Bachelor's Thesis

Information Technology

Digital Media

2016

Max Lindblad

PROTOTYPING A SYSTEM CAPABLE OF TRACKING THE EXECUTION OF ARBITRARY PRE-RECORDED FULL-BODY MOVEMENTS

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Digital Media

2016 | 42

Instructor: Principal Lecturer Mika Luimula, Ph. D.

Max Lindblad

PROTOTYPING A SYSTEM CAPABLE OF TRACKING THE EXECUTION OF ARBITRARY PRE-RECORDED FULL-BODY MOVEMENTS

This thesis is focused around the concept of full-body tracking and the process of developing a system capable of evaluating the progress of executing a predefined full-body movement.

The first major part of the thesis focuses on the theory of full-body tracking, first with some historical and industrial use of the concept, and then focuses on more recent developments with consumer-grade technologies, with a tour of some of the most common solutions.

The second major part focuses on the development of the actual system, called Clinical Layer, which is the most major element that the thesis focuses on. With the project starting in the summer of 2014, the approximate timeline of different advancements in the project will be described to the end of the development phase of the latest version in the summer of 2015.

The next portion of the second part consists of detailed technical description of the basic parts of the system, including the embedded animation system and different ways of animation data can be passed around in the system. After that the ways of creating content for the system are depicted.

Describing the actual underlying algorithm of the tracking system, the next chapter lays out the subsystem's main principles and discusses vital factors relevant to its operation.

With a more practical outlook, the following part takes a look at a few demonstration/testing applications developed that utilize the system.

The last part before the conclusion takes a look at a comparable technology, called Visual Gesture Builder (VGB), while comparing it's features to those of Clinical Layer. The conclusion lays out that the software project was successful, having received good feedback, but also mentions that an interested party should also look at Visual Gesture Builder.

KEYWORDS:

motion tracking, Kinect, digital games, pattern recognition, repetition tracking, software development

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikka | Mediatekniikka

2016 | 42

Ohjaaja: Yliopettaja, FT Mika Luimula

Max Lindblad

VAPAAMUOTOISTEN ENNALTANAUHOTETTUJEN KOKOVARTALOLIIKKEIDEN SUORITUKSEN SEURANTAAN TEHDYN JÄRJESTELMÄN PROTOTYYPPAUS

Tämä opinnäytetyö on kohdistunut koko vartalon liikekaappauksen konseptin ja tästä saatavaa dataa käyttävän järjestelmän kehityksen ympärille. Tämän järjestelmän tarkoitus on kyetä seuraamaan ja arvioimaan ennaltakaapatun mielivaltaisen kokovartaloliikkeen suoritusta.

Opinnäytetyön ensimmäinen osa keskittyy kokovartaloliikeseurannan teoriaan, aloittaen kuvaamalla ensin konseptin historiaa ja käyttöä ammattialoilla, ja sitten käy läpi viimeaikaisempia kehittymiä, kuvaten muutamia yleisempiä ratkaisuja.

Toinen huomattava osa työtä keskittyy itse järjestelmään, joka onkin työn huomattavin keskittymiskohde. Alkaen vuoden 2014 kesästä, järjestelmän kehitysprojektin eri vaiheet kuvaillaan vuoden 2015 kesään, jolloin projektin viimeisin kehitysvaihe päättyi.

Toisen pääosan seuraava osa koostuu yksityiskohtaisesta teknisestä kuvauksesta järjestelmän pääosista, mukaanlukien järjestelmän sisäisen animaatiojärjestelmän ja eri tapoja miten animaatiodataa voi liikutella järjestelmässä. Tämän jälkeen, eri tavat luoda sisältöä järjestelmän käytettäväksi luonnehditaan.

Kuvaten järjestelmän itse tarkoituksen eli liikkeen seuraamis- ja arviointitoiminnon, seuraava kappale esittää tämän alijärjestelmän pääperiaatteen ja pui prosessin eri huomionarvoisia seikkoja.

Ottaen käytännönläheisemmän näkökannan, seuraava osa kuvailee muutaman projektin aikana kehitetyn, järjestelmää testaus- ja esityskäyttöön käyttävän sovelluksen.

Viimeinen osa ennen loppupäätelmiä ottaa katsannon vertailtavaan järjestelmään nimeltä Visual Gesture Builder, verraten tätä kehitettyyn järjestelmään. Loppuluvussa todetaan projektin olleen onnistunut saatuaan hyvää palautetta, mutta mainitaan että asiasta kiinnostuneen kannattaa myös vilkaista Visual Gesture Builder -järjestelmää.

ASIASANAT:

liikkeentunnistus, kinect, digitaaliset pelit, hahmontunnistus, toistontunnistus, ohjelmistokehitys

CONTENT

PICTURES

LIST OF ABBREVIATIONS

3D	Three-dimensional
LED	Light-emitting diode
IR	Infrared
CMOS	Complementary metal-oxide semiconductor
SDK	Software development kit
API	Application programming interface
JSON	JavaScript Object Notation
FBX	Filmbox
COLLADA	COLLAborative Design Activity
BVH	Biovision Hierarchy
VGB	Visual Gesture Builder

1 INTRODUCTION

Many kinds of new input systems have become more and more commonplace during the recent years, especially in consumer applications. Unlike while using a common keyboard and a mouse, input systems like multi-touch, accelerometers, integrated heart rate meters, fingerprint readers and various types of cameras can make otherwise unnecessarily complex or impossible tasks where the device needs to be able to obtain knowledge of its surroundings and the user, much more feasible. In some uses, for example in health care and fitness, these input methods can also provide information that would also otherwise be unknown by the user or be unreliable.

Making interaction more natural can also result in making the activity much more engaging and fun to the user, for example with body tracking technologies. When the user can't cheat the system but will also get encouraging feedback related to their performance, activities that otherwise are arduous and laborious can become a lot less annoying and, at their best, turn from negative things that the user "needs to do" to fun, game-like experiences that the user will keep coming back to. The latter effect is the aim of an effort called gamification, and can be applied even without the use of advanced input systems, but using them can sometimes be very advantageous for user experience and compliance.

One of the input methods that is getting more common is full-body tracking. This is also the focus of this thesis. In the first major part, the state of the art of consumer-oriented full-body tracking systems will be taken a look at. After that, the process of developing a system utilizing full-body tracking data for tracking the execution of pre-recorded movements will be described. In addition the developed system will also be compared to a similar system in retrospect.

2 FULL-BODY TRACKING

The idea of full-body tracking is that a device can see where and in what pose a person is and output that data. Even though the concept is simple, consumer applications for this have started popping up only somewhat recently.

2.1 Early systems and applications

The earliest full-body tracking systems were used (and are still being used) by movie industry for capturing lifelike animation for 3D (Three-dimensional) animated characters by using human actors.

Here the performance of an actor or actress was captured and stored (Motion capture), and later this sequence of poses was utilized by mapping the performer's joints to the virtual character's matching joints, which often resulted in the movements of the character being more believable than if the character was animated by hand. Especially as physical interactions such as secondary motions, weight and exchange of forces are from a real-world source, the motions seem a lot more natural to the human eye (Gutierrez etc. 2008, 56).

These motion capture systems use several technologies for capturing the motions, including passive (reflective) or active (equipped with light-emitting diode (LED)) markers, which are tracked by cameras, or / and inertial systems, which do the capturing with sensors attached to the joints they are tracking. Sometimes even mechanical exoskeletons are used. Often facial animation is captured simultaneously with full-body data, as capturing it separately would require large amounts of time spent re-timing and re-animating and would tend to seem unnatural. An example setup used in a fully digitally animated movie utilizing infrared (IR) markers placed on the both the joints and the face can be seen in Picture 1, along with the final computer generated shot utilizing the acquired motion capture data. (Bredow etc, 2005, 8)

Picture 1. A motion capture setup using IR markers, side-by side with the resulting computer generated "shot". (Bredow etc, 2005, 3)

The problem with many of the systems mentioned above is that these systems are usually very expensive, even the cheapest systems costing thousands of dollars, and in addition these systems are too cumbersome to be feasible for casual consumer applications. Because if this, the uses of these systems are mainly limited to non-interactive motion capture duties.

2.2 Current consumer-friendly systems

As the feasibility of creating full-body tracking devices that don't require special suits or special room setups, and simultaneously aren't prohibitively expensive for consumer applications has improved, several these kinds of systems have started attracting developers and have started making an impact on the market. Here we will describe several of these systems in regards to full-body tracking capabilities.

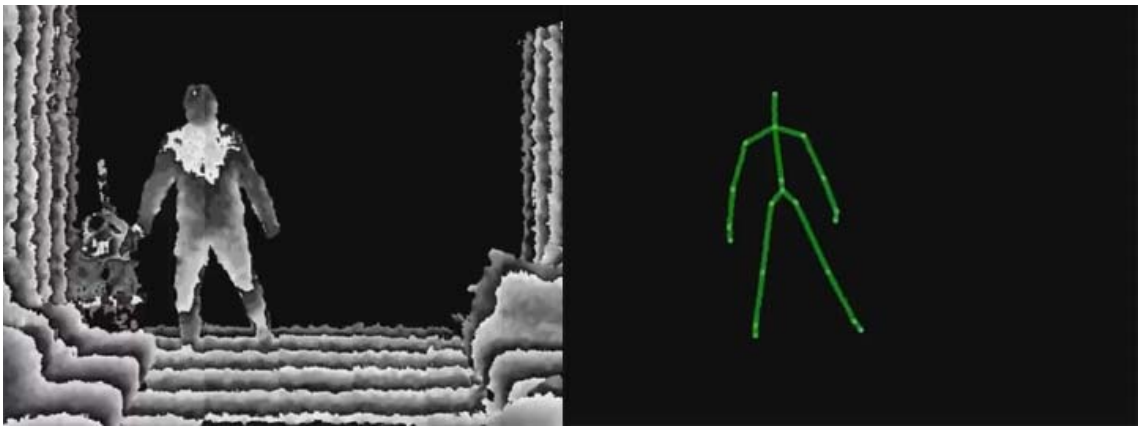
2.2.1 Microsoft Kinect

The Kinect sensor was first introduced in 2010 by Microsoft, who sought to expand the user base of their Xbox 360 console beyond the established gamer niche.

The hardware abilities of the sensor consists of a RGB camera, a depth sensor and a multi-array microphone. The depth sensing ability is made possible by an infrared laser projector combined with a monochrome complementary metal-oxide semiconductor (CMOS) sensor and a technique called Structured Light (Shao 2014, 3).

Initially, the sensor was to only work with the company's then-latest gaming console, the Xbox 360. But as other depth cameras in the market couldn't offer the value brought by the Kinect sensor, it wasn't long until the Kinect was reverse engineered for general-purpose uses by making the device work with normal PC's, and soon after that Microsoft released an official SDK (Software development kit) for developing Windows applications for the Kinect, along with a version of the Kinect designed to be used with Windows and which can be used for commercial applications on the platform. (Paul 2010; Microsoft 2011)

Among other outputs, the Kinect provides skeletal tracking in the form of joint positions in 3D space for 20 individual joints, for up to 2 user's skeletons tracked simultaneously. For up to 6 users total, the Kinect can also show their position, but not their pose. A screen capture showing the depth map and body figure produced by the sensor can be seen below in Picture 2.

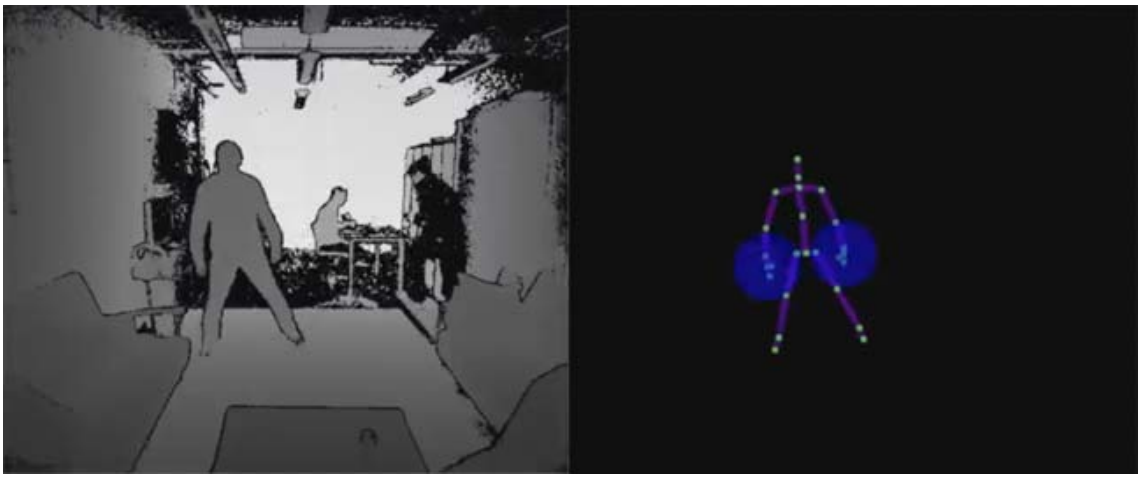


Picture 2. Depth map and skeleton example output from the first-generation Kinect.

2.2.2 Microsoft Kinect for Xbox One / Kinect for Windows v2

The second generation Kinect sensor first became available as a mandatory bundled peripheral for the Xbox One console when it was released in June 2014. Soon after the sensor became available separately and as an early-access "Kinect for windows v2" device for developing with Windows. After the first public release of the Windows SDK for the second-generation Kinect, Microsoft decided to focus on a single device, making it possible to use either the Kinect for Windows v2 or the console version of the device on PC's using a single adapter, and ceased selling the Windows-branded version of the device. The two versions of the sensor are identical, save for a minimal cosmetic alterations. **From now on in this document, this device will mainly be referred to as Kinect 2.** (Fry 2015)

Compared to the first Kinect, the new version is improved in almost in every way. The color camera's resolution was ramped up from 640*480 to 1920*1080 when reading frames at 30 frames per second. The raw resolution of the depth images the device produces has increased, but also the perceived quality of the depth information has improved dramatically, as can be seen in Picture 3. It should be noted though, that the banding effect in the previous picture isn't related to quality of the sensor.



Picture 3. Depth map and skeleton example output from the second-generation Kinect.

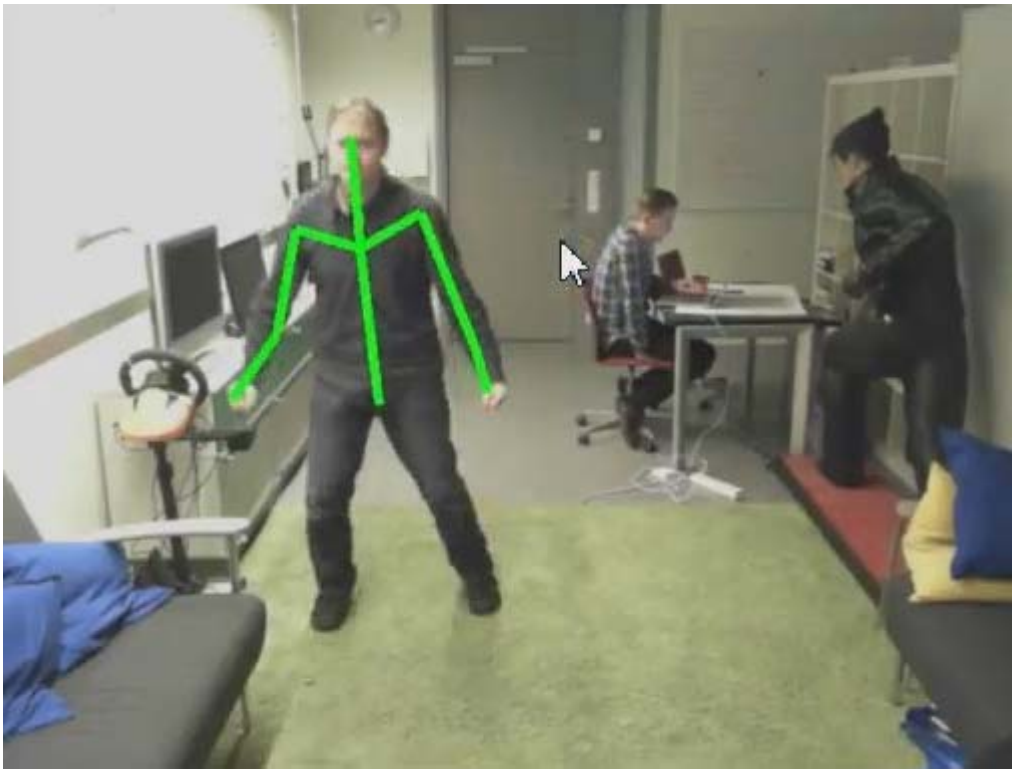
This can at least partially be attributed to the changed technique of obtaining the depth images; Instead of structured light, the v2 sensor calculates the depth pixels by measuring the time it takes for emitted IR light to be reflected from the target back to the sensor, a method called "Time-of-flight". It was also concluded by an experiment carried out by the writer that this also works much better outside, as unlike with the earlier version, the v2 sensor doesn't get blinded on a sunny day, save for a situation where the sun is shining directly at the target and the sensor is facing the same direction as the sunlight is coming from.

With the enhanced depth fidelity, combined with improvements to software processing of the depth image, the skeletal tracking abilities have also been greatly improved. Now the sensor can sense 6 complete human figures, and the amount of tracked joints has increased from 20 to 25. The tracking of the joints is also more stable, so jitter isn't as large as a problem. The field of view of the cameras has also widened, enabling the sensor to be used in smaller spaces and to be able to see more. With the enhanced fidelity, hand states (closed / open / lasso) can also be tracked, along with greatly improved facial tracking. The improved fidelity also allows the application programming interface (API) to provide the roll axis for joints, to some amount of precision. (Microsoft 2015)

2.2.3 Extreme Motion

Developed by an Israeli company called Extreme Reality, Extreme Motion is a software-only solution. The system uses color images provided by normal web cameras, and works with many web cameras out of the box.

The system doesn't provide data for all joints in the body though, only upwards from the pelvis, as can be seen in Picture 4.



Picture 4. Skeleton generated by Extreme Motion overlaid on the input color image.

This isn't too limiting though, as gestures like crouching can easily be detected by tracking the elevation of other joints. As the system only uses color images and not depth, motions like turning your hands towards the sensor can't really be tracked. Tracking the user's relative distance from the camera is possible though, by monitoring changes in apparent size of the user's figure.

Unlike its competitors, the system cannot track multiple bodies at once. It should be noted though that this, along with the missing lower body tracking, would likely be problematic anyway when taking in the account the relatively low field of view of most web cameras, along with physical space limitations in many cases. This point is further validated taking in to account the less demanding nature of the solution, as the user doesn't need to invest in any extra hardware.

In comparison to the latest Kinect sensor, the tracking accuracy is lower, and as mentioned, only mostly works in two dimensions. However, many types of

games with inexact enough controls normally developed for the Kinect are feasible with the system.

In addition to Windows, Extreme motion is also available for Mac, and even works on the mobile platforms Android and iOS.

3 THE DEVELOPED SYSTEM

3.1 Project timeline

The timeline of the project spans from the summer of 2014 to the summer of 2015, during which development was taking place most of the time.

3.1.1 Beginning of the project

The project was begun in the summer of 2014 as a co-operation between Turku University Of Applied Sciences and Serious Games Finland (Now named Goodlife Technology), with funding from the Finnish public funding agency Tekes. The aim of the project initially was to create a general-use system that could be used to track the repetition of different body movements. The system was planned to potentially be a part of Serious Game's software, but to be generic enough so that it could potentially be used as an API for many kinds of applications. The system was named, atleast as a working title, "Clinical Layer". The name comes from the system's planned independence and separation from the actual use case, as the system could be used in many kinds of applications. The hinting towards healthcare wasn't unintentional either.

Because of the previous experience of the team in the tool, and the ease of prototyping, the initial development environment was chosen to be the Unity game engine. The idea was that after a way for the system to work was found, it could be re-implemented without unnecessary dependencies / in a more appropriate language without terrible effort, and possibly condensed to a single add-on library.

As the second version of the Kinect sensor Windows (Kinect for Windows v2) was not yet available when the project began, the development was started using the original Kinect 1 sensor. This was done using a third-party wrapper library, as it was the best choice available that made the data from Kinect availa-

ble inside Unity. Utilizing some of the scripts in the wrapper package and altering them to fit the project's needs, and programming some new functionality, the first simple version of a repetition tracker was prototyped.

The algorithm used was a simple one, where a series of poses would be recorded from the Kinect, recording them 30 per second, basically performing a motion capture. Then to track the repeating of the recorded move, the pointing direction(in relation to parent, local) of chosen joints(arm, forearm, etc) in the body was compared between the corresponding joints of the recording and live feed from the Kinect. If the calculated angle between the live feed and the first recorded frame was small enough (like 30 degrees), the recording was advanced to the next frame. With this logic, the progress in the move could be roughly tracked.

3.1.2 Moving to Kinect 2

Not too long after the start of the project, the Kinect 2 for Windows came available and a prototype device was obtained. After this the test project has been completely rewritten. This is partially because the interface for obtaining data from the Kinect 2 differs greatly to the Kinect 1, as with Kinect 1 the best way to get data from the device to Unity was to use a third-party wrapper, but with Kinect 2 there's an official wrapper that imitates the C# Kinect 2 API. The form of the data that is obtained from the Kinect API was changed also, as the Kinect 2 API provides data about joint orientations, so that was used instead of world-space positions of the joints. The final form of the analyzed data that is fed to the repetition tracking system remains similar though, as the pointing directions of the joints are converted to normalized direction vectors in every version of the repetition tracking algorithm.

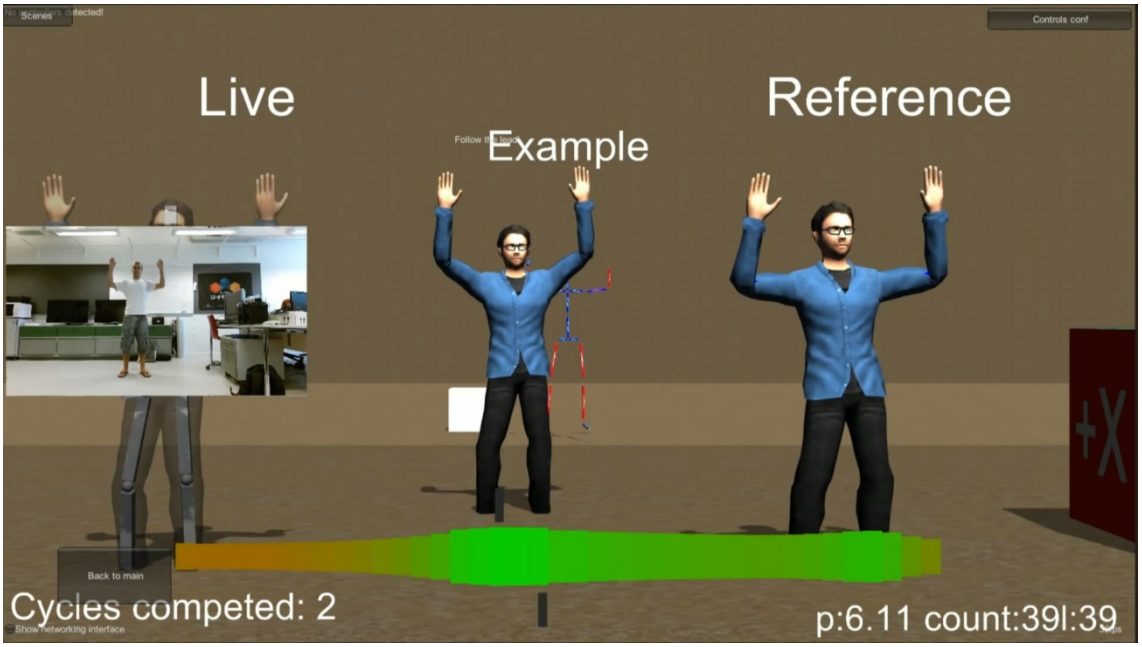
With the final hardware obtained, the first hurdle was to understand how the proper data could be obtained from the official API of the Kinect 2. This proved to be more troublesome than one would expect, as the API was still in closed beta state, and even though there was an official wrapper for Unity, it was even

more incomplete than the API itself. Even a small bug fix to the wrapper itself was required to even access the joint orientation data, which would be the main data source for the project. Documentation was also nearly non-existent and sometimes even wrong, so even being able to simply drive a 3D avatar took some time to figure out and implement. The orientations of the Kinect skeleton in a neutral stance (T-pose) were not documented and were not compatible with normal character rigs, so a system that fits a rig to work with the data from Kinect was made, enabling "avateering" with multiple different kinds of rigs. What "avateering" means is basically driving an "avatar's" (A 3D character representing the user) animations by one's own motions, here meaning successfully being able to animate several characters on the screen simply by going in front of the sensor.

After getting the necessary data to an relatively easily usable state in Unity, the same kind of functionality that existed in the Kinect 1 project was developed, including a move recorder and a repetition tracker with similar algorithm as the Kinect 1 project had. The repetition tracking seemed like it could be improved though, and work on a more accurate and robust repetition tracking method was started.

3.1.3 Onwards

After some time, the basic idea of a better way to track the repetition of moves emerged. Simultaneously work on improving the base framework for the system was taking place, including improving the provider/utilizer model of passing animation frames around, making a basic user interface for recording and cutting the pre-defined moves, preview functionality to the list of predefined moves, etc. With better tools and visualizations to control and see what’s going on, the algorithm was could be better tested and was optimized further. A screenshot from a video of this earlier version can be seen below (Picture 5), and the video in the reference.



Picture 5. Screenshot of an early version of the system (Lindblad 2014a)

Along the length of the project, development was ongoing continuously. Several additional features and additions were developed, including the ability to import input animations from other software, a small exercise game for testing and demonstration purposes, further improved content creation interface bringing together the abilities of the system (recording moves, cutting them, importing them from another source, etc), a bare-bones example to ease the potential integration of the system to other software, and a network-streaming demonstration application that was showcased at a physiotherapist exhibition. These cases, along with the specifications of the internals of the system will be examined in the following chapters.

3.2 Internal logic

As the system consists of many parts and much of control is required for recording the reference moves, post-processing them, testing them with the algorithm, and continuously visualizing the process, the internal structure of the application is somewhat complex. **Note that in the following sub-chapters class and method/function names of the system's (Clinical Layer's) C# source code**

will be used in identifying some components of the system to avoid misunderstandings.

3.2.1 Real-time input data handling

The Kinect 2 sensor and it's API provide access to many kinds of feeds, including the raw depth image, color image, skeleton data, audio from a microphone array, and body index images which color the silhouettes of different bodies detected. But the only feed that Clinical Layer utilizes is the skeleton data (Body frames). On the Kinect 2 API side, the skeleton data is automatically constructed by processing the depth image with advanced image processing algorithms.

The data is first brought available to Unity C# side by using the official Unity wrapper for Kinect 2, where the Unity application polls for new frames in Update function, in the BodySourceManager class. From BodySourceManager the array of bodies (always 6 long) is passed to KinectBodyOrientationSensor. The KinectBodyOrientationSensor picks a body from the array and then reads and converts its position and joint orientations to Unity-specific Vector3 and Quaternion types so they can be processed easier. Next the rotations are converted so that they are relative to T-pose, in which they are at zero. The last processing step is taking in to account the Kinect sensor's rotation offset and height from ground, so that the Y-axis is always pointing in the same direction as real-world gravity, instead of the sensor's upwards facing side, as the sensor might be tilted. After this, "Utilizers" which are subscribed to the KinectBodyOrientationSensor "Provider" are notified that there's a new frame available, which will be further explained below.

3.2.2 BodyOrientationFrames

The most important kind of data the application passes around is BodyOrientationFrames. These contain the world position of the root of a skele-

ton, and the pose of the skeleton as local rotations of a specific kind of hierarchical skeleton rig.

The root's position is saved as a Vector3, and the rotations as an array of Quaternions. The Quaternion array is 25 items long and the order of joint rotations contained conform to the Kinect 2 API's Kinect.JointType enumeration, though not all of the slots are actually used. BodyOrientationFrame data is used for visually animating characters and also for tracking the repetition of moves.

Additionally each BodyOrientationFrame also has an "isTracking" boolean variable indicating if the body was tracked during this frame (Usually only one frame with this set to true is sent in sequence before pausing), and an array of boolean variables indicating which joints should not followed (ignored) by tracking, though this is currently not in frame-specific use, instead being animation-specific.

A modifier indicating the serialization quality of the frame is also contained. A BodyOrientationFrame can be serialized to a byte array with the ToBytes() method, and constructed from a byte array by using a constructor with a byte array input parameter.

3.2.3 Animations

In the system, animations are stored in a custom "ClinicalMove" type. A ClinicalMove is basically just a collection on BodyOrientationFrames forming an animation. In addition to an array of BodyOrientationFrames, a ClinicalMove also contains a name for the move(string), a description(string), the frame rate of the move, a value indicating the angle comparison strictness used by repetition tracking of the move (angleComparisonStrictness), a modifier indicating the serialization quality of the move (clipQuality), and integers for the minor and major versions of the file format.

ClinicalMoves can be serialized to JSON strings using the ToJSON() method, and created from JSON by using the CreateFromJSON() function. JSON stands

for JavaScript Object Notation, and is a human-readable textual data serialization format.

3.2.4 Provider / Utilizer model

Because there's many components in the system that pass BodyOrientationFrames around in similar ways, there's a base class for components that provide frames (BodyOrientationProvider), for example the KinectBodyOrientationSensor, and a base class for components that do something with the frames (BodyOrientationUtilizer), for example MoveRecorder.

The way the data is passed around is so that first, a BodyOrientationUtilizer needs to subscribe to a BodyOrientationProvider, then the provider will notify the utilizer when a new frame is available. The utilizer may then get the data from the provider. **The most important Providers and Utilizers will be listed below.**

3.2.5 KinectBodyOrientationProvider (Provider)

This is the the original source for a Kinect 2 feed, as pointed out earlier. Either picks any body that it sees is active (might change abruptly if the sensor sees more than 1 person), or a person from a specific array index in the bodies array. Developer note: For a more controlled obtaining of bodies, it is recommended to use the KinectActiveAreaBodySelectorSource component with BodyPositionMarkers, like in the "Remote Proto" app, that will be mentioned in a latter chapter.

3.2.6 MoveRepeater (Provider)

This component plays back previously saved ClinicalMove animations and provides a feed of frames from them. Can be used to play the moves at their natural speed (using their set frame rate) or have the frame changes being set ex-

ternally, for example when the animation cutting timeline slider is dragged, or showing the pose in the played animation that most resembles a live feed from kinect (using MoveRepeatTracker).

3.2.7 BodyFrameReceiver (Provider)

Receives frames sent over the network by BodyFrameBroadcaster, and makes them available to Utilizers. This component, along with BodyFrameBroadcaster, is very experimental and not error-proof.

3.2.8 ModelControllerFollow (Utilizer)

This component will use the provided frames to drive a rig for a 3D character (Playing back animation). Developer note: For joints (Transforms in Unity) that need to be driven, the TrackedJoint script should be attached, and for child joints of those joints, the child joint should be assigned. The RigAdjusterForKinect script may need to be used if the joint’s rotations aren’t zeroed at T-pose, otherwise the rig won’t animate properly.

3.2.9 MoveRecorder (Utilizer)

MoveRecorder is used simply for recording BodyOrientationFrames from a BodyOrientationProvider and assembling them to a ClinicalMove.

3.2.10 MoveRepeatTracker (Utilizer)

This is the “main” component of the system, as this provides repetition tracking information (user's progress in executing the predetermined move) from a feed of BodyOrientationFrames that is fed to it. Preferrably a live feed from a KinectBodyOrientationSensor is supplied.

3.2.11 BodyFrameBroadcaster

Sends BodyOrientationFrames over the network to a BodyFrameReceiver. Developer note: Quite simple of a component, depends on Unity's legacy networking system and having a connection, and a NetworkView (an Unity-specific component) target. The target is the GameObject (an Unity-specific concept) with the BodyFrameReceiver script.

3.2.12 Combining Providers and Utilizers

The Providers and Utilizers can be used in conjunction to implement many different kinds of functionalities. For example, a MoveRepeater and a ModelControllerFollow can be connected together to create a simple animation player. Alternatively, combining a KinectBodyOrientationProvider to a MoveRecorder can be used to record the skeleton output from the sensor.

To set up a more meaningful system in a graphical application, the KinectBodyOrientationProvider feed can be attached to both a ModelControllerFollow and a MoveRepeatTracker to give greater feedback to the user while they use the application, as they can see themselves on the screen while the system also tracks if they are progressing with an exercise. To clarify to the user what they are supposed to be trying to do, a MoveRepeater+ModelControllerFollow combo could also be added to show the reference move side-by side with the user's avatar (The virtual representation of the user on the screen). Optionally the networking components BodyFrameBroadcaster and BodyFrameReceiver could also be added for remote observation by a physiotherapist for example, or even to add remote players to a game.

3.2.13 BodyOrientationFrame serialization

BodyOrientationFrames can be serialized to sequences of bytes of predictable lengths, and back. Efficient use of bytes is achieved by manually writing all relevant data to to a bytestream using the standard library class BinaryWriter for writing, and the class BinaryReader for reading. It is critical to know the exact order in which the data making up a BodyOrientationFrame is encoded to the bytestream as the binary format doesn't really have any kind of headers for data. The order can be determined from the BodyOrientationFrame.cs source file.

The way the main payload (the array of body joint orientations) is encoded to bytes differs depending on the quality setting used. There is 3 different quality settings, each with their own quality / size characteristics. The settings are:

- **High**
  The rotations are encoded as they are natively stored: as 4 32-bit floating-point numbers, making up all the components of the Quaternion: w,x,y and z.
- **Medium**
  For this quality setting, the Quaternion is converted to an Euler angles representation, and the 3 components (x,y,z) are stored as 16-bit integers.
- **Low**
  For the lowest quality setting, the Quaternion is again converted to an Euler angles representation, and the 3 components are stored as single bytes. The quality degradation may be quite visible as there's approximately 1.41 degrees of maximum error.

As the individual frames take less than a kilobyte even on the highest quality setting, the frames can be streamed over a network even if the bandwidth is poor, as long as the connection quality is reasonable.

3.2.14 ClinicalMove serialization

ClinicalMove animations can be serialized to human-readable JSON strings, as seen in Picture 6. The smaller parts (Name of the move, description, framerate, angle comparison strictness variable, quality of the frames in the clip, major and minor versions of the file format) are simply saved as their textual representations. The main bulk of the data, the array of BodyOrientationFrames, however, is first serialized to bytes using the previously described method, byte sequences for individual frames being placed one after another, and then the combined byte sequence converted to base64-encoded text. This decreases the size efficiency by a factor (33%), but enabling the files to be human-readable and easier to process is worth it, noting that the animation data never does take that much space anyway (approximately 17 kilobytes a second at highest quality).


```
{
    "lenghtInFrames" : 61,
    "timeLenght"     : 2.0333313941955566,
    "fileFormatVersionMajor" : 1,
    "fileFormatVersionMinor" : 1,
    "moveName"               : "3. Pull resistance band from side",
    "moveNotes"              : "Resistance band connected to somethin
    "unLoadedLenghtInFrames" : 61,
    "frameRate"              : 30.000028610229492,
    "angleComparingStrictness" : 30.0,
    "clipQuality"              : 2,
    "orientationFrameDataString" :
    "AAI3ENc9SK+YvR6Zz7//+f8BAAAAAAAAAAAAAAAAAACAPyxWzDtUkty9vqJxPFl6
    b+jwu2+E54eP86HkrxYkQa+1TidvOOwfT9+RrM8SY5wvBpBPL7Xi3s/LFbMO1SS3I
    8kdY95tBRvSy1obsCQH4/AAAAAAAAAAAAAAAAAACAPwAAAAAAAAAAAAAAAAAgD8A
```


Picture 6. Partially shown contents of a ClinicalMove file.

3.3 Content creation methods

For the movement repetition to work, the system obviously needs reference moves to compare against. These need to be in the ClinicalMove format for the system to be able to understand them. The system supports two ways of bringing new animations to the system, described below.

3.3.1 Direct recording

The easiest way to produce new movements is to simply record them directly from the Kinect's skeleton input. This was good enough to produce good material for testing the system, and there would be no compatibility problems when comparing the rotations of the joints in the move repetition algorithm, as the source for the poses was the same, the Kinect sensor. The ease of this method could also potentially unlock new opportunities for content creation, as for example in physiotherapist use, the recording could be created during an appointment, possibly even by the patient, which could give useful feedback the the physiotherapist and the patient.

3.3.2 Animation importing

Even with all the good features the direct recording has, it still had some drawbacks. One of these was the restless nature of the unfiltered Kinect input, as the joints would jump around and rotate around their twist axes, sometimes to produce a quite frightening visage.

To remedy this, it was established that it's necessary to find a way to bring animations to the application from external sources. Common export formats from 3D software packages seemed interesting, but the problem was that even though these could be imported to Unity while in its development environment, this would not be possible for built executables that Unity outputs (let alone if the system was to be made not dependent on Unity). There also didn't seem to

be easily accessible and usable add-on libraries that could do this task. The formats themselves, for example the formats FBX (Filmbox) or COLLADA (COLLAborative Design Activity), also were so complex that writing an importer for any of these could be a too large task.

3.3.2.1 BVH animation format importer

After some research, a somewhat common and simple enough format was found: BioVision Hierarchy (BVH), with the file extension .bvh . BioVision Hiearchy is a format that was originally developed by a now-defunct motion capture company BioVision to serve the function of providing motion capture data to their customers (Thingvold, 1999). BVH seemed perfect for the task, as the format barely has any extra features, beside storing animation. Many existing programs that are used for creating animation also had support for this format. As the structure of the BVH file format is quite simple, decision was made to develop an importer that could be used to read the contents of a BVH file and produce a compatible ClinicalMove animation from that data.

It still took a while to write the code to read through a BVH file, and it wasn't as trouble-free as it seemed at start. Especially problems with reading the rotations for different rotation axes in the right order, and the fact that some software handle rotations differently (Right-handed versus Left-handed coordinate systems for example) caused headaches. Also making animations compatible proved a challenge, as joints like the left elbow for example had different names depending on the animation rig, and at worst cases, the hierarchy of the body joints was different.

In the end, most of these problems were solved, and the importer was in working order. Even though it wasn't necessary for the importing process, support for reading the joint hierarchy and visualizing the original animation, even if it was incompatible, was developed for debugging purposes.

3.4 Content creation interface

To facilitate developing the different parts of the system, developing and testing the repetition tracking algorithm, and readily testing it with different kinds of movements that could be easily fed to the system, it was needed to develop an user interface alongside the system.

The initial user interface was emergent, developed with the very first parts of the system. It served its purpose, but eventually caused too much friction, especially with the user interface code not properly being separated from rest of the system. The interface used Unity's legacy "IMGUI" system, and its implementation wasn't planned ahead when it was developed, so when it was needed to add features to the system and test them, hacking away at the old interface didn't feel right, especially with Unity's long-awaited new UI system having been released.

So decision was made to make a new user interface for the system. This would not only serve for debugging the system while developing it, but also for being the tool for producing content (moves to repeat) for the system. This could then be used when the system was utilized in another application as a control mechanism, for example.

With the main needed features of the system now being known, and having noted where there were shortcomings with the older interface, most of the features that the interface would end up having were planned from the start, and are described below.

3.4.1 New move creation

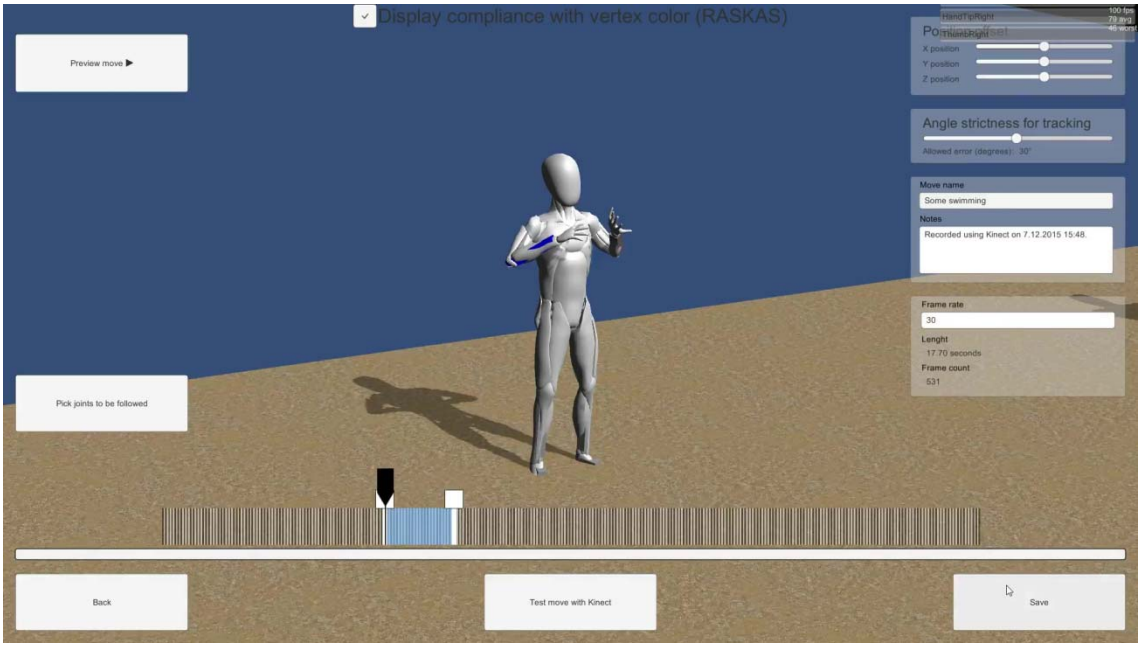
Functionalities were created to the user interface for the two ways to introduce new animation assets to the system:

- Direct recording
  A simple recording interface, with a 3D avatar displaying the kinect's skeleton feed. When entered, the system first waits for a body to be detected by the Kinect. When the Kinect properly sees a body (Not just partially, all joints need to be non-inferred), a 3-second countdown for the start of the recording begins. After the countdown, the actual recording starts. The recording continues until the "Stop recording" button is pressed. After this, the "Properties" view is entered with the just captured raw recording having been loaded in.

- Animation importing
  For the BVH animation file import option, the user is presented with a file picker dialog, asking for a file with a .bvh extension. When a proper file is selected, the file is given to the importer and if the conversion is successful, the BVH-animation that is now converted to a ClinicalMove is loaded to the "Properties" view, which is shown.

3.4.2 Managing existing moves (Properties view)

The Properties view can be accessed by either opening an existing ClinicalMove file with the file picker, or by going through the direct recorder or the BVH file import option, which will, after doing their work, pass the yet unsaved ClinicalMove to this view. The Properties view allows the user to, view, modify, and save the ClinicalMoves in several ways, described below. The Testing mode can also be accessed from this view. A screenshot from a video that shows parts of the interface be seen below (Picture 7), and the video in the reference. The screenshot shows the Properties view.



Picture 7. Screenshot of the properties view. (Lindblad 2015)

3.4.3 The timeline

In the bottom of the screen, there is a timeline. On it there is a black pointer, which signifies the current shown point in the animation. If the animation is playing, this can be seen moving. This can also be dragged with the mouse, which will scrub through the animation.

There is also pointers signifying the start and end points of the animation. These can be moved, which in part crops the part of the animation that will be taken in to account, effectively performing a cut, similar to using a video editing tool. This cutting functionality is very useful for deleting unnecessary frames from the start and end of the animation and making seamlessy repeating, loopable movements. Few moves could be imported without moving the markers at least a small amount. The functionality can also be used for extracting multiple smaller movements from a long recording, saving them to separate files.

3.4.4 Setting and metadata fields

- Position offset
  The position offset setting allows the user to correct the location of the character in the animation. This can be useful if for some reason the character is under the ground or levitating in the air for example. The offset won't itself be saved, this setting will just modify the animation frames by applying the position offset to all of them, and the sliders will always be at zero position when the properties screen is entered again.

- Angle strictness for tracking
  This will modify the setting that the move repetition tracking will use for this specific move. This may sometimes need to be lowered or increased, if there is a relatively small amount of movement for example, but this might cause other problems.

- Move name
  Simply a name for the move. This may be different from the filename, although this is not recommended. An yet unsaved move will be saved with this as the filename, although sanitized, by default.

- Move notes
  This is simply a field where extra information about the move can be put. By default the system will state the source of the animation here (recorded with Kinect or imported from a BVH file).

- Frame rate
  This field will show the frame rate setting of the animation. This won't normally be needed to be modified, as the Kinect recordings will always have a frame rate of 30 and the frame rate of imported BVH animations will be set according to the setting in the BVH file.

- Length and frame count fields
  These fields are uneditable, and will show the count of frames in the animation, and the length of the animation in seconds, calculated from the count of frames and the frame rate.

3.4.5 Choosing ignored joints

By pressing a button in the properties view, the user can go to a mode where they can choose the body joints that will be taken in to account in the repetition tracking. The 3D character will go in to a T-pose and joints that are selected to be ignored are shown grey and those that aren't are shown colored. The list of ignored joints and not ignored joints will also be shown on a list for extra clarification. These states can be toggled either by clicking the joints themselves, or by using the automatic mode.

In the automatic mode, the statuses of the joints will be decided automatically, depending on if they rotated over a certain degree threshold from their starting positions at any point during the move. This threshold can be changed from a slider in the view, and the effects can be seen instantly.

3.4.6 Saving

When the desired changes to the animation have been made, the move should be saved. Pressing the save button will bring up the file dialog, now prompting the user to choose a filename and a folder to place the move to. If the move was opened from an existing file, the same filename will be suggested, otherwise the name of the move will be suggested. The user can also change the animation quality of the move in a dropdown menu, which will cause the system to serialize the frames using a different quality setting when the save button in the file dialog is pressed.

3.4.7 Testing mode

The Testing mode can be accessed from the properties view after opening a move, by pressing the “Test move with Kinect” button. In the testing mode, two 3D characters will be shown. The first one is displaying the raw Kinect feed, and the second one is displaying the reference animation, playing independently with a normal speed. The user will also be shown a visualization that shows how the repetition tracking algorithm is working. This visualization will present the weights that all the frames in the animation are given, and also showing the current position in the move, and a repetition counter.

3.5 Repetition tracking algorithm

The heart of the system is the process which evaluates if the user is doing the move the system is tracking or not. By the simplest definition, when the system is started it is first given a pre-recorded reference move (a recording, ClinicalMove) and then the system starts taking in a stream of poses (preferably the live feed from the Kinect), and outputs the user's progress in executing the move that is being tracked.

If one would be to describe the core logic of the process the system repeats tens of times a second in a single sentence, it would go:

Find a point in the reference animation that resembles the input pose the most, while preferring selecting a point close to the point that was selected in the previous iteration.

In practice, the system works by continuously comparing all the individual poses (skeleton animation frames) it's being fed at the rate of 30 poses a second to all the individual poses in the reference move. During a single iteration, the system takes in to account all the joints that have been pre-selected as the joints relevant to the move and makes comparisons to the frames of the reference animation. The system attempts to select the point in time in the reference animation that the user's motions correspond with the most, by taking in to account several factors including angular differences, the frame's positions in time and velocities. During the comparison, it's important to take in to account that some joints might be in identical poses during multiple different points in the animation. Results of the previous iterations are also taken in to account. The "winning" pose is selected by scoring the different options according to several factors, including those mentioned, rewarding some traits and penalizing others. An assumption is made and taken in to account that the user is progressing forwards in time in executing the move to smoothen the process, although the system does still retain some ability to keep the tracking contiguous if the user is executing the move in a reversed manner.

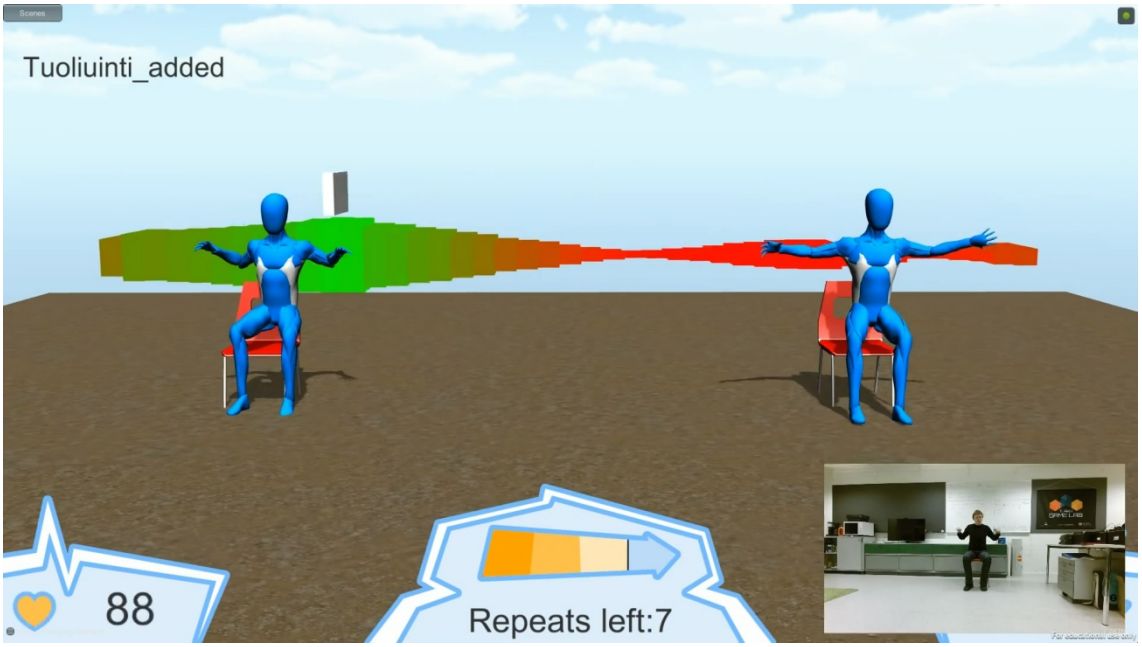
3.6 Developed test and demonstration applications

As mentioned earlier, for testing and demonstrating purposes a few additional small applications were created, described below.

3.6.1 Chair exercising game (Tuolijumppa)

Although an exercise game with the simple objective of repeating sets of pre-defined exercises isn't exactly the most original of game ideas, for the purpose of quickly validating the performance of the repeat recognition system in a realistic setting it is great. Hereby this was the idea that was chosen for the first demo app. There was a small specialty in the game idea though, as it was planned that the exercise routine would be focused on utilizing a normal chair for an accessory.

Some custom logic, like a Kinect-driven cursor and an unique way of choosing the workout to execute, were developed as well. There was only one complete move set was defined and recorded, but this was done with the help of a physiotherapist. The screenshot below (Picture 8) is taken from a video of the application. The video can be seen from the reference.



Picture 8. Chair game prototype. (Lindblad 2014b)

As can be seen in the video, the user is guided to do the correct movement by showing the user's live animated character on the left, and an identical character, which is repeating the move correctly, on the right. The user can repeat exercises at their own pace, and when a predefined amount of repeats is executed, the move changes. When all the different moves are completed, the game ends.

3.6.2 Remote streaming prototype (Remote Proto)

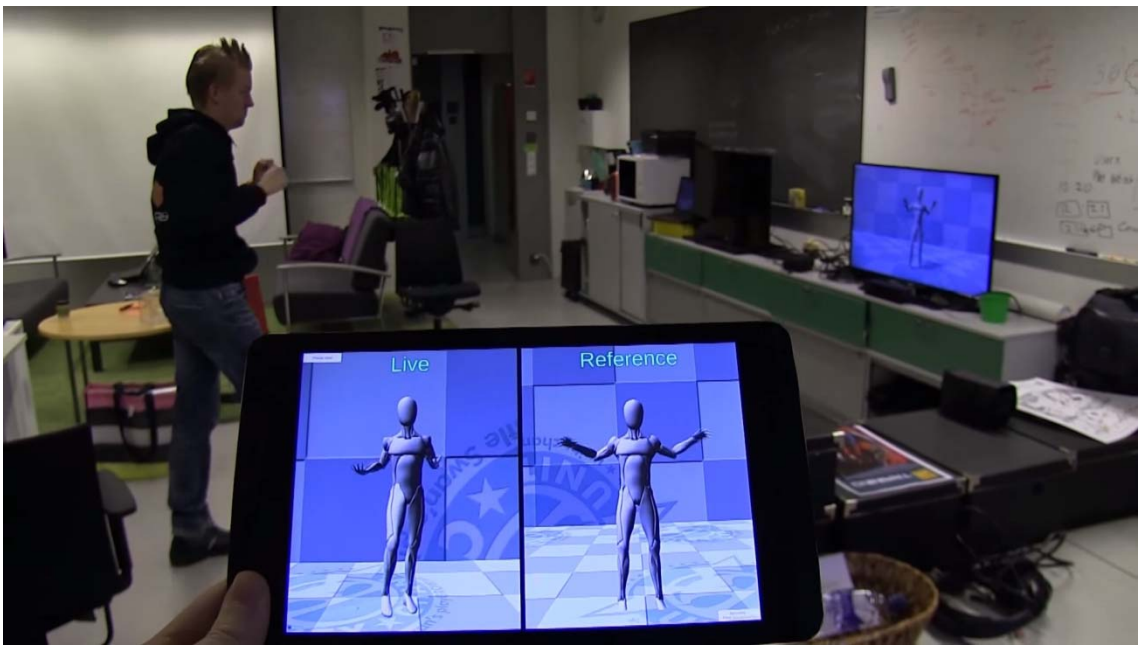
The idea of this prototype application was to demonstrate remote viewing of the Kinect's skeleton feed on a tablet device by a would-be-physiotherapist, while a patient is repeating physiotherapist-chosen movements using their device, connected to a TV for example. This application was developed mainly to be demonstrated at a booth during the World Confederation for Physical Therapy Congress 2015 exhibition.

First thing the application requires is connecting the tablet and the TV to each other. In the prototype this was achieved by having the tablet and the TV in the

same local network, and reading the TV's IP address with the tablet's camera using a QR code. After a connection is established, the " physiotherapist " is in control of the session, and can choose an exercise for the "patient" to repeat. This exercise (a ClinicalMove) would then be transmitted to the patient's device, and the patient would be asked to begin executing it.

Now as the physiotherapist can see what the user is doing, they can verify that the patient is doing the move correctly, while in addition the automatic progress tracking algorithm's output is being shown on both devices. The physiotherapist can also change the tracked move at any time, and see the reference move, in split-screen view with the actual live stream from the patient.

What the patient sees depends on if they are repeating the asked move or not. If the repeat recognition algorithm sees that the user isn't executing the move, the character on the patient's screen turns transparent and starts demonstrating the move by playing the reference move. If the user is recognized executing the move, the character turns solid and starts playing the reference move so that it's in sync with the patient's pace. The screenshot below (Picture 9) shows an early version of the application in action.



Picture 9. A screenshot of an early version of the "Remote Proto" demo.

This application used much the developed functionality, with the moves first being prepared by recording them with Kinect and then importing hand-animated versions of them with the BVH importing functionality. Unity's built-in multiplayer networking functionality was used to facilitate the client-server relationship and the experimental BodyFrameBroadcaster and BodyFrameReceiver components for sending and receiving the raw reference move data (ClinicalMove JSON) and the raw live animation frames (BodyOrientationFrame byte arrays) between the devices. In addition to the basic application, work was put to make the file browsing view to play previews of the ClinicalMove files in the open folder, not by using pre-generated image data but by actually reading the files in a background thread and rendering the thumbnails in real-time.

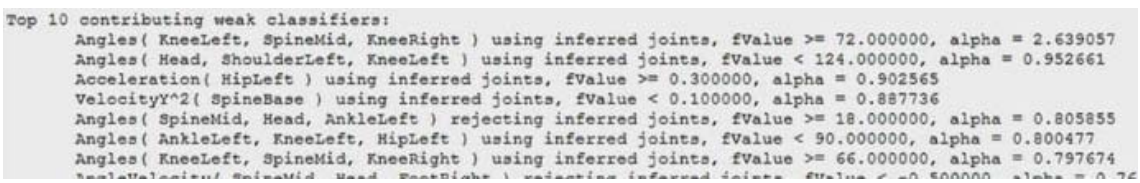
From feedback received, the demo did it's duty and functioned throughout the exhibition, and received positive feedback. The largest problem was that one of the tablet devices used for demonstrating couldn't stay on continuously even while connected to an outlet, as the power draw was too high. This could have potentially been avoided with performance optimizations and frame rate limitations.

4 VISUAL GESTURE BUILDER

Visual Gesture Builder (VGB) is a system developed by Microsoft for the second-generation Kinect. VGB is described as an "Data-driven solution for gesture detection", and is mentioned in this work because of its similarity to Clinical Layer, and the great relevance to the research problem of this thesis. (Microsoft, 2015)

The main outline of VGB as a solution is very similar to Clinical Layer. VGB allows the user to make recordings from the Kinect's output, then process and configure them manually for use with gesture recognition, test them, and finally use them in a Kinect-enabled application to track the completion of moves/gestures.

VGB Consists of several pieces of support software. To record the raw clips, Kinect Studio is used. One might then want to use KSConvert to do a file conversion. Next one needs to open the Visual Gesture Builder itself and create a project if one doesn't yet exist, and go through a wizard setting particular settings for the gesture, for example, if the move is symmetrical or not. VGB benefits from using multiple recordings as input information, so now user needs to go through a manual tagging process where they basically go through the clips and tell the system when the user is executing the gesture. After this the system goes through the training phase, during which it uses machine learning algorithms to find out the most optimal way of tracking the execution of the move. An example partial output of the training process is visible in Picture 10. This process may take a long amount of time, depending on the amount of clips evaluated. (Lower & Hillier 2014)



Picture 10. Example partial output from VGB's training process. (Lower & Hillier 2014)

After this, the user can test the moves by launching the live preview app. When the time comes to use the gesture recognition data within one's own application, they need to use the VGB's runtime library, which is contained in the Kinect's API, to read the user-created gesture database, run the tracking, and then use the output of the gesture tracker to drive their application.

The reason a more thorough comparison with Clinical Layer with testing wasn't completed is because of the late point in the project when VGB came available with good documentation to use in the same environment as Clinical Layer, combined with having no extra time to conduct a proper, in-depth test during the hurried development phase in the project, among while working on another project utilizing motion controls.

Doing an educated guess, one could expect to get better results with VGB in comparison to Clinical Layer, when taking in to account that VGB can intelligently choose to track the best classifiers to conduct it's tracking. It should be noted though that it does have some disadvantage in comparison to Clinical Layer. Because of the overall complexity of the VGB's content creation process, producing trackable gestures/moves is likely to take longer in both manual and automatic processing steps, require the use of multiple different pieces of software, multiple different custom file formats, and possibly the use of cumbersome command-line tools. In comparison, Clinical Layer can quickly produce all the data needed to track a move, including testing it, without leaving the single content creation interface. In addition, Clinical Layer is subjectively easier to use from application code.

Some additional differences between VGB and Clinical Layer are that Clinical Layer is designed for tracking the "analog" progress variable in looped moves, while VGB can do the same (continuous gestures, using the "RFRProgress" machine learning algorithm) without expecting them to be looped, but also discrete gestures (for example, seated or not seated, done using the "AdaBoostTrigger" machine learning algorithm). (Microsoft 2015)

5 CONCLUSION

At the end of the first development phase, in the summer of 2015, the system was functional, and fitted the definition that was laid out at the start of the project, so it could be stated that the project was successful. The performance of the system was evaluated with the two test/demonstration projects, the chair exercise game and the exhibition demo, and both got positive feedback.

A package purposely built for integrating the system to other projects utilizing the Unity engine was also created, including a bare-bones simple example app, which makes utilizing the system in future applications relatively easy. Some additional documentation not included in this thesis was also produced.

It could still be said that the system's performance isn't as it good as it could be, as it does have trouble with some kinds of moves, and there's still some other problems present, for example not all the moves brought to the system the importing functionality can be used for tracking. The repetition tracking algorithm could also be further developed and refined, for example to take in to account angular velocities of joints, and by conducting more structured testing of different move sets and taking the learned results in to account.

If one was searching for a solution for bringing continuous tracking of the repeating of arbitrary moves to an application, Clinical Layer could possibly fit the bill. But they should evaluate if either Clinical Layer or the previously mentioned Visual Gesture Builder works better in their project.

REFERENCES

Bredow, R; Hastings, A; Schaub, D; Kramer, D, Engle, R. 2005. From Mocap to Movie: The Polar Express. USA: Sony Pictures Imageworks. Referenced 5.12.2015. http://library.imageworks.com/pdfs/imageworks-library-from-Mocap-to-Movie-The-Polar-Express.pdf

Gutierrez, M; Vexo, F; Thalmann, D. 2008. Stepping into Virtual Reality. Germany: Springer

Shao, L. 2014. Computer Vision and Machine Learning with RGB-D Sensors. Germany: Springer

Paul, I. 2011. Kinect Hacked To Work On PCs. Referenced 3.12.2015. http://www.pcworld.com/article/210494/Kinect_Hacked_To_Work_On_PCs.html

Meisner, J. 2011. Kinect for Windows SDK – It's Here!. Referenced 6.12.2015. http://blogs.microsoft.com/blog/2011/06/16/kinect-for-windows-sdk-its-here/

Fry, M. 2015. Microsoft to consolidate the Kinect for Windows experience around a single sensor. Referenced 3.12.2015. http://blogs.msdn.com/b/kinectforwindows/archive/2015/04/02/microsoft-to-consolidate-the-kinect-for-windows-experience-around-a-single-sensor.aspx

Microsoft 2015. Kinect hardware. Referenced 3.12.2015. https://dev.windows.com/en-us/kinect/hardware

Lindblad, M. 2014a. Clinical Layer Prototype (recording & playing). Referenced 8.2.2016. https://www.youtube.com/watch?v=bGzZdBcwU0A

Thingvold, J. 1999. Biovision BVH. Referenced 6.12.2015. http://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html

Lindblad, M. 2015. Clinical Layer content creation interface demo. Referenced 8.12.2016. https://www.youtube.com/watch?v=58Aw5OSQS5o

Lindblad, M. 2014b. Tuolijumppa. Referenced 8.2.2016. https://www.youtube.com/watch?v=G1qeMTqbI0A

Microsoft 2014. Visual Gesture Builder: A Data-Driven Solution to Gesture Detection. Referenced 6.12.2015. https://onedrive.live.com/view.aspx?resid=1A0C78068E0550B5!77743&app=WordPdf

Lower, B; Hillier, A. 2014. Custom Gestures End to End with Kinect and Visual Gesture Builder. Referenced 6.12.2015. https://channel9.msdn.com/Blogs/k4wdev/Custom-Gestures-End-to-End-with-Kinect-and-Visual-Gesture-Builder

Microsoft, 2015. Detection Technologies. Referenced 6.12.2015. https://msdn.microsoft.com/en-us/library/dn785523.aspx