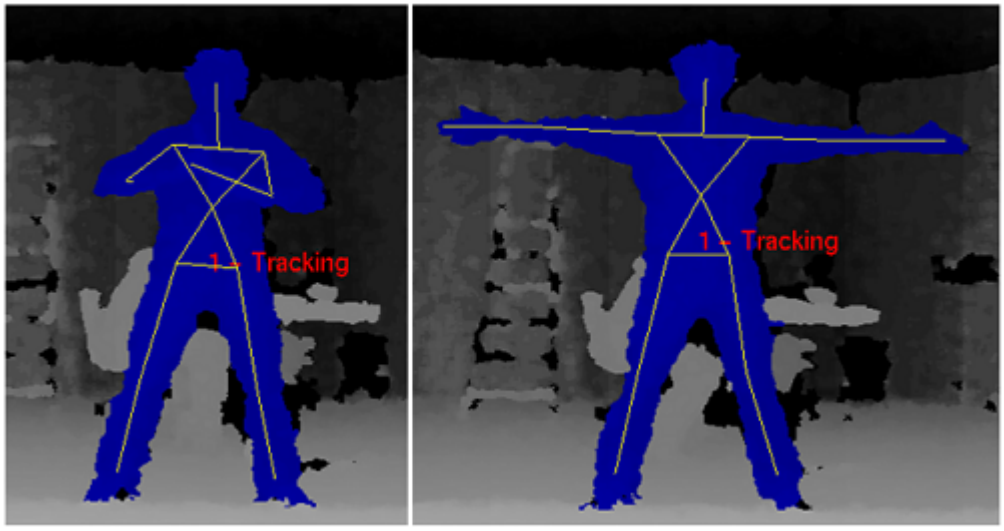

Figure 4.11: Shoulders Instability

The noise of the values and the wrong tracking of the shoulders were enough for deciding to try a more stable couple of joints: the hips. In the end, this choice showed to be better: the tracking process of the hips was not affected by other joints.

4.6.2 Bending

The implementation of the bending requires the use of one joint: the head (Figure 4.12). The calibration in this case is an essential step for the correctness of the function because it depends from the user's height. The only coordinates used, belong to the position of the head in the y-axis.

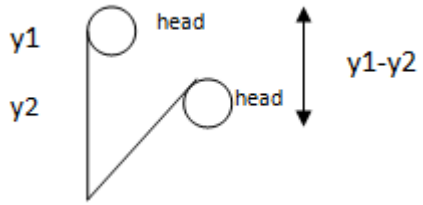

Figure 4.12: Bending

4.6.3 Virtual Mat

One of the most difficult challenges was the realization of the mat via software. The first solution was to manually set up the upper and lower bounds of the zones that belong to a particular colour.
This implementation had some complications:

1. Twelve bounds needed to be set up;
2. Equivalent number of if-then-else sentence in the Java class;
3. Sometimes the expected colour wasn't selected;
4. Problem in matching all the bounds caused by the kinect noise;
5. Feet tracking is still unstable and noisy.

The idea was building a virtual mat divided in some areas, which were univocally identified and which have the possibility to decide if a couple of numbers belonged to themselves. This solution solved most of the problems (listed above) that arouse in the first implementation of the mat which consisted in a list of if-then-else sentences. The coloured zone is drawn automatically by the code and the colours are surely selected when the foot

moves in the proper area. The colour selection doesn't depend anymore by a list of tricky if-then-else sentences. The only variable points now are the couple of points that represent the feet joints and every zone have fixed coordinate instead of having unstable bounds that were chosen through a long test. The Java class Matto and OMat implement this idea.

Circular Virtual Mat

The Max External object, Matto, implements the software version of the Colour (wood) Dance Mat (Figure 4.13). This object, realized in Java, builds a Frame-window of dimension 500x500 in which is drawn an octagon. The vertices of the polygon are used like reference points for the calculation of the squares. The ray of the circumference is about 31 cm and the square edge is about 20.5 cm.

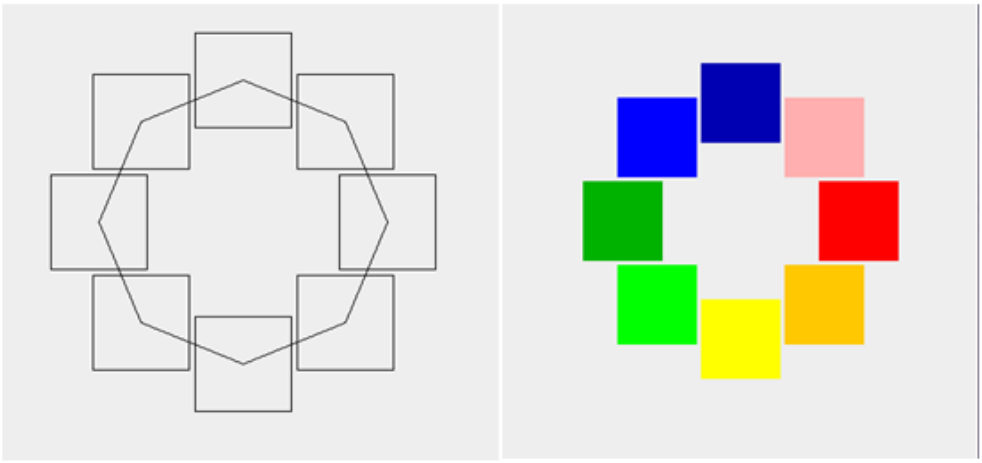

Figure 4.13: Circular Virtual Mat

This object receives in input the X and Z position values and when these points intersect one of the areas they trigger the colour value. About the coordinates used, they were transformed from the OSCeleton z-points into the OSCeleton x-points, for having an equivalent values in both axes. The calibration of the feet, placed one close the other one in the middle of the circle, is necessary for the correct functioning of the mat.

Horizontal Virtual Mat

For highlighting the easiness of changing type of mat, it was realized a Horizontal Virtual Mat (Figure 4.14), implemented by the object OMat. The object draws four adjacent rectangles and its behaviour is the same of the circular one. The width is about 25.5 cm and the height is about 31 cm.

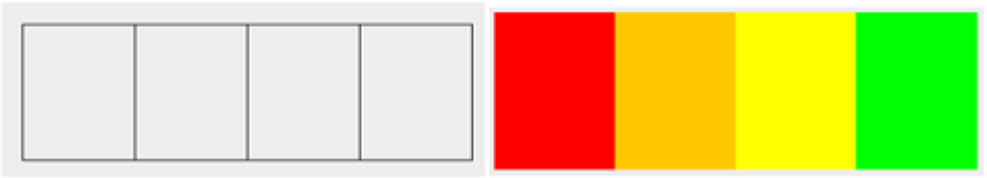

Figure 4.14: Horizontal Virtual Mat

Virtual Circular/Horizontal Mat-Patch example

As explained in the paragraph above, the picture below (Figure 4.15) represent the implementation of the virtual mat using Max/Msp/Jitter. The Java external objects Matto and OMat have a fundamental role in the execution.

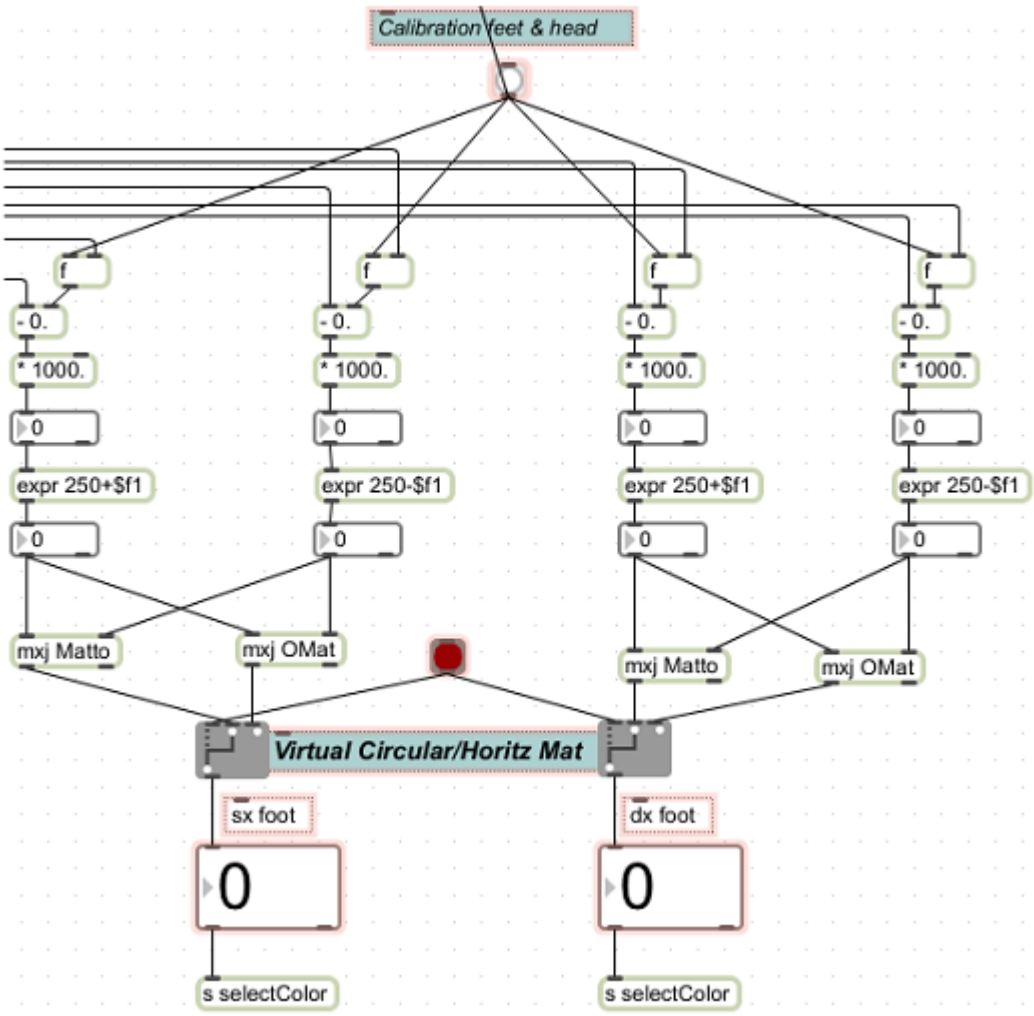

Figure 4.15: Patch Example

4.6.4 Clearing the Screen

A very useful function is clearing the screen with the gesture of the hand (Figure 4.16). In the MPM v2.0 this action was accomplished by pressing some buttons on a USB-Midi control. Every time the player had to stop the music performance, go near the controller and press the right button. Now moving the right hand over the head send a bang that clears the screen. During a test, it was noticed that, when the hand is over the head, the program starts to send a large number of consecutive bangs and the action of clearing the screen is performed after the last bang is executed. This behaviour produced a visible delay. A time function is implemented for sending only one bang. This feature is realized by the class ClrScr.

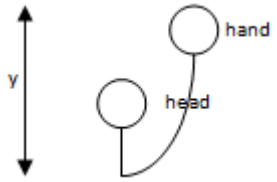

Figure 4.16: Clearing Screen

Chapter 5

Kinect Music Paint Machine Functioning

This chapter will explain the essential steps required to use the Music Paint Machine with the Kinect. The suggested environment for testing or playing with this music interactive system is a common room, provided with a beamer (typically a lab) or the own dining room with a big television with a laptop connection. Using only the own one's laptop and the Kinect is also recommended but this kind of configuration doesn't permit to fully enjoy the painting and to have a useful feedback due to the smallness of the screen. The place, where this project was tested and developed, is the EMIEL lab in the Institute for Psychoacoustics and Electronic Music (IPEM) located in Gent (Belgium).

5.1 Setting up the Music Paint Machine

The preparation of the interactive music system can be accomplished by one person (the player) or at most by two users (player and assistant). In case there is only the player the most annoying part will be the one that requires the calibration of the feet or running some programs. Using a bluetooth mouse, everything can be set up staying in the middle of plastic mat while watching the screen projected by the beamer. This guide takes into consideration the case of two users (Figure 5.1): the player is placed in the middle of the plastic mat and the assistant is behind the laptop outside the Kinect vision field.

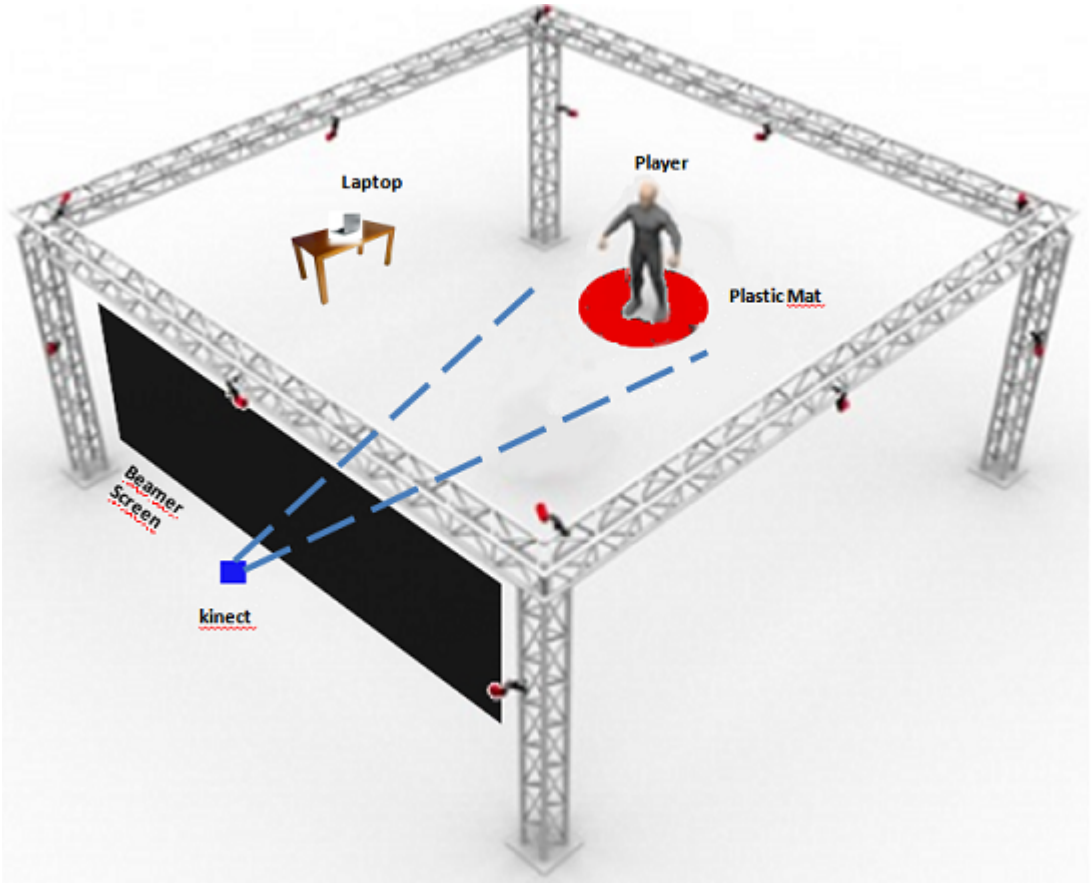

Figure 5.1: Test Environment

5.1.1 Guide

Step 1.

Connect the laptop to the beamer and to the Kinect, located under the screen. If it is necessary use a USB cable extension, in case the Kinect one is too short.

Step 2.

Place the plastic mat in front of the Kinect and 2.5 meter away from it

Step 3.

Open the Max patch. Click on **Options** and then the on **Audio status** and check if the patch uses as input device the Kinect microphones. If not, select **Microphone Array(4-Kinect USB)** from the input device list (Figure 5.2).

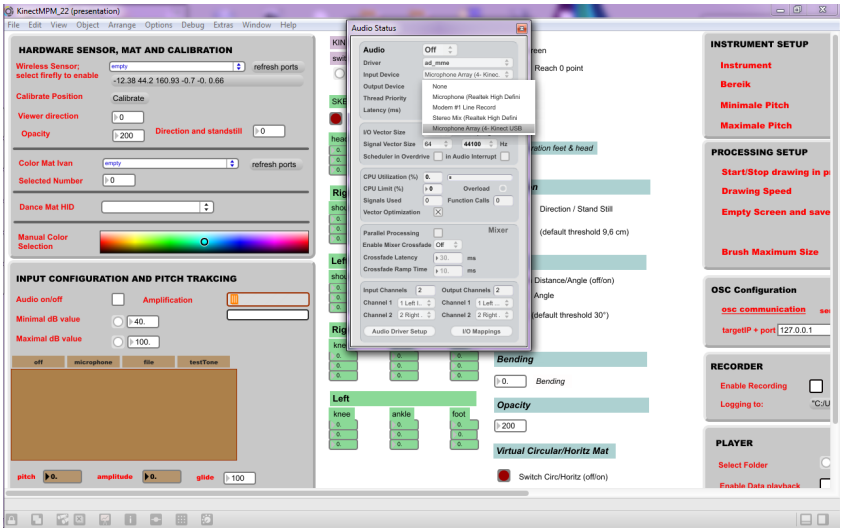

Figure 5.2: Audio Configuration

Step 4.

In the **Input configuration and pitch tracking section**, put a tickle in the **audio on/off** box and then click on the **microphone** tab. If everything is working, in the display immediately below it should appear the audio signal (Figure 5.3).

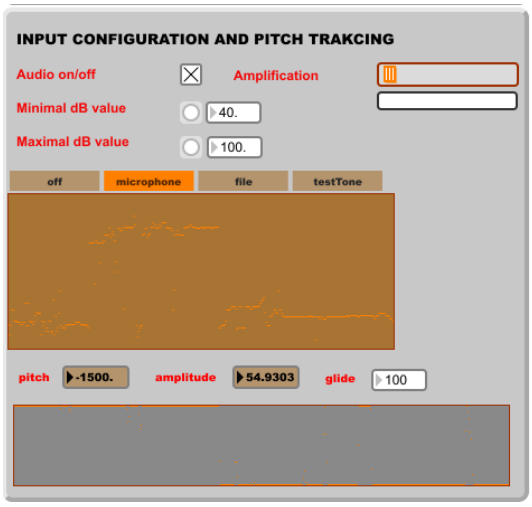

Figure 5.3: Audio Configuration

Step 5.

With the player on the plastic mat facing the Kinect, run OSCeleton and ask the player to do the T-pose for the calibration (Figure 5.4).

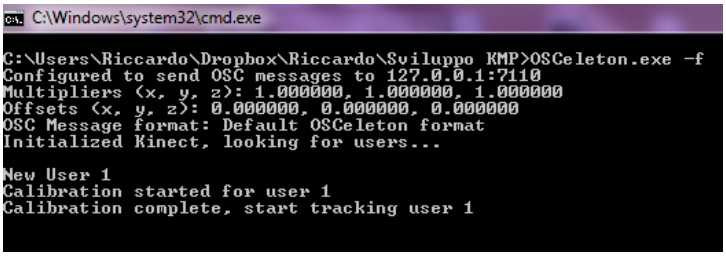

Figure 5.4: OSCeleton

Step 6.

Once the calibration is complete, click on the **Start** buttons in the Kinect section on the Max Patch (Figure 5.5).

Step 7.

Ask the player to put the feet, near to each other, in the middle of the mat and press the **calibration feet & head** button (Figure 5.5).

Step 8.

Make sure that **direction, angle, bending, virtual mats and clearing the screen** are working correctly (Figure 5.5).

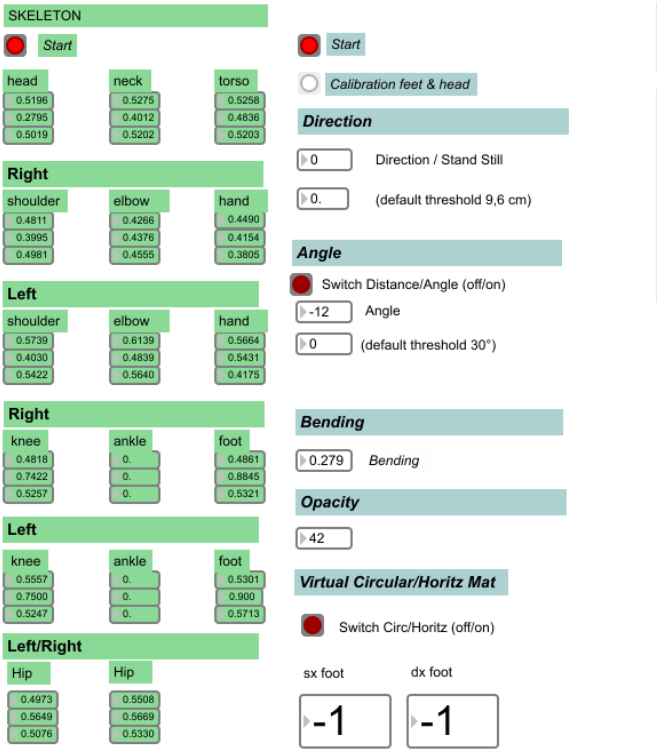

Figure 5.5: Kinect Section

Step 9.

Open the Processing file **brusher.pde** (Figure 5.6) and click play(run). A black screen should appear in few seconds.

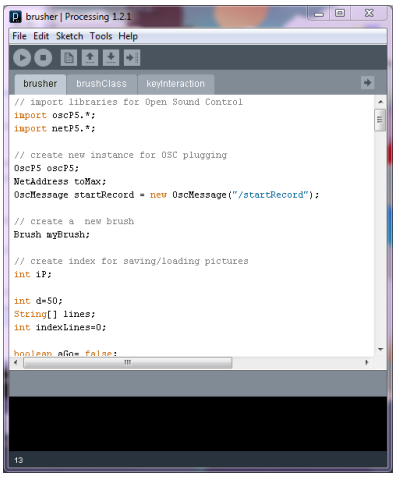

Figure 5.6: brusher.pde

Step 10.

Switch to Max window, pressing **Start + Tab (or Alt + Tab)**.

Step 11.

In the Processing setup (Figure 5.7), section click on **Start/Stop Drawing in Processing**. ***Step 12.***

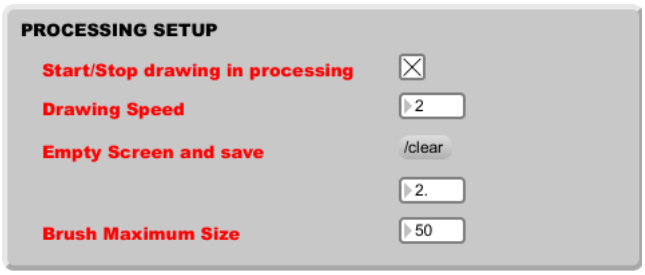

Figure 5.7: Processing Setup

Press **Start + Tab (or Alt + Tab)** for switching back to the Processig black screen and start play (Figure 5.8).

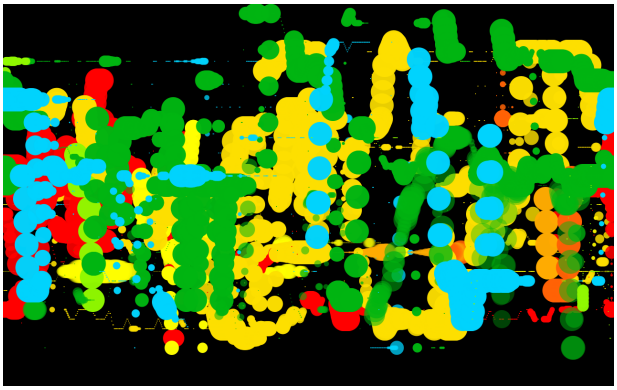

Figure 5.8: Music Paint Machine painting

Chapter 6

Music Paint Machine Usage

The design of the Music Paint Machine, goes around the music experiences of its creator, Luc Nijs, as clarinet and chamber music teacher in formal music education. Gradually, during his lessons, it was more evident the positive contribute of a visual feedback in the process of learning to feel the music and to the development of listening and playing skills [6].

6.1 Examples of Possible Exercises

The tests with Music Paint Machine will concern twelve children and the lessons already started at the beginning of October. The exercises, developed through the years by the teacher, are several and they regard specific task. The most representative exercises, that were used as reference points during the developing of the Kinect Music Paint Machine are:

- moving with the instrument to feel the timing of musical phrases (Figure 6.1);
- step exercises to learn about strong and weak beats (Figure 6.2);
- learning how to emit a constant amount of air for having a enjoyable sound (Figure 6.3).

This three exercises helped respectively in:

- mapping the direction of the brusher;
- implementing the virtual mat;
- checking the sound tracking.

Figure 6.1: Didactic exercise: turn body to feel the timing of a phrase

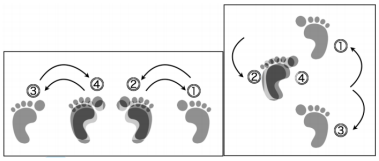

Figure 6.2: Didactic exercises: stepping to feel weak and strong beats (numbers represent beats in a 4/4 measure)

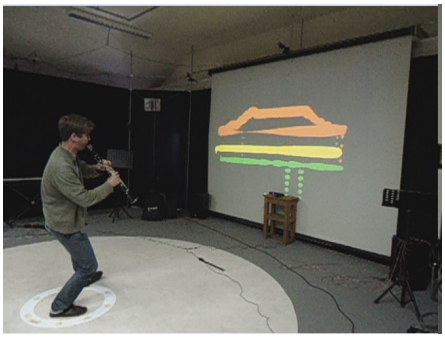

Figure 6.3: Sound Tracking

6.2 Examples of Possible Game Task

The Music Paint Machine can be also an original game. Some challenges could be:

- drawing simple geometric figure (Figure 6.4);
- experimenting abstract landscape (Figure 6.5).

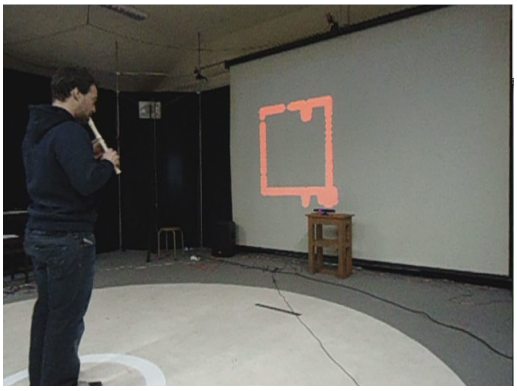

Figure 6.4: A Square

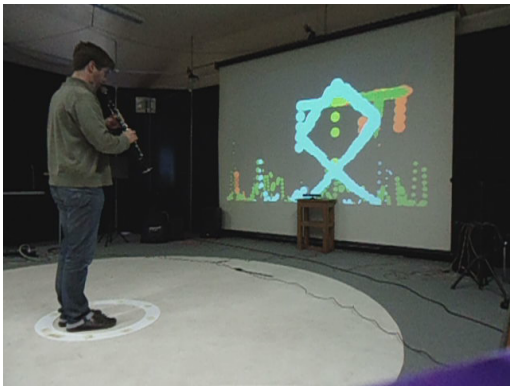

Figure 6.5: Abstract Composition

Chapter 7

PureData Development

The Max patch was developed also for PureData. Pd (aka Pure Data) is a real-time graphical programming environment for audio, video, and graphical processing [22].

7.1 Patch Overview

The reason behind the software porting from Max/Msp/Jitter was to see if the performances reached with the commercial software were comparable with the open-source version of it. The PureData patch (Figure 7.1) is composed by the essential parts of the Max one:

- Pitch and Sound tracking;
- Skeleton Joints;
- Movements;
- OSC, Instrument Setup and Processing.

The Log, Player and Registration section wasn't developed on purpose.

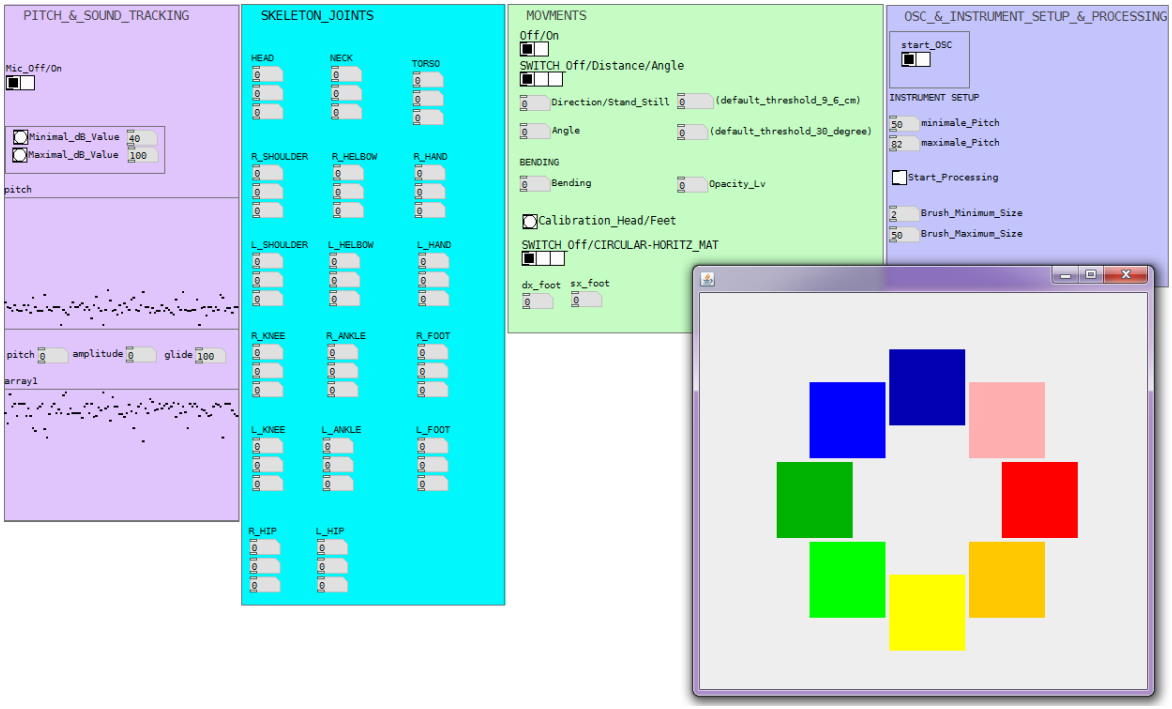

Figure 7.1: PureData Patch

7.2 Patch Functioning

The functioning of the patch is quite similar to the Max one (see chapter 5) and in this section, only the basic steps are listed.

7.2.1 Guide

Step 1.
Open the PureData patch, **presentation_KMPM.pd** (Figure 7.1). Click on **Media** and then the on **Audio settings** and check if the patch uses as input device the Kinect microphones. If not, select **Microphone Array(4-Kinect USB)** from the input device list.

Step 2.
In the **Pitch & Sound tracking section**, click on the **Mic Off/On** box. If everything is working, in the display immediately below it should appear the audio signal (Figure 7.1).

Step 3.
In the **Skeleton Joints** section, click on the **Off/On** box and then select between direction or angle in the **SWITCH_Off/Distance/Angle** box (Figure 7.1).

Step 4.
Ask the player to put the feet, near to each other, in the middle of the mat and press the **calibration Head/Feet** button (Figure 7.1).

Step 5.
Open the Processing file **brusher.pde** and click play(run). In the **OSC & INSTRUMENT SETUP & PROCESSING**, click on **start OSC** box and then on **start Processing** (Figure 7.1).

7.3 Patch Analysis

The performances of the PureData patch are quite comparable of the Max one and in conclusion there is no significance differences in the structure of the two software. Most of the objects used in Max/MSP/Jitter, have their correspondent ones in PureData and also their functioning is equal or quite similar (except rare cases). The PureData, is an open-source software, with a big community behind and its potentially is unlimited. The only weak point consists in the basic and not so user friendly graphic interface, which is fundamental for having an high level of software development productivity.

Table 3 - PD vs Max

	PureData	**Max**
software	**open-source**	proprietary
architecture	**plugin-like**	no plugin-like
add-on	**more extensions and libraries**	absent
tool	**structured data type management**	absent

Chapter 8

Conclusion

8.1 Cons

8.1.1 OSCeleton

It's a very simple program but not very performing while executing particular gestures in short spaces (like moving the feet on a virtual mat). Even if there is the possibility of calling a smoothing function passing the argument –f from the console (osceleton -f), there aren't significant improvements. Looking in deep into the OpenNI API, the documentation about the smoothing function (setSmooth) is unclear, not detailed and it's not also mentioned which kind of filter they implemented for smoothing the data. Another controversial aspect is the developer choice to convert the Kinect coordinates (expressed in millimeter) in pixel values. One reason could be the necessity of having useful values for painting something on a screen.

8.1.2 Absence of a real time filter

The noise of the data is a big problem also working on Max/Msp/Jitter. The implementation of some filters affected so much the response of the movements that the requirement of a real time interaction was not respected and it was preferred working with good bounds. The solution to the data noise could be the usage of a Max object that implements the Kalman Filter but the realization of this kind of filter requires software developing skills and mathematics knowledges that are impossible to acquire in few months.

8.1.3 No pressure and power detection

In the old wood mat, the colours are chosen by pressing the respective colour with the feet while in the virtual mat the colours are chosen by putting the feet on a desired one. In this way the creator's idea, to have a haptic feedback and the possibility of controlling the intensity of the colour through the power involved in the act, run out. A number of alternative solutions were tested but due the Kinect noise of working in short spaces and due the objective impossibility of simulate the act of pressing through software, it was decided to put all the efforts in finding the most accurate way of selecting the colours.

8.1.4 Microsoft Kinect SDK beta vs OpenNI/PrimeSense

This project was developed before the official release of the Microsoft Kinect SDK and the use of OpenNI and OSCeleton were a inevitable choice. Here are explained the main differences between the two development kit.

Skeleton

OpenNI's tracking system requires a user to hold a calibration pose until the tracking system can identify enough of the joints to make out a player[23]. The time this requires varies wildly depending on environment conditions and processing power. The Windows SDK uses a specialized system (for more information *http://research.microsoft.com/pubs/145347/BodyPartRecognition.pdf*) that compares known images of humans to the incoming data from the depth stream allowing them to quickly (less than a second in most cases) determine human shaped objects, and from that generate a joint set. This is great in a walk in/walk out situation, however, it is more prone to false positives than OpenNI.

Joint Accuracy

Kinect tracking system can do predictive tracking of joints, which allows them to maintain a relatively high degree of accuracy in situations where the sensor loses track of the user. This technique however, has two side effects

- it can cause the sensor to generate false positives where it can track non-human objects as humans;
- it requires significantly more processing power to use.

OpenNI, on the other hand, cannot do the predictive analysis that Microsoft can but it generates very few false positives and it requires much less computational power.

Color Video Stream

Windows SDK is able to grab the full 1024×768 resolution of the camera, while OpenNI is only able to get a scaled down 800 x 600 version.

Audio Recording

OpenNI has a driver that permits to access the Microphone array on the Kinect Sensor, but actually there is no support for it with in the API while the Kinect Windows API, allows full access, and recording capabilities. It even allows for the noise canceling effect of the Microphone array.

Speech Recognition

OpenNI has no capability to record audio from the device. The Windows Kinect SDK gives you access to the same speech recognition library used on the Xbox 360. This means you can quickly and easily generate speech files using XML, and import and test them long before you application is completed, using the speech tools. Noise filtering and user selection is built into the system, using the Sensor's microphone array.

Platform Dependence

Microsoft has only deployed the Kinect SDK on Windows. OpenNI is completely supported across platforms, including most major Linux distributions and OS X. This is great if you need to build an application that is multi-platform, and since the code is open source, theoretically developers could port it to any future platform as well.

Summary

Table 4 - Summary

	Kinect SDK			OpenNI		
	Win	Tie	Lost	Win	Tie	Lost
Skeleton	X					X
Joint Accuracy		X			X	
Color Video Stream	X					X
Audio Recording	X					X
Speech Recognition	X					X
Platform Independence			X	X		

8.2 Pro

8.2.1 Hardware: Only Kinect is needed

All the external devices used in the Music Paint Machine v2.0, are replaced by the Kinect. No more accelerometers, magnetometers, backpack transmitters, receivers and midi controllers. The heavy wood mat is also replaced by a virtual software mat and the player is supported by a plastic light mat for selecting the colours. The benefits of using only one device are:

1. easiness of setting up the environment;
2. increase of system portability;
3. increase of time for testing and playing.

8.2.2 Freedom of Movements

The Kinect introduction permits to execute a more large number of movements without the interference of other external devices. In this way the player can concentrate more on the movements, on the tasks given by the instrumental teacher and can focus more on the visual feedback that is receiving from the interactive music system. The movements tracked, are a little subset of all the possibilities this device gives.

8.2.3 Future big Improvements

This work focused only in experimenting what was possible or not with this new technology. It was proved that Kinect could be a valuable solution for this interactive music system, applied to the instrumental teaching. It wasn't tested only with the clarinet but also with the guitar, flute, violin, record and voice. All the players gave so good opinions that they also suggested some customizations for their particular use. The Music Paint Machine v3.0 will utilize some of the solutions tested in the Kinect Music Paint Machine and they will be supported by the Microsoft SDK that grants more data stability and reliability. The usage of the Microsoft development tool will be a great help in further improvements like:

- mapping new movements;
- developing other virtual mats;

- creating a more friendly graphic user interface.

8.2.4 Future Commercialization

Even if the Music Paint Machine's creator first intentions were to design an interactive music system as a tool for supporting his research, no one can hide the commercial potentialities of this system. The Music Paint Machine, if well developed, can be transformed into a Kinect game that has a double role:

- an educational one;
- and an entertaining one.

. The user can exercise himself with his instrument, or otherwise he can have fun in painting something on the television screen. Challenging the other members of the family or friends and see which paint is the most beautiful or more closest to the music composition, could be an interesting game mode too.

Important Stimuli

This job gives four important stimuli:

1. The possibility to apply my music background;
2. The opportunity to participate in the development of a new way of instrumental teaching;
3. The experience of researching in the not so explored field of the interactive music system;
4. The chance to satisfy the technology addiction working with the Kinect.

Bibliography

[1] Marc Leman. *Embodied music cognition and mediation technology.* The MIT Press, 2007.

[2] Marc Leman. *Music, gesture, and the formation of embodied meaning*, page 126??153. Musical gestures : sound, movement, and meaning. Routledge, 2009.

[3] K. Anders Ericsson, Ralf T. Krampe, and Clemens Tesch-Römer. The role of deliberate practice in the acquisition of expert performance. 100(3):363–406+.

[4] D. Hoppe, M. Sadakata, and P. Desain. Development of real-time visual feedback assistance in singing training: a review. *Journal of Computer Assisted Learning*, 22(4):308–316, 2006.

[5] Lehmann A. Acquired mental representations in music performance: Anecdotal and preliminary empirical evidence does practice make perfect? current theory and research on instrumental music practice. (pp. 141-163). oslo: Norges musikkhùgskole. 1997.

[6] Luc Nijs, Bart Moens, Micheline Lesaffre, and Marc Leman. The music paint machine: stimulating self-monitoring through the generation of creative visual output. In *JNMR, special issue on monitoring*, 2011.

[7] W. Altenmuller E., & Gruhn. Brain mechanisms. in r. parncutt & g. e. mcpherson (eds.), the science and psychology of music performance: Creative strategies for teaching and learning (pp. 63-81). new york: Oxford university press. In *JNMR, special issue on monitoring*, 2002.

[8] Goolsby T. W. Portfolio assessment for better evaluation. music educators journal, 82 (3), 39. 1995.

[9] Caroline Palmer and Carolyn Drake. Monitoring and planning capacities in the acquisition of music performance skills. *Canadian Journal of Experimental Psychology*, 51, 1997.

[10] Woody R. H. Learning from the experts: Applying research in expert performance to music education. update: Appliations of research in music education, 19(2), 9-14. 2001.

[11] Luc Nijs, Pieter Coussement, Chris Muller, Micheline Lesaffre, and Marc Leman. The music paint machine - a multimodal interactive platform to stimulate musical creativity in instrumental practice. In *CSEDU (1)*, pages 331–336, 2010.

[12] Max/msp/jitter. Web Site. `http://cycling74.com/docs/max5/vignettes/intro/docintro.html`.

[13] Processing. Web Site. `http://en.wikipedia.org/wiki/Processing_(programming_language)`.

[14] Stephanie Crawford. How microsoft kinect works. Web Site. `http://electronics.howstuffworks.com/microsoft-kinect.htm`.

[15] Kinect. Wikipedia. `http://en.wikipedia.org/wiki/Kinect`.

[16] Ps1080. PrimeSense. `http://www.primesense.com/en/technology/114-the-ps1080`.

[17] Primesensor. PrimeSense. `http://en.souvr.com/product/201004/6180.html`.

[18] Openni user guide. OpenNI Documentation. `http://75.98.78.94/Documentation.aspx`.

[19] Primesense nite control user guide. PrimeSense Documentation. Inside Software Documentation.

[20] Osceleton. Sensebloom. `https://github.com/Sensebloom/OSCeleton`.

[21] Kinect sdk. Microsoft. `http://research.microsoft.com/en-us/um/redmond/projects/kinectsdk/about.aspx`.

[22] Puredata. WebSite. http://puredata.info/.

[23] Kinect for windows sdk beta vs. openni. WebSite. http://labs.vectorform.com/2011/06/windows-kinect-sdk-vs-openni-2/.

Acknowledgements

My heartfelt thanks to my family and my closest relatives (dad, mom, sister, aunts, uncles, grandmothers, grandfathers, cousins, godfathers, dog, family friends, Canadians, everybody that gave me some money) for supporting me till the end of this important step of my life. My not so heartfelt thanks to my colleagues (Nicola Scattolin, Stefano Giusto, Marco Randon, Simone Rinaldo, Luca Zenatti) at DEI that helped me to waste my precious study-time playing with the video-game Hedgewars during the lessons. In this game I'm the BEST (sorry dear colleagues). I really appreciate the generosity of my unaware ex room-mates (Danny Dverin, Jacopo Monticelli) for letting me take their food from the fridge (ahahah i'm a master thief) because i don't like going to supermarket. I felt so guilty for this constant robbery that every six months i offered them only one spritz. Thanks to my actual room-mates for not letting me clean the house for one long month (i wanted but they forced me to relax). Special thanks to my childhood friends (Anrea Ninotti, Alessandro Segatto, Andrea Porta, Massimo Montagner, Alessandro De Gasper) for all the funny moments spent together. My gratitude goes also to Professor Giovanni De Poli, Professor Marc Leman, and Professor Sergio Canazza for giving me the possibility to work six months in the Institute for Psychoacoustics and Electronic Music in Ghent. A heartfelt thanks to Luc Nijs for all the support and materials he gave to me for developing this thesis work.