FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Assessing the Sensory Functionality of Children using Depth Data

Development of a Reproducible and Stable Test for Measuring the Kinesthesia of Children (with ASD)

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Medieninformatik

eingereicht von

Johannes Hartl
Matrikelnummer 0825347

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Mag. Dr. Hannes Kaufmann
Mitwirkung: Dipl.-Ing. Christian Schönauer

Wien, 02.03.2015

(Unterschrift Verfasser) (Unterschrift Betreuer)

Eidesstattliche Erklärung

Johannes Hartl

Eichenweg 4-10/5/1
2464 Göttlesbrunn

„Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe."

Johannes Hartl, Wien, 02.03.2015

Abstract

Autism spectrum disorder (ASD) is a pervasive developmental disorder that often involves sensory and motor problems. A sense that has not yet been well studied is proprioception, the sense of joint motion and joint position, which is the basis for motor planning and purposeful action. This sense can be tested with a repositioning test where the subjects have to move their limbs until they believe a certain reference angle is reached. Existing systems using exoskeleton robotic arms are limited regarding the measureable parameters and their usability in everyday clinical practice.

In this thesis I will present the design and implementation of a new method to measure proprioceptive functions of children with ASD. An RGB-D sensor is used to deliver the depth data that are transferred to a PC and analyzed with the application I developed. The system will be used for a reproducible and stable test for non-intrusively assessing the kinesthesia of children, where the children only have to stand in front of the sensor without any constraining instruments. Since the application works in 3D space, two rotation axes for each joint are used.

The system was evaluated for reproducibility, stability and accuracy with ten participants. In addition, it was pilot tested with normally developed children for functionality and usability of the system, where the children had to assume different body positions without visual control. The system will be used in a larger study on children with ASD.

Keywords: autism spectrum disorder; proprioception; kinesthesia; depth camera

Kurzfassung

Autismus ist eine tiefgreifende Entwicklungsstörung die häufig sensorische und motorische Probleme mit sich bringt. Ein Sinn der noch nicht ausgiebig erforscht wurde ist die Propriozeption, der Lage- und Bewegungssinn des Körpers, der die Grundlage für die Bewegungsplanung und zielgerichtetes Handeln bildet. Dieser Sinn kann mittels eines Umpositionierungs-Tests getestet werden, wobei das zu testende Kind ihre oder seine Gliedmaßen so weit bewegen muss, bis sie oder er denkt einen Referenz-Winkel erreicht zu haben. Existierende Systeme, die einen Außenskelett-Roboterarm verwenden, sind sowohl bezüglich der messbaren Parameter als auch bezüglich Benutzbarkeit im klinischen Alltag eingeschränkt.

In dieser Arbeit stelle ich das Design und die Umsetzung einer neuen Methode vor die propriozeptiven Funktionen von Kindern mit Autismus zu messen, ohne beeinflussende und unkomfortable Maschinen. Ein Tiefenbild-Sensor liefert die Tiefendaten, welche zu einem PC übertragen werden, wo sie anschließend mit der entwickelten Applikation analysiert werden. Das System wird für den reproduzierbaren und stabilen Test der Kinästhesie von Kindern verwendet, wobei die Kinder lediglich vor dem Sensor stehen, ohne einschränkende andere Instrumente. Da die Applikation im 3D-Raum arbeitet werden für jedes Gelenk jeweils zwei Rotationsachsen gemessen.

Das System wurde hinsichtlich Reproduzierbarkeit, Stabilität und Genauigkeit mit zehn Testpersonen evaluiert. Zusätzlich wurde ein Pilot-Test mit normal entwickelten Kindern durchgeführt, um die Funktionalität und Benutzbarkeit des Systems zu testen. Hierbei mussten die Kinder eine Reihe unterschiedlicher Posen ohne Sichtkontrolle einnehmen. Außerdem wird das System für eine größere Studie mit Kindern mit Autismus verwendet.

Schlagwörter: Autismus; Propriozeption; Kinesthäsie; Tiefenbild-Sensor

Contents

Eidesstattliche Erklärung **2**

Abstract **3**

Kurzfassung **4**

Contents **5**

1 Introduction **7**

1.1 Problem definition 7

1.2 Expected results 8

1.3 Structure of the thesis 9

2 Related Work **10**

2.1 Autism spectrum disorder (ASD) 10

2.2 Skeleton tracking 12

2.3 Analysis of human motion 13

2.4 Hand tracking 15

2.5 Point Clouds 16

2.6 Validity of Kinect 17

3 Comparison methods **19**

3.1 Skeletal joints 19

3.2 Point clouds 20

3.3 Hybrid approaches 21

4 Design and Implementation **23**

4.1 Setup and Configuration 23

4.1.1 The Sensor Hardware – Microsoft Kinect sensor for XBOX 360 23

4.1.2 Framework – OpenNI/NiTE 26

4.1.3 Unity3D, C# 26

4.1.4 Wrappers, Drivers and .DLLs 27

4.2 Prototype 27

4.2.1 The First Prototype 28

4.2.2 The Second Prototype 41

4.2.3 The final prototype 47

5 Evaluation **52**

5.1 Methods 52

5.2 Hypotheses 52

5.3 Experimental Design 52

5.4 Participants 54
5.5 Evaluation of the Skeleton Analysis 54
5.5.1 Validity of the Measurement Algorithm 58
5.6 Evaluation of the Hand Point Cloud Analysis 58
5.7 Evaluation with Children 61
5.8 Discussion of the Results 62

6 Conclusion and Future Work 64

List of Figures 65

List of Tables 67

Acknowledgements 68

Appendix A: Installing all applications necessary to run the prototype 69

Appendix B: Attempts to Improve the Application with PCL and Mesh-Rendering 70

B.1 Including the PCL 70
B.2 Rendering of a Mesh 70

Bibliography 71

1 Introduction

Based on the definition in the 10th version of the International Statistical Classification of Diseases and Related Health Problems (ICD-10), the autism spectrum disorder (ASD) is a pervasive developmental disorder [45] that often involves sensory and motor problems [16]. Proprioception is the sense of joint motion and joint position, induced by special nerve endings located in joints, capsules, muscles, skin etc., as explained by Lee et al. [22]. This sense can be tested with active and passive repositioning tests. During the active repositioning tests the subjects have to move their limbs, e.g. their arms, until they believe the reference angle is reached. By contrast, in the passive repositioning tests their limbs are moved either by machines or therapists, and the subjects have to report, e.g. by pushing a button, when they feel that the reference angle is reached [22].

1.1 Problem definition

As Fuentes et al. [16] describe in their paper, there are technical setups to test the proprioception of autistic children that are similar to the ones explained by Lee et al. in [22]. In their system a child's arm is strapped to an exoskeleton robotic arm that is hidden under a projection screen. As visualized in Figure 1, a schematic representation of the child's arm is projected onto this screen. A dot above the elbow joint and lines above the upper and forearm help and lead the children during the tests. This robotic arm measures the angles of the shoulder and the elbow joints. Because the mechanical construction acts on a shoulder-level horizontal plane, it only measures deviances regarding position in two dimensions or rotation with one degree of freedom (DOF).

Figure 1 - Illustration of the test setup with the projection screen and the schematic representation of the child's arm. [16]

By the use of an exoskeleton robotic arm, the children's arms are strapped onto or fixed in the machine, which is neither comfortable nor does it lead to uninfluenced results. Non-intrusive measurements like approaches using depth cameras will on the one hand raise comfort for the subjects. The children simply have to stand (or move) in

front of a camera, allowing unhindered movements, which may lead to more reliable data than the previously mentioned constraining instruments. On the other hand it will increase the possibilities for measurement, because it will no longer be limited to only one *DOF*, but will support three dimensions by assessing two *DOFs* of each joint. Additionally, low cost systems based on depth cameras instead of expensive equipment like exoskeleton arms are much easier to use.

1.2 Expected results

The goal of the diploma thesis is a standardized and reproducible test for assessing the sensory functionality, the kinesthesia, of children, in order to detect the level of the autism spectrum disorder (ASD). This is realized with a 3D desktop application using a depth camera such as the Microsoft Kinect sensor.

The depth sensor records the area in front of the camera. Based on the depth information the sensor's logic distinguishes the visible users from the background and then the framework calculates the users' skeletal tracking information. With this tracking data the application computes the horizontal and vertical body axes, divides the depth and skeletal data into the two halves of the body and then the shoulders' and elbows' angles of the left and the right half of the body are compared.

This setting enables a measurement of the deviance without strapping the children onto a machine, which may have been uncomfortable for them during the former tests and also took some time.

Furthermore, an analysis of all three dimensions, or two degrees of freedom (DOF) per joint, is possible, which we hypothesize will give us more accurate and more meaningful results. Several different measurement values are implemented and evaluated in order to offer a variety of useful values to the therapists or psychologists working with autistic children.

Our target is to measure the children's proprioception. This sensory functionality can be influenced by visual information, for example when children look at their arms. Therefore, children tracked by the depth sensor have to have their eyes closed or covered, or at least have to look straight into the camera and not towards their arms.

The first question that already will be answered during the analysis of related work in chapter 2 is: Which measurement values provide a reliable and stable assessment for our special case, when posture depth information and skeletal information are available?

The other questions that will be answered after the evaluation of the developed prototype application in chapter 5 are: Is the measurement of the skeleton and hand point cloud stable and reproducible? Are all test persons able to assume a set of poses during the testing of their kinesthesia?

1.3 Structure of the thesis

This thesis consists of the following parts:

- In chapter 2 a literature review provides an overview of the state of the art and related work with focus on ASD (2.1), skeleton tracking (2.2), human motion analysis (2.3), hand tracking (2.4), and point cloud library (2.5), as well as validity information about the Microsoft Kinect sensor (2.6).
- In chapter 3 methods for the analysis of depth and skeleton information are compared based on the skeletal joints (3.1), point clouds (3.2) and hybrid approaches (3.3).
- As the main part of this thesis, chapter 4 gives information on the setup and configuration of the system (4.1) on the one hand, and the implementation of the prototype in three developmental stages (4.2) on the other hand.
- In chapter 5 the evaluation is documented. Apart from the methods (5.1), the hypotheses (5.2), the experimental design (5.3) and information about the participants (5.4), the evaluations regarding the skeleton values (5.5) and the hand point cloud (5.6) are followed by an evaluation with normally developed children and a discussion of the results (5.8).
- Finally, in chapter 6 the contents and findings of this thesis are summarized and ideas for future work are suggested.

2 Related Work

There are several topics that are related to this thesis. The outcome of the literature review in this chapter provides an overview of the state of the art and the related work. The focus lies on ASD and testing children with ASD (2.1), skeleton tracking (2.2), human motion analysis (2.3), hand tracking (2.3), point cloud library (2.5) and the validity of the Microsoft Kinect sensor (2.6).

2.1 Autism spectrum disorder (ASD)

Autism spectrum disorder (ASD) is a pervasive developmental disorder [45] that often involves sensory and motor problems [16]. Fuentes et al. [16] further mention differences in auditory, visual, tactile, and movement processing. Proprioception is the sense responsible for motor skills and for interacting with the environment. Because of their deficiencies regarding somatosensory or vestibular information for maintaining balance, children with ASD have to rely on visual information instead. In Fuentes et al.'s [16] study, participants with ASD state that in childhood they had problems with proprioception, they didn't know where their limbs would move to next or had last been positioned. The authors found out that although the participants with ASD suffered from motor and sensory impairments, their performance regarding proprioceptive accuracy and precision while identifying the absolute position of their fingertips as comparable to the performance of typically developed people. They suggest that this results from the central nervous system that directly calculates the limb endpoint position [16].

In their literature review, Lord and Bishop [24] describe ASD as a neurodevelopmental disorder with a strong genetic component, although there is no valid biological test to prove that. Being male increases the risk factor for ASD about four times, as well as having parents of advanced age. The deficits occur prior to three years of age in the areas of social interaction, communication and restricted or repetitive behaviors like repetitive hand and finger movements, mannerisms affecting the whole body, obsessive behaviors and rituals. Deficits in joint attention are also mentioned.

Several other papers have been published on systems targeting assessment and treatment of deficits in these areas. Some of these use advanced tracking technologies such as the Microsoft Kinect sensor, or virtual and augmented reality scenarios. As already mentioned in the introductory part in section 1.1, Fuentes et al. [16] write about their methods for testing the proprioception of autistic children. They use an exoskeleton robotic arm and perform active and passive tests. The concept of these tests has been developed and is explained by Lee et al. [22]. During the active repositioning tests the subjects have to move their limbs, e.g. their arms, until they believe a reference angle is reached. Whereas in the passive repositioning tests their limbs are moved, either by machines or therapists, and the subjects have to report, e.g. by push-

ing a button, when they feel that the reference angle is obtained [22]. Lee et al. [22] avoided visual clues by blindfolding their subjects, whereas Fuentes et al. [16] hide the exoskeleton robotic arm under a projection screen. A schematic representation of the child's arm is projected onto this screen as a dot above the elbow joint and lines above the upper and forearm, as visualized in Figure 1, while the real arm is strapped to the mechanical arm. It measures the angles of the shoulder and the elbow joints during the tests to compare them to the target or the counterpart joint angles. However, the exoskeleton robotic arm only measures deviances regarding two dimensions respectively one rotation-axis, because it acts on a shoulder-level horizontal plane.

Blanche et al. [7] use the *Comprehensive Observations of Proprioception (COP)*, a scale for measuring proprioceptive processing by direct observation, to evaluate difficulties among children with ASD. Some of the test items are joint hypermobility, poor joint alignment or inefficient ankle strategies. Although the performance of the children with ASD was similar to that of children with developmental disabilities regarding most of the observed items, the study showed that the COP is a useful tool for observing proprioceptive difficulties of children with ASD.

Sparks et al. [39] observed the volumes of different parts of the brain of three to four-year-old children with ASD and compared them to children with typical development on the one hand, and children with developmental delay on the other hand. They found out that the cerebral volumes of autistic children were significantly increased, regardless of their intelligence quotient (IQ). Based on their structural findings, Sparks et al. suggest unusual processes in brain development of young children with ASD to be the cause.

There are also some research groups working with children with ASD by using virtual and augmented reality. Casas et al. [8] built an augmented reality system for training and testing children with ASD. Their project *Pictogramm room* mixes reality and computer-generated information, hence abstraction capacities are not as much required as in virtual reality applications. This makes it easier to use for people with autism who have fundamental problems with abstraction. They were able to build an application using the Kinect sensor, where the children are trained to move their body for example to match certain postures. They tested their system with typically developed children to gather reference values. By comparing the results of children with ASD with reference values, they got an idea of which skills had developed more and which still had to be improved. Other authors, like Bartoli et al. [5] and Bai et al. [4], also used virtual and augmented reality systems for training autistic children's development, although they assessed the children's learning skills manually. Bartoli et al. [5] used the Kinect sensor and motion-based touch-less video games, whereas Bai et al. [4] used a marker-based augmented reality application, where augmented-reality tags were put on wooden blocks serving as real equivalents of virtual toys in order to promote the children's pretend play. However, in their findings both groups could not find significant evidence that the measured benefits rely on the specific settings the games were played in, or if the benefits were persistent.

In this thesis I will describe a method and technical system to measure the proprioception of children with ASD. To do so, I will use a virtual reality system for automatically assessing the angles of the shoulder and elbow joints regarding two rotation-axes. I will further compare the angles of the corresponding joints of the two body halves, as well as the point clouds of both segmented hands to calculate deviations. Details on how the algorithm works can be found in section 4.2.

2.2 Skeleton tracking

In the context of this thesis skeleton tracking is used to capture gross motor movements for the assessment of proprioception. Gaming, human-computer interaction, security or healthcare are only some of the application areas for real-time human body tracking, as explained by Shotton et al. [37]. Owing to limitations in accuracy, robustness or scalability, there was no existing system that reached interactive rates on consumer hardware until Microsoft's Kinect platform was launched. Shotton et al. [37] and Zhang [47] explain some technical details of the Microsoft Kinect sensor. The depth sensor, consisting of an infrared (IR) projector and an IR camera, produces a 640 x 480 pixels depth image at 30 frames per second and depth resolution of a few centimeters. It works in low light, is color and texture invariant and simplifies the so far difficult task of background subtraction. Additionally, there is a color camera and a four-microphone array. More details on the Microsoft Kinect sensor can be found in subsection 4.1.1.

Based on an object recognition approach, Shotton et al. designed a high-speed classification-system to estimate body parts invariant to pose, body shape and clothing. By using hundreds of thousands of training depth images of different shapes, sizes, poses, clothing or hairstyles, partly real and partly synthetic, the research team trained a deep randomized decision forest classifier. Self-occlusions and cropped poses are handled as well. The developed algorithm is a core component of the Microsoft Kinect platform. As shown in Figure 2, it works on single input depth images and divides objects into parts in order to reach computational efficiency and robustness. Each pixel of the image is evaluated separately and the classifier can therefore be run completely parallel on each pixel on a GPU. After every pixel is classified and labeled as a body part or as background, the system calculates temporary joints by using the local modes of density of each body part. Finally, these joints are mapped to a skeleton considering the prior knowledge from the training data and the temporal continuity. The results are 3D coordinates of the skeletal joints. [37,47]

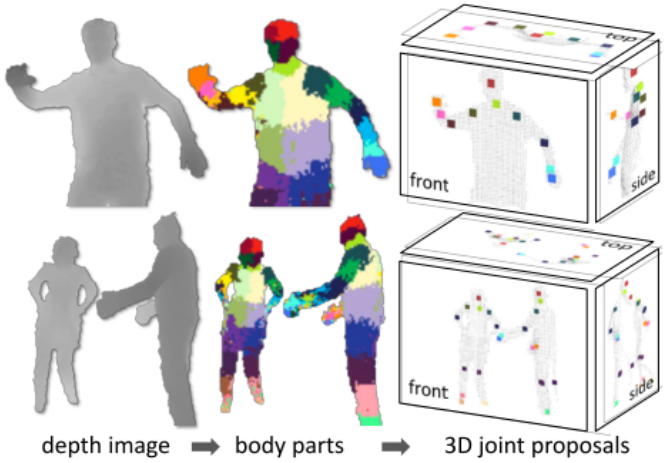

Figure 2 - Kinect Skeleton Tracking Pipeline by Shotton et al. [37]

Another approach to tracking the skeleton using Microsoft's Kinect platform was made by Kar [21]. He used the OpenKinect drivers to access the depth and RGB images of the Kinect sensor. The algorithm uses a Haar cascade classifier to detect the upper body and face. Then the hand positions are defined using skin segmentation and finally a stick skeleton model is fit to the upper body. Although his algorithm only detects the upper body, which would be enough for my context, it does not work correctly on occluded poses, is two thirds slower and not as accurate and stable as Shotten's approach. I use the skeleton tracking approach of Shotton et al. [37], as explained in this chapter and shown in Figure 2, and extend it with some hand tracking functionality, as described in sections 2.4 and 4.2.

Previous research groups working on model-based approaches for full body tracking based their models of human limbs on simple geometric objects such as cylinders. In contrast, Charles and Everingham [10] developed a generative model of limb shapes, trained by silhouette, depth and 3D pose data delivered by Microsoft Kinect. In this way they created a system that was able to estimate poses of highly varied human body shapes in terms of anatomy, clothing, camera parameters and freedom in pose space of the human.

2.3 Analysis of human motion

Several applications use skeletal information for the analysis of human motion. In their paper, Liu et al. [23] describe their application that automatically evaluates the player's energy consumption. To realize this they used the high-level skeleton module of the OpenNI framework, and took average values for the height and weight. Every piece of the body has a percentage of the whole body weight, based on a human model, e.g. the torso has 58% of the body weight. In the real-time analysis the motion of each moving body part is first divided into a horizontal and vertical movement. The partial kinetic energy is computed using the movement speed and the mass of each part, first for each directional plane, then summed up for each part. By summing up the partial ener-

gy values, they get the total energy consumption for the actual frame. Similar to this approach, where the movement is split into a horizontal and a vertical movement to achieve an easier computation, the calculation of the upper and the forearm's rotations in this thesis' algorithm is also separated into the vertical and horizontal rotation axes in order to achieve more accurate and comparable values, as explained in more detail in subsection 4.2.1.8.

Another system using skeletal information in order to analyze the movement of the human body is explained by Martin et al. [26]. They built up a training environment for laborers in factories to learn the correct lifting and carrying methods for the purpose of preventing back injuries. Martin et al. mention some problems regarding the accuracy of the Kinect sensor, such as occlusions and jumpiness of the perceived joint locations when the sensor was positioned in a suboptimal way. During the tests and evaluation of my prototype I was confronted with similar problems. Therefore, the sensor was positioned in a particular way, as explained in detail in section 5.3.

Suma et al. [41,42] developed *FAAST*, a *Flexible Action and Articulated Skeleton Toolkit*. With this middleware it is possible to integrate the full body skeleton information tracked by *OpenNI*-compliant depth sensors like the Microsoft Kinect into virtual reality applications and video games. Details on *OpenNI* can be found in section 4.1.2. By using a *VRPN server* (*Virtual Reality Peripheral Network)*, the user's skeletal joints and customizable actions are transmitted over the network or into another application. These customizable actions trigger mouse or keyboard events when special poses or gestures of the tracked users are recognized. Although this toolkit offers functions such as the extraction and transmission of the skeletal joints for easy integration in applications, it could not be used for my prototype application because of the necessity of direct access to the depth image of the sensor.

ARTIFICe is a VR/AR framework built by Mossel et al. [28], which is very flexible when it comes to offering support for different setups, platforms, and integrating multiple different input and output devices, from low-cost to highly professional and expensive. For retrieving tracking data over the network from remote input devices again *VRPN* is used. The framework is not only used for tracking markers or 3D mice, but also for depth cameras, and because it integrates *OpenNI/NiTE* and *FAAST* the framework supports full-body tracking. More information on *OpenNI* and *NiTE* can be found in subsection 4.1.2. Mossel et al. [28] used Unity3D as a development platform and rendering engine, as explained in further detail in subsection 4.1.3. Because my application setup only needs support for one depth sensor as input device I decided not to use this framework. I used *OpenNI* and *NiTE* directly in my prototype application, as explained in subsection 4.2.1.

2.4 Hand tracking

3D tracking of human hands has a number of application areas, such as human-computer interaction, hand-pose detection or even robot learning. Based on the use cases there are differences in the requirements regarding accuracy and speed of the hand skeleton tracking algorithms. Human-computer interaction as in user-interface navigation has to be very fast to give the user live feedback. Detecting and comparing hand poses has to be very accurate to bring the best fitting result. But problems such as self-occlusions of the hand or at least of some fingers make developing robust vision-based solutions complicated. Special motion capturing hardware or the use of visual markers provide viable solutions, but the setup is complex and the hardware expensive [29]. Extending skeleton tracking to the tracking of hands, or especially parts of them, is another focal point of this thesis. However, not many researchers base their algorithms on the outcome of a full-body skeleton tracking system.

Zafrulla et al. [46] used the Microsoft Kinect sensor for recognition of American Sign Language. For this purpose they used the skeleton tracking library of *OpenNI/NiTE*, configured to only return joints of the upper body to save computing time. Based on these joints they calculated body pose features of about 20 dimensions. To get the hand shape features they collected all the 3D points in the neighborhood of both the left and right hand-related joints. They found that while using the Kinect for skeleton tracking of people sitting in front of the sensor, some tracking errors occur and further calculations may fail, whereas using it while standing works quite well. Similar to this approach, I used the 3D neighborhood of the hand joints to collect the hand-related depth points in order to calculate the difference of both hand shapes, as explained in more detail in subsection 4.2.2.1.

Oikonomidis et al. [29] developed another method for hand tracking using the Microsoft Kinect sensor. Their algorithm uses the RGB image and related depth map. The hand segmentation is done by detection of the skin color and depth segmentation, which means that the hand always has to be in front of the rest of the body. Afterwards a 3D hand model containing palm, fingers and segments of the fingers, represented by 27 parameters, is defined. These model parameters are estimated in order to find the one most compatible with the visual hand segmented in the previous step, using a distance measure. (The authors optimized this operation by using *Particle Swarm Optimization*.) This workflow leads to a solution for accurate 3D tracking of the human hand in near real-time, which requires neither complex hardware setups nor visual markers. But the users have to hold their hand in a special starting pose until the system has analyzed the hand and modified the model.

A similar approach was developed by Melax et al. [27] although they used a physical simulation for fitting a 3D hand. They did not use the Microsoft Kinect sensor but an ASUS Xtion Pro depth camera. When a fast hand movement was detected, the entire hand was updated. They also had a classifier looking for a spread five-finger pose. Additionally they simulated the flipping of fingers and the grasping in order to find the

best fitting model, which is the one with the smallest error. This algorithm is much faster than that of Oikonomidis et al. [29] and can tolerate holes as well as sparse depth maps because it works with the point cloud of the depth image and minimal error calculations.

Regarding real-time hand and fingertip tracking and detection Feng et al. [15] developed a powerful algorithm using the Microsoft Kinect sensor for a new writing-in-the-air system. Based on their assumption and constraint, that the hand and the fingers are always in front of the torso, they computed the depth histogram of the whole human body to segment the hand region. Via the K-means algorithm the segmented region is classified into the finger-hand part and the hand-arm part, as they call them. The fingertip, as part of the finger-hand cluster, is the farthest point from the hand-arm cluster's center.

Pedersoli et al. [31] developed *XKin*, a library-project for human-computer communication via hand-gestures. This open-source framework offers functionalities for detecting the body, the hand, hand postures or hand gestures and it works on raw Kinect data streams coming from *libfreenect*, an open-source driver for the Kinect sensor, on which the framework is based on.

Another hand gesture recognition system was developed by Ren et al. [33]. They used depth thresholding for hand segmentation and a black belt around the wrist for more accurate hand segmentation. They used so-called time series curves to identify the fingers, which means that they recorded the relative distance between each contour vertex and the center of the palm.

Varkonyi-Koczy and Tusor [43] used a stereo 2D camera setup for capturing the scene in order to identify fuzzy hand postures and gestures. The system detects the human hand by skin color detection, followed by extraction of feature points based on curvature extrema, peaks such as fingertips, and valleys such as the roots of the fingers. These feature points are then matched in a stereo image pair to get a 3D coordinate model that is then used for posture and gesture detection.

2.5 Point Clouds

The emergence of affordable depth cameras based on time-of-flight or structured light approaches, as mentioned in subsection 4.1.1, created an increased demand for frameworks and algorithms that are able to process these data robustly and efficiently. Therefore, recent research and software development projects aim at filling this gap.

The *Point Cloud Library* or *PCL*, as presented by Rusu et al. [34], is a library of functions and mechanisms necessary in order to handle and work with n-D point clouds and 3D geometries. Mathematical operations as well as a fast k-nearest neighbor registration are implemented. All modules pass the data around using shared pointers to avoid the recopying of the point cloud data. It also contains more complex functions

such as the *StatisticalOutlierRemoval,* which is very useful for applications working with point clouds based on depth maps because it rejects outliers in the point cloud.

Applications using 3D point clouds combined with depth sensors are often used in robotics to make robots “see”. The system presented by Rusu et al. [35] builds object maps of kitchens or other indoor household environments. Anand et al. [2] used segmentation of 3D point clouds of indoor scenes recorded with the Kinect sensor to label objects in the scene, quite similar to Ali et al. [1], who combined the segmented objects with a context model derived from social media metadata. Another approach using the Kinect is described by Henry et al. [18], where they used the depth sensor for building colored and dense 3D models of indoor environments.

With *KinectFusion*, Izadi et al. [20] built a real-time interactive real-time reconstruction system that only uses the depth data delivered by the Kinect sensor for tracking the 3D pose of the sensor in the environment as well as reconstructing 3D models of the physical scene. While the user moves the Kinect it delivers depth data that is directly merged to the already known 3D model. The system is thus able to “scan” whole objects, people, environments, as well as fill holes and track high-quality details. The more depth measurements are added, the more complete the model becomes.

For the algorithmic part of my application regarding the hand segmentation and especially the comparison of the two hands’ orientation and appearance, the use of some point cloud library or framework could have been reasonable. But in *Appendix* B.1 Including the PCL some of the occurring problems during the attempts to include the library are further explained.

2.6 Validity of Kinect

Clark et al. [11] analyzed the validity of the Microsoft Kinect sensor’s skeletal joint tracking system to be able to use it for gait retraining instead of marker-based systems. They used the shoulder center and hip center to calculate the lateral trunk lean angle. This lean angle is used during training in a real-time biofeedback system visualized on a monitor. For their special use, they found out that the Kinect sensor reaches without any further calibration an error of more than 3.2°±2.2°, with global calibration an error of 1.7°±1.5°, and with individualized calibration an error of 0.8°±0.8°. Whereas in [12] Clark et al. found out that the Microsoft Kinect provides comparable anatomical landmark data to a 3D motion analysis system. They tested it during three postural control tests: forward reach, lateral reach and single-leg eyes-closed standing balance. Although there are some differences between the outputs of the two systems, for example the height of the sternum, those differences are systematic and therefore may only have insignificant influence on the actual use of these results. Based on their findings, the authors suggest that the Kinect “can validly assess kinematic strategies of postural control” ([12], p.1). Galna et al. [17] agree with the findings of Clark et al. in [11,12], and add that the Kinect sensor does not notice tiny movements and rotations in joints as accurate as other tracking systems do as they tested with the actions of hand clasping

and toe tapping. As the two papers by Clark et al. [11,12] as well as the paper of Galna et al. [17] show, using the Microsoft Kinect sensor for the assessment of the sensory functionality of children by measuring the kinesthesia of children using depth data is possible, and after some small preceding adaptations like detecting and removing systematic errors, a stable and reproducible test can be developed. Especially by penalizing bigger divergences more than smaller ones, the little error values mentioned above can be minimized and neglected.

3 Comparison methods

As the aim of this thesis is the development of a reproducible and stable test for measuring the kinesthesia of children, one research aspect are comparison methods of body postures. To briefly introduce my implemented methods in relation to the common methods in this research area, this chapter describes and analyzes different comparison methods. In section 3.1 the presented algorithms work on the tracked skeletal joints. Section 3.2 is about point clouds based on the depth images' data. And section 3.3 describes hybrid approaches, meaning methods in which the skeletal joints and the point clouds come together, or the point clouds complement the information on the skeletal joints.

3.1 Skeletal joints

As the main functionality of my application works with information based on the tracked skeletal joints, this section deals with joint-based comparison methods such as Liu et al.'s [23], who used the change in the skeletal joint nodes' 3D coordinates, delivered by the MS Kinect sensor. They used the points for calculation of motion in order to calculate the user's energy consumption.

Chaaraoui et al. [9] developed an evolutionary algorithm to improve action recognition using depth images. Therefore, the skeletal joints are composed to large feature vectors based on their 3D coordinates, although quaternions or distances between different joints could have been used to increase the level of detail. In order to correctly compare the extracted features, they proposed an algorithm to normalize the skeletal joints regarding their positions, scales and rotations, based on the distance from torso to neck.

My application tracks the user in front of the camera. By using the depth image, the *OpenNI/NiTE* framework segments the user from the background. Based on a ground truth the framework analyzes the depth information and extracts the skeletal joints, as explained in section 2.2, developed by Shotton et al. [37]. Similar to Liu et al. [23], I used the 3D coordinates for further processing. Every joint is connected with its relating neighbor to track the bones and finally a full-body skeleton. The bones are necessary for calculation of rotations and angles, whereas the full-body skeleton is primarily used for presentation on the virtual stage of the application. For the normalization of the joints' and bones' coordinates, an algorithm, similar to the process Chaaroui et al. [9] explained in their paper, is used, where the distance between torso and neck acts as the normalizing factor. However, instead of rescaling the whole skeleton, all joints and bones, the normalizing factor is used at the very end, after the calculation of the error values. In general, this step is necessary in my application to deliver reproducible and stable skeleton data for the measuring of the kinesthesia of children. Based on the ori-

entation and position of the connection between torso and neck, the skeleton is rotated in order to have the skeleton directed towards the camera on the virtual stage of the application. This should result in a more consistent presentation of the tracked information and improve user experience and usability, because in general the user is not directed exactly towards the Kinect sensor. Technical details about the mentioned algorithms can be found in subsections 4.2.1.5, 4.2.1.6 and 4.2.1.8.

3.2 Point clouds

Apart from the joints of the shoulders, elbows and wrists, information regarding fingers and palms is necessary for the application in order to measure kinesthesia. Because with the current configuration of my application's prototype the hands are segmented out of the sensor's depth image and are therefore available as clouds of points in 3D space, the following part contrasts different comparison methods with each other that are based on point clouds.

Ly et al. [25] used a single depth image captured with a Kinect sensor and a kinematic skeleton to predict the kinematic pose of the whole skeleton. The algorithm manipulates a kinematic skeleton to best fit the point cloud. For comparison of the algorithms they used the following four metrics: a fitness metric that gives higher scores when the model better explains the point cloud; the mean point distance error between the cloud points and the related closest points on the model surface; the root mean squared point error of the distance between the cloud points and the related closest surface point; and the mean joint distance error between the inferred joint locations and those from the ground truth model. For the error calculation of the skeleton joints in my application both the mean point distance and the mean squared error metrics were implemented and used. For the error calculation of the point clouds the mean squared error metric was implemented and used.

Another method of comparing information based on point clouds was presented by Baak et al. [3]. Their application transforms the distance values of the depth image into a 3D point cloud, subtracts the background, deletes the contour pixels to get rid of mixed pixels and filters using a 3x3 median filter to reduce noise. Then five geodesic extrema representing the head, feet and hands are extracted, the best matching entry out of a pose database is compared to the optimized previous pose. A voting approach then fuses them to the final pose.

Stone and Skubic [40] measure the stride-to-stride gait variability of old people in order to predict future falls using the Kinect sensor's depth image while people walk through a room. From the 3D point cloud of the foreground, which represents the walking people, gait information is extracted. By projecting the points below 20 inches of height onto the ground, and after using normalization, the correlation coefficient is computed, which represents the number of left and right steps in a walking sequence as local maxima and minima. Further information is estimated by combining the occurrence time of each step with the movement of the centroid of the 3D point cloud. The authors

mention that especially for assessing the variation in stride velocity, the Kinect system has sufficient accuracy.

As already mentioned in section 2.5, *PCL* is a common library for working on datasets like point clouds. One of the most basic functions is the *iterative closest point* algorithm or *ICP*. By applying the *ICP* algorithm on two separate 3D point clouds, as explained by Besl and McKay [6], the algorithm finds corresponding points between the two clouds and tries to minimize the Euclidean distance, or another error metric, between the matched points by translating and rotating the clouds until they fit best. This implementation is called *point-to-point ICP* and the result represents the parameters for translation and rotation. In my application a varied *ICP* is implemented, which calculates the current Euclidean distance values between the two point clouds of the right and the left hand. Therefore, the point cloud of the user's depth image is segmented based on the hand joints to separate the hand-relating parts of the cloud from the rest, as mentioned in section 3.3 and explained in technical details in subsection 4.2.2.1. The two point clouds consisting of the hands' points are re-rotated based on the orientations of the shoulder and elbow joints and repositioned based on their hand joints to result in a reproducible and normalized situation. Additionally, one hand point cloud is mirrored to be directly comparable to the other hand. Then my varied implementation of the *ICP* is triggered, where every cloud point of the one hand is compared to every cloud point of the other hand to find the smallest squared distance. The quadratic distance is calculated in order to penalize the bigger distances more. By summing up the smallest distance values of all the points, similar to Ly et al.'s [25] mean point distance and root mean squared error metrics, the application calculates the total squared error value and the mean squared error value between the two hands' point clouds. Technical details about the mentioned algorithms can be found in subsection 4.2.2.3 and evaluation results can be found in section 5.6.

3.3 Hybrid approaches

In this research area, hybrid approaches that combine skeletal joints and point cloud information are developed as soon as more information than the framework-given skeletal joints is necessary, like it is in this thesis. Du and Zhang [13] developed a system for remote teleoperation of robot manipulators based on Microsoft Kinect. Using the skeletal joints, the coordinates of the shoulders, elbows and wrists can be extracted, but thumb and index finger tips are not included. By building up a hand bounding box based on the coordinates of elbow and wrist, the point cloud segment regarding the hand and the fingers can be selected. In their special use case the thumb and index finger tips were the two points farthest away from the wrist. Additionally, they used the shoulders coordinates as reference points to prevent human body movement from affecting the robot arms.

For their approach on hand segmentation, Huang et al. [19] used a combination of skeleton and depth information and also defined a bounding box. But because they

used the *Microsoft SDK*, they additionally had the point of the hand, not only the wrist. So they positioned the bounding box around the hands' joints positions. Then the points in the box were segmented as the hand.

Ponce et al. [32] used Microsoft Kinect to detect all-day activities such as *brushing teeth* or *drinking water*. They extracted features from the skeletal joints to get information about body pose and hand position with respect to torso and head. Additionally, they used the *Histogram of Oriented Gradients* or *HOG* feature descriptor of the RGB and depth images, especially for skeletal segments such as the head, torso, left or right arm, in order to detect the current activity by comparing it with the trained model of activities.

Building up a hand bounding box for the pre-selection of the hand-points is also used in my application, although a bit different compared to the approach of Du and Zhang [13]. They used the hand bounding box based on the coordinates of elbow and wrist and the direction the forearm points to. But in my case, the hand bounding box had to be more flexible because the fingers or the hand itself could have any possible rotation. Therefore, a much bigger box was used. The positioning of the hand bounding box in my application is a bit more complicated than Huang et al.'s [19] method. By using another framework than the *Microsoft SDK*, the joint which is related to the hand itself is missing, which is why I had to use a bigger bounding box and also reposition the box to select all hand-related depth points. Because of the bigger bounding box, it can happen that many points that are not part of the hand are included in the selection. Therefore, my algorithm had to refine the segmentation to get rid of all accidentally selected points of other body-parts or the environment. Beginning with the selection of the nearest point to the hand joint, the algorithm iterates through all the points in the hand point cloud and selects all the points that are in the direct neighborhood of the already selected ones. More technical details about the segmentation, its refinement and the optimizations that were made are explained in detail in subsections 4.2.2 and 4.2.3.

4 Design and Implementation

As the title of this thesis already mentions, it is about the development of a reproducible and stable test for measuring the kinesthesia of children with ASD by assessing the sensory functionality of children using depth data. Therefore, in addition to the literature reviews of this and related research areas and the listing and comparison of methods for comparing and measuring tracked skeletal joints and depth data, I developed an application prototype. In this main chapter section 4.1 describes setup and configuration of the application, whereas section 4.2 contains details about the prototype versions including the different refinements based on iterative user tests up to the final prototype with detailed information about the implemented algorithms and functionalities.

4.1 Setup and Configuration

At the beginning of the development process of an application the whole system has to be setup and configured. This requires the use of a combination of special hardware, systems, frameworks and maybe plugins, libraries or APIs. Details about the sensor hardware used in my application setup can be found in subsection 4.1.1, about the used framework and plugins in subsection 4.1.2, about the editor, programming language and IDE in subsection 4.1.3, and important information about used wrappers, drivers and *dll*-files in subsection 4.1.4.

4.1.1 The Sensor Hardware – Microsoft Kinect sensor for XBOX 360

The sensor hardware used is the *Microsoft Kinect sensor for XBOX 360*. As Shotton et al. [37] and Zhang [47] write in their papers, it consists of a depth sensor, an RGB camera and a four-microphone array. Figure 3 shows the Kinect and illustrates the positions of its components.

Figure 3 - The Microsoft Kinect sensor for XBOX 360 with components indicated.

The RGB camera is comparable to a webcam and produces color images with up to 1280 x 960 pixels at 12 frames per second, depending on the configuration of the application setting.

Figure 4 - RGB image, produced by the RGB camera. Certain sections were blacked out for reasons of anonymity.

The depth sensor, composed of an IR projector and an IR camera, makes a 640 pixels wide and 480 pixels high depth image at 30 frames per second and a depth resolution of a few centimeters. The IR camera uses a monochrome CMOS sensor, works in low light, is color and texture invariant and simplifies the so far difficult task of background subtraction. The depth sensor works on the *structured light* principle, meaning that the laser beam of the IR projector, which is an IR laser, is turned into a set of IR dots by being filtered by a diffraction grating. Because of the known displacement between the IR projector and the IR camera and the known IR dot pattern, every observed dot in an image can be matched with a dot in the projector pattern, and therefore can be triangulated and reconstructed in 3D. For matching the dots with the pattern efficiently, small groups of dots, so-called neighborhoods, are compared. The generated dot pattern of a scene as in Figure 4, as seen by the IR camera, is visualized in Figure 5. The right part of the image shows a close-up of the red boxed pattern, and the yellow box visualizes a window in the background of the scene, where no reflected IR dots can be seen. This is a problem of the Kinect sensor, which is why users have to take care to avoid highly reflective surfaces. The Kinect sensor produces a depth map based on the IR image, see Figure 6. Based on the depth value, the pixel is colored. The farther away the point is, the darker the gray value is colored, the closer to the camera the lighter a pixel is colored. Black regions occur in several cases. Either when no depth values can be computed, points are too close, or points are in the shadow of the IR projector where no IR dots are founded in the displacement of IR projector and IR camera, or reflective surfaces cause poor IR lights.

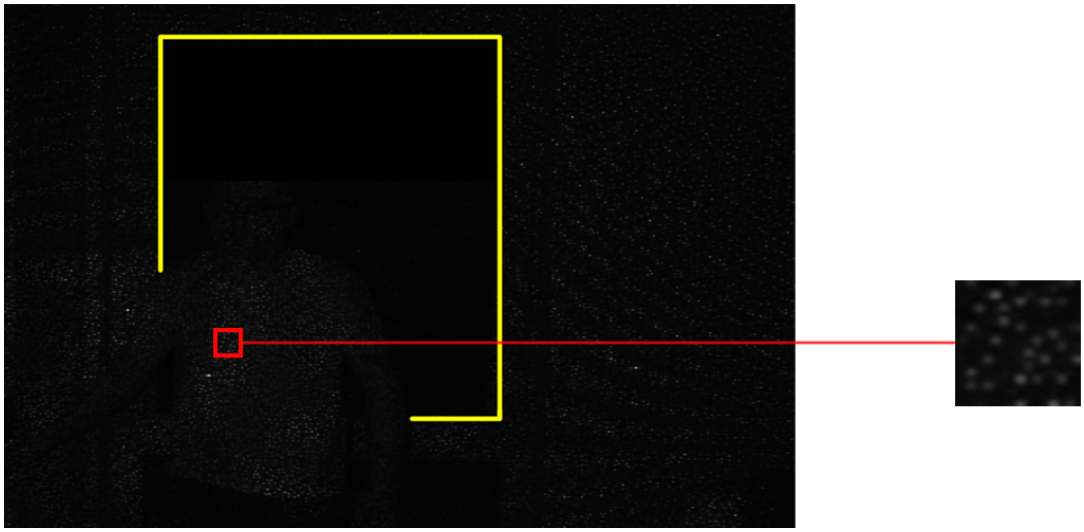

Figure 5 - The infrared dots seen by the IR camera. The image on the right shows a close-up of the red boxed area. The yellow boxed area in the background shows part of a window, a reflective surface.

Figure 6 - The depth image, produced by the Kinect sensor, based on the dot image in Figure 5, and colored yellow for better visibility.

During one of the last development phases of the application prototype for this thesis the newer version of the sensor, the *Microsoft Kinect sensor for XBOX One*, was announced. Its technology, the *time-of-flight* principle the new sensor works on, measures the time an amplitude-modulated light takes from the light source to the target object and back again to the camera. This is done for each pixel. State of the art time-of-flight cameras provide high pixel resolutions, a wide field of view and the motion blur is reduced significantly from 65ms to 14ms compared to the previous sensors. Also a high dynamic range can be realized, which is needed for rendering high-reflective objects near the camera as well as low-reflective objects far away. By combining multiple captured images, depth and active IR images are produced, as explained in [30] by Payne et al. Although the higher resolution of the new Kinect sensor would have been advantageous regarding more accurate measurement, more available joints and depth points, greater speed and better segmentation of the hands, the requirements for work-

ing with the new sensor were not practicable for my purposes. As it needs Microsoft Windows 8 and the regarding Microsoft Kinect SDK for developing applications, a complete reengineering of the whole application prototype would have been necessary. For future work, the new sensor with the compatible SDK offers a new level of accuracy and performance. In the next sections 4.1.2, 4.1.3 and 4.1.4, the framework, IDE and drivers that were used are explained in more detail.

4.1.2 Framework – OpenNI/NiTE

For accessing the data the Kinect sensor offers, I used the open-source framework *OpenNI* [48], short for *"Open Natural Interaction"*, that was developed and supported by a group of companies around *PrimeSense Ltd.* [49] as an industry-standard framework for the interoperability of natural interaction devices. Accompanied by middleware libraries that can convert raw sensor data from an OpenNI-compliant device such as Kinect, this framework can be used in order to develop custom applications written in *C, C++* and *C#.* The framework runs on all popular platforms like Windows, Linux, MacOS, as mentioned by Villaroman et al. [44]. One of the previously mentioned middleware libraries is *NiTE*, which extends the functionality of *OpenNI* with focus on image processing and allows tracking of the whole skeleton as a set of 15 joints, as Sinthanayothin et al. [38] and Chaaraoui et al. [9] explain briefly. OpenNI as well as *NiTE* were used in version *2.2* because of the most accurate tracking result and because no calibration pose is necessary at the beginning of the tracking session.

During the time of writing this thesis and developing the application prototype, the company *PrimeSense* was taken over by *Apple* and shortly after, *Apple* stopped the *OpenNI/NiTE* support. To support the *OpenNI/NiTE* community, the company *Occipital* [50] set up a support and download platform for the last stable version of the framework.

4.1.3 Unity3D, C#

For the development of the application prototype *Unity3D* [51] was used as an editor for the 3D content, the stage and to include and load the scripts. It offers a powerful rendering engine and the possibility to build the final application for all desktop and mobile platforms, game consoles or web clients without making changes to the code or the content, as pointed out by Mossel et al. in [28]. This is possible because *Unity3D* is built up on *Mono*, which supports only *.Net 2.0* and earlier. All scripts that the application prototype consists of were developed with *Microsoft Visual Studio 2012*. I used the programming language *C#* for the whole application, because *Unity3D* only supports *C#*, *JavaScript* and *Boo*, and the wrappers and libraries for accessing the *OpenNI/NiTE* framework (see subsection 4.1.4) were only available for *C#* out of the three mentioned languages.

4.1.4 Wrappers, Drivers and .DLLs

While for *OpenNI* versions prior to 2.0 a special wrapper for *Unity3D* was available to be imported into the editor and access the framework's and middleware's functionalities, with *OpenNI* version 2.0 and higher no official wrappers could be found. However, Falahati [14] developed wrappers for *OpenNI* 2.2 and *NiTE* 2.2, under *GNU LGPL*, based on the *.Net 4.0 Client Profile* framework. He also delivered the *Microsoft Visual Studio Solution File* for *Microsoft Visual Studio 2010*. With this file, I was able to edit the wrapper project, and adapt and extend the functionalities offered by the wrappers for my own needs. I recompiled the wrapper files for further use in my application prototype. The functionality I needed to use in my application was accessing the camera elevation, to move the camera to a certain tilt angle. Another necessary feature was the functionality to activate and deactivate the mirroring of the video stream, which is used to receive and display the captured video frames in the virtual scene as if there was a mirror, and not inverted. This is indispensable for the usability of the application. As the Kinect's original driver file had not supported these options, I had to swap it with a customized driver file for the Kinect sensor. A developer of the *OpenNI*-community, *ed-diecohen* [52], published a patch with which it was possible to recompile the *OpenNI*-drivers for the Kinect, including the enhanced options. Based on this driver file I was able to add camera elevation methods to the wrapper's source code. But as *Unity3D* supports *.Net 2.0* only, and the wrappers developed by Falahati were based on *.Net 4.0 CP*, I had to reimplement those classes and functions that the wrappers use but which are not supported in *.Net 2.0,* such as the classes in the namespace *System.Windows.Media.Media3D*. After recompiling the wrapper files to *.dll*-files and including them in the editor, I was able to access the functionality via the *C#*-scripts of my application.

4.2 Prototype

After the system configuration and setup explained in Appendix A: Installing all applications necessary to run the prototype, my system was prepared for the development. As nowadays quite common, an agile development method was used in order to directly implement the experts' feedback in the iterations of the process and to already see a lot of functionality in an early development state. With this method, several versions of my application prototype arose, with growing functionality, usability and a modified user-interface. The steps from the beginning until the first prototype are explained in subsection 4.2.1, the functionalities and improvements for the second prototype can be found in subsection 4.2.2 and the final prototype is described in subsection 4.2.3.

As it is common in game-development environments, all steps for initialization and configuration are done in the application's start-function, named *Start*. The iteratively called loop-function contains the main functionality of the application. It is called once per frame and is named *Update*. In addition, the method *OnGUI* is called whenever the GUI is repainted. And the method *OnApplicationQuit* is called immediately before the

application is quit, so it is the right place to save the state and information that should not be lost.

4.2.1 The First Prototype

I initially started with the first attempts of getting any information out of the Kinect sensor via the *OpenNI/NiTE* framework. By including the wrapper classes of the framework, mentioned in subsection 4.1.4, into my main application, I was able to step by step call the basic functions of first *OpenNI* (4.2.1.1) and later *NiTE* (4.2.1.3), to build up my application. The prototype's functionalities are explained in this subsection. After a short elaboration of the difference between depth and world coordinates (4.2.1.2), the tilting of the camera (4.2.1.4), the building of the skeleton using bones (4.2.1.5) followed by the alignment, rotation and overlay of the skeleton and the point cloud (4.2.1.6) are explained. The next section is about the mirroring of the bones and joints of the skeleton (4.2.1.7), which is necessary for the measuring of the divergence between the two body halves (4.2.1.8). For giving the user control about the function, a GUI is created (4.2.1.9) and completes the first prototype.

By putting all the functionalities mentioned in this subsection together, I created the first prototype version of the application. For this purpose I removed unused code, formatted the debug-outputs and exported a standalone player for Windows using the Unity3D editor. After installing the necessary tools as listed in Appendix A: Installing all applications necessary to run the prototype on a test computer, the executable file can be started. The process begins with the initialization of the system, followed by the search for users in front of the sensor. As soon as a user and especially the user's skeleton are tracked, the record button is enabled. This records the current scene and displays the tracked skeleton on the virtual stage. The calculate button overlays the point cloud over the skeleton. The triggered process further aligns and rotates the whole data, mirrors the bones and joints and finally measures the divergence. Finally, the person using the application can use the functions for visualizing details, can record the next pose or can close the application.

The application window, as displayed in Figure 7, collects all the parts of the application explained in this subsection. In the screenshot, part (1) represents the depth map of the tracked user, and part (2) the RGB-image of the camera. The visualization of the whole skeleton, including the mirrored bones and the overlaid point cloud are labeled with (3). For direct user feedback, section (4) displays the live depth map of all tracked users. The area number (5) is the GUI for controlling the application's functions, as described in subsection 4.2.1.9.

After the evaluation of the first prototype with the experts, some improvements such as the missing handling of the hands' point clouds, a control unit for the camera tilting and an export of the measured data to a text file were suggested. In addition, an xml-file for configuration variables and keyboard shortcuts, as well as a dialog window for entering the test person's name and displaying the current pose number are parts of the functionalities for the next development phase.

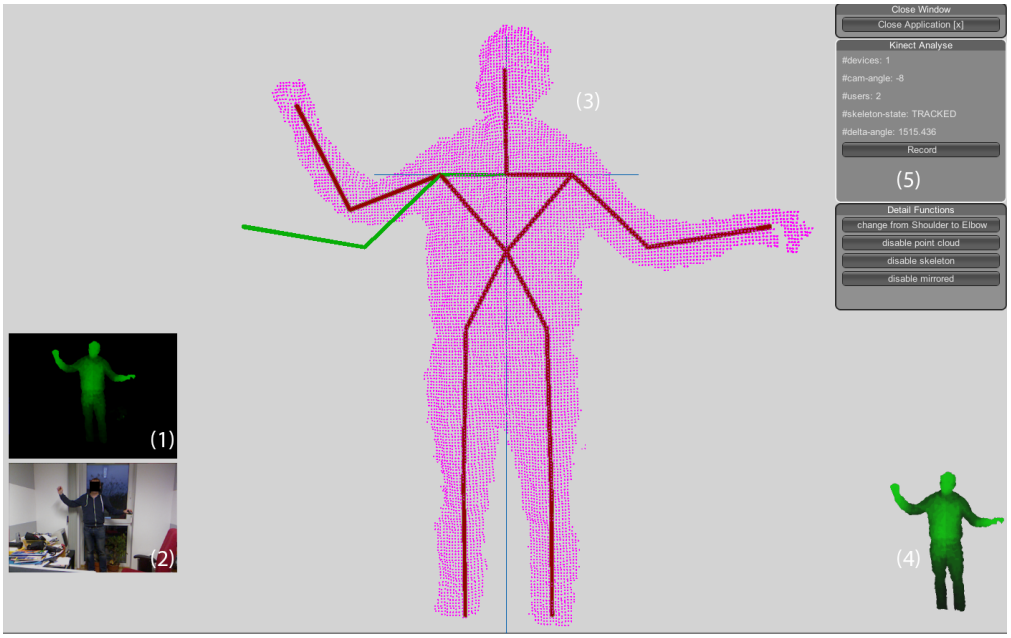

Figure 7 - Screenshot of the first prototype version of the application. (1) The depth map of the tracked user. (2) The RGB-image. (3) The visualized skeleton, with mirrored bones and overlaid point cloud. (4) The live depth map of all tracked users. (5) The GUI for controlling the application's functions.

4.2.1.1 Basic OpenNI-Functions

By initializing the *OpenNI*-object, the framework delivers the current *OpenNI*-version as well as a list of sensors available at the moment. If multiple USB-controllers are supported on the computer, multiple sensors can be used in the same application environment. From the list device-related information can be requested, such as the names of the producers and the devices. Then the RGB-stream and the depth-stream of a specific device can be created. Therefore, the framework offers information about the supported video modes, especially the different combinations of resolution and frames per second (fps) of the special sensor types. They depend on the device, the sensor type and the available drivers. The supported video modes of both sensors in our application setting are listed in Table 1, and the modes used are highlighted.

Table 1 - Available video modes of depth and RGB sensors. The highlighted rows represent the video modes used in the application.

Sensor	**Resolution**	**FPS**	**Sensor**	**Resolution**	**FPS**
Depth	*640 x 480*	*30*	RGB	1280 x 960	12
	320 x 240	30		*640 x 480*	*30*
	80 x 60	30		640 x 480	15

In addition, a special video stream option called mirroring can be activated or deactivated. By changing the mirroring option, the captured frames are delivered as the original or inverted. This can be useful depending on the application. For my purpose, receiving the mirrored frames was necessary to improve usability of the application's GUI. The virtual avatar, representing the person in front of the sensor, is facing the camera, and therefore it is facing the user in front of the screen. Another option is responsible for the synchronization of the depth and the RGB stream. If this option is enabled, the device is forced to only deliver synchronized frames of both sensors, which belong together. This is very useful, especially if the people in front of the sensor are moving.

By creating and starting a video stream, either depth stream or RGB stream, the current frame can be iteratively read in the application's loop-function. For reading the frames, a pointer to the current frame's memory is delivered by the framework. The copying has to be done with the *Marshal.Copy*-method, which is offered by the *.Net Framework 2.0* and copies data from an unmanaged memory pointer to a managed array in the application context. To do so, the application already has to know the length of the array that has to be filled up. This length can be calculated based on the previously configured video mode by multiplying the width and the height of the chosen resolution. After the copying has finished, a one-dimensional array contains the whole information of the frame. For better use, it can be helpful to transform it into a two-dimensional array by iterating through the elements and splitting up the array into rows. For the depth stream, the array contains depth values for each pixel. For the RGB stream a different process is possible. By preparing an empty Bitmap object with the correct width, height and color format, the frame can be copied to the application's memory with the command *updateBitmap*. Then the received bitmap can be saved to the file system or transformed into a two-dimensional texture. The texture-based visualization of the RGB stream can be seen in Figure 8 - (2).

The framework offers the possibility of adding custom callbacks for special events related to the devices. Those events are triggered when the state of a device changes, a device is disconnected or connected. My application reacts to those events and gives feedback for a better user experience, such as "Please connect a device", or "Please restart your application".

The sensor is initialized and configured with the basic *OpenNI*-functions. Furthermore, the available sensors can be accessed and their video streams can be copied into the application and visualized for example on the application's stage as a two-dimensional texture on a plane. These are the necessary basics the *NiTE*-functions are built on, as you can read about in the next subsection 4.2.1.2. Every *OpenNI*- and *NiTE*-function always returns with a status that indicates whether something went wrong or everything is alright.

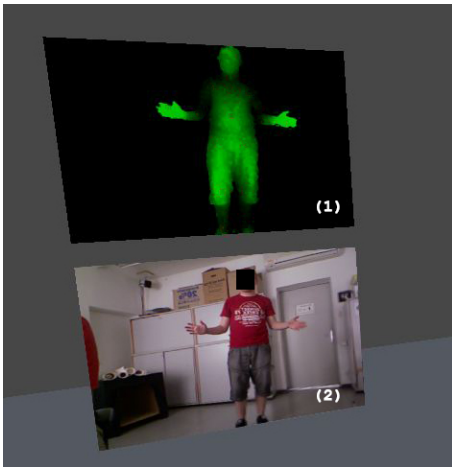

Figure 8 - Screenshot of the application's stage (1) Texture-based visualization of the histogram-equalized depth stream segment relating to the tracked user. (2) Texture-based visualization of the RGB stream.

4.2.1.2 Depth and World Coordinate Systems

As described in *OpenNI's* documentation [48], the framework uses two different coordinate systems for the representation of depth values, called depth and world representation.

The depth coordinates are the native data representation, where the depth value of each pixel of the depth map represents the distance between the camera plane and the object the pixel belongs to. The x- and y-coordinates are the location on the map, which is a two-dimensional array. The origin of the map is the top left corner of the field of view.

The world coordinates represent points in a quite common 3D Cartesian coordinate system. The origin of this coordinate system is the camera lens. Every point is defined by the 3 coordinates x, y and z. The x-axis goes from left to right, the y-axis goes from bottom to top, and the z-axis begins at the camera lens and leads straight ahead into the scene.

The depth representation is the projection of the scene on the IR-sensor. If the sensor's angular field of view and resolution are known, each pixel can be converted. For the conversion the stream-pointer itself has to be provided to these functions, because it determines parameters for the specific points. During conversion from depth to world coordinates or vice versa, the z value always remains the same. The conversion is done by the framework's coordinate-converter.

4.2.1.3 Basic NiTE-Functions

Similar to the *OpenNI*-object, by initializing the *NiTE*-object, the framework delivers the current *NiTE*-version. *NiTE* offers functions for automatic tracking of users in front of the depth sensor. Therefore, a user tracker has to be created for the current device. The framework numbers the tracked users automatically, beginning with number one.

In addition to the video streams of *OpenNI*, the user tracker is used as a stream too. Again, by using the *Marshal.Copy*-method, the data can be copied into an array in the application context. This time a labeled map of all visible users in the current frame is delivered. This map has properties similar to the depth stream. But instead of the depth values, each pixel has an integer value that stands for a label. If the pixel's value is zero, no user, or at least no tracked user, is visible at this point. A value unequal to zero stands for the number of the tracked user the pixel belongs to in the current context. With this information the depth frame can easily be segmented into the foreground and the background. The foreground represents the tracked user in front of the sensor, and the background is the rest of the depth image including the floor, the walls and any other object in the scene.

Figure 8 - (1) shows the two-dimensional texture of the segmented depth stream. The colored parts represent the tracked user, whereas the rest is colored in black, because it is identified as unnecessary background. For a better visualization the contrast of the tracked user's depth image is enhanced by equalization of the segmented depth histogram. Therefore, the depth values of the segmented depth stream are analyzed. Based on the lowest and the highest depth value, the color representing each pixel is linearly interpolated. The lightest pixels are closest to the sensor, whereas the darkest pixels have the biggest distance to the camera.

By converting the pixels of the tracked user into world coordinates, my application creates a point cloud representing the user in front of the camera. Therefore, one of my subroutines iterates through all points of the depth image. Each point is converted from depth coordinates to the world coordinates using the framework's coordinate converter. This requires the memory pointer to the video stream, which holds information about the field of view and the resolution of the sensor, the x, y and z-coordinates of the depth point, as well as placeholders for the x, y and z-coordinates of the converted world point. Once the world coordinates of all the points of the user's point cloud are available, a sphere for each point is generated and displayed on the application's virtual stage. In Figure 9 - (3) the point cloud representing the tracked user is visualized as tiny white spheres. Spheres are primitive game objects in the Unity3D game development environments.

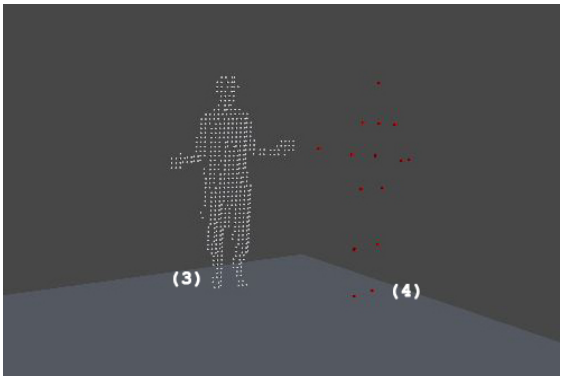

Figure 9 - Screenshot of the application's stage (3) The point cloud representing the tracked user, visualized as tiny white spheres. (4) The skeletal joints representing the tracked user's skeleton, visualized as tiny red cubes.

Once the framework tracks a user, a skeleton tracker can be started for this user. By simply passing the user's identifier, which is equivalent to the label of the label map, the framework tries to track the skeleton. The more the user moves in front of the sensor, the easier it is for the framework to identify the skeleton. If the skeleton is validly tracked, the application can request each of the joints, identified by the joint type's identifier. As already mentioned in 4.1.2 and as Sinthanayothin et al. [38] and Chaaraoui et al. [9] briefly explain in their papers, *OpenNI/NiTE* provides a set of 15 different joints. These joint types are listed in Table 2, together with their numeric identifier. By requesting the different joints, the world coordinates are received. With this information they can be displayed on the application's stage, similar to the point cloud mentioned before. Each joint consists of a position, an orientation and a confidence factor for both values. These confidence factors stand for the reliability of the given position or orientation values. For example, the higher the confidence factor, the more accurate is the position. This is necessary, because if parts of the user's body are not visible for the sensor, the hidden joints are interpolated by the frameworks logic, and thus have low confidence. In Figure 9 - (4) the skeletal joints representing the fully visible tracked user's skeleton are visualized as tiny red cubes.

As it is possible to convert coordinates from depth to world system, the coordinate converter is also able to convert it the other way round. In my application, after the skeletal joints are retrieved and visualized on the application's stage as 3D cubes, they are converted into the depth coordinate system. This process results in x and y-coordinates of the depth image the joints belong to. Based on this information, it is straightforward to paint little squares, e.g. with a side length of 10 pixels for better visibility, around the joints' positions on the depth image. In addition to Figure 8 - (1), Figure 10 shows the texture-based visualization of the histogram-equalized depth stream segment that relates to the tracked user. The skeletal joints are additionally visualized as tiny red squares.

Table 2 - The different joint types provided by the framework, with numeric identifier. The highlighted rows represent the joint types that are used in the application.

Id	Joint Type	Id	Joint Type	Id	Joint Type
0	Head	5	Right Elbow	10	Right Hip
1	Neck	6	Left Hand	11	Left Knee
2	Left Shoulder	7	Right Hand	12	Right Knee
3	Right Shoulder	8	Torso	13	Left Foot
4	Left Elbow	9	Left Hip	14	Right Foot

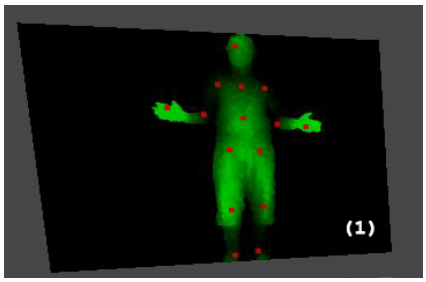

Figure 10 - Texture-based visualization of the histogram-equalized depth stream segment relating to the tracked user, displayed on the application's virtual stage, similar to Figure 8 - (1). The skeletal joints are additionally visualized as tiny red squares.

4.2.1.4 Tilting the Camera

In contrast to the previous *OpenNI*- and *NiTE*-versions, the up-to-date versions do not need a calibration pose any more. When the person in front of the sensor is moving around a bit, the framework is much faster at identifying people and tracking their skeleton. Therefore, I implemented a camera-mover script, which is responsible for tilting the whole Kinect sensor bar upwards and downwards until a user is identified and tracked. By tilting the sensor, the perspective view of the scene changes. Because objects near the sensor move differently to objects far away, an effect is reached that is similar to the moving of the people in front of the sensor. In addition, there is the possibility to manually adjust the tilt angle of the camera via application.

4.2.1.5 Building Skeletons with Bones

As shown in Figure 10 and especially in Figure 9 - (4), the displaying of the skeletal joints themselves does not result in a very meaningful representation of the user's skeleton. To make it more usable, my application generates skeleton bones. Similar to Liu et al. [23], I, too, used the 3D coordinates of the joints for further processing, as mentioned in section 3.1. Based on a script for the drawing of skeleton bones provided by Christian Schönauer [36], Interactive Media Systems Group of TU Vienna, I developed a class for the automatic generation of skeleton bones. This class manages the calculation, generation and transformation of the virtual bones based on two joints, e.g.

the upper arm bone is generated based on the information of the shoulder and the elbow joints. It offers functions for updating the position, direction and scaling, as well as for building the hierarchy of the skeleton bones. This hierarchy means for example that the forearm is related to the upper arm, the upper arm is related to the torso, and so on. But one of the most important and fundamental functions is to visualize the bones. Based on the basic script the algorithm builds up a cylinder, another type of primitive game object, transforms, rotates, scales and displays it at the correct position on the stage. Figure 11 shows a skeleton representation of the tracked user's bones, based on the previously fetched skeletal joints.

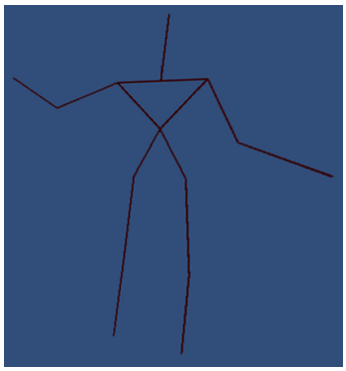

Figure 11 - Skeletal joints and bones build up a representation of the tracked user's human skeleton.

4.2.1.6 Alignment and Rotation

The skeleton in Figure 11 is a representation of the tracked skeleton. It illustrates that the user in front of the sensor was not standing straight ahead, facing the camera exactly from the front and along the camera's z-axis. The left parts of the skeleton seem to be farther away than the right parts. This is observable when comparing the shoulders, the hips or the forearms. Another orientation that can affect the visual appearance and possibly the further calculations is whether the user is leaning forward or backward. To get rid of these influencing displacements and orientations, two more bones are introduced. These two bones represent the vertical and horizontal body axes. The horizontal body axis extends the connection between the two shoulder joints. Whereas the vertical body axis extends the virtually non-visualized connection between the joints of the neck and the torso. The two body axes are orthogonal to each other and the neck is the intersection point. Based on the two body axes, my application translates the whole skeleton in order to move its intersection point to the origin of the application's 3D coordinate system. After this step, the neck joint is in the origin of the coordinate system, which is a basic requirement for all further manipulations, the mirroring and the measuring.

As a next step, the axes' orientations are calculated. In Unity3D there are different angles of orientations and rotations. At first, there are the so-called Euler angles. These consist of x, y and z-angles, and represent the rotation of each axis in degrees. In addi-

tion, there are local Euler angles, which show the rotation in relation to the parent object's rotation. The rotation attribute stands for the rotation in world space and is stored in so-called quaternions. Again, the local rotation shows the rotation in relation to the parent object's rotation. A quaternion is a special way of representing a rotation. This is necessary to avoid the gimbal lock, which is the loss of one degree of freedom as soon as two rotation axes become parallel. Therefore, all rotations in Unity3D are represented as quaternions, which are based on complex numbers. The functions for rotations used Euler angles, which are indispensable for the measurements as well. Hence, my application has to work with Euler angles. In general, working with rotations of game objects depends on many factors. They are influenced by translations, rotations and scalings of themselves and especially of their parent objects. To solve this and avoid similar problems, my algorithm projects the objects onto the xy-, xz- and yz-planes. By projecting the vector onto the xy-plane for example, its y-coordinates are removed. It further calculates the angles between the projected vectors and the coordinate system's x-, y- and z-axes. Unity3D provides this angles function, used to calculate the angles between two vectors. For resulting in the same rotation, it is possible to use a positive or a negative angle. An important characteristic is that Unity3D's angles function always returns the acute angle, which is the smaller of the two possible angles, because it disregards the vectors' orientations. But, accounting for this fact, my algorithm checks the circumstances and adapts the angles as soon as the obtuse angle is required. In detail, it tests whether the projected vector's coordinates point into the same or the opposite direction as the coordinate system's axes they are compared to. Depending on this test, either the acute angle is used or the obtuse angle is calculated by subtracting the acute angle from 360 degrees.

Based on the body axes' orientation angles, the adjustment can be implemented. For this purpose, a rotation container, which is the parent object of the whole skeleton, is rotated using the quaternions function for Euler rotation by passing the calculated x, y and z-angles. By using the angles of the vertical and the horizontal body axes, only one rotation is necessary. The vertical body axis is used for x-rotation, to reach an orientation of 90 degrees in order to counteract the forward or backward leaning. For y-rotation, the horizontal body axis is used in order to get the direction of 180 degrees. This is necessary for adjusting the body to be looking exactly towards the camera, with the horizontal body axis parallel to the camera plane. And, finally, we used the vertical body axis for the z-rotation, to have both body axes perfectly aligned with the x and y-axes of the coordinate system. After this rotation an additional test is done, in case the rotation did not work quite well, and a special refinement step is used to align the whole skeleton. Finally, the directions of all bones of the skeleton are updated, which is an inevitable step for the reproducibility and the stability of the whole application. In Figure 12, the aligned and rotated skeleton is visualized with red-colored bones. The thin, blue lines represent the body axes.

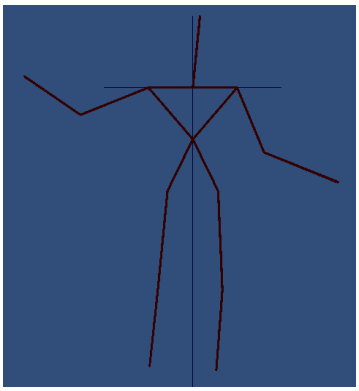

Figure 12 - Rotated, aligned skeleton.

Because usually people have different body sizes, it would be useful to normalize all the joints and bones, similar to the process as Chaaroui et al. [9] explained in their paper. But, as already mentioned in section 3.1, it is not used at this point of my application. Since at a later development state the prototype works with the point clouds of the hands, it would be an unnecessary execution cost to rescale the whole point cloud. It is much faster to normalize the error values during or after the measuring. This has the same effect, but takes less execution time.

The point cloud representing the tracked user's depth image is built up of spheres. These spheres also have to be aligned and rotated with the skeleton to reach an optimal visualization. Before the spheres are generated, the z-offset of the torso joint is selected, if the skeleton already is available. Otherwise, as a fallback, the depth images middle point is used as a reference point for the z-offset. Then the z-offset is subtracted from the depth point's world coordinate, to reach a perfect overlay of the skeleton and the point cloud, as visualized in Figure 13.

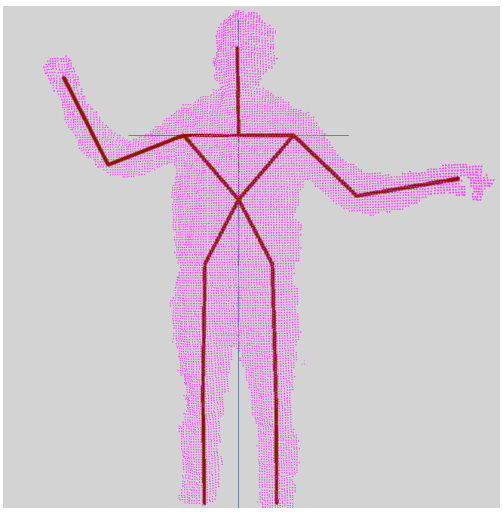

Figure 13 - The overlay of the skeleton and the depth image's point cloud.

4.2.1.7 Mirroring the Joints and Bones

The main purpose of this application is the measurement of the divergence between angles of the left and right shoulders and elbows. Therefore the application has to visualize this divergence as good as possible, the next processing step is the mirroring of one arm on the vertical body axis. First of all, primitive cubes are created for the duplicates of the joints of the left shoulder, elbow and hand, which are colored green for better perceptibility. For each duplicated joint the rotation is set to zero, the so-called identity, which simply means that there is no rotation at all. In addition, the scale values of the original joints are copied to the duplicated joints. The yz-plane acts as the mirroring plane. That means that we have to mirror the x-coordinate of the joints' position on the neck joint. To do so, the algorithm takes the horizontal distance from the original joint and subtracts twice the horizontal distance between the original joint and the neck to get the new x-coordinate of the mirrored joint's position. The y and z-coordinates are copied from the original joints. After that, the mirrored bones are generated based on the mirrored joints. In addition to the upper arm and the forearm, the clavicle, which connects the neck with the shoulder, is built as well. A reference to the original bone is saved too. Similar to the joints, the bones are also colored green. Finally, the correct object hierarchy is created, and the directions of all mirrored bones are updated. As all the bones and joints in the application are named, the mirrored objects become meaningful names as well. The bones' names get the prefix "mirrored_", and the joints names get the prefix "mirrored_joint_". Figure 14 shows the skeleton with the mirrored joints and bones, colored in green.

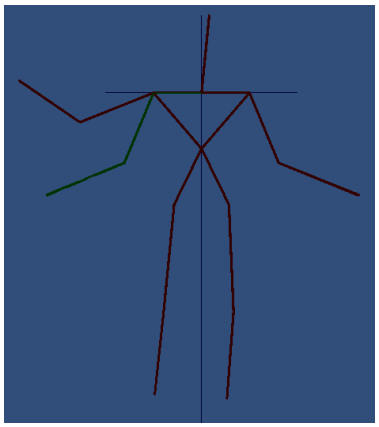

Figure 14 - Skeleton with mirrored joints and bones.

4.2.1.8 Measuring the Skeleton

Based on the delivered joints, the generated bones, the alignment and rotation of the skeleton and the mirrored joints and bones, the measuring of the divergence can be done and the error values can be calculated. The first thing my algorithm does is to calculate the angles of the shoulder and the elbow joints of the mirrored left and the original right body half. Therefore, the orientation of the two bones that are connected in a joint has to be updated. This is done by projecting the bones onto two orthogonal

planes and measuring the angles between the projections and the corresponding axes. In order to get a joint's y-rotation, the bones' y-rotation values are subtracted from each other and further reduced by 180 degrees. If the angle is lower than zero, 360 degrees are added in order to avoid negative angles. The z-rotation is calculated in the same way. The result is a two-dimensional rotation vector for each joint. This vector contains the transversal and the longitudinal rotation, meaning the rotation of the x- and the y-axis of the joint. The following code-sample represents this functionality.

```
public Vector2 calculateAngles (bone1, bone2) {
   xyAngle = bone1.orientation.z - bone2.orientation.z - 180;
   if (xyAngle < 0) {
      xyAngle += 360;
   }
   xzAngle = bone2.orientation.y - bone1.orientation.y - 180;
   if (xzAngle < 0) {
      xzAngle += 360;
   }
   return new Vector2(xyAngle, xzAngle);
}
```

To calculate the divergence, the coordinate pairs simply have to be subtracted. By setting the differences between the rotation values as absolute, or by squaring them, the algorithm delivers two different error metrics for each of the two joints. In order to get the total absolute or squared error of both joints, the values are summed up.

4.2.1.9 The First GUI

To give the user of the application the possibility to control all the functionalities, a graphical user interface is necessary. As in Unity3D the *OnGUI* method is called every time the GUI is repainted, it is the right place to insert the UI's logic. In addition to the menu elements, dialogue windows can be generated as well. For the first version of the prototype, with the previously explained functions, I created the GUI as shown in Figure 15. This GUI is split into three windows. The top window is for closing the application. With a click on the "Close Application" button, a prompt appears, asking the user if he or she is sure to close the application. This is a very important function, because all tracked and measured data are saved to files, the connections are closed and the resources are freed.

Figure 15 - The first prototype's GUI.

The middle section is for controlling the main functionality. Apart from the buttons for recording a frame and calculating the error values, a short statistic of the current setting is displayed, containing the number of connected devices, the current tilt angle of the camera, the number of users registered so far, the visible skeleton's state and the last calculated delta of the angles.

The lower section contains buttons for the controlling of different functionalities necessary for analyzing details of the tracked information. The "change from elbow to shoulder" button triggers the visual overlay of the mirrored left arm and the original right arm. It rotates the mirrored arm at the shoulder joint, based on the upper arms' orientations. The result is that the upper arms and the elbows are congruent, as visualized in Figure 16. This is a nice visual helper for directly seeing the divergence of the elbow joint. Simultaneously the button changes its label to "change from shoulder to elbow". It is straightforward that a click on the button will undo the rotation. The other three buttons are for disabling or enabling the visibility of the point cloud, the skeleton, and the mirrored bones and joints. By clicking them and depending on the current state of visibility, the related objects are hidden or displayed.

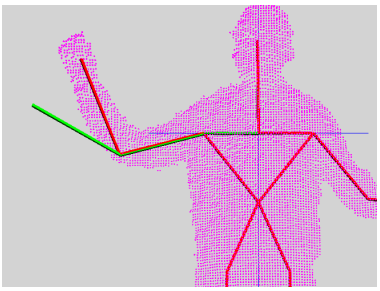

Figure 16 - The visualization of the functionality "change from elbow to shoulder".

4.2.2 The Second Prototype

In this chapter I will explain the new and improved functionalities, based on the expert's feedback, functional requirements after the test and evaluation of the first prototype. The most important part deals with the segmentation of the hands' point clouds (4.2.2.1), followed by the mirroring of the hand point cloud (4.2.2.2) and the measuring of the divergence between the two hand point clouds (4.2.2.3). In addition, this subsection provides information about the XML-file-based configuration file (4.2.2.4), the export of the analyzed data as text files (4.2.2.6), as well as extended camera controls (4.2.2.7). Finally, some improvements regarding the GUI (4.2.2.4) are followed by some improvements of the runtime (4.2.2.8).

All the changes and new functionalities that are explained in the following subsections put together result in the second prototype. This prototype version is fully functional, and could already be used for testing people's proprioception. As visualized in Figure 17, the GUI of the application's second stable version including the virtual stage contains the new result window with the details of the current pose (1), the down-sampled and live RGB image, as well as the improved menu boxes for controlling the application's functions (3).

After evaluating the second prototype with the experts, again some improvements were suggested for the final development phase. During the segmentation phase, which takes some seconds, and the other processing phases, a pop-up prompting the current processing state should be displayed. In order to speed up the segmentation some point cloud optimizations are necessary. Another functionality is the automated tracking of a sequence of multiple frames for testing the stability of the measurement algorithm. The functions of the debug modes are extended in order to save screenshots of the RGB textures, depth textures and the applications' stage. Lastly the possibility to track a completely new person without having to restart the application needs to be implemented.

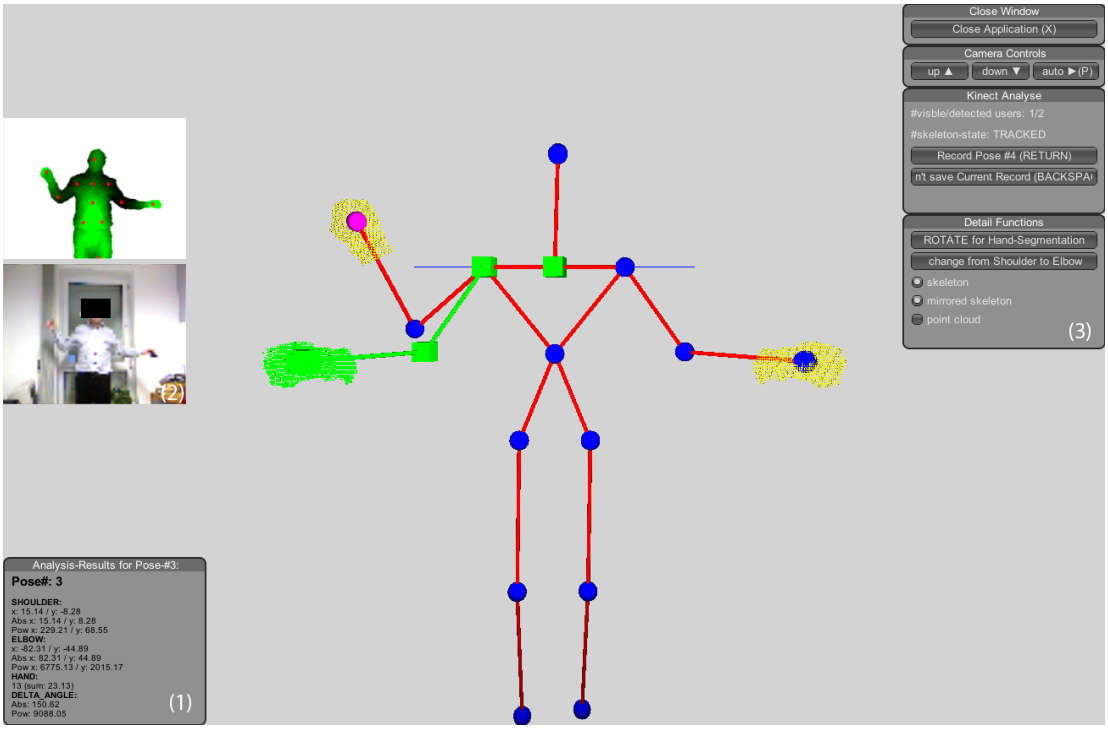

Figure 17 - Screenshot of the second prototype version of the application. (1) The result window displaying the analysis of the current pose. (2) The down-sampled RGB image. (3) The improved GUI for controlling the application's functions.

4.2.2.1 Segmenting the Hands' Point Clouds

For the segmentation of the point clouds representing the hands, a bounding box is used, similar to the system of Du and Zhang [13] mentioned in section 3.3. While they used the hand bounding box based on the coordinates of elbow and wrist and the direction the forearm points to, in my case the bounding box has to be more flexible. The fingers and the hand itself can have any possible rotation. Hence, a bigger box is used for the pre-selection of the points, followed by a refinement step. The positioning of the bounding box is more complicated than Huang et al.'s [19] method because of the differences between the frameworks. While they used *Microsoft's SDK*, the joint representing the palm of the hand itself is not available in the *OpenNI/NiTE* framework. Therefore, the box has to be bigger and repositioned for my purposes.

Depending on the angles of the elbows, a disadvantageous position in front of the camera, or loose clothes of the test person, it can happen that the tracked elbow joints are not exactly on the right position, but a bit offset. To avoid continuing with these false positions when segmenting the hand point clouds, the lengths of the forearms are calculated. The difference between the two lengths is defined for later use. Now the parent object of all skeletal joints and the whole point cloud is transformed by using rotations in order to align the forearm with the y-axis, the so-called up-vector, and translations to move the hand joint to the coordinate system's origin. Then Unity's *Bounds*-object can be used to generate the bounding box. The side lengths of the bounding box are scaled with the normalization factor, which represents the length be-

tween the torso and the neck joint. The iteration over all spheres takes place to check whether they are inside the bounds or not. The spheres inside the bounds are saved as a part of the resulting point cloud of the hand. The hand's point cloud is colored yellow for better visual feedback, as displayed in Figure 18. Then the parent object is translated and rotated back again.

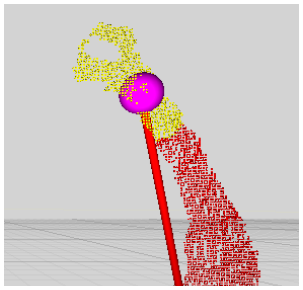

Figure 18 - The segmentation of the hand's point cloud.

Depending on the position of the hands and based on the big bounding box, unwanted spheres might be selected. In a refinement step they are then unselected again. Figure 19 visualizes the outcome of the refinement step, where the points of the head are not part of the new point cloud any more, although they are within the bounding box. Therefore the sphere nearest to the hand joint is selected as starting point of the new point cloud. The algorithm then loops over the pre-selected spheres and checks the distance to the spheres that are already in the new point cloud. As long as new points are selected, this neighborhood continues to grow and selects all hand-related points. The squared magnitude between two points is used for comparison, and the distance has to lie under a predefined maximum. In addition, a reverse loop is added in order to improve performance, because the new spheres are added to the list and are therefore located at the end of it. By looping through the new list of points in the reverse direction, a possible match is found much quicker. Then the spheres that were not selected are reset again. Then, the point cloud containing the unused spheres is deactivated because they are not used any more.

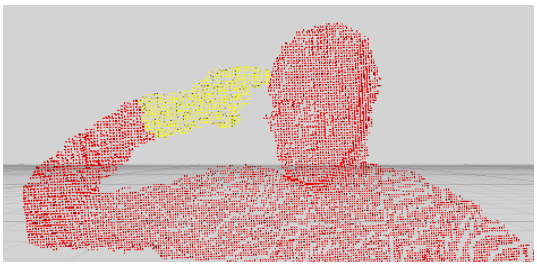

Figure 19 - The segmented point cloud after the refinement step.

4.2.2.2 Mirroring the Hand Point Cloud

The next step necessary for the comparison is to mirror the previously segmented hand point cloud, similar to the skeletal joints and bones in subsection 4.2.1.7. The algorithm loops through the cloud containing the hand points and mirrors each of the spheres. The original sphere is duplicated and recolored in green. The outcome is displayed in Figure 20.

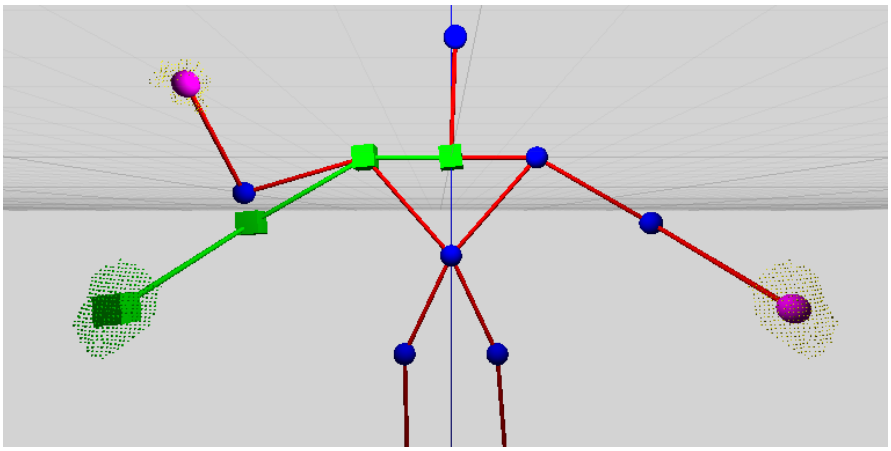

Figure 20 - The mirrored hand point cloud.

4.2.2.3 Measuring the Point Cloud Distance

For measuring the difference between the point clouds of the two hands, the hand point clouds, together with the hand and elbow joints and the forearm bones, are rotated to be parallel to the up-vector, and translated to the origin, similar to the segmentation process. But this time not both hand joints are aligned with the origin. The previously calculated difference between the lengths of the two forearms is used for the exact overlay of both hand point clouds. Figure 21 shows the overlay of two hand point clouds when both hands are in almost the same pose on the left. For a better usability and visibility of the overlay, I programmed a debug function that moves the two forearms apart, which is visualized in Figure 21 on the right.

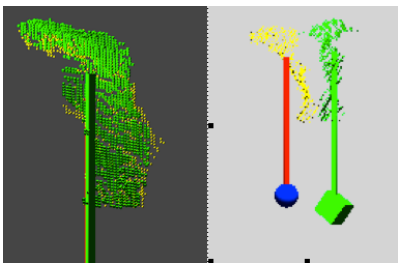

Figure 21 - Exact overlay of two hand point clouds of the same hand position, on the left. The same two hand point clouds, from another perspective, moved apart for visual debugging, on the right.

Once the two hand point clouds are adjusted, the measurement of the difference between them can be processed. My algorithm, implemented as a variation of the *ICP* mentioned in section 3.2, calculates the squared distance values between the point clouds. After the segmentation and mirroring, as explained in the previous subsections 4.2.2.1 and 4.2.2.2, and re-rotation based on the rotations of the shoulder and elbow joints and the repositioning based on the hand joints, my varied implementation of the *ICP* is triggered. Every cloud point of the one hand is compared to every point of the other hand to find the relating points with the smallest quadratic distance. Basically the squared magnitude is used, because it performs much faster and it penalizes the bigger differences more. The distance values are then summed up to reach the total squared distance. Similar to Ly et al.'s [25] mean point distance and root mean squared error metrics, the sum is divided by the amount of compared points to get the mean error values, in my case the mean squared distance. To achieve normalized error values independent of the person's size, the mean squared distance as well as the total squared distance are further divided by the normalization factor, as visualized in the following formula. The normalization factor is the distance between the torso and the neck joint as mentioned in section 3.1.

$$\frac{\sum min(sqrMagnitude(p_i - q_j))}{pointsCount * normalizationFactor}$$

4.2.2.4 GUI-Improvements

Based on the expert's feedback after the first prototype's evaluation, the possibility to type in the name of the test person at the start of the application is realized by displaying a pop-up dialog. This dialog contains a text box for entering the name and a button for saving the value and closing the pop-up, as displayed in Figure 22.

Figure 22 - Pop-up dialog for entering the test person's name.

Another improvement is the visualization of the pose number. Therefore, the record button is modified and displays the number of the next pose to record, as visualized at the top of Figure 23. In addition, a button for "not saving the currently recorded set of data" was introduced, which is displayed at the bottom of fFigure 23. This functionality is necessary in order to delete the current record including its values and data in case something went wrong. The letters in the brackets represent the keyboard shortcuts for the buttons' functionalities. These letters can be configured in the configuration file, as explained in subsection 4.2.2.5.

Figure 23 - The record button including the pose number, at the top, and the „don't save current record" button, at the bottom. The letters in the brackets represent the keyboard shortcuts for the buttons' functionalities.

After the measurements and calculations are done, the results have to be visualized. A window is displayed in the left bottom corner. This window shows detailed information of the currently recorded and measured pose. The information contains the total error values, as well as partial information regarding the shoulders, elbows and hands, and is visualized in Figure 24.

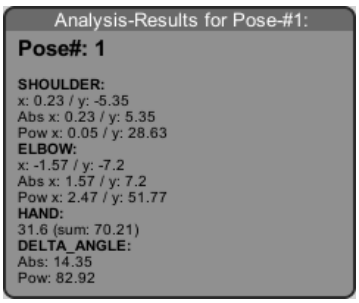

Figure 24 - The result window displaying the analysis of the current pose.

4.2.2.5 XML-File for Configuration

At this stage of development it was necessary to implement some customizable configuration for defining different types of flags, keyboard shortcuts, mouse buttons, debug mode settings or the depth point cloud's level of details. This configuration is loaded at startup and overwrites the predefined default values of the application. After using an XML-serializer for de-serializing and parsing the configuration, a loop iterates through the settings. Based on the type of the variable, the value is parsed as integer, float or boolean. After loading the keyboard shortcuts are displayed in the GUI. As visualized in Figure 23, the related shortcut is indicated in brackets after the button's labels.

4.2.2.6 Exporting Data as Text Files

For the purpose of post-processing, analysis and persistence, the measured and calculated values are saved permanently as a comma-separated file. The class for exporting, written by Chris[1] and modified by Munchies[2], uses a list, representing the rows of the spreadsheet. Each row is defined as a dictionary, with associative indices such as "pose#", "deltaangle", etc. and the corresponding values. When the application is

[1] http://stackoverflow.com/posts/2422212/revisions
[2] http://stackoverflow.com/posts/19959674/revisions

closed, the whole list of dictionaries is exported, named with the proband's name, the date and the file ending ".csv" into the exports-directory as a comma-separated file.

4.2.2.7 Camera Control

The camera mover in the first prototype (see subsection 4.2.1.4) was automatic. There had been no possibility to control the movement manually via the user interface. After the first evaluation, the experts asked for a control mechanism. Therefore, I created a GUI and the functionalities for pausing and stopping the automatic movement. This automatic movement is realized with an elapsed event handler that calls a function based on a timed interval. The possible tilt angle goes from approximately +27 to -27 degrees. I also added buttons for manual tilting up and down, as shown in Figure 25. This is suggested for the adjustment of the sensor's perspective.

Figure 25 - The GUI for the camera controls.

4.2.2.8 Runtime Improvements

For the visualization of the depth image of the tracked user on the virtual stage, lots of little spheres are used. As the initialization of huge amounts of game objects like spheres or cubes takes a lot of time, I generated packages of spheres, each containing thousands of objects, and saved them as a prefabbed asset, a so-called *prefab*. By initializing this prefab in the application, the time used for the creation of 1000 spheres decreased from several seconds to a few hundredths of a second.

As another measure for improving the runtime, I further decreased the resolution of the color image, because the permanent rendering of the color texture takes a lot of computation time and results in a low frame rate. The color image is not very important for the work with the application and displaying only every 64th pixel leads to an increased frame rate.

4.2.3 The final prototype

In this chapter, I will explain the final prototype's new and improved functionalities, based on the expert's feedback and functional requirements after the test and evaluation of the previous prototype version. The first part of this subsection presents a solution for decreasing the processing time of the point cloud segmentation (4.2.3.1). Furthermore, functionality for debugging, testing and evaluating the application are explained, such as the tracking of multiple frames (4.2.3.2) or the saving of several different screenshots of RGB and depth textures, as well as screen captures of the whole applications' stage (4.2.3.3). Finally, the solution for increasing the tracking quality and accuracy (4.2.3.4) is followed by steps for improving the application's usability (4.2.3.5).

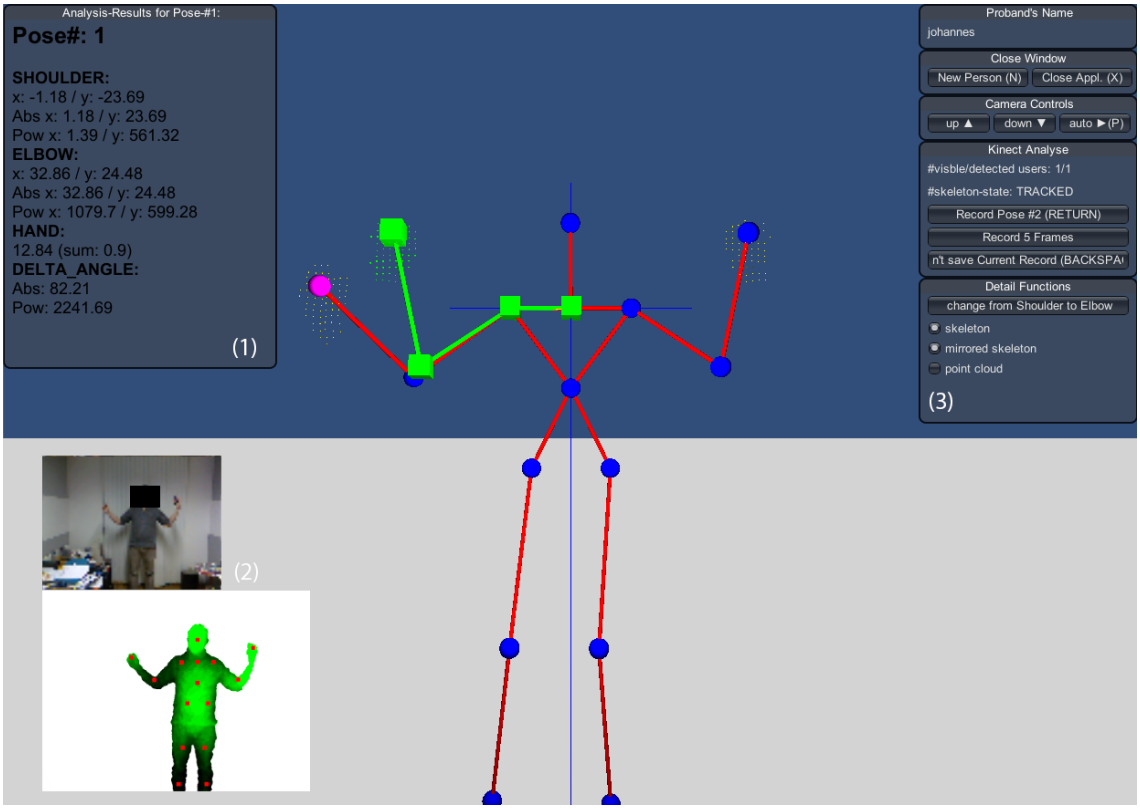

Figure 26 - The final prototype's application window. (1) The prominent result window displaying the analysis of the current pose. (2) The smaller monitor containing the down-sampled RGB image and the bigger one visualizing the real-time depth image with the overlay of the live skeletal joints. (3) The GUI including the debugging functions like recording multiple frames and the current test person's name.

The following subsections describe the requested changes and functionalities as suggested by the experts during the prototype tests of the prior versions. Figure 26 visualizes the application window of the final prototype that already contains the implemented functions, objects and interface elements. The result window displaying the analysis of the current pose (1) is now very prominent, and the font size of it's content is much bigger compared to the previous prototype version in Figure 17. A small box contains the real-time and down-sampled RGB image that is primarily used for the adjustment of the camera on the stage in order to see the whole body of the test person. And a bigger box shows the real-time depth image with the overlay of the live skeletal joints (2). The GUI includes the debugging controls such as the recording of multiple frames, a button for restarting the session for a new test person, and a box containing the current test person's name (3).

4.2.3.1 Optimization of the Point Cloud

The segmentation of the hand point clouds requires a great computational load because the algorithm iterates through all visible points of the full body depth map, as explained in subsection 4.2.2.1. Furthermore, each point is compared to the bounding box of both hands in order to find the segmented point cloud.

For optimizing the segmentation and refinement processes, the size of the point cloud is drastically reduced. Therefore, based on the depth map, squared boxes with side lengths of 150 pixels are generated around the coordinates of the hands' joints. Only points within these squares, marked as red boxes in the left part of Figure 27, stay in the user's point cloud and are used for the segmentation and the refinement of the hand point clouds. This optimization results in a significant reduction of the processing time. The right part of Figure 27 visualizes the outcome of the segmentation, based on the optimized point cloud as it is displayed on the applications' stage.

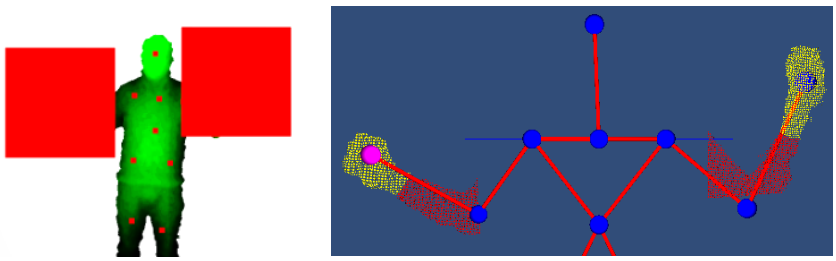

Figure 27 - Depth map of the tracked user, including the skeletal joints and the two big, red boxes for optimization of the point cloud, on the left. The segmented hand points based on the optimized point cloud, on the right.

4.2.3.2 Multi-Frame Tracking

I developed some useful functions for debugging and testing. Apart from debug visualizations, outputs and loggings that were already mentioned, the possibility of tracking a sequence of multiple frames with only one click is the most important functionality. The number of frames to record is configurable in the configuration file, explained in 4.2.2.5, with a default value of 5 consecutive frames.

At first, all the frames are recorded because the tracking is done very quickly. After the frames are tracked and temporarily saved, the application iterates through the saved frames. Each frame is passed through the whole process, beginning with alignment and rotation, the overlay of skeleton and point cloud, segmentation of the hand point clouds, the mirroring and finally the measuring of the errors, followed by the saving of the data measured.

4.2.3.3 Saving Screenshots

For the analysis of the measured data after the test sessions, the saving of screenshots of the tracked scene is very useful. The application automatically saves the visualized textures of the RGB stream and the user depth map including the highlighted skeletal joints at the time of recording the frame. After the process is completed, the whole application window is captured and exported also as an image file. In this way an image of the application's visual output is saved as well, including the skeleton, the mirrored bones and joints, the segmented hands and all GUI-Elements. If, for example, a frame produces weird values, the therapist can lookup the corresponding screen-

shots. As they are labeled with the test person's name, date, time and pose number, it is straightforward to correlate the measurement data of the .csv-file and the corresponding screenshots.

4.2.3.4 Quality Improvements

The quality of the application relies on the quality of the skeletal joints, which are delivered by the sensor and the framework. For increasing the accuracy of the delivered positions of the skeletal joints, the time a user's skeleton is tracked has to be maximized. The longer the system tracks a user, the better the tracking and the more reliable is the generated data. Unlike the earlier versions, the final prototype starts tracking as soon as a user is detected and can be tracked, and only stops tracking at the end of the application or when a user leaves the sensor's field of view. By continuously tracking the skeleton, the framework refines the accuracy of the calculated joints for reaching the best possible quality. In addition, the algorithm does not need any time for reinitializing the skeleton tracker. Figure 28 shows the depth texture of the live captured user map, as it is displayed on the application's virtual stage. The red squares highlight the real-time positions of the skeletal joints.

Figure 28 - Depth texture of the live captured user map. The red squares mark the real-time positions of the skeletal joints.

4.2.3.5 Usability Improvements

For the final version of the prototype, some improvements regarding the user interface and its usability were suggested, as the evaluation of the second prototype showed. In the prior versions, the application had to be closed and restarted in order to begin a new tracking session for a new test person. Now, the menu for closing the application was changed a little bit. A second button was added, for restarting the measuring process for a new test person, as the left part of Figure 29 shows. By clicking on the "New Person" button, a pop-up dialog for entering the new person's name appears. This dialog is displayed in the right part of Figure 29. After the name is entered, the previously measured data is exported to a .csv-file, whereas the measurement counters, buffers and values are reset to their initial states. The session can thereafter be started for the new test person.

Figure 29 - The GUI menu for closing the current measurement session, either for closing the application or for restarting the measuring process for a new test person, on the left. The pop-up dialog, used for entering the new person's name, on the right.

For being aware of the current test person's name, a box was added to the area of the GUI menu. This box contains the name of the current test person and is visualized in Figure 30.

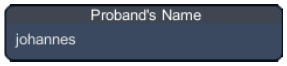

Figure 30 - The box displaying the name of the current test person.

After the user clicked the button for recording the pose, the GUI is blocked and the application does something. The processing of the data and the measuring of the error values take a while, but the user has no knowledge about the current state. Hence, I implemented a little pop-up for live stating of the current processing state. This output is displayed in Figure 31. The example shows that the mirroring of the skeletal bones and joints, as well as the hand point cloud is already done.

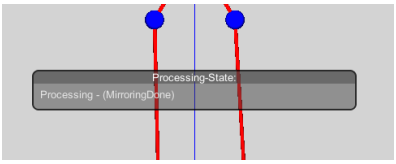

Figure 31 - A pop-up containing information on the processing state.

5 Evaluation

After the development of the application prototype in its final version, as previously explained in subsection 4.2.3, I carried out several evaluations for testing the stability and reproducibility of the measurements produced by the application. After an explanation of the evaluation methods in section 5.1, three hypotheses are listed in section 5.2, followed by the experimental design in section 5.3 and a description of the participants in section 5.4. The tests themselves are split into basic comparison of the skeleton values on the one hand, see section 5.5, and the hand point cloud part on the other hand, see section 5.6. In section 5.7 an additional evaluation is presented, provided by Elisabeth Soechting who tested fourteen normally developed children between seven and eight years of age. Finally the results are discussed in section 5.8.

5.1 Methods

The aim of the evaluation was to test the application prototype regarding stability and reproducibility of the measurement processes for the skeleton part on the one hand and the hand point cloud part on the other hand. Furthermore, the usability and feasibility of the test exercises were been tested. Finally, my colleague Elisabeth Soechting tested some children with a first version of the application in a separate study, and provided her results for this thesis.

5.2 Hypotheses

The following hypotheses were made to test the application prototype:

H1: The measurement of the skeleton is stable and reproducible.

H2: The measurement of the hand point cloud is stable and reproducible.

H3: All test persons are able to assume a set of special poses during the testing of their kinesthesia.

5.3 Experimental Design

The evaluation tested the two separate measurement processes of the application prototype. On the one hand the algorithm for measuring the error value of the skeleton, as described in subsection 4.2.1.8, was tested. On the other hand the error value calculation of the hand point cloud, as described in subsection 4.2.2.3, was evaluated.

For the evaluation of the prototype, a room with enough space was necessary to avoid disturbing objects in the sensor’s field of view. The room was about 7,6 meters long and 6,1 meters wide, whereas the area used for the evaluation was 3 meters long and

4 meters wide. The minimum space for working with the system is 3 meters long and 2,5 meters wide. Figure 32 shows the system's setup in the testing room. The sensor was fixed on a tripod 1,40 meters above the ground. The test persons would pose in front of the sensor keeping a distance of 2,10 meters. After some previous tests, this distance seemed to be ideal, because the upper body could be tracked to the full extent for all participants, and the legs were not required for tests at all. Their standing position was marked on the floor with a colored adhesive tape.

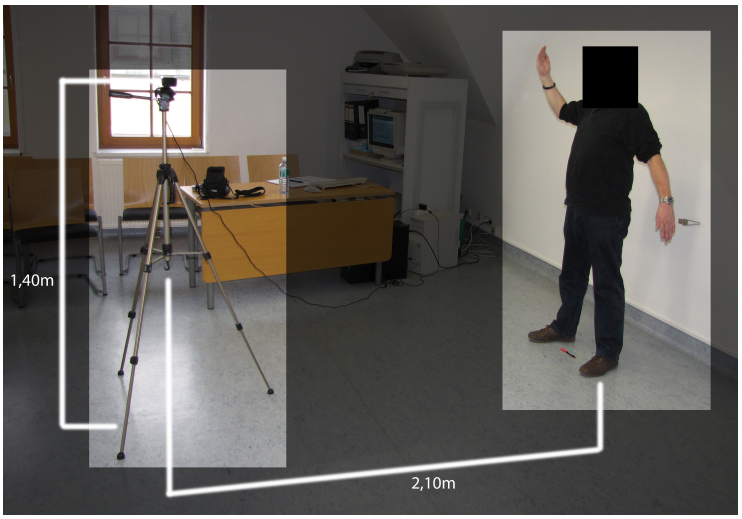

Figure 32 - The setup of the system for the evaluations with the test persons.

Some tracking results during the measurement of the equal skeleton poses had to be deleted because of temporary loss of the tracking accuracy, caused by a slightly defective USB-port of the test computer. The defect occurred during multi-frame tracking after some evaluation sessions, and the temperature of the computer was high. The normal single-frame tracking was not influenced by the defect. Detecting these outliers was straightforward, because during the multi-frame tracking the skeleton's tracking state changed from "tracking" to "not tracking" and the relating data items had absolute error values of about 5 to10 times higher than the normal values. Among other things, the depth values of the hand joints were not tracked with the same accuracy. This is visible in the elbows' y-angles of the extended logging functionality and in the screen captures in Figure 33. Other data items had to be removed because the participants moved before the tracking had finished or they did not stand still.

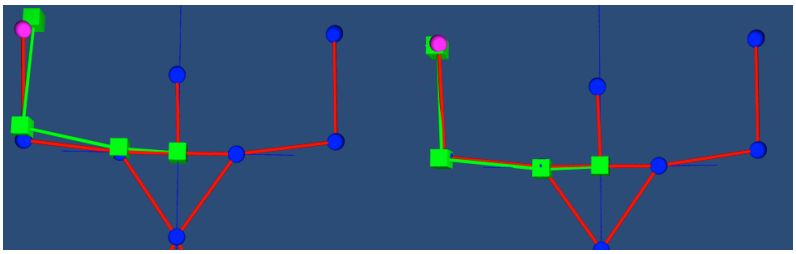

Figure 33 - Outcome of the incorrectly tracked joints, caused by a defective USB-port.

5.4 Participants

For the tests I found ten volunteers. The group of participants consisted of four women and six men. These ten people were aged between 21 and 60 years old and their height was between 1,58 and 1,97 meters. They were all normally developed and had no disabilities. Only two of them had previous experience with AR and the Kinect sensor. The participants agreed that the data measured and images taken be used in this thesis, and will receive a digital copy of this thesis as remuneration.

5.5 Evaluation of the Skeleton Analysis

Based on hypothesis H1, the first question to answer is whether poses, in which both halves of the body are aligned equally, are detected correctly and result in very low error values. For this purpose the test persons had to assume five different poses. These poses, as illustrated in Figure 34, were selected to evaluate the minimum error values. The left and right halves of the body, especially the arms and hands, were in the same positions. For the first pose the participants had to stretch out their arms horizontally. Pose number two was to bend the elbows orthogonally and point to the ceiling with the forearms. For the third pose they had to bend their elbows orthogonally and point to the floor with their forearms. The fourth pose was like pose number one, but the arms were lowered to about 30 degrees. Pose number five was quite similar to pose number two but for this pose the shoulder was rotated towards the front at an angle of about 45 degrees.

Figure 34 - The five poses for the first skeleton evaluation.

Each test person was aligned as exactly as possible to gather valid and useful tracking data, representing the reproducibility and stability of the test. By using the multi-frame tracking, as explained in subsection 4.2.3.2, multiple images of the same pose were recorded sequentially and processed afterwards. Thus, 420 valid tracking results for these five poses were taken. The statistical analysis of the resulting data for the absolute delta angles yielded a mean value of $\mu=32,51°$ with a standard deviation of $\partial=18,36°$, whereas the squared delta angles resulted in a mean value of $\mu=636,03°$ and a standard deviation of $\partial=877,64°$. Table 3 lists these analytical data, while Figure 35 visualizes the distributions in two boxplots, highlighting the mean values, the medians, and the quartiles. While the mean value and standard deviation of the absolute delta angle seem quite compelling, the box plot gives information about the whole distribution and shows the range of values. Since the values of the absolute error metric are quite low, the minimum values for equal poses are identified and the upper boundaries

for equal, or at least nearly equal, poses are an absolute delta angle of approximately 50 degrees on the one hand and a squared delta angle of approximately 1500 degrees. These values will be useful later on when testing normally developed children, as well as children with ASD.

Table 3 - Analysis of the first evaluation of the skeleton measurement.

	absolute delta angle	**squared delta angle**
mean value	32,51°	*636,03°*
standard deviation	18,36°	*877,64°*

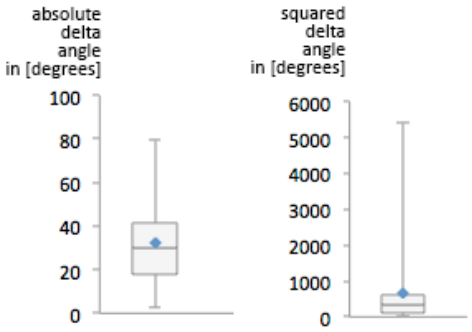

Figure 35 - Boxplots for the first evaluation of the skeleton measurement.

Based on hypothesis H1, the second question to be answered is whether the error values of different poses are detected as a divergence. Therefore, four sets of test poses were defined, as shown in Figure 36. Set one consisted of poses where one arm was stretched out horizontally, and the other arm was bent at the elbow orthogonally while pointing to the ceiling with the forearm. The second set consisted of poses where one arm was stretched out horizontally and the other arm was stretched out as well, but lowered to 30 degrees. Set number three was quite similar to set one, but the shoulder of the side with the bent elbow was rotated towards the front to about 45 degrees. The poses of the fourth set had one outstretched arm that was lowered to 30 degrees, and the shoulder of the other side with the bent elbow was rotated towards the front to about 45 degrees, similar to the poses of set three.

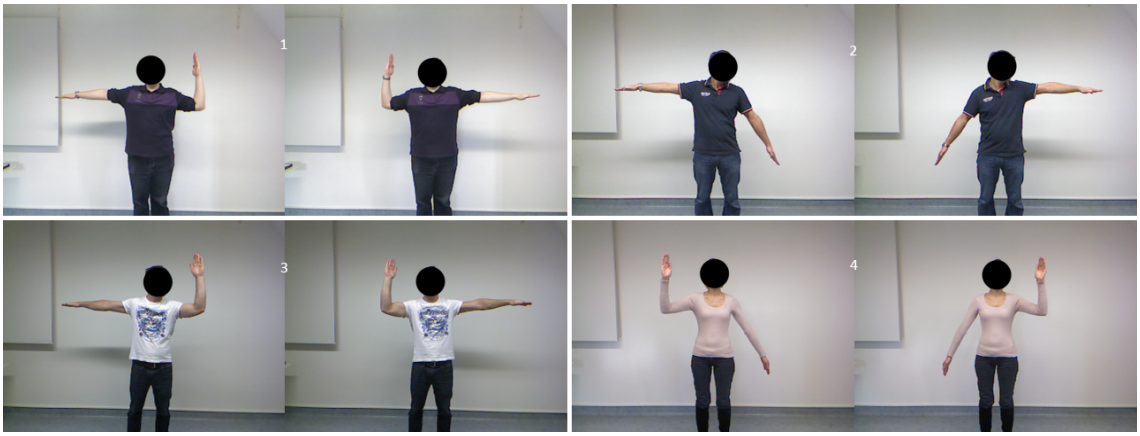

Figure 36 - The four sets of poses for the second skeleton evaluation.

The second evaluation resulted in 560 valid data items, divided into the four sets of poses. Table 4 lists the analytical data, while Figure 37 illustrates the distributions of the absolute and squared error values in two groups of boxplots. For the first set 146 valid results were recorded, with a mean value of $\mu=203,46°$ and a standard deviation of $\partial=53,29°$ for the absolute error metric. The squared delta angles resulted in a mean value of $\mu=20270,02°$ and a standard deviation of $\partial=10301,42°$. The 140 items of the third set and the 126 sets of the fourth set look quite similar, as listed in Table 4. All these values are much higher than the previously identified ones bound for equal poses. But the absolute error values of the 146 data items of the second set have a mean value of $\mu=92,05°$ and a standard deviation of $\partial=20,58°$. The squared error values have a mean value of $\mu=4560,61°$ and a standard deviation of $\partial=1140,25°$. Because in the second set of poses, only the shoulder joint of one side is rotated a little bit, both error metrics are significantly lower than the results of the other three sets, which is visible in Figure 37. However, the values are still much higher than the ones bound for the equal poses.

Table 4 - Analysis of the four pose sets of the second evaluation of the skeleton measurement. (“abs.”=”absolute”; “squ.”=”squared”)

pose set	**1**		**2**		**3**		**4**	
delta angle	*abs.*	*squ.*	*abs.*	*squ.*	*abs.*	*squ.*	*abs.*	*squ.*
mean value	203,46°	20270,02°	92,05°	4560,61°	265,32°	30536,73°	306,95°	33101,70°
standard deviation	53,29°	10301,42°	20,58°	1140,25°	66,11°	15519,18°	79,56°	16913,37°

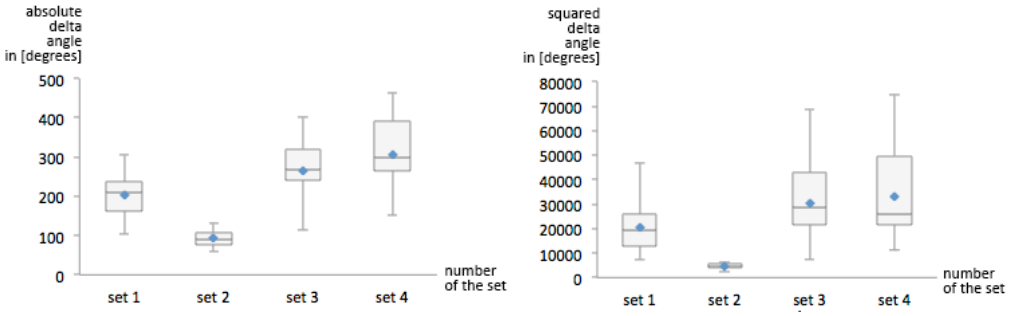

Figure 37 - Boxplots for the second evaluation of the skeleton measurement.

Another issue to be investigated within the second evaluation is whether the error values of different poses are similar when compared with their mirrored poses. Therefore, the absolute error values of both poses within each set were compared to their mirror poses. The 280 tracked sets were divided into 73 pairs of set one and two, 71 pairs of set three and 63 pairs of set four. The analytical data for each set are listed in Table 5. Figure 38 visualizes the boxplots for the calculated divergences within the four pose sets. For the poses of set two only the shoulder joint of one side is rotated a little bit. Due to the small change, it was very easy for the test persons to exactly reproduce this pose for the other side. That's the reason for the low mean value of $\mu=24,05°$ and standard deviation of $\partial=16,75°$ of the divergence between both poses of this set. The more changes had to be made for a pose, the more difficult it was for the test persons to precisely reproduce the poses, which caused bigger differences among the poses of a set. This is why the sets one, three and four resulted in higher mean values and standard deviations, as listed in Table 5.

Table 5 - Analysis of the divergences within the four pose sets of the second evaluation of the skeleton measurement.

set	**1**	**2**	**3**	**4**
mean value	44,30°	24,05°	73,81°	62,78°
standard deviation	36,67°	16,75°	57,17°	50,54°

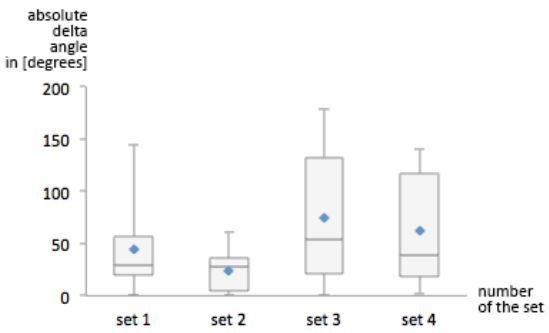

Figure 38 - Boxplots for the calculated divergences within the four pose sets of the second evaluation of the skeleton measurement.

5.5.1 Validity of the Measurement Algorithm

Since the previous evaluation showed huge differences in the error values even though the poses were quite equal, I did a separate evaluation of the algorithmic part of the application. For this purpose the application was started in the Unity3D editor in order to use its pause function. Another benefit of the editor is the possibility to change the main camera while running the application. After pausing the running application, a screenshot was taken. By using an image manipulation program, the angles of the joints were measured and compared to the calculated angles. Figure 39 shows an example documenting this process and the results. The outcome of this validity evaluation showed that a measurement error of less than one degree was reached, which may originate from inaccuracy during the manual measurement.

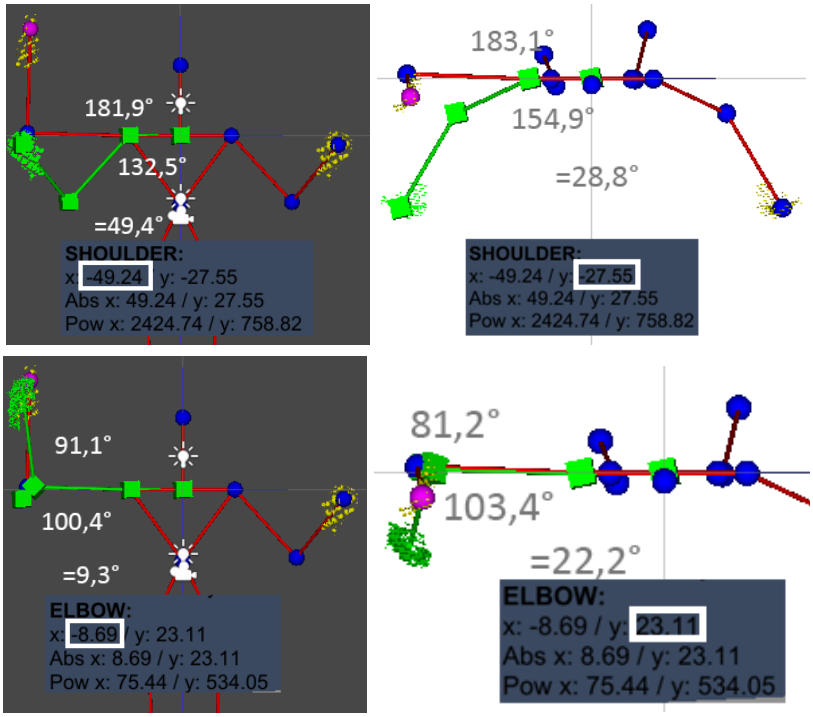

Figure 39 - Screenshots of one random pose of the measurement algorithm's manual validity evaluation. The images in the first row show the angles of the shoulder, the images in the second row show the angles of the elbow.

5.6 Evaluation of the Hand Point Cloud Analysis

Based on hypothesis H2, the first question regarding the hand point cloud is whether poses in which both hands are positioned in a similar way are detected correctly and result in very low error values. To investigate this the test persons had to assume four different hand poses. These poses, as shown in fFigure 40, were selected to evaluate the minimum error values. The left and right hands were in the same pose. For the first pose the participants point to show their palms towards the sensor. Pose number two required making two fists. The third pose was to open the hands again but rotate the

flat hands so that they were facing each other. For pose number four the test persons had to bend their fingers at the base joints so that they point towards each other, while the palms stay facing each other.

Figure 40 - The four poses for the first hand point cloud evaluation.

Again, each test person's hands were aligned as exactly as possible to gather valid and useful tracking data, representing the reproducibility and stability of the test by using the multi-frame tracking mode. As the compared hand point clouds in each of the four poses are quite similar, the delta between them is very low. Based on the 385 valid tracking data items the mean squared delta of the hand points resulted in a mean value of $\mu=8,42$ units and a standard deviation of $\partial=4,89$ units, and the squared delta of the hand points resulted in $\mu=8,00$ units and $\partial=5,02$ units. These analytical values are listed in Table 6, and visualized as boxplots in Figure 41. Except for some outliers, the values for both the mean squared delta and the squared delta are quite low. Therefore, we can predefine the upper boundaries for equality of the hand point deltas of approximately 13 units for both metrics.

Table 6 - Analysis of the first hand point cloud evaluation. (in [units])

	mean squared delta hand points	**squared delta hand points**
mean value	8,42	8,00
standard deviation	4,89	5,02

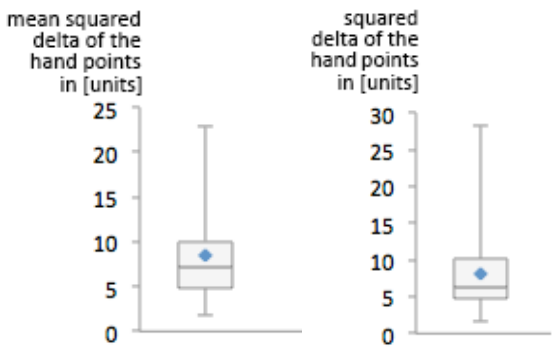

Figure 41 - Boxplots for the first evaluation of the skeleton measurement.

Based on hypothesis H2, the second question regarding the hand point cloud is whether the error values of different poses are detected as a divergence. In order to

investigate this, three sets of hand poses were defined, as visualized in Figure 42. The first set consisted of poses with one flat hand facing the sensor and the other making a fist. The poses of the second set required one flat hand facing the sensor while the other hand has had bent fingers at the base joint that were horizontally pointing to the first hand. The third set consisted of poses similar to set two, but the flat hand was rotated so that it was horizontally facing the hand with the bent fingers.

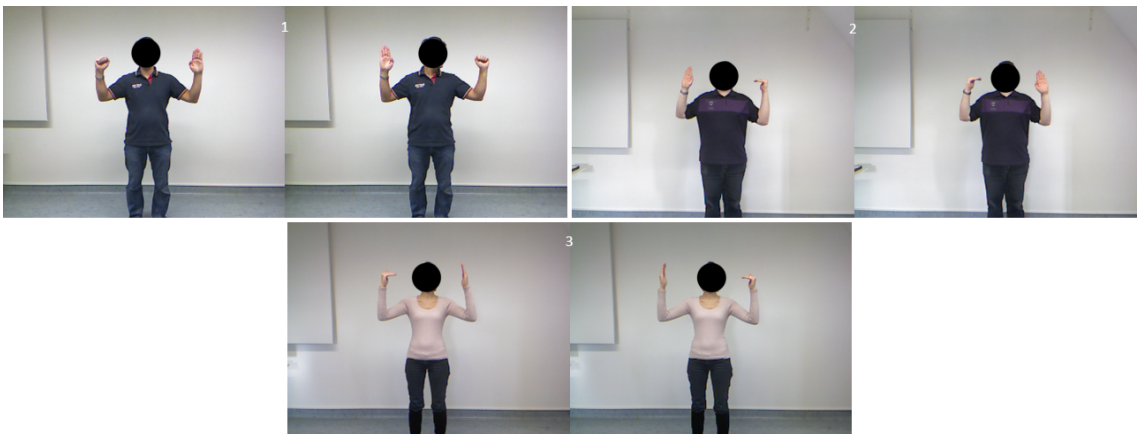

Figure 42 - The three sets of poses for the second hand point cloud evaluation.

The second evaluation resulted in 436 valid data items, divided into the three sets of poses. Table 7 lists the analytical data, while Figure 43 illustrates the distributions of the mean squared and squared error values in two groups of boxplots. For the first set 144 valid results were recorded, with a mean value of $\mu=18,87$ units and a standard deviation of $\partial=11,84$ units for the mean squared error metric. The squared delta resulted in a mean value of $\mu=26,87$ units and a standard deviation of $\partial=25,30$ units. The 146 items of the third set had slightly different squared error metrics, whereas the mean squared error values were significantly bigger with a mean value of $\mu=29,57$ units and a standard deviation of $\partial=20,60$ units. These mean squared error values were similar to the values of the second set. The 146 tracked data items of the second pose set resulted in a mean value of $\mu=35,65$ units and a standard deviation of $\partial=30,33$ units of the squared error values. The error values were significantly higher than the boundaries for the equal poses, although some outliers may lie under these boundaries.

Table 7 - Analysis of the three pose sets of the second hand point cloud evaluation. (“m.s.”=”mean squared”; “squ.”=”squared”; in [units])

pose set	**1**		**2**		**3**	
delta hand points	*m.s.*	*squ.*	*m.s.*	*squ.*	*m.s.*	*squ.*
mean value	18,87	26,87	28,90	35,65	29,57	26,64
standard deviation	11,84	25,30	18,93	30,33	20,60	26,07

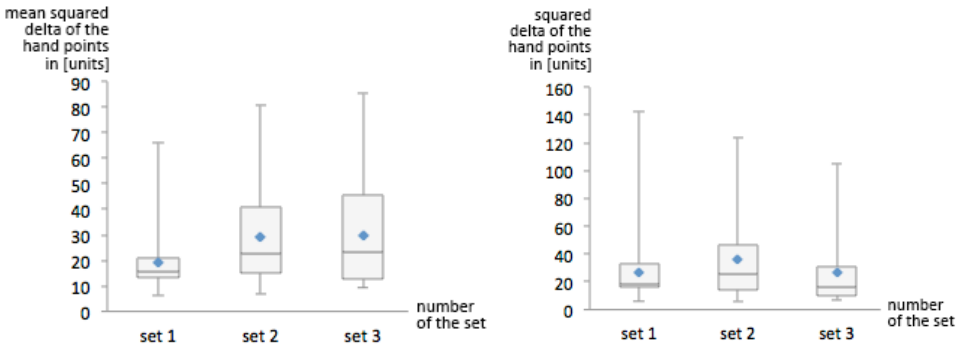

Figure 43 - Boxplots for the second hand point cloud evaluation.

5.7 Evaluation with Children

Based on hypothesis H3, and by using a first version of the application prototype, Elisabeth Soechting from the Department of Basic Psychological Research at the University of Vienna in Austria tested fourteen normally developed children between seven and eight years of age. A sample of screenshots with several different poses is given in Figure 44.

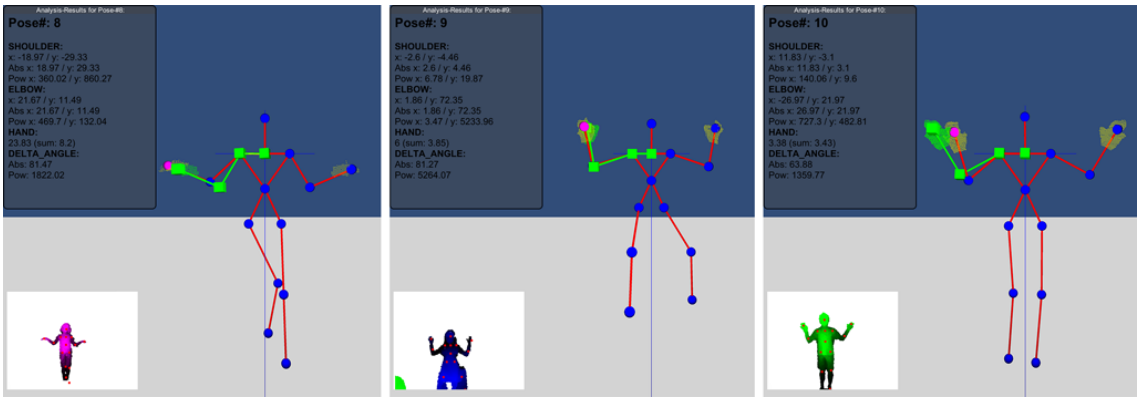

Figure 44 - Sampled screenshots of the tests with normally developed children.

Elisabeth Soechting tested twelve different poses in order to find out whether they could be useful for the future work with the application or not, as well as for collecting measurement data of a reference group of normally developed children. Most of the poses worked well with the application and with the children, for example the ones displayed in Figure 45. On the contrary, some poses were not very useful for the application with the Kinect sensor; one is illustrated in Figure 46. This pose may not work because some children would try and touch their hands above the head, which lead to a completely useless value for the hand point measurement. In general, the first tests showed that the system works and can be used for the measurement of the kinesthesia of children who are either normally developed or present with ASD.

Figure 45 - Some poses for the tests with children.

Figure 46 - A pose not working well with the application.

5.8 Discussion of the Results

Hypothesis H1 was confirmed by the evaluations' results in section 5.5. The analysis of the skeleton measurement of equal poses resulted in very low error values. For the test of the detection of differences between poses in which both halves of the body are not equally aligned, the hypothesis was also confirmed, because the values were significantly higher than the identified upper boundaries for equality. The more changes had to be made for a pose, the more difficult it was for the test persons to precisely reproduce the poses, which caused bigger differences between the poses within a set.

Hypothesis H2 was confirmed by the evaluations' results in section 5.6. The analysis of the hand point cloud measurement of equal hand poses resulted in low error values. Apart from some outliers caused by slightly inaccurate posing of the participants during multi-frame tracking, values for the upper boundaries for equality could be identified. The tests regarding the differences between hand poses confirmed the hypothesis.

With hypothesis H3 I assumed that the process of measuring one posture after the other during a test session is feasible for the participants and the examiner. Section 5.7 describes the pilot testing of a sequence of twelve test items. Two out of the twelve items were not practicable and will be replaced for the field study.

The outcome of the validity evaluation in subsection 5.5.1 showed that a measurement error of less than one degree was reached, which may originate from inaccuracies during the manual measurement in an image processing application. Regarding tracking accuracy, as explained in section 2.6, a tracking error of 3.2°±2.2° was reached. But as the evaluation showed, such small tracking error values will not significantly affect the outcome of the measurement process. The accuracy and stability during the tracking

can be improved in future work by using the new Kinect One sensor, mentioned in subsection 4.1.1.

The results of the evaluations show that the application prototype is able to track and measure the participants in front of the sensor in a reproducible and stable way. By choosing the right postures, the system can be used for the field study of measuring the kinesthesia of children who are normally developed or who present ASD.

6 Conclusion and Future Work

In this thesis I presented the design and implementation of an application for the measurement of proprioceptive functionality with the use of depth data. The application was used for a reproducible and stable test for assessing the kinesthesia of children. The prototype implementation was evaluated regarding functionality and accuracy. In addition, it was pilot tested with normally developed children, who had to assume different body positions without visual control. The evaluation shows that the application prototype tracks and measures the participants in front of the sensor in a reproducible and stable way. Furthermore, all test persons were able to assume a set of poses during the testing of their kinesthesia.

The system presented in this thesis uses a *Microsoft Kinect for XBOX 360* as RGB-D sensor connected to a PC, where the application collects and analyzes posture data. In future work with depth sensors several improvements can be implemented. In order to increase accuracy and speed, the new *Kinect One* sensor could be used. For this purpose the framework would have to be changed to the new *Microsoft Kinect SDK*. This would require refactoring of the application. But the new system would support the tracking of more joints, especially more joints in the hands. Another possibility to improve the accuracy of the hand point cloud part would be to use multiple Kinect sensors that are synced and calibrated in order to gather a more detailed point cloud representation of the hands. The *PCL* would especially be useful for merging the depth point clouds of multiple Kinect sensors. As the conversion of depth to world coordinates is very expensive, working with the depth coordinates for as long as possible and only converting a selection of points to world coordinates at the end of the algorithm would also increase the speed.

In her future study, the collaborating psychologist will use the application to provide meaningful data that discriminate between children with ASD and typical children. In order to gather more data, she will use additional measurement instruments such as force and touch sensors to test hypotheses regarding differences in proprioceptive accuracy.

List of Figures

Figure 1 - Illustration of the test setup with the projection screen and the schematic representation of the child's arm. [16].....7
Figure 2 - Kinect Skeleton Tracking Pipeline by Shotton et al. [37].....13
Figure 3 - The Microsoft Kinect sensor for XBOX 360 with components indicated.....23
Figure 4 - RGB image, produced by the RGB camera. Certain sections were blacked out for reasons of anonymity. 24
Figure 5 - The infrared dots seen by the IR camera. The image on the right shows a close-up of the red boxed area. The yellow boxed area in the background shows part of a window, a reflective surface. 25
Figure 6 - The depth image, produced by the Kinect sensor, based on the dot image in figure 5, and colored yellow for better visibility.....25
Figure 7 - Screenshot of the first prototype version of the application. (1) The depth map of the tracked user. (2) The RGB-image. (3) The visualized skeleton, with mirrored bones and overlaid point cloud. (4) The live depth map of all tracked users. (5) The GUI for controlling the application's functions.....29
Figure 8 - Screenshot of the application's stage (1) Texture-based visualization of the histogram-equalized depth stream segment relating to the tracked user. (2) Texture-based visualization of the RGB stream. 31
Figure 9 - Screenshot of the application's stage (3) The point cloud representing the tracked user, visualized as tiny white spheres. (4) The skeletal joints representing the tracked user's skeleton, visualized as tiny red cubes. 33
Figure 10 - Texture-based visualization of the histogram-equalized depth stream segment relating to the tracked user, displayed on the application's virtual stage, similar to figure 8 - (1). The skeletal joints are additionally visualized as tiny red squares.....34
Figure 11 - Skeletal joints and bones build up a representation of the tracked user's human skeleton. 35
Figure 12 - Rotated, aligned skeleton.....37
Figure 13 - The overlay of the skeleton and the depth image's point cloud. 37
Figure 14 - Skeleton with mirrored joints and bones. 38
Figure 15 - The first prototype's GUI. 40
Figure 16 - The visualization of the functionality "change from elbow to shoulder".....40
Figure 17 - Screenshot of the second prototype version of the application. (1) The result window displaying the analysis of the current pose. (2) The down-sampled RGB image. (3) The improved GUI for controlling the application's functions.....42
Figure 18 - The segmentation of the hand's point cloud. 43
Figure 19 - The segmented point cloud after the refinement step.....43
Figure 20 - The mirrored hand point cloud. 44
Figure 21 - Exact overlay of two hand point clouds of the same hand position, on the left. The same two hand point clouds, from another perspective, moved apart for visual debugging, on the right.....44
Figure 22 - Pop-up dialog for entering the test person's name. 45
Figure 23 - The record button including the pose number, at the top, and the „don't save current record" button, at the bottom. The letters in the brackets represent the keyboard shortcuts for the buttons' functionalities.....46

Figure 24 - The result window displaying the analysis of the current pose. 46
Figure 25 - The GUI for the camera controls. 47
Figure 26 - The final prototype's application window. (1) The prominent result window displaying the analysis of the current pose. (2) The smaller monitor containing the down-sampled RGB image and the bigger one visualizing the real-time depth image with the overlay of the live skeletal joints. (3) The GUI including the debugging functions like recording multiple frames and the current test person's name. 48
Figure 27 - Depth map of the tracked user, including the skeletal joints and the two big, red boxes for optimization of the point cloud, on the left. The segmented hand points based on the optimized point cloud, on the right. 49
Figure 28 - Depth texture of the live captured user map. The red squares mark the real-time positions of the skeletal joints. 50
Figure 29 - The GUI menu for closing the current measurement session, either for closing the application or for restarting the measuring process for a new test person, on the left. The pop-up dialog, used for entering the new person's name, on the right. 51
Figure 30 - The box displaying the name of the current test person. 51
Figure 31 - A pop-up containing information on the processing state. 51
Figure 32 - The setup of the system for the evaluations with the test persons. 53
Figure 33 - Outcome of the incorrectly tracked joints, caused by a defective USB-port. 53
Figure 34 - The five poses for the first skeleton evaluation. 54
Figure 35 - Boxplots for the first evaluation of the skeleton measurement. 55
Figure 36 - The four sets of poses for the second skeleton evaluation. 56
Figure 37 - Boxplots for the second evaluation of the skeleton measurement. 57
Figure 38 - Boxplots for the calculated divergences within the four pose sets of the second evaluation of the skeleton measurement. 57
Figure 39 - Screenshots of one random pose of the measurement algorithm's manual validity evaluation. The images in the first row show the angles of the shoulder, the images in the second row show the angles of the elbow. 58
Figure 40 - The four poses for the first hand point cloud evaluation. 59
Figure 41 - Boxplots for the first evaluation of the skeleton measurement. 59
Figure 42 - The three sets of poses for the second hand point cloud evaluation. 60
Figure 43 - Boxplots for the second hand point cloud evaluation. 61
Figure 44 - Sampled screenshots of the tests with normally developed children. 61
Figure 45 - Some poses for the tests with children. 62
Figure 46 - A pose not working well with the application. 62
Figure 47 - Outcome of the attempt to create a mesh based on the depth points. 70

List of Tables

Table 1 - Available video modes of depth and RGB sensors. The highlighted rows represent the video modes used in the application.....29
Table 2 - The different joint types provided by the framework, with numeric identifier. The highlighted rows represent the joint types that are used in the application.....34
Table 3 - Analysis of the first evaluation of the skeleton measurement. 55
Table 4 - Analysis of the four pose sets of the second evaluation of the skeleton measurement. (“abs.”=”absolute”; “squ.”=”squared”).....56
Table 5 - Analysis of the divergences within the four pose sets of the second evaluation of the skeleton measurement. 57
Table 6 - Analysis of the first hand point cloud evaluation. (in [units]).....59
Table 7 - Analysis of the three pose sets of the second hand point cloud evaluation. (“m.s.”=”mean squared”; “squ.”=”squared”; in [units]) 60

Acknowledgements

I would like to thank all the people who helped me throughout my studies and in reaching my goals. I would like to especially thank

my advisors Dipl.-Ing. Christian Schoenauer and Mag. Dr. Hannes Kaufmann who always had an open door and helped me with solving my problems.

my colleague Elisabeth Soechting for the idea for this project, her input regarding functionality and usability, and the provided test data.

Andreas, Christian, Gerhard, Harald, Johann, Madeleine, Marco, Renate, Sandra and Viktoria for being my test persons for the evaluation.

Esther Conway for proofreading this thesis.

my parents for their unconditional support during my whole life, especially in my school days and during my studies.

my friends for being my friends.

my love Sandra, for being who you are, for your patience, uniqueness and support.

Appendix A: Installing all applications necessary to run the prototype

For the execution of the application prototype, several steps and installations are necessary, which are listed in the following:

1. Download and install *Kinect for Windows SDK v1.6* or *v1.8*, which is needed for the basic USB-drivers for the Kinect sensor bar. http://cw-hartl.at/TUVienna/autism_installs/KinectSDK-v1.6-Setup.exe
2. Download and install *OpenNI 2.2.0.30 Beta(x86)*, which is the framework everything is built on. http://cw-hartl.at/TUVienna/autism_installs/OpenNI-Windows-x86-2.2.msi
3. Download and install *NiTE 2.2.0.10(x86)*, for tracking the skeleton of the users in front of the sensor. http://cw-hartl.at/TUVienna/autism_installs/NiTE-Windows-x86-2.2.msi
4. Download and move the *enhanced Kinect.dll* to *OpenNI's* installation directory, e.g. "C:/Program Files (x86)/OpenNI2/Redist/OpenNI2/Drivers/". http://cw-hartl.at/TUVienna/autism_installs/enhancedKinect.zip

Appendix B: Attempts to Improve the Application with PCL and Mesh-Rendering

B.1 Including the PCL

For the processing of the hand point clouds, the use of the point cloud library or *PCL* (http://pointclouds.org/) would have been advantageous. However, similar to the problems described in subsection 4.1.4, no official wrappers for the use in my *C#*-application could be found. For other tools such as the visualization toolkit or *VTK* (http://vtk.org) *C#*-wrappers did exist. In order to be able to use this, another workaround would have been necessary. A small wrapper for transferring the *PCL* data into the *VTK* format could have been used. Afterwards open toolkit or *openTK* (http://opentk.com) could have been used. But this was not a nice and easy-to-use solution. Apart from that, I did not get it to work as it should. Therefore, and because I just would have needed the functionality of the *ICP*, I decided not to hack around with multiple frameworks and libraries, and instead implement a simple *ICP* on my own, as described in subsection 4.2.2.3.

B.2 Rendering of a Mesh

As an attempt to improve the visual presentation of the point cloud the use of a mesh rendering was considered. I used the mesh object of a library by a developer named HoshiKata (http://www.codeproject.com/Articles/492435/Delaunay-Triangulation-For-Fast-Mesh-Generation). But as visualized in Figure 47, the outcome was not satisfactory. Because it was not a must-have, and solving the problem seemed to be quite difficult and time-consuming, I did not incorporate this function.

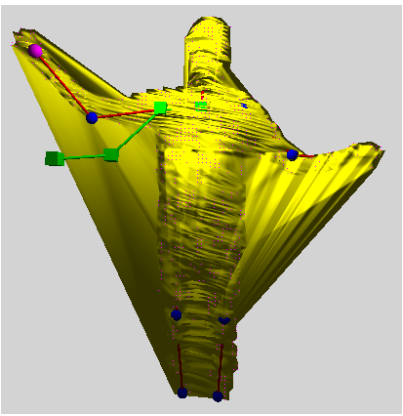

Figure 47 - Outcome of the attempt to create a mesh based on the depth points.

Bibliography

1. Ali, H., Shafait, F., Giannakidou, E., et al.Contextual object category recognition for RGB-D scene labeling. *Robotics and Autonomous Systems 62*, 2 (2014), 241–256.

2. Anand, A., Koppula, H.S., Joachims, T., and Saxena, A.Contextually guided semantic labeling and search for three-dimensional point clouds. *The International Journal of Robotics Research 32*, 1 (2012), 19–34.

3. Baak, A., Müller, M., and Bharaj, G.A Data-Driven Approach for Real-Time Full Body Pose Reconstruction from a Depth Camera. In A. Fossati, J. Gall, H. Grabner, X. Ren and K. Konolige, eds., *Consumer Depth Cameras for Computer Vision*. Springer London, London, 2013, 71–98.

4. Bai[1], Z., Blackwell, A., and Coulouris, G.Through the Looking Glass: Pretend Play for Children with Autism. *ISMAR13*, , 49–58.

5. Bartoli, L., Corradi, C., Garzotto, F., and Valoriani, M.Exploring motion-based touchless games for autistic children's learning. *Proceedings of the 12th International Conference on Interaction Design and Children - IDC '13*, ACM Press (2013), 102–111.

6. Besl, P.J. and McKay, N.D.A Method for Registration of 3-D Shapes. (1992), 586–606.

7. Blanche, E.I., Reinoso, G., Chang, M.C., and Bodison, S.Proprioceptive processing difficulties among children with autism spectrum disorders and developmental disabilities. *The American journal of occupational therapy : official publication of the American Occupational Therapy Association 66*, 5 (2009), 621–4.

8. Casas, X., Herrera, G., Coma, I., and Fernández, M.A KINECT-BASED AUGMENTED REALITY SYSTEM FOR INDIVIDUALS WITH AUTISM SPECTRUM DISORDERS. *Proceedings of the International Conference on Computer Graphics Theory and Applications and International Conference on Information Visualization Theory and Applications - GRAPP/IVAPP 2012*, (2012), 440–446.

9. Chaaraoui, A.A., Padilla-López, J.R., Climent-Pérez, P., and Flórez-Revuelta, F.Evolutionary joint selection to improve human action recognition with RGB-D devices. *Expert Systems with Applications 41*, 3 (2014), 786–794.

10. Charles, J. and Everingham, M.Learning shape models for monocular human pose estimation from the Microsoft Xbox Kinect. *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, IEEE (2011), 1202–1208.

11. Clark, R.A., Pua, Y.-H., Bryant, A.L., and Hunt, M.A.Validity of the Microsoft Kinect for providing lateral trunk lean feedback during gait retraining. *Gait & posture 38*, 4 (2013), 1064–6.

12. Clark, R.A., Pua, Y.-H., Fortin, K., et al.Validity of the Microsoft Kinect for assessment of postural control. *Gait & posture 36*, 3 (2012), 372–7.

13. Du, G. and Zhang, P.Markerless human–robot interface for dual robot manipulators using Kinect sensor. *Robotics and Computer-Integrated Manufacturing 30*, 2 (2014), 150–159.

14. Falahati, S.NiWrapper: OpenNI 2 .Net Wrapper. falahati.net/my-projects/96-openni-2-net-wrapper#.Ue77tVP0buN.

15. Feng, Z., Xu, S., Zhang, X., Jin, L., Ye, Z., and Yang, W.Real-time fingertip tracking and detection using Kinect depth sensor for a new writing-in-the air system. *Proceedings of the 4th International Conference on Internet Multimedia Computing and Service - ICIMCS '12*, ACM Press (2012), 70.

16. Fuentes, C.T., Mostofsky, S.H., and Bastian, A.J.No proprioceptive deficits in autism despite movement-related sensory and execution impairments. *Journal of autism and developmental disorders 41*, 10 (2011), 1352–61.

17. Galna, B., Barry, G., Jackson, D., Mhiripiri, D., Olivier, P., and Rochester, L.Accuracy of the Microsoft Kinect sensor for measuring movement in people with Parkinson's disease. *Gait & posture 39*, 4 (2014), 1062–8.

18. Henry, P., Krainin, M., Herbst, E., Ren, X., and Fox, D.RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments. *The International Journal of Robotics Research 31*, 5 (2012), 647–663.

19. Huang, Z., Xu, Z., Li, Z., Zhao, Z., and Tao, D.Depth and Skeleton Information Model for Kinect Based Hand Segmentation. *Proceedings of the 2013 International Conference on Advanced ICT*, Atlantis Press (2013), 622–626.

20. Izadi, S., Davison, A., Fitzgibbon, A., et al.KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera*. *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11*, ACM Press (2011), 559.

21. Kar, A.Skeletal Tracking using Microsoft Kinect. *Methodology*, 2010, 1–11. http://www.mendeley.com/research/skeletal-tracking-using-microsoft-kinect/.

22. Lee, H.-M., Liau, J.-J., Cheng, C.-K., Tan, C.-M., and Shih, J.-T.Evaluation of shoulder proprioception following muscle fatigue. *Clinical Biomechanics 18*, 9 (2003), 843–847.

23. Liu, Z., Tang, S., Qin, H., and Bu, S.Evaluating user's energy consumption using kinect based skeleton tracking. *Proceedings of the 20th ACM international conference on Multimedia - MM '12*, (2012), 1373.

24. Lord, C. and Bishop, S.L.Autism Spectrum Disorders: Diagnosis, Prevalence, and Services for Children and Families. *Social Policy Report - sharing child and youth development knowledge 24*, 2 (2010).

25. Ly, D.L., Saxena, A., and Lipson, H.Co-evolutionary predictors for kinematic pose inference from RGBD images. *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference - GECCO '12*, ACM Press (2012), 967.

26. Martin, C.C., Burkert, D.C., Choi, K.R., et al.A real-time ergonomic monitoring system using the Microsoft Kinect. *2012 IEEE Systems and Information Engineering Design Symposium*, IEEE (2012), 50–55.

27. Melax, S., Keselman, L., and Orsten, S.Dynamics based 3D skeletal hand tracking. *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games - I3D '13*, ACM Press (2013), 184.

28. Mossel, A., Schönauer, C., Gerstweiler, G., and Kaufmann, H.ARTiFICe-Augmented Reality Framework for Distributed Collaboration. *International Journal of Virtual Reality (presented at Workshop on Off-The-Shelf Virtual Reality, IEEE VR, USA, 2012) 11*, 3 (2012), 1–7.

29. Oikonomidis, I., Kyriazis, N., and Argyros, A.Efficient model-based 3D tracking of hand articulations using Kinect. *Procedings of the British Machine Vision Conference 2011*, (2011), 101.1–101.11.

30. Payne, A., Daniel, A., Mehta, A., et al.7.6 A 512×424 CMOS 3D Time-of-Flight image sensor with multi-frequency photo-demodulation up to 130MHz and 2GS/s ADC. *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, IEEE (2014), 134–135.

31. Pedersoli, F., Adami, N., Benini, S., and Leonardi, R.XKin -. *Proceedings of the 20th ACM international conference on Multimedia - MM '12*, ACM Press (2012), 1465.

32. Ponce, C., Selman, B., and Saxena, A.Unstructured human activity detection from RGBD images. *2012 IEEE International Conference on Robotics and Automation*, IEEE (2012), 842–849.

33. Ren, Z., Meng, J., Yuan, J., and Zhang, Z.Robust Hand Gesture Recognition with Kinect Sensor. *Proceedings of the 19th ACM …*, (2011), 759–760.

34. Rusu, R.B. and Cousins, S.3D is here: Point Cloud Library (PCL). *2011 IEEE International Conference on Robotics and Automation*, IEEE (2011), 1–4.

35. Rusu, R.B., Marton, Z.C., Blodow, N., Dolha, M., and Beetz, M.Towards 3D Point cloud based object maps for household environments. *Robotics and Autonomous Systems 56*, 11 (2008), 927–941.

36. Schönauer, C. and Kaufmann, H.Wide Area Motion Tracking Using Consumer Hardware. *The International Journal of Virtual Reality 12*, 1 (2013), 1–9.

37. Shotton, J., Fitzgibbon, A., Cook, M., et al.Real-time human pose recognition in parts from single depth images. *Proceedings of the IEEE Computer Vision and Pattern Recognition - CVPR 2011*, IEEE (2011), 1297–1304.

38. Sinthanayothin, C., Wongwaen, N., and Bholsithi, W.Skeleton Tracking using Kinect Sensor & Displaying in 3D Virtual Scene. *International Journal of Advancements in Computing Technology 4*, 11 (2012), 213–223.

39. Sparks, B.F., Friedman, S.D., Shaw, D.W., et al.Brain structural abnormalities in young children with autism spectrum disorder. *Neurology 59*, 2 (2002), 184–92.

40. Stone, E.E. and Skubic, M.Passive in-home measurement of stride-to-stride gait variability comparing vision and Kinect sensing. *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference 2011*, (2011), 6491–4.

41. Suma, E. a., Krum, D.M., Lange, B., Koenig, S., Rizzo, A., and Bolas, M.Adapting user interfaces for gestural interaction with the flexible action and articulated skeleton toolkit. *Computers & Graphics 37*, 3 (2013), 193–201.

42. Suma, E. a., Lange, B., Rizzo, A.S., Krum, D.M., and Bolas, M.FAAST: The Flexible Action and Articulated Skeleton Toolkit. *2011 IEEE Virtual Reality Conference*, (2011), 247–248.

43. Varkonyi-Koczy, A.R. and Tusor, B.Human–Computer Interaction for Smart Environment Applications Using Fuzzy Hand Posture and Gesture Models. *IEEE Transactions on Instrumentation and Measurement 60*, 5 (2011), 1505–1514.

44. Villaroman, N., Rowe, D., and Swan, B.Teaching natural user interaction using OpenNI and the Microsoft Kinect sensor. *Proceedings of the 2011 conference on Information technology education - SIGITE '11*, ACM Press (2011), 227.

45. World Health Organization.The ICD-10 Classification of Mental and Behavioural Disorders. *International Classification 10*, (1992), 1–267.

46. Zafrulla, Z., Brashear, H., Starner, T., Hamilton, H., and Presti, P.American sign language recognition with the kinect. *Proceedings of the 13th international conference on multimodal interfaces - ICMI '11*, (2011), 279.

47. Zhang, Z.Microsoft Kinect Sensor and Its Effect. *IEEE Multimedia 19*, 2 (2012), 4–10.

48. OpenNI. http://www.openni.org.

49. Prime Sense Ltd. http://www.primesense.com/.

50. Occipital Inc. http://structure.io/openni.

51. Unity3D - Game Development Tool. http://unity3d.com.

52. enhancedKinect.dll. https://github.com/OpenNI/OpenNI2/commit/8fea30e1cad144ea1e2fe03410bbed95a3a79638.