UPPSALA
UNIVERSITET

IT 12 043
Examensarbete 15 hp
September 2012

Dance quantification with Kinect

Adjusting music volume by using depth data from a Kinect sensor

Christoffer Brodd-Reijer

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Dance quantification with Kinect

Christoffer Brodd-Reijer

Humans interact with computers in many various ways. Different interaction models are suited for different situations and tasks. This thesis explores the motion based interaction made possible by the Kinect for Windows device. A stand-alone library is built for interpreting, analyzing and quantifying movements in a room. It highlights the complexity in movement analysis in real time and discusses various algorithms for interpolation and filtering. The finished library is integrated into an existing music application by expanding its plugin system, allowing users to control the volume level of the playing music with the quantity of their body movements. The results are evaluated by subjective interviews with test subjects and objective measurements such as execution time and memory consumption. The results show that it is possible to properly quantify movement in real time with little memory consumption while still getting satisfying results, ending in a greater incentive to dance.

Handledare: Bengt Sandblad
Ämnesgranskare: Iordanis Kavathatzopoulus
Examinator: Olle Gällmo
IT 12 043
Tryckt av: Reprocentralen ITC

Summary in Swedish

Detta exjobb presenterar ett bibliotek som använder djupdata från Microsofts Kinect for Windows enhet för att kvantifiera rörelser i ett rum. Förbehandling, analysering och kvantifiering sker i realtid med minimal minneskonsumption. Exjobbet går igenom olika metoder av förbehandling av den råa djupdatan och evaluerar deras tillämplighet utifrån dessa krav. Vidare testas bibliotekets genom integrering med ett existerande musikprogram. Programmets tilläggssystem utökas för att kunna köra tillägg som manipulerar volymen i programmet. Biblioteket läggs sedan i ett sådant tillägg och den absolute mängden rörelse i rummet kopplas sedan ihop med volymen på musiken.

Resultatet undersöks både subjektivt och objektivt. Nio personer inbjuds att testa systemet genom att själva undersöka och komma fram till hur deras rörelser påverkar musikens volym. Efter detta får de fylla i ett formulär som undersöker hur intuitivt interaktionen var, vilken grad av kontroll de kände att deras rörelser hade och hur mycket det bjöd in till dans. Objektiva mätningar genomfördes också genom att undersöka körtid och minneskonsumption på två olika datorer.

Resultatet visar att tack vare enkla och effektiva metoder kunde kvantifiering ske i realtid med minimal minneskonsumption, men ändå ge tillfredsställande effekter.

Biblioteket och tillägget är skrivet i C# och släppt som öppen källkod.

Table of Contents

1 Introduction

This bachelor thesis is made at the Uppsala University's department of human-computer interaction and aim at using Microsoft's Kinect for Windows [1] hardware to directly tie movements of people to the volume level of a music application.

The code is packaged as a library which can be integrated into other projects. The library provides access to the raw data and mechanics of the device but also adds some helpful ways to perform more common tasks such as detecting presence of the device. It also exposes a property indicating the quantity of movement in front of the IR sensor.

For illustrative purposes of how the library can be used, a plugin for the Stoffi Music Player[1] has been created which essentially presents the various configurations of the library to the user and sends the quantity of movement, reported by the library, to the music player for volume control.

The project is built using the C# language on top of the .NET platform. All code is released under the open source license GNU General Public License version 3 to enable other software engineers to use, build upon and extend the functionality of the project.

1.1 Background

In consumer oriented technological products, such as gaming systems, the use of motion control has been growing. Products such as the EyeToy and PlayStation Eye for the PlayStation systems from Sony, along with the controller for the Wii system from Nintendo are some examples of motion based input devices. The Kinect hardware was originally developed by Microsoft for use with its Xbox gaming console. However, in light of the widespread reverse engineering of the device for other uses, Microsoft has since released for use with PCs in order to broaden its use [2].

Allowing motion to be used for controlling the interface of a system is not a new idea, but the area is growing and more products are reaching the average consumer. This helps highlight some of the advantages that motion based input has, but it also shows some of the challenges that developers and consumer face when creating and using these interfaces.

The biggest disadvantage of motion based systems is that they are imprecise. It is very difficult to perform a certain motion within margins in the millimeter range. Another challenge is that of discovery and memory. Various people have various definitions of what feels as an intuitive and easy to remember motion. Describing a motion is not always easy and if two motions are too similar it may end up confusing the user.

The guidelines[2] presented by Microsoft aims to help developers create interfaces and motions that are easy to learn, easy to perform and easy to remember.

To make the motion control easy, both to develop and to use, this thesis will focus on a very narrow usage scenario – that of a crowd socializing in a living room – and will only try to supplement existing interactions such as keyboard and mouse, not replace them fully. This means that if the motion based interface fails the user will be able to use another set of input to perform the task.

The task that will be performed by the motion based input is that of volume control. The goal is be to eliminate the step of changing volume when the crowd moves from sitting idly and chatting to standing up and dancing in the room.

[1] www.stoffiplayer.com
[2] http://go.microsoft.com/fwlink/?LinkID=247735

This will both provide an intuitive way of controlling the volume and it will serve to invite to dance as the higher volume, connected to movements of the body, creates a psychological connection to dance.

1.2 Problem description

The project needs to reach the following goals:

- Accurate movement analysis
- High efficiency
- Easy to integrate in third party code
- Invite to dance

The function of the library – that of quantifying movement - needs to work in many different situations and scenarios. It must therefore be configurable so that third party developers and/or users can change the parameters and thus behavior of the library so that it fits the current use case.

High efficiency is required since the quantification need to be done in real time with as little delay as possible. A delay below 300 milliseconds would be preferred to ensure a subjective feeling of instant response [3].

The library needs to be easy to use by any third party programmer so as to provide the ability to quantify the dancing movements of a crowd in a room. By creating a library the code can be used inside other projects and can thus be extended and adapted easier. It is therefore important that the code is abstracted and easy to integrate into other systems. This means that a well-defined Application Programming Interface (API) is needed.

A more subjective but still important goal is to invite people to dance. The library is integrated into an existing music player by a plugin system. The plugin is connected the movement quantification with the volume of the music application. This enables users to control the volume with dance movements. The goal is to have the plugin be installed and used with ease.

1.3 Related research

There has been extensive research in the area of human detection from still and video images. Most common is the usage of histograms [4, 5] but there is also the technique called Partial Least Square (PLS) which is a dimensionality reduction technique [6]. These are all based on color images from visible light. This present a few challenges, mostly when trying to perceive humans against a noisy background or in dim lightning.

Other research uses 3D data from stereoscopic cameras [7], time-of-flight cameras [8] or IR-sensors [9]. These improves upon the detection techniques using visible light as detection can be done regardless of lighting conditions, color or texture of background or clothing. Some of these techniques, such as Relational Depth Similarity Features (RDSF) allow the detection to be done in real time [9]. Research has also been done using the IR-sensor of the Kinect device for Xbox 360 [10, 11]. These however do not do detection in real time and take tens of seconds to complete.

A related field is the research of crowd analysis. Most of the studies in this field are focused on modeling the behavior of pedestrians [12] or crowds in public spaces to avoid or minimize disasters [13]. The models used to analyze crowd behavior and flow are helpful in situations of complex geometry [14]. Some techniques for modeling crowd dynamics use self-learning algorithms using statistical data of normal crowd behavior coming from sources such as surveillance cameras [15].

2 Application Programming Interface

Since the part of the code will be packaged as a Dynamic-link library (DLL) it needs a well-defined Application Programming Interface (API) that will fit in most scenarios where this code will be used. The API should be extendable without needing to break the Application Binary Interface (ABI), making it possible to keep the library backwards compatible, increasing the incentive for developers to upgrade to newer versions of the code.

The API will also need to allow for quick integration and deployment of the code into an existing system at the same time as it should allow for fine grained control of the Kinect device and full access to all the raw data from the sensors.

Lastly the API needs to expose some configuration parameters to allow developers to either them themselves, or their users via some sort of interface, manipulate the behavior of the quantification code.

To meet all these requirements the properties described in Table 1 where created. Each property behaves as a variable for the using developer where some are read-only and some are read-write.

Name	**Type**	**Description**
`Sensor`	`KinectSensor`	Gets the currently connected sensor.
`Range`	`DepthRange`	Gets or sets the range of the depth sensor.
`Format`	`DepthImageFormat`	Gets the format of the depth stream.
`MinDepth`	`int`	Gets the minimum depth value that can be detected by the sensor.
`MaxDepth`	`int`	Gets the maximum depth value that can be detected by the sensor.
`IsConnected`	`bool`	Gets whether or not the Kinect device is connected.
`Elevation`	`double`	Gets or sets the angle of the Kinect device. Between 0 (down) and 10 (up).
`IsEnabled`	`bool`	Gets or sets whether the analyzer is running.
`FrameWidth`	`int`	Gets the width of the depth data frame. -1 if no device detected.
`FrameHeight`	`int`	Gets the height of the depth data frame. -1 if no device detected.
`Quantity`	`float`	Gets the quantity of dancing. Between 0 (no dancing) and 10 (full on party).
`Viscosity`	`Double`	Gets or sets the amount of sensitivity to changes in movement that the Quantity property has. Between 0 (fast changes) and 10 (slow changes). Default: 5.
`Sensitivity`	`Double`	Gets or sets the amount of sensitivity to movement that the Quantity property has. Between 0 (much movement needed) and 10 (little movement needed). Default: 5.

Table 1. Properties of the analyzer library.

The `Sensor` property gives the developer raw access to the Kinect sensor device, allowing for full access to the raw data. The `Viscosity` and `Sensitivity` properties allow for adjustment of the analyzer's behavior.

3 Implementation

The code can be divided into three parts. First there's the detection and initialization of the Kinect device. Secondly there's the fetching and preprocessing of the depth data from the infrared (IR) sensor. Lastly the data is processed and analyzed for final quantification of the movement in front of the depth sensor.

In order to work with the Kinect for Windows device the Software Development Kit (SDK) must be downloaded and installed. Microsoft provides both the SDK and a Toolkit with code samples and software called Kinect Studio which let developers apply hooks into all calls to the Kinect device. The Toolkit also provides the `KinectSensorChooser` class which is further discussed in section 3.1.

3.1 Device detection

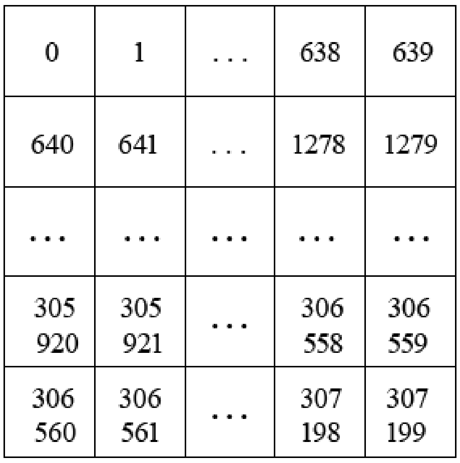
Detecting the device is done using the class `KinectSensorChooser`. After creating an instance of this class and starting it, the class will fire an event called `KinectChanged` which occurs when the status of a device is changed, such as when it is connected, disconnected, or there is a problem with the hardware.

Whenever the `KinectChanged` event, or the `PropertyChanged` event, which indicates that a property of the chooser itself has changed, is fired the details of the event will not be further examined. Instead a function called `RefreshDevices` is called which will update the relevant properties. This makes for less than and it ensures that in every situation where a relevant event is fired, the class will update the appropriate properties.

Device detection has been tested in various ways. Both hardware and software can change the connectivity of the device. Detection is made properly if the device is plugged in or unplugged using either the Universal Serial Bus (USB) connector or the power connector of the device itself. Detection of when driver is uninstalled or crashes also occurs properly, as well as if the driver is installed or recovers.

Only one device can be detected and used at the same time. This is a limitation in the project and not the SDK or Toolkit. Support for multiple devices is possible and the advantages such support would bring are further discussed in section 5.

3.2 Depth data



0	1	. . .	638	639
640	641	. . .	1278	1279
. . .	. . .	. . .	. . .	. . .
305920	305921	. . .	306558	306559
306560	306561	. . .	307198	307199

Figure 1. The depth frame with the array index for each pixel.

The depth data received from the IR sensor on the Kinect device comes as an array of shorts. Each short corresponds to a pixel in the depth frame image. The sensor supports the resolutions 80x60, 320x240 and 640x480, all at 30 frames per second (FPS). For this particular project a resolution of 640x480 was used.

By using the 640x480 resolution the short array is 307,200 elements long. Each short represents a pixel in the frame as seen in Figure 1.

A short is 16 bits and the first (highest order) bits represents the actual depth at that pixel while the last (lowest order) 3 bits represents the player segmentation data. This layout is illustrated in Figure 2. In this project the player segmentation is not used and the shorts are thus shifted 3 bits to discard that information.

This means that the data is preprocessed by shifting all pixel values three bits to the right. The result will be the number of millimeters to the point at that pixel in the room. The sensor has a lower limit of 800 mm and an upper limit of 4000 mm. All pixels where the distance is further than 4 meters will thus register as 4000 and all pixels where the distance is less than 8 decimeters will register as -1. These pixels are shown in black in Figure 3.

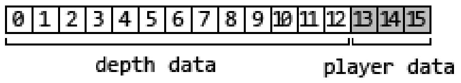


Figure 2. The layout of a depth pixel.

On certain surfaces the IR laser will be unable to get a proper reading of the distance due to refraction or diffraction.

This causes a value of -1 to be registered at the pixel. The nearest neighbor interpolation algorithm is described in Figure 4. The middle pixel is the one being interpolated. The algorithm looks at all neighbors, starting with those directly next to the pixel being interpolated and working outward. As soon as a pixel with a value other than -1 is encountered the value of that pixel is copied into the pixel being interpolated.

There are various techniques to mitigate this problem with unknown distances. One is to apply what is called Nearest Neighbor Interpolation [10, 11]. This is an interpolation of the unknown pixels which looks at the neighbor pixels and applies the same value as the nearest neighbor with a proper distance value.

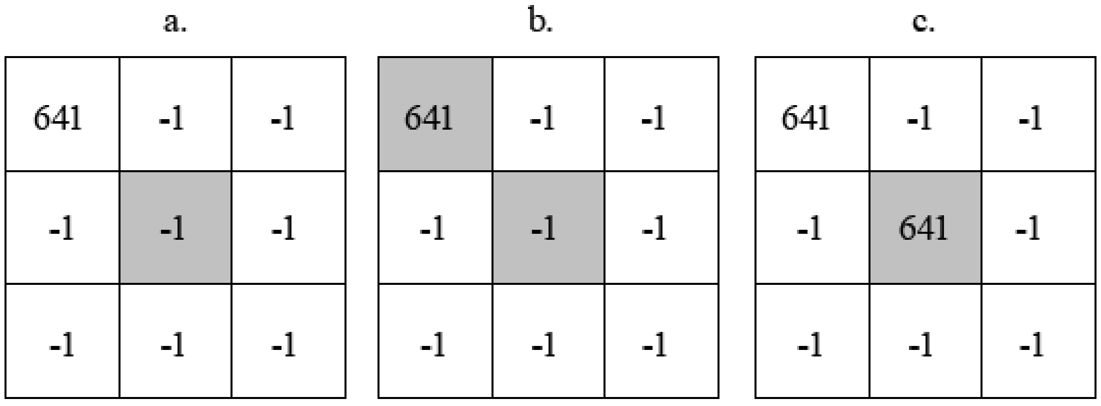


Figure 3. Nearest neighbor interpolation. a) shows the pixel to interpolate. b) shows the nearest pixel with a proper value and c) shows the value copied to the interpolated pixel

In addition to this algorithm another interpolation was also evaluated. It worked by looking in each of the four directions (left, right, up and down) and interpolated the pixel with the mean value of all pixels encountered holding a proper value. The Nearest Neighbor Interpolation algorithm is slightly faster since it only needs to encounter a single proper valued pixel for the process to come to a stop.

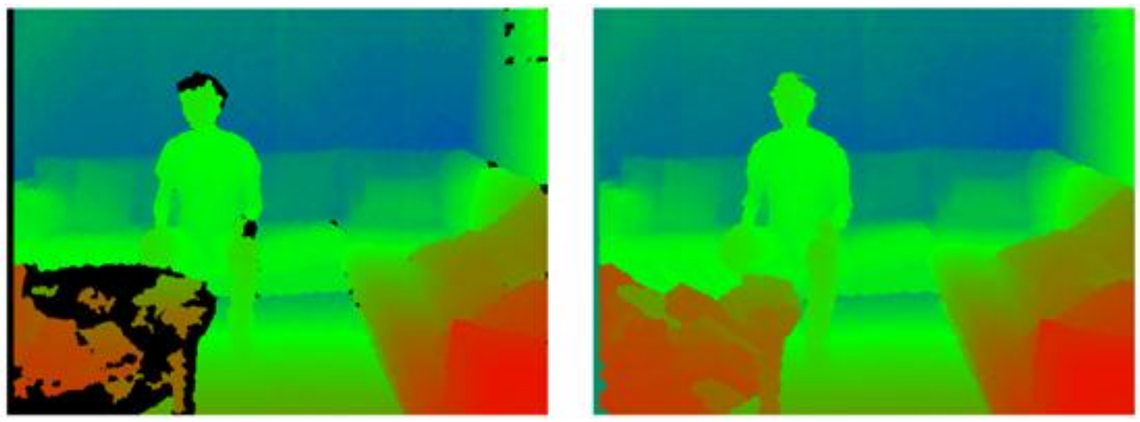


Figure 4. The effects of interpolation. To the left is the color coded depth image before interpolation (black is pixels with unknown depth, value of -1) and to the right is the image after interpolation has taken place using the Nearest Neighbor Interpolation.

The interpolation creates a block like artifacts as seen in Figure 3. This is because the frame is processed from left to right, down to bottom and will thus only need to look to either the left or above to find a proper valued pixel. This means that large blocks of pixels with a -1 value will be filled by the value either above or to the left of the area.

To counter this, a median filter can be used [11]. The algorithm for median filters is illustrated in Figure 4. Usually a 3x3 block is used when applying median filters, but the principle of the algorithm is the same no matter how large the block is. All pixels inside the block are ordered and the median value is used to replace the value of the pixel or pixels being filtered.

A mean filter works in the same way but instead the mean value of all values is inserted at the filtered position instead of the median value.

The median filter has the property of keeping edges better than the mean filter and is thus preferred for working with images where edges are of interest [10, 11].

The time needed to perform bit shifting, interpolation using the Nearest Neighbor Interpolation algorithm and applying a median filter in C# on an AMD Athlon II X2 3 GHz processor is several seconds if more than 50% of the image needs to be interpolated. The operations take around 50 milliseconds if less than 5% needs to be interpolated.

Even though the code has been optimized and only a single iteration over the frame is performed, the operation is too slow to meet the goal of a maximum of 300 milliseconds between action and result.

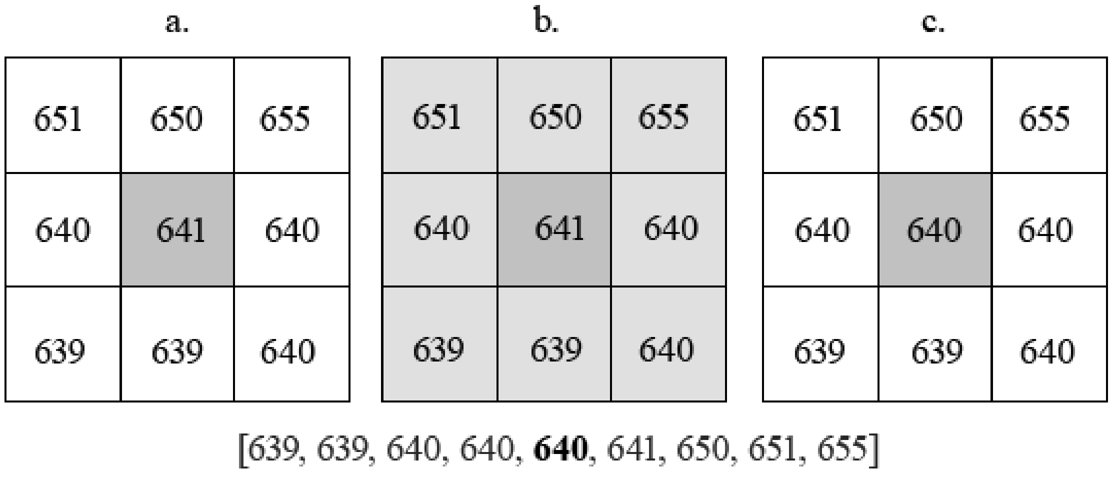


Figure 5. Median filter. a) shows the pixel that will be filtered. b) select all surrounding pixels (including the to-be-filtered pixel) and extract the median value. c) replace the value of the filtered pixel.

A faster approach is to keep the pixels with a -1 value and not perform interpolation. As discussed in section 3.3 this can be used as an advantage and increases the accuracy of the motion analysis. The interpolation also ran at different speeds depending on how many pixels carried the -1 value. If the camera was obstructed by an object that was closer than the minimum detectable depth (8 dm) the majority of the picture would need to be interpolated, adding significant processing time just to prepare the frame for analysis.

Additionally, if the interpolation is skipped then the filtering is not needed anymore. The processing time after discarding interpolation is thus less than 25 milliseconds on the same CPU.

3.3 Motion analysis

The analysis of motion is done by using two consecutive depth frames. Thus two arrays are needed; one array carries the pixels of the current depth frame, and the other array carries the pixels of the previous depth frame. The first time the code is run the latter will be empty which means that analysis will be skipped that iteration. For every other run the two frames will be compared and analyzed, after which the current depth frame will be copied over to the array carrying the previous depth frame's pixels.

Since the depth frame is 640x480 pixels and each pixel is represented by a short which is 16 bits, a single array is 614,400 bytes. Two of these arrays will thus take up 1,200 kilobytes of memory, or 1.2 megabytes.

When the two arrays are compared, they are iterated over and the pixel at each position in one frame is compared with the corresponding pixel in the other frame at the same position in the other frame. If the difference between the values of those pixels is over a certain threshold the pixel is considered to be changed, otherwise it is considered to be identical.

The reason to use a threshold is because of false positives. Even in a room with no movement there are changes to the values of the depth frame over time. These changes are mostly small (less than 100 millimeters) but there are some pixels where the change is over one meter. To eliminate most of these false positives a threshold is used. This threshold is described in Equation 1. The value of `Sensitivity` is described in Table 1 and is set by the developer using the code (who may have gotten the value from a user via an interface). The default value for `Sensitivity` is 5.

$$t = 600 - Sensitivity * 50$$

Equation 1. The threshold used to determine if a change in depth is significant.

The number of pixels where the change in depth is over the threshold is then calculated as a percentage of the total number of pixels in the depth frame. Any pixel with a value of -1 is discarded. If a pixel has a value of -1 it means that the object is either unreadable (due to diffraction or refraction) or closer than the minimum detectable depth. If these pixels are discarded from the whole analysis it means that there is no need for interpolation or other forms of preprocessing. See section 3.2 for discussion on why this is preferred.

The full equation for getting the quantity of movement between two depth frames is described in Equation 2. The `Sensitivity` value is again used to boost the resulting value.

$$q = \frac{3p * Sensitivity^3}{l}$$

Equation 2. Quantity of movement where p is the number of pixels where significant change was detected and l is the total number of pixels with a proper (not -1) value.

The result from Equation 2 is set to 10 if it exceeds the value 10, or set to 0 if it is negative. The result is a quantification of movement in the room between two frames.

$$f = \begin{cases} 1, & Viscosity \leq 0 \\ \dfrac{0.3}{Viscosity^2}, & Viscosity > 0 \end{cases}$$

Equation 3. The factor used in the EWMA calculation for the new Quantity.

The `Quantity` property is updated using the Exponentially Weighted Moving Average (EWMA). Equation 3 shows the usage of the `Viscosity` property to calculate a factor between 0 and 1 which is used in Equation 4 to calculate the new `Quantity` property.

$$Quantity = Quantity * (1 - f) + q * f$$

Equation 4. The EWMA used to calculate the new Quantity value.

By using the EWMA the Quantity becomes resistant to large fluctuations in movement. Just as with `Sensitivity`, the value of `Viscosity` is set to 5 by default.

4 Integration with player

The library is integrated with the Stoffi Music Player, allowing it to control its volume according to the value of the `Quantity` property. This player was chosen for integration because of both the author's deep knowledge of its code and because of the fact that it’s open source and uses the same programming language and development platform – C# and .NET respectively – as this project, thus making the integration easier.

While the music player has a plugin system in place [16], it needs to be expanded to allow for manipulation of the sound by the plugin. Prior to this project the plugin system only allowed for visualizing plugins which displayed graphics and had access to Fast Fourier Transform (FFT) data from the audio stream.

The integration of this library into the music player via the plugin system also serves as an evaluation of how well the library can be used by a third party.

4.1 Extension of plugin system

A new class of plugin is created called a `Filter` plugin. Plugins of this class can specify various manipulations of the sound, ranging from distortions and echo to pitch and volume control.

As a limit of the scope of this project, only the volume control is implemented into the player itself. While the plugins can specify other manipulations to be performed, they are discarded at the player’s code.

To allow the plugin to present the `Viscosity` and `Sensitivity` property to the user, a setting system is needed. Since the plugin system is cross-platform and plugins cannot contain any code from the `System.Windows` namespace of .NET, the plugins cannot specify the actual graphical elements to present to the user. The plugins instead specify the setting as a class containing the settings ID (a string identifying the setting) the type of the setting’s value and optionally a list of possible values as well as maximum and minimum values (used for number types). A default value and whether or not the setting should be visible are also specified by the plugin.

Name	**Type**	**Description**
`Volume`	`double`	Sets the volume. Between 0 (no volume) and 100 (full volume).
`Chorus`	`BASS_DX8_CHORUS`	Sets the chorus.
`Compressor`	`BASS_DX8_COMPRESSOR`	Sets the compressor.
`Distortion`	`BASS_DX8_DISTORTION`	Sets the distortion.
`Echo`	`BASS_DX8_ECHO`	Sets the echo.
`Flanger`	`BASS_DX8_FLANGER`	Sets the flanger.
`Gargle`	`BASS_DX8_GARGLE`	Sets the gargle (amplitude modulation).
`I3DL2Reverb`	`BASS_DX8_I3DL2REVERB`	Sets the interactive 3D audio level 2 reverb.
`ParamEq`	`BASS_DX8_PARAMEQ`	Sets the parametric equalizer
`Reverb`	`BASS_DX8_REVERB`	Sets the reverb.

Table 2. The properties of the Filter plugin class.

The player then displays these settings specified by the plugin to the user in its graphical user interface. All labels are looked up in a translation file shipped with the plugin by using the ID string of the setting. As a setting is changed an event is fired and if the plugin is subscribed to that event its code is run. The same flow applies when any property of a setting is changed by the plugin. It fires an event which propagates into the player which in turn updates the interface.

The `Filter` plugin class provides ten properties which the plugin can adjust. These are described in Table 2. As stated earlier only the `Volume` property is handled by the player. Similarly to the flow of settings and changes to their values, the change to the `Volume` property fires an event which propagates into the player where the actual volume is adjusted.

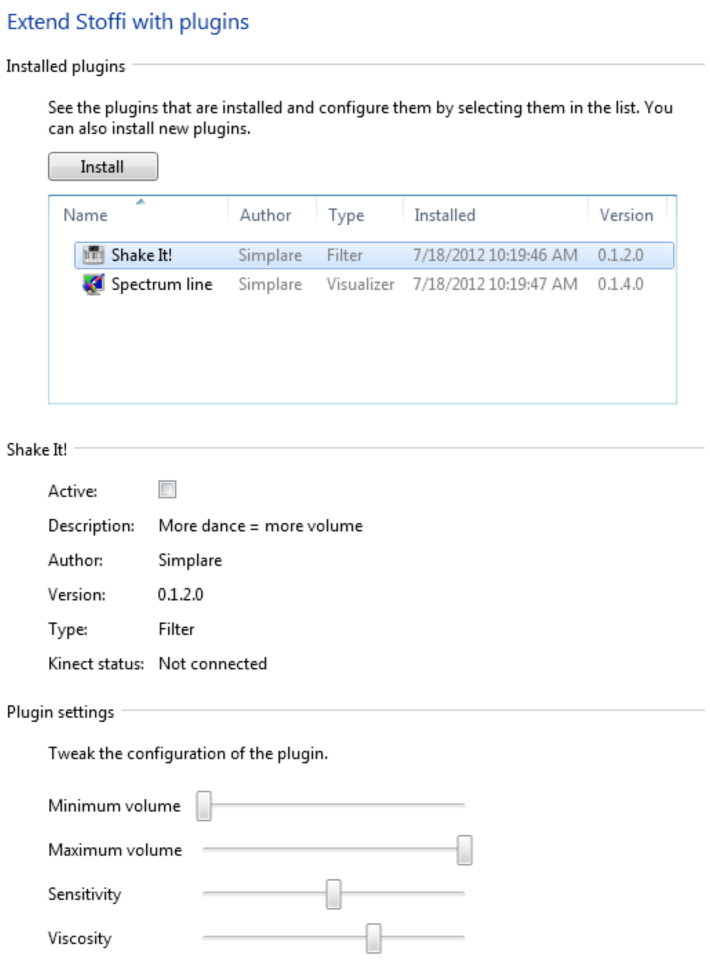
4.2 Using library in plugin

The library is put inside a plugin which exposes the four settings `Viscosity`, `Sensitivity`, `MaximumVolume` and `MinimumVolume`. These settings are set to 5, 5, 100 and 0 respectively, and changeable by the user via the interface and mechanisms described in section 4.1. The Viscosity and Sensitivity settings are sent to the library unaltered. The `MaximumVolume` and `MinimumVolume` are used to calculate the volume sent to the player according to equation 5.

$$v = v_{min} + \frac{q(v_{max} - v_{min})}{10}$$

Equation 5. The volume sent to the player (v) as expressed by the minimum volume and maximum volume set by the user. Where q is the Quantity property reported from the library.

The calculation is run and the resulting value sent to the player by a method which is hooked to the `PropertyChanged` event of the library. The plugin also subscribes to the `Connect` and `Disconnect` events where it updates a label showing the user the status of the Kinect device hardware.


Extend Stoffi with plugins

Installed plugins

See the plugins that are installed and configure them by selecting them in the list. You can also install new plugins.

Name	Author	Type	Installed	Version
Shake It!	Simplare	Filter	7/18/2012 10:19:46 AM	0.1.2.0
Spectrum line	Simplare	Visualizer	7/18/2012 10:19:47 AM	0.1.4.0

Shake It!

Active:	☐
Description:	More dance = more volume
Author:	Simplare
Version:	0.1.2.0
Type:	Filter
Kinect status:	Not connected

Plugin settings

Tweak the configuration of the plugin.

Figure 6. The management interface of the plugin.

The libraries IsEnabled property is set to true or false when the plugin is started and stopped respectively. The stopping and starting of the plugin is detected by overriding the `OnStart()` and `OnStop()` methods.

5 Result

The goals stated in Section 1.2 demanded that the library should function in many different situations with a various number of people in the room, different rooms and with different types of dance styles (depending on music genres). The library also needed to work in an efficient way and deliver results in a timely manner, preferably within 300 milliseconds for the subjective feeling of “instantaneous”. Lastly, since the library is built to be integrated into third party code, it needs to be easy for developers familiar with the C# and .NET platform.

Most of these goals are directly tested against, while some are simulated in various ways. The plugin is tested with different sets of people performing different kinds of dances in front of the camera. Testing is only done in two rooms but from different angles and positions within those rooms. The people involved in testing are asked some short questions about how they perceive the accuracy and usability of the system.

The testing of the integration of the library into existing code comes only from the actions described in Section 4.2 and is thus highly biased as the author of the plugin is also the author of the library. It does however provide a few insights into how well the integration of the plugin can be performed. Especially the amount of code needed and the number of required changes to the project configuration.

Lastly the efficiency of the plugin is evaluated by measurements of its speed and usage of resources such as memory and processor cycles. The speed is evaluated by counting the number of ticks – where a tick is 100 nanoseconds – between the beginning and finish of a certain operation. These are then averaged over 10,000 iterations. Resource consumption is reported by the operating system.

5.1 Accuracy

At a `Sensitivity` setting of 5 the plugin will register movements such as the extending and full contraction of the arm (touching the left shoulder with the left hand and then extending the left arm perpendicular to the body), giving a value `Quantity` of about 3-4 if done 3 feet from the camera. Movement of only the head or hands is barely registered and will yield a value of below 1 at the same distance. Moving both arms from straight up to straight down, alongside the body, having the fully extended all the time (similar to mimicking the flapping of wings) will yield a value of 7-8 at the same distance by one person, and reach a steady 9-10 by two.

At zero sensitivity no movement will be registered and at full sensitivity anything more than the waving of the arm from the elbow will register a full 10.

The `Viscosity` provides a mean for short-lived by large fluctuations to be muted. At full viscosity the `Quantity` value will take one minute and thirteen seconds to reach 99.9 % of its value and seven seconds to reach 50%. While at a viscosity of 5 it takes eighteen seconds to reach 99.9% and two seconds to reach 50%. At a viscosity value of 2 it takes two seconds to reach 99.9% of the value.

The plugin has more false positives than false negatives at detecting dance movements. A person walking past the camera will yield a positive result, as will the act of gesticulating with the arms or changing position from sitting to standing up. It will however not give a positive result for “light head banging” (moving the chin up and down a few centimeters) at a sensitivity level of above 2.

5.2 Efficiency

The efficiency measurements are done on two computers using an Intel processor and an AMD processor. Code is placed before and after the call of the method to be measured, measuring the number of ticks between both points. Each tick is 100 nanoseconds.

	System 1	System 2
Processor	Intel Core i3 540	AMD Athlon II X2 250
Clock speed	3.07 GHz	3.0 GHz
Cores	2	2
Bus speed	133.8 MHz	200 MHz
L2 cache	2x256 Kb	2x1024 Kb
L3 cache	4 Mb	N/A
Memory type	DDR3	DDR3
DRAM frequency	668.9 MHz	533.3 MHz

Two methods are measured: the preprocessing of the depth image and the analysis of the movements between two frames. Each method is measured 10,000 times and the average is used.

	System 1	System 2
Preprocessing	6.09 ms	7.42 ms
Analysis	12.81 ms	14.93 ms

The memory consumption is measured by looking at the values reported by the operating system, in this case Windows 7. The values are taken from the system application Task Manager and are reported as the Private Working Set for the process.

Memory used	9,683 kilobytes

5.3 Integration into existing code

As the library has only been integrated into a single plugin and the author of the library is also the author of the plugin and the music player hosting the plugin, it is not suitable to use this experience as an indicator on how well suited the library is for integration into third party code.

However, conclusions can be drawn on the relative amount of code needed to get the library functional into an existing code base. It is also possible to evaluate the extent of changes that need to be made to the project configuration in order to accommodate for the library and its dependencies on the Kinect SDK libraries.

The minimum amount of code needed to integrate the library is six rows

- One row for including the namespace
- Two rows for creating a method to handle the event `PropertyChanged`
- One row for connecting that method to the event
- One row for reading the `Quantity` property
- One row for starting the library

Additionally, in the plugin, two more methods are created for handling the event of a device connecting and disconnecting. One row is also added for stopping the library.

The amount of configuration needed to the project itself is to add three libraries to the project folder and include them as references into the project. Two of these libraries are the Kinect SDK and the Kinect Toolkit.

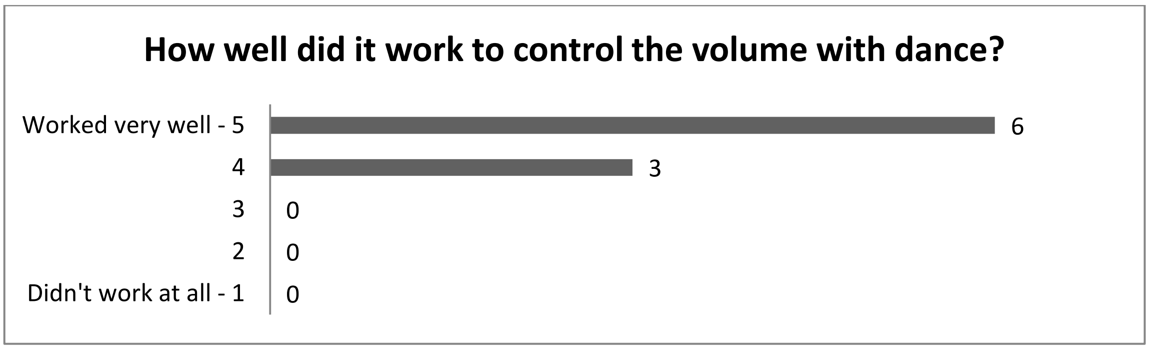
5.4 Surveys

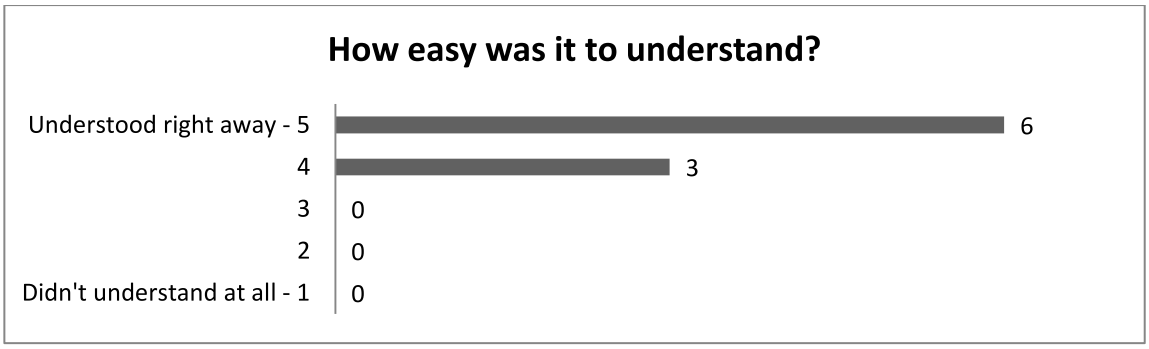
A total of ten people were asked via an anonymous survey how they perceived the plugin after testing it. The survey was created online and emailed, thus allowing people to send in answers in an anonymous way and allowing for more honest feedback.

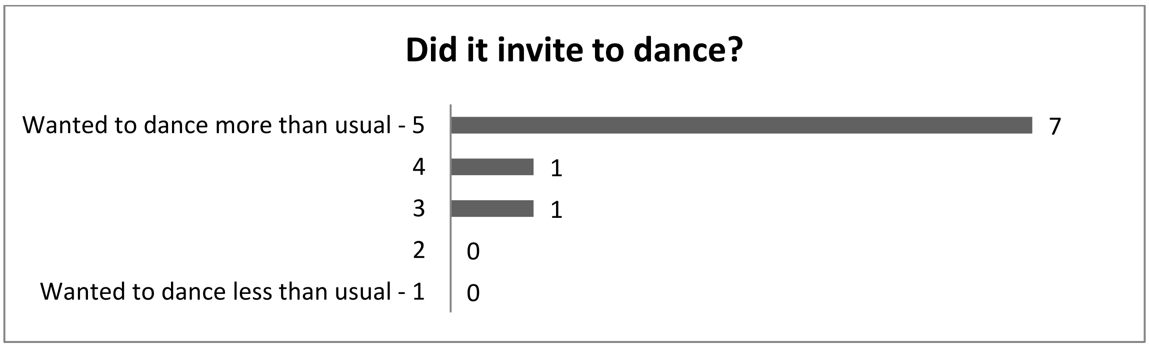
The people where asked three questions which they answered by grading their experience on a 1-5 scale. The questions where

- How well did it work to control the volume with dance?
- How easy was it to understand?
- Did it invite to dance?

The surveyed people where allowed to experience with the plugin. Feedback was presented both visually and by sound. The author of the plugin was available during the testing to answer questions. The testers were in ages 14 to 60, both male and female. Various professions were represented: teacher, nurse, carpenter, computer technician, construction worker and self-employed in mental health care.







6 Conclusion and future work

The goals stated have been reached fully or in part. The code is efficient enough to provide feedback less than 300 milliseconds from start of input which, according to studies, leads to the feeling of immediate feedback [3]. The systems where the efficiency of the code was measured are well within the range of specifications of most computers sold the last couple of years.

The code might need adaptation to run efficiently on lesser capable systems such as mobile phones, tablets or embedded systems but since the Kinect for Windows hardware is not available on anything other than Windows desktop computers it is less of an issue.

The accuracy of the system is generally good but there are many false positives. Non-dance movements often cause the volume to go up but this could be a positive effect as it might increase the willingness of the person to dance, which was one of the goals. The dance movements which are not detected are very few. Most of the undetected dance movements were those of the legs but that is an issue with the position of the camera and not the code itself. This can be improved by adding more cameras and increase the depth data, an enhancement which is covered in Section 6.2.

Integrating the library into an existing code base was not sufficiently tested to allow for any definite conclusions. However, the amount of code required was in the single digit lines of code and the library could be added to a project with very few operations. Given good enough instructions and assuming that the third party project is using the same framework, programming language and Integrated Development Environment it should be fairly easy for any developer to start using the library.

The people who tested the plugin gave very positive feedback. Most were happily surprised at the very first interaction with the system. The survey showed that the system was intuitive to use which also showed during testing. Not anyone needed to ask any questions on how to use the plugin but could quickly figure out how their movements affected the volume.

The survey only included ten people and only as many were available to test the plugin before this report. This means that no definite conclusions can be reached by going by the survey and testing alone. More extensive testing is needed to understand how people use the system. No testing of the installation and setup of the plugin was done which is an important step before the plugin can be used.

6.1 Known issues

The plugin stops reporting accurate values after the computer resumes from going into hibernation while the plugin is running. The reported value moves from 0 and 5 but cannot go any further and is constantly fluctuating by a value of 1-2. It is not known if this issue is within the plugin or the library. The first order of investigation for resolving this issue would be to look at the depth data received by the library after a resume from hibernation.

If this data is in any way flawed the issue is with the hardware and it might be possible to fix the bug by forcing the hardware to be restarted after hibernation has occurred. If the data is correct the code can be analyzed all the way from raw data to the value sent to the volume control of the music player, in search of a deviation or flaw.

A second issue is the rather limited field of view of the camera. The depth sensor can see 43.5 degrees vertically and 57.5 degrees horizontally. Object farther than 4 meters or closer than 0.8 meters cannot be detected. This makes it problematic to include more than a handful of people in the field of vision. A way to mitigate this issue is to use more cameras. Some enhancements is required to the library, they are discussed in Section 6.2.

Since the library detects changes in depth and at a very fast rate (30 times per second) it is not possible to increase the volume by moving something toward and away from the camera. The solution to this is not trivial. It would either include a second analysis over several frames or a new way to eliminate false positives of movement reported by the sensor. Fortunately the problem is minor as this movement is difficult to perform isolated from other movements which will yield detection.

The last issue is that movement near the camera will result in a larger `Quantity` value than movement farther away from the camera. This is due to the fact that an object closer to the camera will affect more pixels. This issue can be mitigated by weighting the number of changed pixels to the distance they have changed. This works since the background will always be no more than 4 meters away as that is the maximum depth the sensor can reach.

6.2 Potential enhancements

The library can currently only use a single Kinect for Windows device. By expanding the number of cameras the field of vision can be enlarged and thus enable the library to analyze the movement of a larger crowd. As long as the SDK can properly handle additional cameras it would be trivial to expand the function of the library. The depth frames from the various cameras could be combined into a single array and iterated over as before. To increase efficiency cluster of cameras could be created whose depth frames are combined and iterated over by a single thread. Each of these clusters could then be analyzed simultaneously.

The plugin can also be extended to use more gestures for controlling the application. Gestures can be added to skip a song, play or pause, have the computer read the name of the song and artist out loud, or close the application. Voice control could also be added but care would have to be taken as the application may be playing loud music and thus making voice pattern recognition harder.

To fully allow third party developers to use the library in their code more documentation is needed. A fully working code example and a tutorial would lower the barrier and making the library easier to use.

Lastly, the library could be ported to more platforms. It might be possible to get the library to run on OS X or Linux under the Mono framework.

References

[1] T. Leyvand, C. Meekhof, Y.-C. Wei, J. Sun och B. Guo, ”Kinect Identity: Technology and Experience,” *Computer,* vol. 44, nr 4, pp. 94-96, 2011.

[2] J. Giles, ”Inside the race to hack the Kinect,” *The New Scientist,* pp. 22-23, 4 December 2010.

[3] D. Benyon, Designing Interactive Systems: A Comprehensive Guide to HCI and Interaction Design, Pearson Education Limited, 2010.

[4] N. Dalal och B. Triggs, ”Histograms of oriented gradients for human detection,” CVPR, San Diego, CA, 2005.

[5] N. Dalal, B. Triggs och C. Schmid, ”Human detection using oriented histograms of flow and appearance,” European Conference on Computer Vision, Graz, Austria, 2006.

[6] W. Schwartz, A. Kembhavi, D. Harwood och L. Davis, ”Human detection using partial least square analysis,” ICCV, 2009.

[7] H.-D. Yang och S.-W. Lee, ”Reconstruction of 3D human body pose from stereo image sequences based on top-down learning,” *Pattern Recognition,* vol. 40, nr 11, pp. 3120-3131, 2007.

[8] V. Ganapathi, C. Plagemann, D. Koller och S. Thrun, ”Real time motion capture using a single time-of-flight camera,” *CVPR,* pp. 755-762, 2010.

[9] S. Ikemura och H. Fujiyoshi, ”Real-Time Human Detection using Relational Depth Similarity Features,” 2010.

[10] S. N. Krishnamurthy, ”Human Detection and Extraction using Kinect Depth Images,” Bournemouth University, 2011.

[11] L. Xia, C.-C. Chen och J. K. Aggarwal, ”Human Detection Using Depth Information by Kinect,” The University of Texas, Austin, 2011.

[12] F. Feurtey, ”Simulating the Collision Avoidance Behavior of Pedestrians,” 2000.

[13] D. Helbing, I. J. Farkas och T. Vicsek, ”Simulating Dynamical Features of Escape Panic,” *Nature,* vol. 407, pp. 487-490, 2000.

[14] R. L. Hughes, ”The Flow of Human Crowds,” *Annual Review of Fluid Mechanics,* vol. 35, pp. 169-182, 2003.

[15] E. L. Andrade och R. B. Fisher, ”Simulation of Crowd Problems for Computer Vision,” The University of Edinburgh, 2005.

[16] C. Brodd-Reijer, F. Gadnell, C. Carenvall och M. Tibblin, ”Engaging in music: making the enjoyment of music an integrated part of life,” 2012.