IuK Praxisprojektarbeit

Erfassung und Speicherung von menschlichen Bewegungsdaten mit einer Kinect One Kamera zur Weiterverarbeitung mit der Bewegungsanalysesoftware OpenSim

Autor : Oliver Mayr

Matrikelnummer :

Studiengang : Informations- und Kommunikationstechnik

Erstprüfer :

Zweitprüferin :

Vorgelegt am : [Platzhalter Datum]

Abstrakt

In dieser Arbeit werden Körperdaten in der Bewegung, die mit Hilfe einer Kinect Kamera aufgenommen werden, für die Bearbeitung mit der Software OpenSim verarbeitet. Hierfür werden die Daten in das TRC Datenformat konvertiert. Dies passiert in der Entwicklungsumgebung von Microsoft Visual Studio in der Programmiersprache C#. In OpenSim werden die Körperdaten mit einem simulierten Körperskelett-Model verbunden, um die aufgenommene Bewegung in einer Simulation zu rekonstruieren.

Abstract

This assigment deals with the connection between the motion analysis software OpenSim and the Microsoft Kinect Camera. The body data of a human is recorded through the Camera and converted into the motion data format TRC. The record and convertion is realized in the development tool Visual Studio in the programming language C#. The converted body data then is connected with a full skeletal modell in OpenSim, in order to achieve a full reconstruction of the recorded motion.

Inhaltsverzeichnis

1 Einleitung

Dieses Projekt entstand im Rahmen des Modules “IuK Praxisprojekt“ im Bachelorstudiengang “Informations- und Kommunikationstechnik“ mit dem Schwerpunkt Informationstechnik. Die Arbeit wurde komplett im Institut für Informationstechnik an der FH Dortmund durchgeführt. Der Zeitraum für die Bearbeitung war das Sommersemester 2016.

1.1 Motivation

Die Hauptmotivation hinter diesem Projekt findet sich in dem großen Interesse an der Kinect Kamera und der Erkennung und Arbeit mit Körperdaten. Schon seit Beginn der Arbeit mit der Kinect bestand das Interesse, eine Erkennung von Körpergestiken zu konstruieren. Nach Fertigstellung des IuK Projektes, in dem eine Bewegungsteuerung eines NXT Humanoiden durch die Kinect Kamera realisiert wurde, entstand daraus die Idee in diesem Praxisprojekt auf die Körperbewegungsanalyse einzugehen.

Abbildung 1.1 IuK NXT Projektarbeit

Statt den Körper als Steuerungseinheit zu sehen, sollte diesmal die Aufnahme und die Rekonstruierbarkeit von Bewegungen im Vordergrund stehen.

Für die Verarbeitung der Körperdaten wurde das Programm OpenSim ausgewählt. Diese ambitionierte Software weckte Interesse aufgrund seiner Vielfalt an Möglichkeiten in Bezug auf die Körperbewegungsanalyse. Auch besteht durch den Opensource Status des Programmes eine große Kompatibilität zu anderen Entwicklungsumgebungen wie zum Beispiel Visual Studio oder Matlab.

1.2 Gliederung

Begonnen werden soll mit einer Einführung in die Inbetriebnahme der verwendeten Hardware. Hierbei soll auf die Kinect Kamera und ihre Funktionen eingegangen werden. Dabei werden die technischen Details, ein Vergleich zwischen alter und neuer Version der Kamera und die einzelnen Sensoren genauer betrachtet.

Im darauffolgenden Kapitel soll auf die verwendete Software eingegangen werden. Hierfür wird mit der Software OpenSim und ihren Hauptfunktionen begonnen. Abschließend wird die Software Visual Studio beschrieben. Hierfür wird eine Einführung in das Öffnen der Projektmappe und das Arbeiten mit Verweisen gegeben.

Nach Erläuterung der verwendeten Hardware und Software soll das Ziel und die Funktionsweise des Projektprogrammes erläutert werden. Hierbei wird mit Hilfe von Programmablaufplänen und Quellcodeausschnitten ein besseres Verständnis für den Ablauf der Applikation gegeben.

Abschließend soll die Verwendung von OpenSim beschrieben werden. In diesem Rahmen wird das verwendete Modell und das Inverse Kinematics Tool genauer betrachtet.

Im Fazit wird zu Beginn auf die aufgekommenen Probleme während der Entwicklungsphase eingegangen. Außerdem wird im zweiten Teil des Fazits ein Zukunftsausblick für das Projekt gegeben.

2 Funktionsumfang und Inbetriebnahme der Hardware

Damit ein besseres Verständnis für den Inhalt dieses Iuk Praxisprojektes vorhanden ist, sollen in diesem Kapitel die Grundlagen für die eingesetzte Hardware geschaffen werden. Hardwaretechnisch wird in diesem Projekt mit der Kinect Kamera Modell 1520 für die Konsole XBOX ONE gearbeitet. Um einen Einblick in die Möglichkeiten dieser Kamera zu erhalten, wird in den nächsten Unterkapiteln auf die wichtigsten Funktionen und Bauteile eingegangen. Des Weiteren wird ein Vergleich mit der älteren Version der Kinect Kamera gezogen.

2.1 Kinect Kamera

Die in diesem Projekt verwendete Kinect ONE basiert auf der Kinect Kamera der XBOX 360 und wurde zusammen mit der XBOX ONE am 22. November 2013 der Öffentlichkeit vorgestellt [wik16a]. Neben diverser Verbesserungen an der Hardware, wurde mit Veröffentlichung der Kamera ein neues Development Tool bereitgestellt, welches eine bessere Einbindung der Kinect ONE in einer Entwicklungsumgebung ermöglicht.

2.1.1 Kinect ONE im direkten Vergleich mit der Kinect 360

Um die Verbesserung zwischen der älteren Kinect XBOX 360 Variante und der neuen Kinect XBOX ONE aufzuzeigen, wurde in Tabelle 2.1 ein direkter Vergleich der beiden Varianten gegenübergestellt.

Eigenschaften	**Kinect 360**	**Kinect ONE**
Color Kamera	640 x 480 mit 30fps	1920 x 1080 mit 30fps
Depth Kamera	320 x 240	512 x 424
Max. Depth Distanz	Ca. 4.5m	Ca. 4.5m
Min. Depth Distanz	40 cm	50 cm
Horizontales Sichtfeld	57	70
Vertikales Sichtfeld	43	60
Motor	Ja	Nein
Gelenkpunkte	20	25
Aktiv auslesbare Personen	2	6

Tabelle 2.1 Direkter Vergleich der Kinect Generationen

Neben den höheren Auflösungen bei Tiefen- und Farbkamera wurden besonders bei der Körpererkennung Fortschritte gemacht. Statt 20 können nun 25 Gelenkpunkte ausgelesen werden. Bei den neuen Punkten handelt es sich um Gelenkpunkte für Daumen und Zeigefinger sowie einen neuen Punkt an dem Nacken des Benutzers. Außerdem wurde die Lage der Torso Gelenkpunkte verändert [pte14]. Die Gelenkpunkte werden im späteren Verlauf dieser Dokumentation im Detail erläutert. Eine weitere wichtige Neuerung ist die Möglichkeit mit sechs statt nur zwei Personen zu arbeiten. So kann die Kinect ONE sechs Personen simultan erkennen und die Körperdaten aufzeichnen.

2.1.2 Aufbau und Komponenten

Die Kinect ONE besteht aus einer schwarzen rechteckigen Haupteinheit, die einen integrierten Standfuß besitzt. In dieser Haupteinheit befinden sich der Infrarotprojektor, die Infrarotkamera und die Farbkamera. Darüber hinaus befindet sich an der Unterseite eine Leiste in der die vier Mikrofone der Kamera verbaut sind. Befindet sich die Kinect in einem falschen Winkel zu dem Benutzer, kann dieser mit dem Standfuß eine Bewegung in horizontaler Richtung durchführen. Um die Daten der Sensoren direkt verarbeiten zu können, ist außerdem ein Prozessor der Firma PrimeSense verbaut. [Han13]

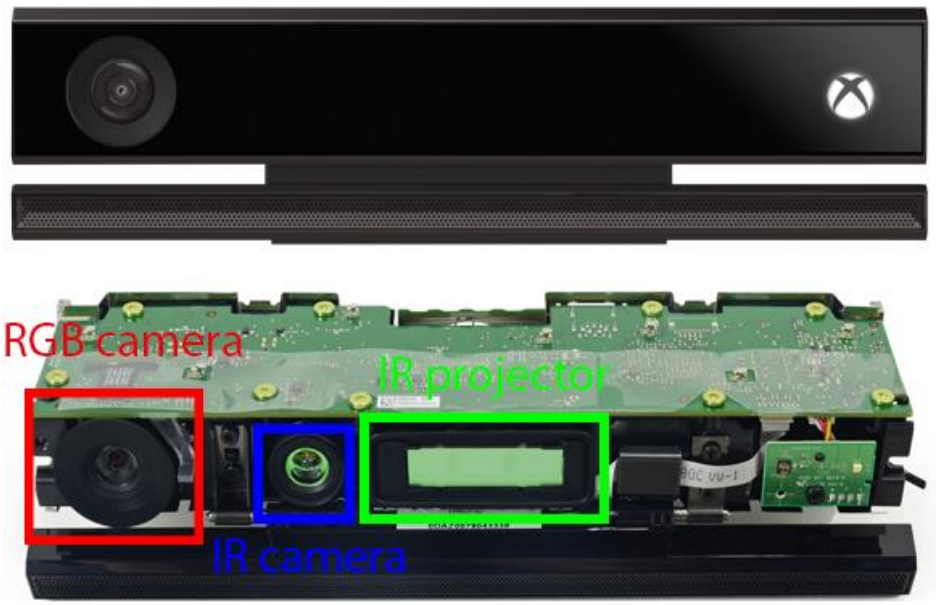


Abbildung 2.1 Kinect ONE Kamera [ifi13]

2.1.3 Farbkamera

Die Farbkamera arbeitet mit einer Auflösung von 1920x1080 Pixel in einer Taktrate von 30 Bildern pro Sekunde. Bei schlechten Lichtverhältnissen sinkt die Taktrate auf bis zu 15 Bildern pro Sekunde [dev16].

2.1.4 Infrarotprojektor

Bei dem Infrarotprojektor wird ein Laser der Klasse 1 verwendet. Klasse 1 steht hierbei für einen Laser, der eine Wellenlänge zwischen 400 nm bis 1400 nm besitzt und gefahrlos für das menschliche Auge ist. Bei dem Infrarotprojektor der Kinect wird mit einer Wellenlänge von 830nm gearbeitet. Die primäre Aufgabe des Projektors besteht darin, ein kontinuierliches Lichtmuster auszustrahlen. Durchgeführt wird dies mit Hilfe einer Lochschablone. Hiermit fällt der Laser immer in ein

vordefiniertes Muster. Diese Fächerung wird in Abbildung 3 deutlich. Die Funktion dieses Musters soll im Kapitel der Infrarotkamera im Detail erläutert werden. [sig13]

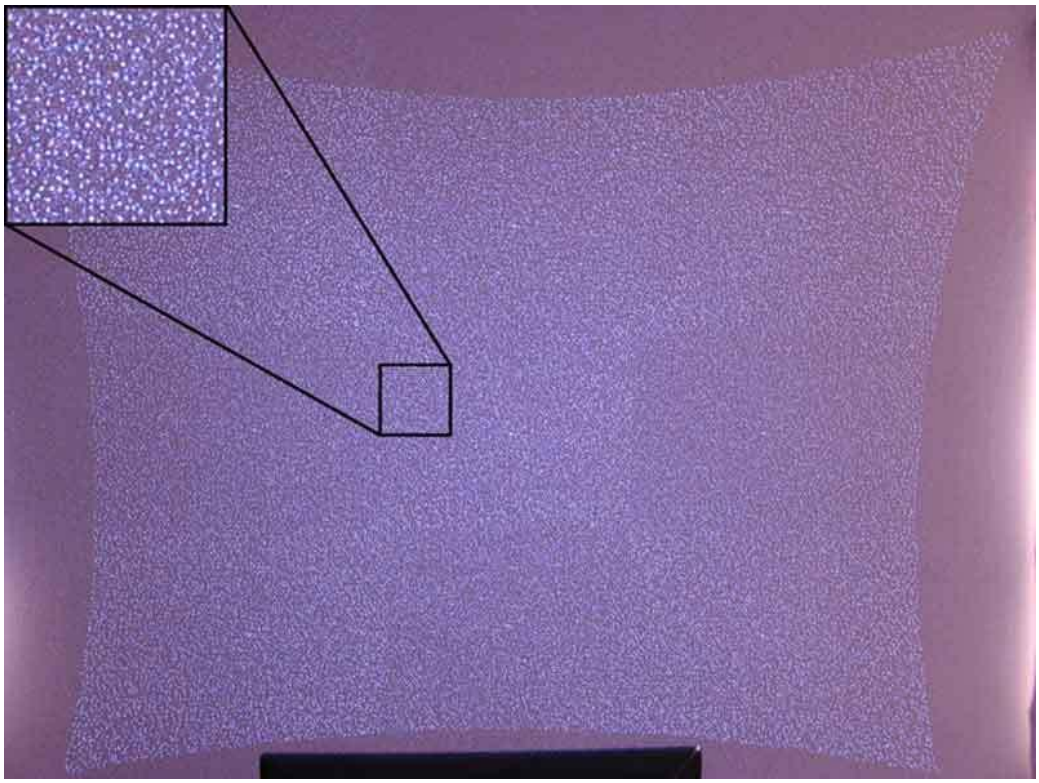


Abbildung 2.2 Infrarotprojektor [dna15]

2.1.5 Infrarotkamera

Die Infrarotkamera befindet sich direkt neben dem Infrarotprojektor und hat mit Hilfe eines monochromatischen CMOS Sensor die Möglichkeit Infrarotlicht zu empfangen. Sie arbeitet mit einer Auflösung von 512x424 Pixel. Jeder Pixel enthält 16 Bit Informationen. Diese setzen sich durch 13 Bit Tiefendaten und 3 Bit Indexdaten zusammen. Die Zusammensetzung ermöglicht das Arbeiten mit dem im vorherigen Unterkapitel beschriebene Lichtmuster. Die Infrarotkamera nimmt Veränderungen des abgesendeten Lichtmusters wahr. Wenn sich beispielsweise wie in Abbildung 2.3 ein Objekt im Sichtfeld befindet, passen die Pixel des Hintergrundes und des Objektes nicht mehr zusammen. Die Infrarotkamera registriert dies und kann mit Hilfe von Triangulation die Entfernung zu eben diesem Objekt berechnen [mir10]. Eine fehlerfreie Berechnung kann nur bei eingehaltenem Abstand von 0,5 bis 4,5 Metern gewährleistet werden [dev16].

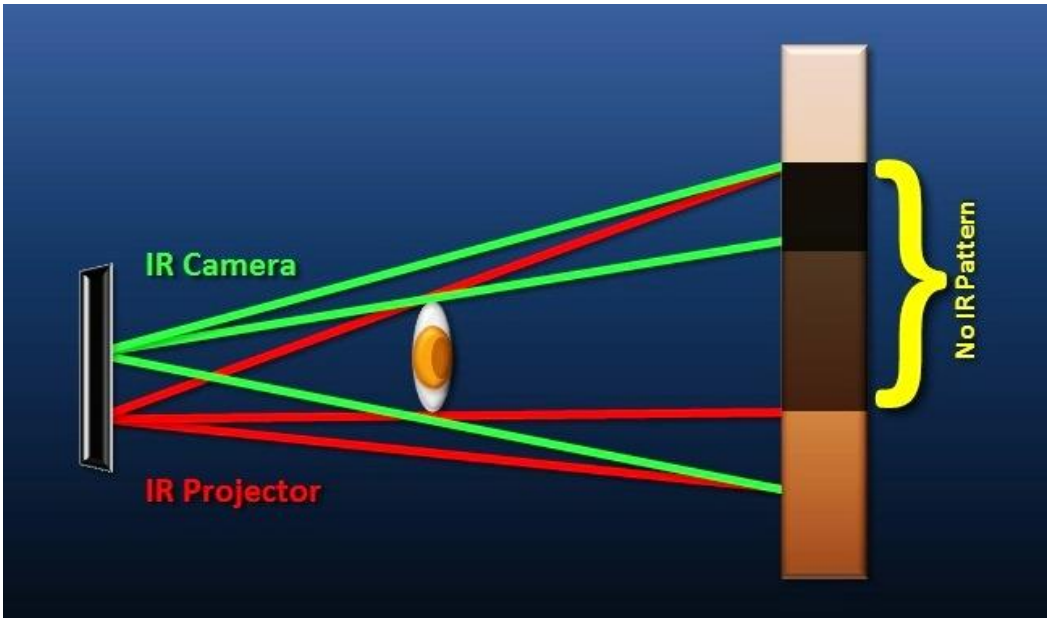


Abbildung 2.3 Kinect Infrarotkamera [msdn15]

2.1.6 Mikrofonarray

In der Kinect Kamera sind vier Mikrofone verbaut. Diese befinden sich in der Schiene an der Unterseite der Kamera. Die Mikrofone sind in einem vordefinierten gleichen Abstand voneinander angebracht. Diese voneinander versetzte Anordnung ermöglicht eine Lokalisierung der Geräuschquelle. Im Rahmen des Projektes wurde auf die Verwendung dieser Funktion verzichtet [sig13].

3 Funktionsumfang und Inbetriebnahme der Software

In diesem Teil der Dokumentation soll ein Überblick über die Simulationssoftware OpenSim gegeben werden. Des Weiteren werden die wichtigsten Aspekte der Benutzeroberfläche vorgestellt. Anschließend soll die Entwicklungsumgebung von Microsoft Visual Studio genauer betrachtet werden. Nach einer Einführung in das Öffnen des Projektes wird ein Einblick in die Einbindung der Kinect ONE Kamera gegeben.

3.1 OpenSim

OpenSim ist eine frei erhältliche Software mit der ein Computermodel des menschlichen Muskel-Skelettes erstellt und analysiert werden kann. Ferner können diese Modelle in eine dynamische Bewegungsimulation eingebunden werden. In diesem Projekt wird mit der Version 3.3 (Jul.2 2015) von OpenSim gearbeitet.

3.1.1 Überblick

OpenSim wurde erstmals 2007 im Rahmen der „Amercian Society of Biomechanics Conference“ vorgestellt. Seit dieser Erstveröffentlichung wird die Software in einem weiten Spektrum an Aufgabenfeldern eingesetzt. Dies beinhaltet den Bereich der Rehabilitation, der Orthopädie, der Robotik, der Ergonomie und des Designs [ops16].

OpenSim basiert auf der Simbody Bibliothek. Hierbei handelt es sich um eine Open Source Engine, die eine Erstellung von mathematischen Modellen mit einer biologischen Dynamik ermöglicht. Programmiert wird diese von Simbios, einem Programmiererteam des „National Center for Biomedical Computation“ an der Universität Stanford [simt15].

3.1.2 Hauptmerkmale

OpenSim beinhaltet ein weites Feld von Tools und Möglichkeiten, um das vom Nutzer ausgewählte Aufgabengebiet bestmöglich zu bearbeiten. In diesem Zusammenhang wird auf die wichtigsten Fähigkeiten der Software eingegangen [sik13].

- Aufnahme von Bildern und Videos des Muskel-Skelett Models
- Plotten von Ergebnissen
- Individuelle Skalierung der Größe des Muskel-Skelett Models
- Durchführung einer invertierten Kinematik Analyse zur Verarbeitung von Markierungspunkten wie sie beispielsweise bei einem Kinect System vorkommen
- Analyse der Simulierungen

3.1.3 Benutzeroberfläche

Um ein besseres Verständnis der Entwicklungsumgebung von OpenSim zu gewährleisten, wird in diesem Abschnitt auf die wichtigsten Punkte der Benutzeroberfläche eingegangen.

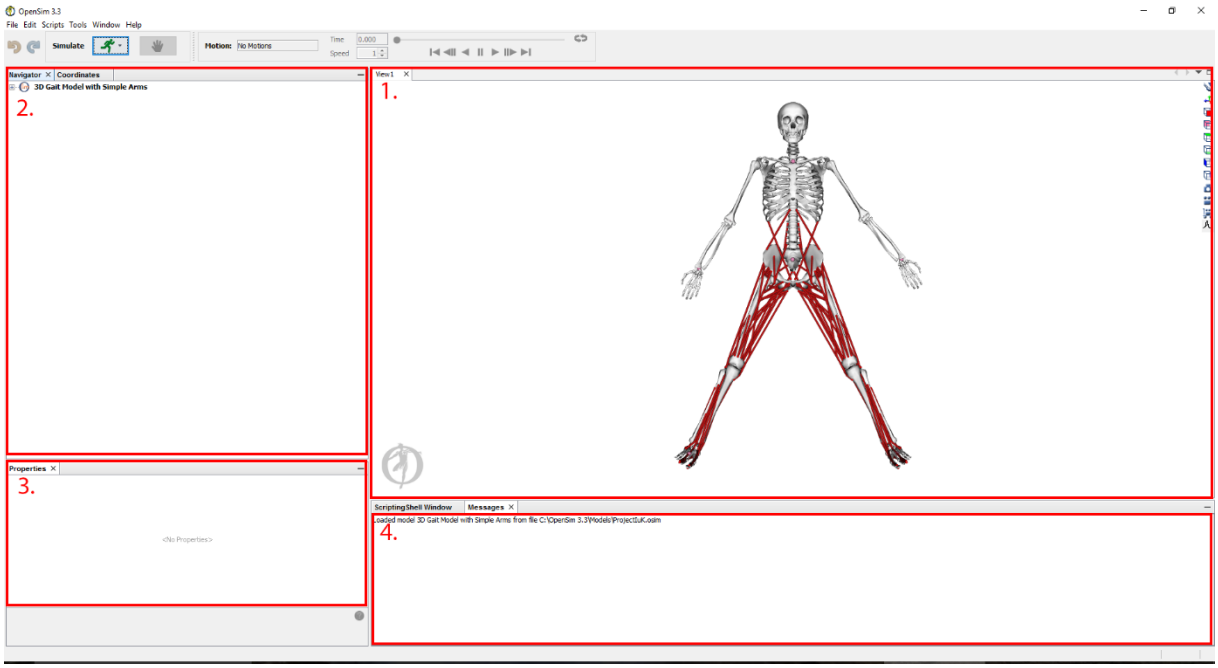


Abbildung 3.1 OpenSim Benutzeroberfläche

Sichtbar ist diese Oberfläche in Abbildung 3.1. Auf die folgenden rot markierten Punkte wird im Detail eingegangen.

1. Viewbox Window
2. Navigator Window
3. Properties Window
4. Message Window

3.1.3.1 Viewbox

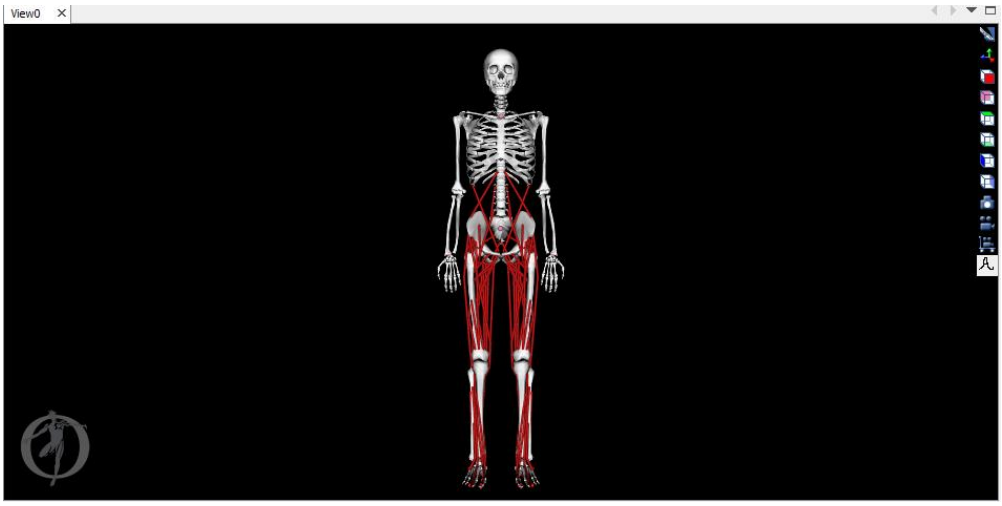


Abbildung 3.2 OpenSim Viewbox

Auf der rechten Seite der Benutzeroberfläche befindet sich die „Viewbox“. In dieser wird das ausgewählte Model angezeigt und jede Bewegungssimulation abgespielt. An der rechten Seite der „Viewbox“ befindet sich eine Navigationsleiste mit verschiedenen Funktionen.

	Farbe des Hintergrundes verändern
	Anzeige des Koordinatensystems
	Sicht aus Richtung der negativen X-Achse
	Sicht aus Richtung der positiven X-Achse
	Sicht aus Richtung der negativen Y-Achse
	Sicht aus Richtung der positiven Y-Achse
	Sicht aus Richtung der negativen Z-Achse
	Sicht aus Richtung der positiven Z-Achse
	Aufnahme eines Fotos vom Model
	Aufnahme eines Videos vom Model
	Erstellung einer Kamera-Fahrt
	Kommentierung von Objekten in dreidimensionaler Ansicht

Tabelle 3.1 Navigationsleiste

3.1.3.2 Navigator

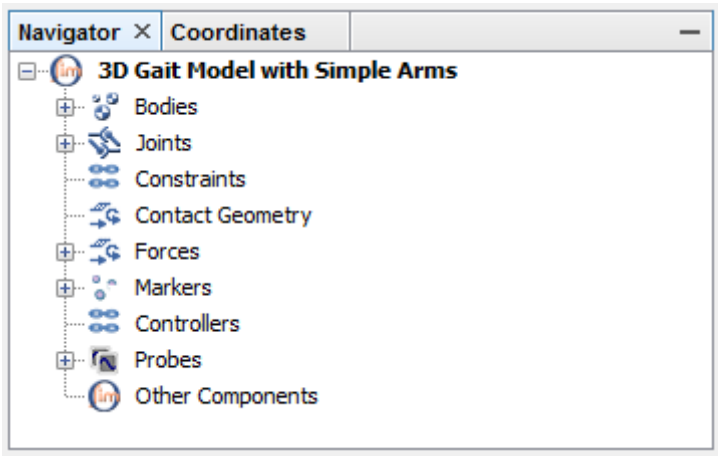


Abbildung 3.3 OpenSim Navigator

Das Navigator Feld kann auf der linken Seite der Benutzeroberfläche gefunden werden. In diesem Beispiel befindet sich ein eingebundenes Model des menschlichen Körpers in dem Feld. Alle wichtigen Optionen des Models können über dieses Fenster erreicht werden. Auf die genauen Punkte des Models wird im späteren Verlauf dieser Dokumentation eingegangen.

3.1.3.3 Properties

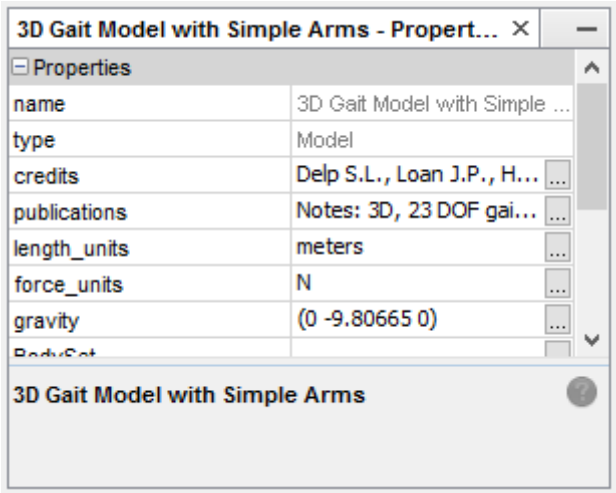


Abbildung 3.4 OpenSim Properties

In der unteren Ecke links des Benutzerinterfaces befindet sich der Bereich *Properties*. Hier sind die Eigenschaften des jeweils ausgewählten Objektes zu erkennen. In diesem Fall wurde das komplette

Model ausgewählt, wodurch die allgemeinen Eigenschaften von diesem angezeigt werden. Im Fall des oben angeführten Beispiels wären das die jeweils verwendeten Einheitentypen, Informationen über die Publikation und Namen der Entwickler. Weiterhin können die gewünschten Maßeinheiten ausgewählt werden. Auch eine Veränderung der simulierten Gravitationskraft ist möglich.

3.1.3.4 Messages Window

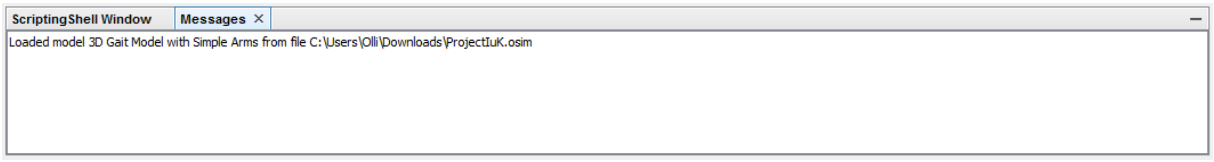


Abbildung 3.5 OpenSim Messages Window

Bei Abbildung 3.5 handelt es sich um das *Messages Window*, in dem alle durchgeführten Aktionen in Textform ausgegeben werden. Auch von OpenSim erkannte Error werden hier ausgegeben, wodurch das Erkennen von Fehlern vereinfacht wird.

3.2 Visual Studio

Für die Durchführung der Programmierung des Aufnahmeprogrammes wurde die Entwicklungsumgebung Visual Studio 2013 (Version 12.0.30501.00 Update 2) verwendet. Für das Einbinden des Programmes muss in einem ersten Schritt über die Schaltfläche Datei, Öffnen Projekt/Projektmappe geöffnet werden. In dem neu geöffneten Fenster muss in das Verzeichnis `PraxisProjektIuK` verwiesen werden. Dort befindet sich die auszuwählende Datei `PraxisprojektIuk.sln`. Das SLN Datenformat steht bei Visual Studio für eine Projektmappe. In einer Mappe befinden sich alle für das Programm notwendigen Verweise und Programmierteile.

Das Projekt wurde in dem *WPF Framework* unter der Programmiersprache *C#* erstellt. *WPF* steht für *Windows Presentation Framework* und ermöglicht das Bearbeiten der Ausgabeoberfläche mit Multimediaelementen.

Nach Öffnen des Projektes kann auf der rechten Seite zwischen den einzelnen *C#*-Teilen gewechselt werden. Des Weiteren kann über den Punkt Verweise die Bibliotheken der Kinect eingesehen werden. Dieses Verfahren wird in dem folgenden Abschnitt genauer betrachtet.

3.2.1 Die Kinect in Visual Studio betreiben

Um eine Verbindung zwischen Desktop-Computer und Kinect aufzubauen, muss das SDK v 2.0 (Software Development Kit) installiert werden. Dieses wird von Microsoft zur freien Verfügung gestellt [mic16]. Da die Kinect ursprünglich für die Konsole XBOX ONE konzipiert wurde, besitzt sie einen Anschluss, der nicht mit einem Desktop-Computer kompatibel ist. Aus diesem Grund muss die Kinect über ein spezielles Netzteil an den Computer angeschlossen werden. Sichtbar ist dieses Netzteil mit USB Adapter in Abbildung 3.6.

Abbildung 3.6 Kinect Adapter [thu16]

Mit der Kinect Bibliothek, die über das Kinect SDK geliefert wird, besteht die Möglichkeit auf alle vorhandenen Funktionen zuzugreifen. Nach Installation des SDKs kann die komplette Bibliothek über den Punkt Verweise in das Programm eingebunden werden. Da es sich um ein Microsoft Verweis handelt, befindet er sich nach Installation bei den Standard Bibliotheken. In Abbildung 3.7 wird das Einbinden eines Verweises zum besseren Verständnis dargestellt.

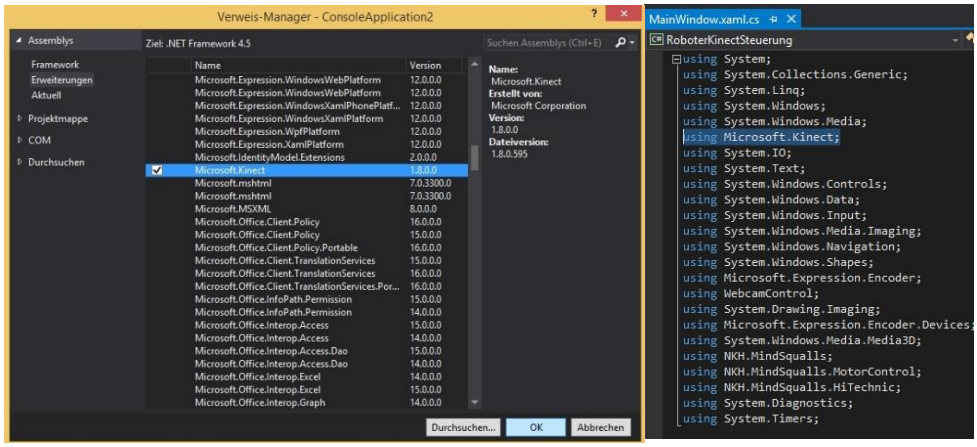


Abbildung 3.7 Hinzufügen des Kinect Verweises

4 Projektbeschreibung

Im Rahmen dieses Kapitels wird die Zielsetzung und der dafür verwendete Lösungsweg dieses Projektes genauer betrachtet. Hierfür soll im ersten Unterkapitel das detaillierte Projektziel thematisiert werden. Anschließend wird ein Einblick in die durchgeführte Programmierung unter Visual Studio gegeben, um ein Verständnis für die Funktionen und Entstehung der einzelnen Programmteile zu erhalten.

4.1 Projektziel

Das Ziel dieses IuK Praxisprojektes ist die Speicherung von Körperdaten während einer Bewegung und das Einbindung dieser Daten in die Simmulationssoftware OpenSim. Hierfür sollen die Koordinaten der Gelenkpunkte in einem vom Nutzer selbst festgelegten Zeitraum aufgenommen werden.

4.2 Programmierung in Visual Studio

In den folgenden Unterkapiteln soll auf den Aufbau des Programmes eingegangen werden. Zu Beginn wird die Körpererkennung der Kinect erläutert. Außerdem wird die Betrachtung des aufgenommenen Bildes durch die Farbkamera und die Einbindung der Gelenkpunkte als rote Ellipsen beschrieben. Anschließend soll die Programmierung der Aufnahmefunktion erläutert werden. Um dies zu gewährleisten muss auf die Speicherung der Gelenkpunkte als Koordinaten und dessen Berechnungsweg genauer eingegangen werden. Abschließend wird die Formatierung in das richtige Datenformat und das Einbinden in OpenSim erklärt.

4.2.1 Programmablauf

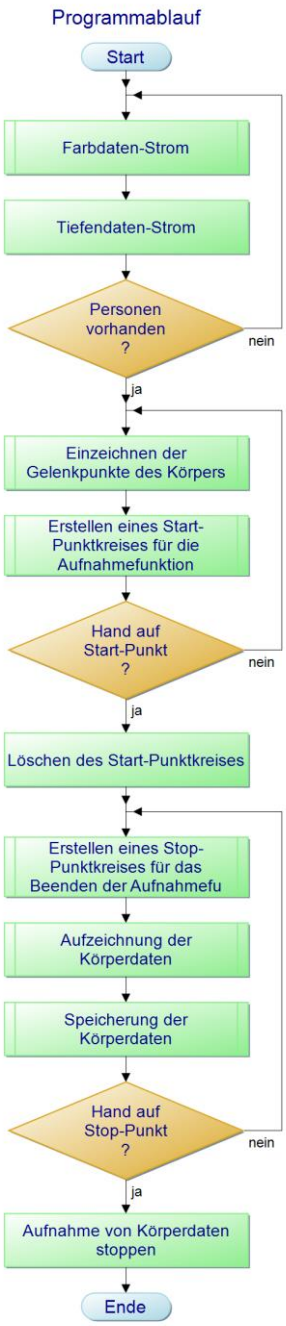


Abbildung 4.1 Programmablauf

Bei dem Start der Applikation werden das Video- und das Tiefensignal der Kinect Kamera gestartet. Wird ein menschlicher Körper erkannt, dann werden die Gelenkpunkte in Form von roten Ellipsen als Overlay auf das Bild gelegt. Gleichzeitig wird ein grüner Kreis auf Höhe der Brust erstellt, der bei manueller Berührung die Aufnahme startet. Nach fünf Sekunden werden die Daten der markierten Gelenkpunkte aufgenommen. Gleichzeitig wird der Kreis, der zum Starten erstellt wurde, gelöscht und durch einen neuen Kreis ersetzt. Bei Betätigung dieses Kreises durch die Hand beendet sich die Applikation und die Aufzeichnung der Daten wird gestoppt. Die Daten sind nun in einer Datei im Verzeichnis des Programmes zu finden.

4.2.2 Datenströme der Kinect Kamera

Das Video- und Tiefensignal in diesem Programm funktioniert über die `MultiSourceFrameReader` Klasse der Kinect Libary. In dieser sind alle Datenstreams der Kinect gebündelt und über einen *Eventhandler* kann auf diese in einer Methode zugegriffen werden [msdn16b]. Bei einem Eventhandler handelt es sich um eine Methode, die sich nach dem starten in einer Schleife befindet. Ein weiterer Quellcodeausschnit 4.1 zeigt die Verwendung der `MultiSourceFrameReader` Klasse in diesem Projekt.

```
msfr = kinectSensor.OpenMultiSourceFrameReader(FrameSourceTypes.Color |
FrameSourceTypes.Body);

msfr.MultiSourceFrameArrived += msfr_MultiSourceFrameArrived;
```

Quellcodeausschnitt 4.1 MulitsourceFrameReader

Der *Eventhandler* sendet kontinuierlich die ausgewählten Daten-Ströme an die Methode `MuliSourceFrameArrived`. Welches Signal hierbei übertragen wird, bestimmt sich aus dem gewählten `FrameSourceType` [msdn16c]. Alle Auswählmöglichkeiten können in Tabelle 4.1 gefunden werden.

FrameSourceType	**Beschreibung**
Audio	Audio-Signal
Body	Körper-Signal
BodyIndex	Körpersilouette-Signal
Color	Video-Signal
Depth	Tiefen-Signal
Infrared	Infrarot-Signal
LongExposureInfrared	Infrarot-Signal mit längerer Belichtungszeit
None	Kein Signal

Tabelle 4.1 SourceTypen

In einem nächsten Schritt soll auf die Methode `MultiSourceFrameArrived` eingegangen werden. Diese arbeitet mit dem Video-Signal und dem Körper-Signal. In dem folgenden Quellcodeausschnitt 4.2 soll die Verarbeitung dieser Signale genauer betrachtet werden.

```
var reference = e.FrameReference.AcquireFrame();
using (var frame = reference.ColorFrameReference.AcquireFrame())
       {
        if (frame != null)
              {
                     (…)
              }
       }

using (var frame = reference.BodyFrameReference.AcquireFrame())
       {
        if (frame != null)
              {
                     (…)
              }

       }
```

Quellcodeausschnitt 4.2 Methode MultiSourceFrameArrived

In einem ersten Schritt werden die übertragenen *Datenstreams* der lokalen *Objectvariable* `reference` zugewiesen. Anschließend wird mit Hilfe einer `using` Anweisung das gewünschte Signal ausgewählt. Nach einer Abfrage auf die Vollständigkeit des Signales kann mit den Daten gearbeitet werden.

4.2.3 Videosignal

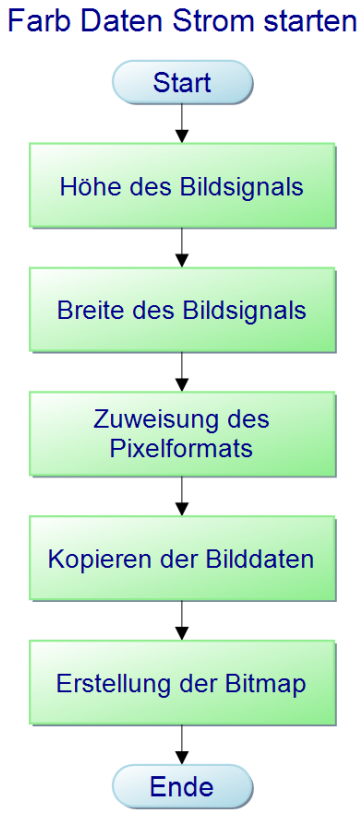


Abbildung 4.2 Farb Daten Srom starten

Das Video-Signal wird über die Klasse `ToBitmap` in die Benutzerfläche der Applikation eingebunden. In dieser werden die Höhe und Breite aus dem Videosignal ausgelesen. Des Weiteren muss das richtige Pixelformat ausgewählt werden. Für das Video-Signal der Kinect Kamera muss hierbei das Bgr32 (32 Bits pro Pixel für jeweils drei Farbkanäle) Format gewählt werden. In einem nächsten Schritt werden die Bilddaten in ein *Array* kopiert.

```
BitmapSource.Create(width, height, dpiX, dpiY, pixelformat, bitmappalette, pixels,
stride);
```

Quellcodeausschnitt 4.3 BitmapSource

Im Quellcodeauschnitt 4.3 befindet sich die abschließende Erstellung der Bitmap. Dies passiert über einen im *.NET Framework* festgelegten Standard [msdn16d]. Für ein besseres Verständnis der benötigten Werte sollen diese in Tabelle 4.2 beschrieben werden.

Width	Höhe der Bitmap
Height	Breite der Bitmap
dpiX	Horizontalen Dots per Inch der Bitmap
dpiY	Vertikalen Dots per Inch der Bitmap
PixelFormat	Das Pixelformat der Bitmap
BitmapPalette	Die Farbpalette der Bitmap
Pixels	Inhalt der Bitmap
Stride	Pixel Größe in Bytes der Bitmap

Tabelle 4.2 Bitmap Eigenschaften

4.2.4 Körper Daten Strom

In den folgenden Unterkapiteln wird die Verwendung der Tiefendaten der Kinect Kamera beschrieben. Hierfür sollen in einem ersten Teil die Grundlagen der Körperdatenerkennung durch die Kamera erläutert werden. Anschließend werden die auf Basis dieser Grundlage erstellten Gelenk-, Start- und Stopp-Punkte betrachtet.

4.2.4.1 Grundlagen

Eine wichtige Basis für dieses Projekt ist die Arbeit mit der Body (Körperdaten) Klasse der Kinect ONE Kamera. Diese ermöglicht den Zugriff auf die Gelenkpunkte von den Personen, die sich im Bereich der Infrarotkamera befinden. Die Gelenkpunkte werden in Form von *Joints* (englischer Begriff für Gelenke) dargestellt. Insgesamt 25 *Joints* können durch die Kinect Kamera erkannt werden [msdn16e]. Mit folgender Abbildung 4.3 soll ein besseres Verständnis für die Position dieser Punkte ermöglicht werden.

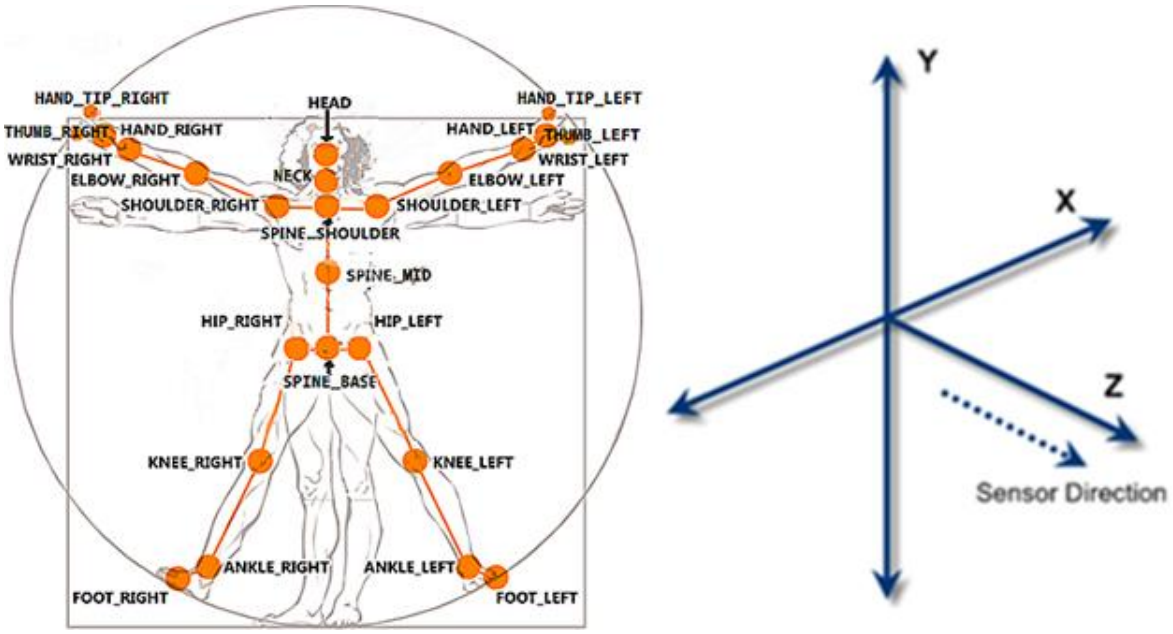


Abbildung 4.3 Position der Joint Punkte [ggp16]

Verbindet man alle *Joints* erhält man ein angedeutetes Skelett, wobei die Verbindungslinien die Knochen zwischen den Gelenken darstellen sollen.

Die Joint-Positionen werden durch X, Y und Z Werte dargestellt. Wie auf der rechten Seite von Abbildung 4.3 sichtbar, handelt es sich hierbei um ein Koordinatensystem, in dem Z für die Entfernung zwischen Joint und Kinect steht.

4.2.4.2 Gelenkpunkte

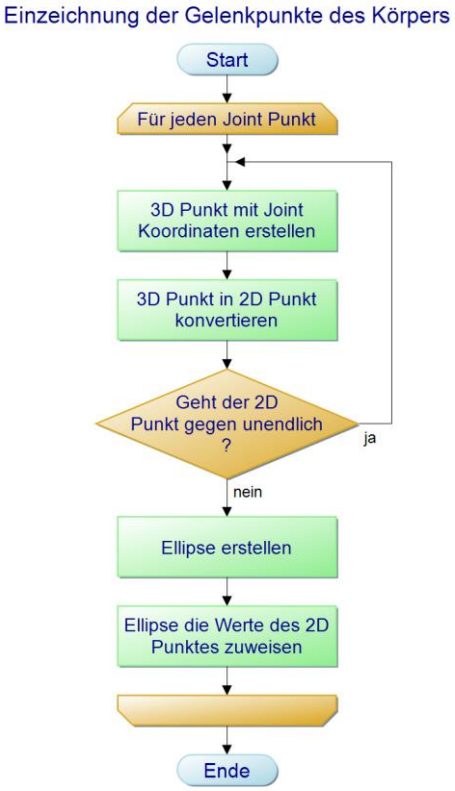


Abbildung 4.4 Einzeichnung der Gelenkpunkte des Körpers

Die Erstellung der *Joint* Punkte als visuelle Rückmeldung für den Nutzer wird über eine for-Schleife gestartet. Nachdem ein Körper erkannt wurde, wird diese Schleife gestartet und läuft für jeden erkannten Joint Punkt einmal durch. In einem ersten Schritt wird ein dreidimensionaler Punkt erstellt. Dieser wird in einem nächsten Schritt in einen zweidimensionalen Punkt umgewandelt. Anschließend muss eine Abfrage durchgeführt werden, die überprüft, ob die Koordinaten des zweidimensionalen Punktes kleiner als unendlich sind. Ist dieses Kriterium erfüllt, wird eine Ellipse erstellt. Diese bekommt die X- und Y-Werte des zweidimensionalen Punktes und wird auf das Videosignal aufgetragen.

```
CameraSpacePoint jointPosition = joint.Position;

ColorSpacePoint colorPoint =
kinectSensor.CoordinateMapper.MapCameraPointToColorSpace(jointPosition);
Point point1 = new Point();

point1.X = float.IsInfinity(colorPoint.X) ? 0 : colorPoint.X;
point1.Y = float.IsInfinity(colorPoint.Y) ? 0 : colorPoint.Y;
```

Quellcodeausschnitt 4.4 Coordinatemapping

Im Quellcodeauschnitt 4.4 wird die Umformung eines Punktes aus dem dreidimensionalen in den zweidimensionalen Raum dargestellt. Verwendet wird dabei die Methode `CoordinateMapper`. Dies ist eine Methode der Kinect Bibliothek von Microsoft [msdn16f]. Die aufgezeichneten Joint Punkte werden in einer Auflösung von 512×424 Pixel aufgezeichnet. Der Farbdatenstrom besitzt eine Auflösung von 1920x1080 Pixel. Die Joint Punkte können also nicht direkt als Overlay auf das Videosignal gelegt werden, sondern müssen erst konvertiert werden. Diese Konvertierung findet über den `CoordinateMapper` statt. Inwiefern diese Methode die Werte im Detail umrechnet, ist nicht bekannt.

4.2.4.3 Einbindung eines Start- und Stopp-Punktes

Damit das Projektziel, die Aufnahme von den Koordinaten der *Joint* Positionen realisiert werden kann, müssen Start- und Stopp-Punkte definiert werden. Da sich der Benutzer entfernt von Tastatur und Maus befindet, werden jene Punkte benötigt. Diese wurden mit Hilfe von Ellipsen durchgeführt.

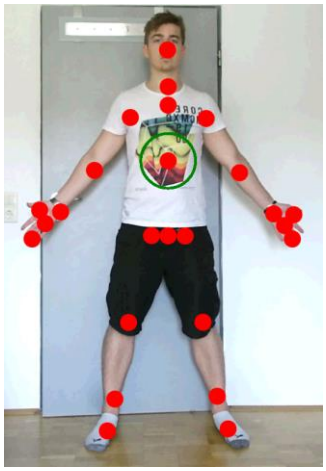


Abbildung 4.5 Start Punkt

Positioniert ist dieser Start-Kreis, wie in Abbildung 4.5 erkennbar, im Bereich des mittleren Torsos.

Für einen Stopp der Aufnahme wurde ein Punkt gewählt, der sich auf einer höhreren Position als der Start-Punkt auf dem Torso befindet.

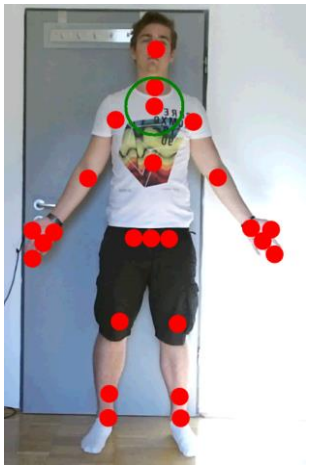


Abbildung 4.6 Stopp-Punkt

Erkennbar in Abbildung 4.6 ist, dass der Kreis in der Mitte zwischen den beiden Schulter-Punkten positioniert wurde. Durch seine Lage wird ein ungewolltes Stoppen der Applikation verhindert.

Programmiertechnisch wurden bei beiden Kreisen auf dem Prinzip der Joint Punkt Erstellung in Kapitel 4.2.4.2 aufgebaut. Statt alle Gelenkpunkte zu markieren, wurden nur einzelne Joint Punkte ausgewählt. Für den Start-Punkt wurde die Position von *SpineMid* (Mittlere Wirbelsäule) gewählt und für den Stopp-Punkt die Position von *SpineShoulder* (Wirbelsäulen-Schulterpunkt). An dieser Stelle ist es wichtig zu erwähnen, dass es sich bei beiden Punkten nur um visuelle Punkte handelt. Das Auslösen der beiden Punkte wird in einem späteren Verlauf dieser Dokumentation erläutert.

Für ein bestmögliches Ergebnis bei der Darstellung der Gelenk-, Start- und Stopp-Punkte wird eine Entfernung von 2.70 Meter zu der Kinect Kamera empfohlen.

4.2.5 Aufnahme

In den folgenden Unterkapiteln soll Schritt für Schritt erklärt werden, wie die Aufnahme der Joint Positionen strukturiert ist. Zu Beginn müssen hierfür die notwendigen Grundlagen für die richtige Formatierung der aufgezeichneten Körperdaten erklärt werden.

4.2.5.1 Vorbereitung des Koordinatensystems

Eine Grundlage bei der Verarbeitung der Gelenk Punkte in diesem Projekt ist die richtige Position des Koordinatenursprunges. Bei der Kinect Kamera befindet sich dieser in der Mitte der Kamera [msdn16g]. Bei OpenSim hingegen befindet sich der Koordinatenursprung exakt auf halben Weg zwischen den Füßen der aufgenommenen Person [sic15b].

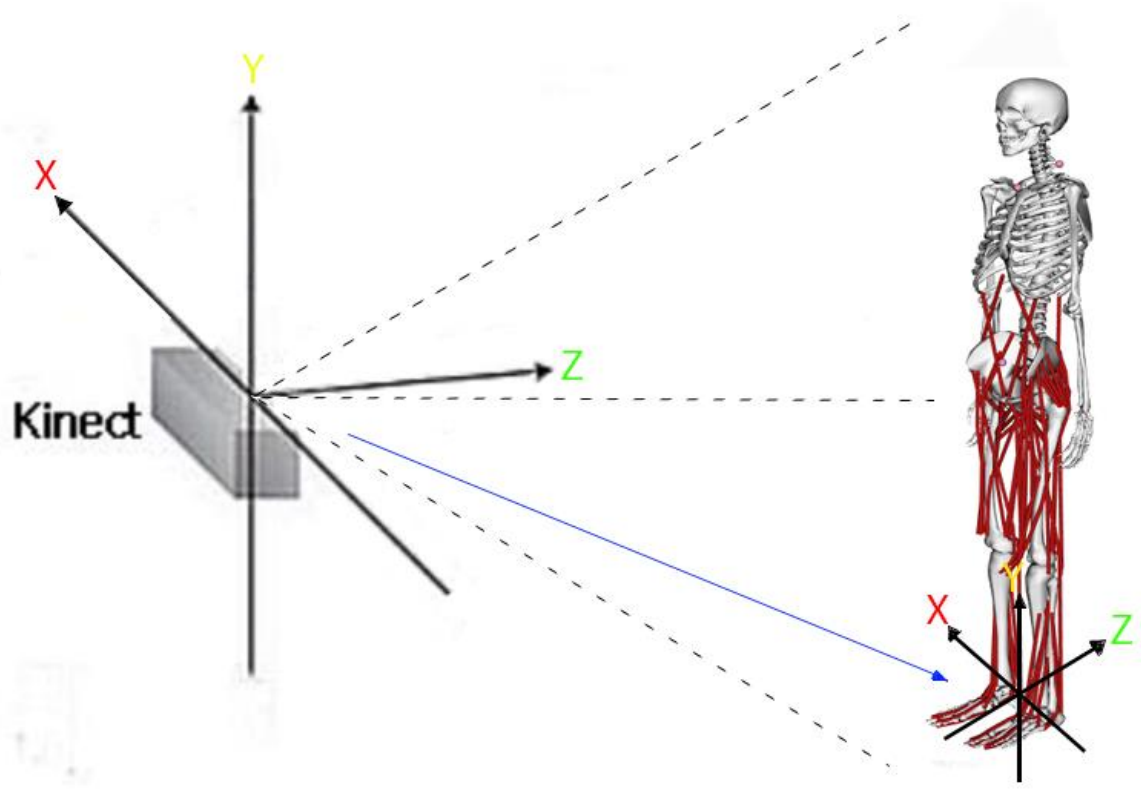


Abbildung 4.7 Verschiebung des Koordinatenursprunges

In Abbildung 4.7 ist sichtbar, wie der Koordinatenursprung von der Kinect (links) auf einen Punkt zwischen die Füße des Benutzers verschoben wird (rechts). Der blaue Pfeil soll diese Verschiebung

visualisieren. Um dies in der Berechnung der Joint Positionen einfließen zu lassen, muss eine Subtraktion mit dem neuen Ursprung durchgeführt werden.

$$\begin{pmatrix} JointPosition_x \\ JointPosition_y \\ JointPosition_z \end{pmatrix} - \begin{pmatrix} NeuesKoordinatensystem_x \\ NeuesKoordinatensystem_y \\ NeuesKoordinatensystem_z \end{pmatrix} = \begin{pmatrix} NeueJointPosition_x \\ NeueJointPosition_y \\ NeueJointPosition_z \end{pmatrix}$$

Formel 4.1 Koordinatenursprung verschieben

Sichtbar ist diese Subtraktion in Formel 4.1. Der Vektor JointPosition steht hierbei für die Jointposition eines beliebigen Gelenkpunktes und der Vektor NeuesKoordinatensystem für den schon erwähnten Punkt zwischen den Füßen. Der Rest der Subtraktion dieser beiden Vektoren ergibt die Entfernung der jeweiligen Joint Positionen zu dem neuen Ursprung.

Nach dieser Verschiebung des Koordinatenursprungs, kann mit der Transformation der Achsen begonnen werden. Die Y-Achse ist bei beiden Koordinatensystemen identisch, aber die X- und Z-Achsen sind vertauscht. Um das Koordinatensystem anpassen zu können, muss eine Rotation um -90 Grad um die Y-Achse durchgeführt werden [bho16].

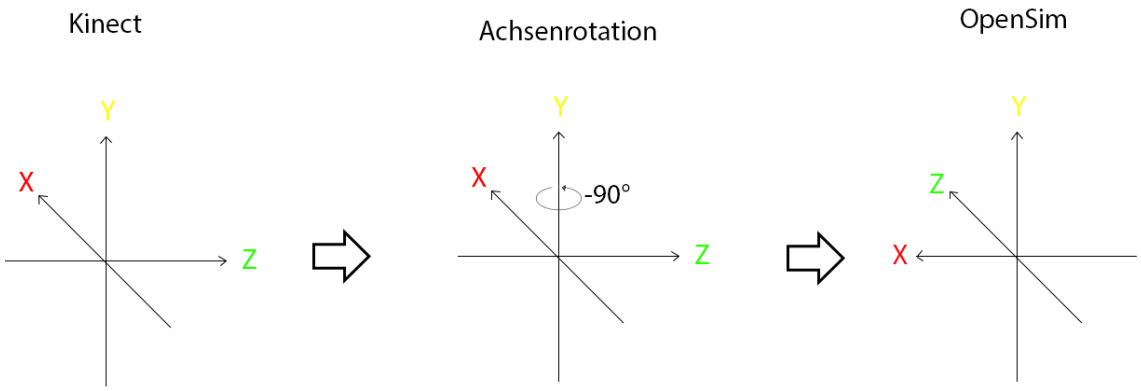


Abbildung 4.8 Achsenrotation

Die hierfür benötigten Schritte wurden schematisch in Abbildung 4.8 dargestellt. Auf der linken Seite ist das Koordinatensystem der Kinect abgebildet und auf der rechten Seite das nach der Rotation hergeleitete OpenSim Koordinatensystem.

Um dies rechnerisch zu realisieren, muss eine Rotationsmatrix für die Drehung um die Y-Achse verwendet werden [bho16].

$$R_y(\alpha) = \begin{pmatrix} \cos(\alpha) & 0 & \sin(\alpha) \\ 0 & 1 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{pmatrix}$$

Formel 4.2 Rotationsmatrix

Die Variable α steht hierbei für den gewählten Drehwinkel. Multizipliert man den in Formel 4.2 erstellten Vektor $NeueJointPosition_{XYZ}$ auf diese Matrix, erhält man die gewünschten Vektoren im Koordinatensystem von OpenSim.

4.2.5.2 Das TRC Datenformat

Eine wichtige Grundlage für das Speichern der aufgenommenen Körperdaten ist das TRC (Track Row Column) Datenformat. Dieses ist für das Verarbeiten der Körper-Daten in OpenSim notwendig. Das TRC Datenformat wurde von der Motion Analysis Corporation eingeführt, um die Positionen von festgelegten Körperpunkten bei einem Motion Capture System in einem zeitlichen Rahmen spezifizieren zu können [sic13c].

1	PathFileType	4	(X/Y/Z)	kinectSkeleton.trc							
2	DataRate	CameraRate	NumFrames	NumMarkers	Units	OrigDataRate	OrigDataStar	OrigNumFrames			
3	60.00	60.00	147	17	mm	60.00	1	300			
4	#Frame	Time	Neck			ShoulderMid			R.Shoulder		
5			X1	Y1	Z1	X2	Y2	Z2	X3	Y3	Z3
6											
7	1	5.170.000	92.260.417	-1.988.188	50.618.210	772.339	14.036.053	42.658	-150.115	13.538.045	1.920.618
8	2	5.330.000	92.219.927	-1.990.031	50.631.953	771.839	14.034.036	426.651	-15.525	13.536.326	1.918.481
9	3	5.400.000	92.052.006	-1.990.951	50.588.668	770.355	1.403.764	426.308	-154.728	13.539.256	1.918.084
10	4	5.460.000	91.951.658	-199.123	50.780.902	769.502	1.403.571	427.851	-155.748	13.537.909	1.918.134
11	5	5.500.000	91.951.658	-199.123	50.780.902	769.502	1.403.571	427.851	-155.748	13.537.909	1.918.134
12	6	5.530.000	91.951.658	-199.123	50.780.902	769.502	1.403.571	427.851	-155.748	13.537.909	1.918.134
13	7	5.590.000	91.762.922	-1.994.341	50.567.372	767.825	14.032.401	425.891	-159.809	13.534.637	1.915.488
14	8	5.630.000	91.762.922	-1.994.341	50.567.372	767.825	14.032.401	425.891	-159.809	13.534.637	1.915.488
15	9	5.660.000	91.762.922	-1.994.341	50.567.372	767.825	14.032.401	425.891	-159.809	13.534.637	1.915.488
16	10	5.700.000	91.762.922	-1.994.341	50.567.372	767.825	14.032.401	425.891	-159.809	13.534.637	1.915.488
17	11	5.760.000	90.887.749	-1.996.496	50.483.104	760.138	14.038.001	425.153	-167.811	13.537.111	191.308

Abbildung 4.9 TRC Datenformat

In den ersten drei Zeilen befindet sich der *Header* (Kopfzeile). In diesem sind alle Grundinformationen für die Datei abgespeichert. Bei der ersten Zeile handelt es sich um drei genormte Attribute des TRC Daten Formates und abschließend den Namen der Datei.In den darauffolgenden zwei Zeilen werden die Eigenschaften dieser TRC File genannt. Die für diese Beschreibung notwendigen Begriffe sollen nun im Einzelnen in Tabelle 4.3 erläutert werden [par09].

DataRate	Abtastrate in Hz
CameraRate	Camera Frequenz
NumFrames	Anzahl an aufgezeichneten Frame
NumMarkers	Anzahl der verwendeten Marker
Units	Einheit in der gemessen wurde
OrigDataRate	Standard TRC Data Rate
OriginalDataStartFrame	Standard TRC Data Start Frame
OriginalNumFrames	Standard TRC Anzahl von Frames

Tabelle 4.3 Eigenschaften des Headers

In Zeile 4 der Abbildung 4.9 wird anschließend das Reihensystem definiert, in das die aufgezeichneten Daten gespeichert werden. Dieses beinhaltet die Nummer des Frames und der dazugehörige Zeitpunkt an dem er aufgenommen wurde. Darüber hinaus werden die verwendeten Marker einzeln aufgezählt. Dies ermöglicht eine genaue Zuweisung der jeweiligen X-, Y- und Z-Werte zu den aufgezeichneten Markern.

Ein weiterer wichtiger Punkt bei dem TRC Datenformat ist die richtige Formatierung. Zwischen den einzelnen Werten und Definitionen wird immer mit einem Tab getrennt [sic13c]. Folgender Quellcodeauschnitt der Erstellung von Zeile 4 beschreibt exemplarisch, wie die Abstände im TRC Datenformat einzuhalten sind.

```
Frame#<tab>Time<tab>R.ASIS<tab><tab><tab>L.ASIS<tab><tab><tab>V.Sacral<tab> (…)
```

Quellcodeausschnitt 4.5 Abstandsformatierung

Zwischen den jeweiligen Marker Punkten muss mit drei Tabs getrennt werden. Bei allen anderen Werten und Definitionen wird nur mit einem Tab getrennt.

4.2.5.3 Beginn der Aufnahme

Damit die Aufnahme der Koordinaten der Joint Positionen beginnen kann muss mit der in Kapitel 4.2.3.3 beschriebenen Start-Ellipse interagiert werden. Durch das „Berühren“ des Kreismittelpunktes mit der rechten Hand, wird ein fünf sekündiger Countdown gestartet. Nach Ablauf der Zeit beginnt die Aufnahme der Joint-Positionen.

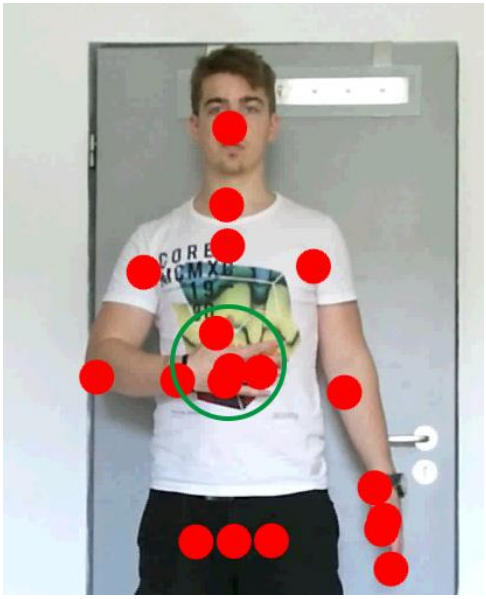


Abbildung 4.10 Berühren des Start Punktes

Abbildung 4.10 zeigt die nötige Bewegung mit dem rechten Arm. Die Hand muss auf den Mittelpunkt der grünen Ellipse bewegt werden. Programmiertechnisch wurde dies durch eine Abfrage realisiert.

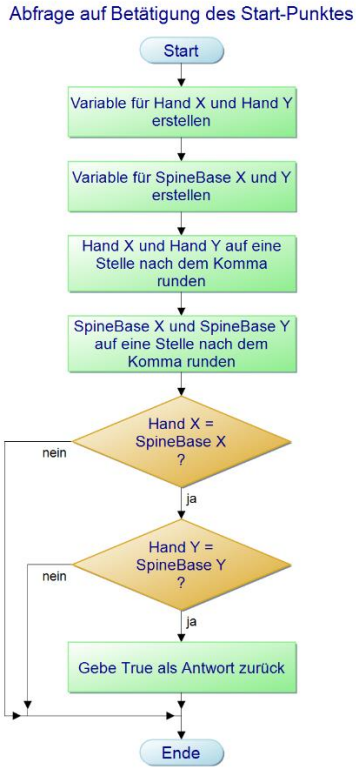


Abbildung 4.11 Abfrage auf Betätigung des Start Punktes

Diese Abfrage wird in dem Programmablaufplan in Abbildung 4.11 beschrieben. Nach der Erstellung von Variablen mit den X und Y Werten von *SpineMid* und *HandRight* werden diese in das Format `double` konvertiert. Dies ermöglicht das Arbeiten mit mathematischen Funktionen. Dadurch können in dem nächsten Schritt die Werte auf eine Stelle nach dem Komma gerundet werden. Realisiert wird dies durch die Funktion `Math.Round [msdn16h]`. Diese rundet die gewünschte `double` Variable mit doppelter Genauigkeit auf eine angegebene Anzahl von Nachkommastellen. Dies ist notwendig, da die Abfrage auf gleiche X und Y Werte sonst eine zu genaue Positionierung der Hand benötigen würde. Abschließend wird die boolesche Abfrage `isAshighAsSpineMid` durchgeführt. Sichtbar ist diese in dem folgenden Quellcodeausschnitt 4.6.

```
bool isAshighAsSpineMid = handXdoubletoeven == spineBaseXdoubletoeven &&
handYdoubletoeven == spineBaseYdoubletoeven;
return isAshighAsSpineMid;
```

Quellcodeausschnitt 4.6 Abfrage auf Gleichheit der Koordinaten

Mit dieser wird abgefragt ob die abgerundeten X und Y Werte von *HandRight* und *SpineMid* übereinstimmen. Das doppelte *UND* Zeichen sichert, dass nur bei Übereinstimmung von beiden Werten ein *true* gesetzt werden kann. Ist dies der Fall wird eine `IsAshighAsSpineMid` als boolescher Wert *true* (wahr) zurückgegeben.

4.2.5.4 Berechnung der Koordinaten

Für die Berechnung der Joint Koordinaten müssen die in Kapitel 4.2.5.1 erläuterten Grundlagen für das verwenden des richtigen Koordinatensystems, verwendet werden.

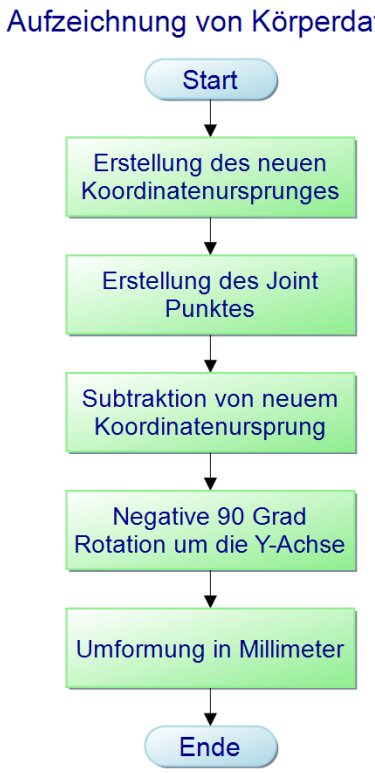


Abbildung 4.12 Aufzeichnung der Körperdaten

In einem ersten Arbeitsschritt muss der neue Koordinatenursprung erstellt werden. Des Weiteren muss in einem nächsten Schritt ein Joint Punkt erstellt und die Koordinaten des neuen Koordinatenursprunges von ihm subtrahiert werden. Anschließend wird die Rotation um Y-Achse durchgeführt. Abschießend werden die Werte in die Messeinheit Millimeter umgeformt. Dies ist notwendig, da das Datenformat für die spätere Speicherung diese Messeinheit verlangt.

Im folgendem Quellcodeausschnitt 4.7 soll die Erstellung des neuen Koordinatenursprunges erläutert werden.

```
Vector3D sourceP = new Vector3D(body.Joints[JointType.SpineBase].Position.X,
body.Joints[JointType.FootRight].Position.Y,
body.Joints[JointType.SpineBase].Position.Z);
```

Quellcodeausschnitt 4.7 Neuer Koordinatenursprung

Damit sich dieser wie in Kapitel 4.2.5.1 auf halben Weg zwischen den Füßen befindet muss ein spezieller Vektor Punkt als neuer Koordinatenursprung erstellt werden. Dieser basiert, wie im Quellcode sichtbar, auf den X und Z Werten von *SpinBase* (Untere Wirbelsäule). Des Weiteren wird der X Wert von *FootRight* (Rechter Fuß) verwendet. Schematisch sichtbar ist dies in Abbildung 4.13.

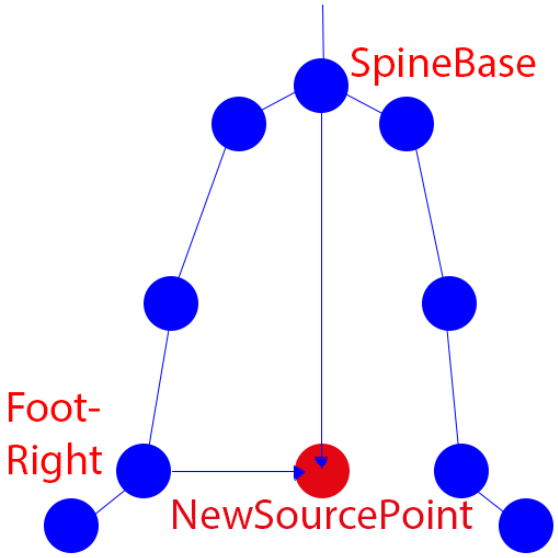


Abbildung 4.13 Zusammensetzung des Koordinatenursprunges

Der folgende Quellcodeausschnitt 4.8 beinhaltet die Erstellung eines Joint Punktes und die Rotation um die Y-Achse. Des Weiteren wird die Konvertierung der Werte in Millimeter beschrieben.

```
var jointPositionX = body.Joints[JointType.Joint].Position.X - sourceP.X;
var jointPositionY = body.Joints[JointType.Joint].Position.Y - sourceP.Y;
var jointPositionZ = body.Joints[JointType.Joint].Position.Z - sourceP.Z;

var rotateJointPositionX = jointPositionX * Math.Cos(-90) + jointPositionZ *
Math.Sin(-90);
var rotateJointPositionY = jointPositionY;
var rotateJointPositionZ = jointPositionX * -Math.Sin(-90) + jointPositionZ
Math.Cos(-90);

double jointPositionXToDouble = Convert.ToDouble(rotateJointPositionX);
double jointPositionYToDouble = Convert.ToDouble(rotateJointPositionY);
double jointPositionZToDouble = Convert.ToDouble(rotateJointPositionZ);

double jointPositionXToMillimeters = jointPositionXToDouble * 1000;
double jointPositionYToMillimeters = jointPositionYToDouble * 1000;
double jointPositionZToMillimeters = jointPositionZToDouble * 1000;

jointX = jointPositionXToMillimeters.ToString();
jointY = jointPositionYToMillimeters.ToString();
jointZ = jointPositionZToMillimeters.ToString();
```

Quellcodeausschnitt 4.8 Erstellung eines Joint Punktes

Zu Beginn werden Variablen mit den X, Y und Z Werten des jeweiligen Joints erstellt. Darüber hinaus wird dieser Joint mit den Koordinaten des neuen Koordinatenursprunges subtrahiert. Anschließend wird die Rotation mit der Rotationsmatrix aus Kapitel 4.2.5.1 durchgeführt. Um die Werte richtig darstellen zu können müssen Sie in einem nächsten Schritt in das *double* Format umgewandelt werden. Das *double* Format ist notwendig um die Werte später in ein ausgebbares *string* Format konvertieren zu können. Damit die Daten in OpenSim einlesbar sind, müssen die Koordinaten in ein anderes Längenformat umgewandelt werden. Hierfür werden die Werte von dem standardisierten Meterformat in das Millimeterformat konvertiert und globalen Strings zugeordnet. Dies ermöglicht den Zugriff auf diese Daten von jeder beliebigen Stelle des Programmes. Wichtig wird dies im folgenden Unterkapitel der Speicherung der Koordinaten.

4.2.5.5 Speicherung der Koordinaten

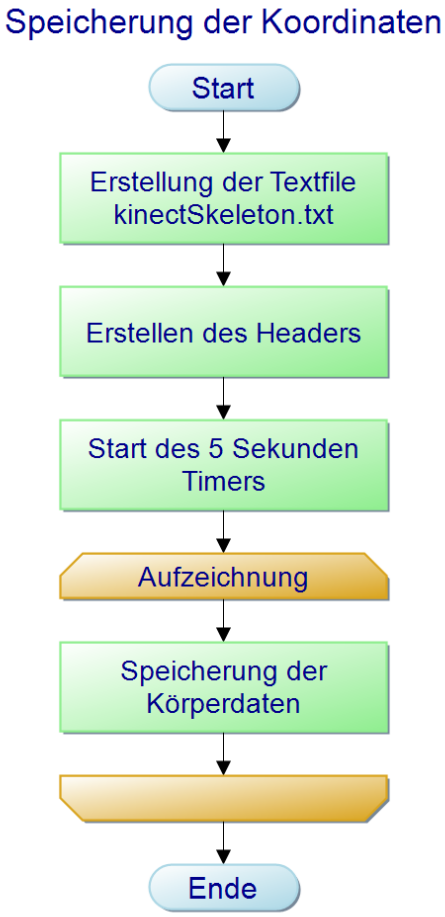


Abbildung 4.14 Abspeicherung der Körperdaten

Die Speicherung der Körperdaten funktioniert nach dem im Abbildung 4.14 sichtbaren System. Nach Erstellung der Textfile kinectSkeleton.txt und des Einfügens des Headers (Ersten 6 Zeilen des TRC Datenformates) wird ein *Timer* von fünf Sekunden gestartet. Nach Ablauf dieser Zeit beginnt die Speicherung der Körperdaten. Für diese Speicherung muss die nötige Formatierung für das TRC Datenformat aus Kapitel 4.2.5.2 berücksichtigt werden. Dies bedeutet auf die festgelegten Standards dieses Formates geachtet werden. Die Realisierung des Headers in C# soll mit Hilfe des folgenden Quellcodeausschnittes 4.9 erklärt werden.

```
using (System.IO.StreamWriter file =
new System.IO.StreamWriter(@"Output/kinectSkeleton.txt", true))
    {
       file.WriteLine(pathFileType + "\t" + top2 + "\t" + coordinateSystem + "\t" +
       filename);
       file.WriteLine(dataRate + "\t" + cameraRate + "\t" + numFrames + "\t" +
       numMarkers + "\t" + units + "\t" + origDataRate + "\t" + origDataStartFrame +
       "\t" + origNumFrames);
       file.WriteLine(dataRateValue + "\t" + cameraRateValue + "\t" + numFrameValue
       + "\t" + numMarkersValue + "\t" + unitsValue + "\t" + origDataRateValue +
       "\t" + origDataStartFrameValue + "\t" + origNumFramesValue);
       file.WriteLine(frame + "\t" + time + "\t" + neck + "\t" + "\t" + "\t" +
       shoulder_C + "\t" + "\t" + "\t" + shoulderRight + "\t" + "\t" + "\t" +
       shoulderLeft + "\t" + "\t" + "\t" + elbowRight + "\t" + "\t" + "\t" +
       elbowLeft
       + "\t" + "\t" + "\t" + rightWrist + "\t" + "\t" + "\t" + leftWrist + "\t" +
       "\t" + "\t" + hipCenter + "\t" + "\t" + "\t" + hipRight + "\t" + "\t" + "\t"
       + hipLeft
       + "\t" + "\t" + "\t" + kneeRight + "\t" + "\t" + "\t" + kneeLeft + "\t" +
       "\t" + "\t" + footRight + "\t" + "\t" + "\t" + footLeft);
       file.WriteLine("\t" + "\t" + "X1" + "\t" + "Y1" + "\t" + "Z1" + "\t" + "X2" +
       "\t" + "Y2" + "\t" + "Z2" +
       (…)
       file.WriteLine("");
    }
```

Quellcodeausschnitt 4.9 Header erstellen

In Quellcode sichtbar wird zu Beginn die Textfile Datei erstellt damit im Anschluss die ersten sechs Zeilen mit dem Header hinzugefügt werden können. Durch die Methode `file.WriteLine` wird eine neue Zeile mit Inhalt erstellt [msdn16i]. Mit `\t` wird ein Abstand in Länge eines Tabs realisiert.

Die Speicherung der Körperdaten funktioniert auf demselben System. Der Unterschied besteht hierbei in der stetigen Wiederholung der Methode. Dies soll durch den folgenden Quellcodeausschnitt 4.10 verdeutlicht werden.

```
using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@"Output/kinectSkeleton.txt", true))
    {
       file.WriteLine(counter + "\t" + elapsedTimeOutput + "\t" + neckX + "\t" +
       neckY + "\t" + neckZ + "\t" + shoulderCX + "\t" + shoulderCY + "\t" +
       shoulderCZ + "\t" + shoulderRX + "\t" + shoulderRY + "\t" + shoulderRZ + "\t"
       +shoulderLX + "\t" + shoulderLY + "\t" +
       (…);
    }
```

Quellcodeausschnitt 4.10 Abspeicherung der Körperdaten

Bei jedem Durchlauf der Aufzeichnung und Messung der Körperkoordinaten wird mit Hilfe der schon bekannten Methode `file.Writeline` eine neue Zeile in die erstellte *Textfile* `kinectSkeleton.txt` geschrieben. Die Variable Counter steht hierbei für die Framenummer, welches gleichzusetzen mit der Anzahl der durchgeführten Durchläufe des Programmes ist. Die *Variable* `elapsedTimeOutput` beinhaltet den derzeitigen Zeitpunkt in der Aufzeichnung der Körperdaten. Anschließend folgen die aufgenommenen aktuellen Körperdaten. Diese werden für den TRC standardisiert in Reihenfolge der Marker abgespeichert. Hierfür muss immer der X-, Y- und Z-Wert eines Markers hintereinander abgespeichert werden. Auch bei der Speicherung der Körperdaten muss zwischen jeder Variable ein Abstand der Länge eines Tabs existieren.

4.2.5.6 Beenden der Aufnahme

Damit die Aufzeichnung der Körperdaten beendet wird, muss die rechte Hand des Nutzers auf eine Position in der Mitte zwischen den beiden Schulter-Punkten gebracht werden. Es handelt sich hierbei um den ellipsenförmigen Stopp-Punkt aus Kapitel 4.2.4.3.

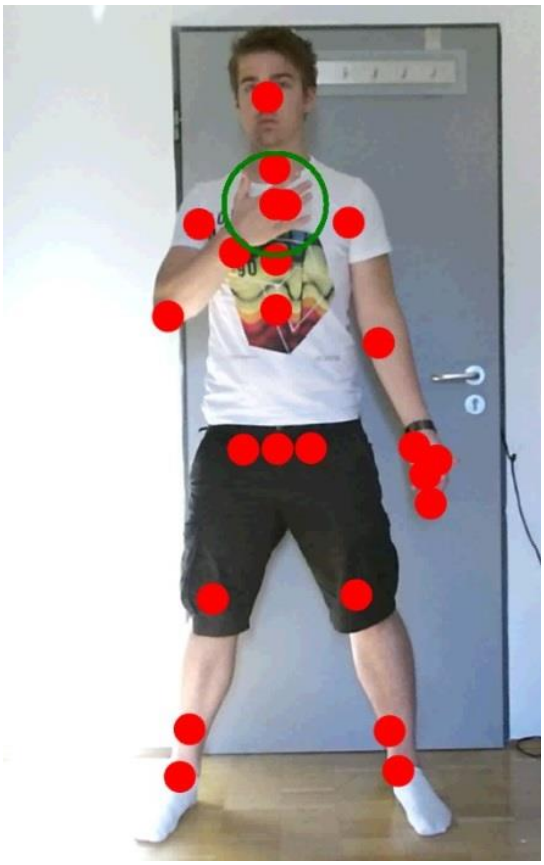


Abbildung 4.15 Berühren des Stopp-Punktes

Wie in Abbildung 4.15 erkennbar, muss die Kinect Joint Position der rechten Hand auf den Mittelpunkt des grünen Kreises gebracht werden. Dies funktioniert über das gleiche Abfrage Prinzip, welches auch zum Begin der Aufnahme verwendet wird. Einzig die Auswahl der abgefragten Joint Positionen unterscheidet sich.

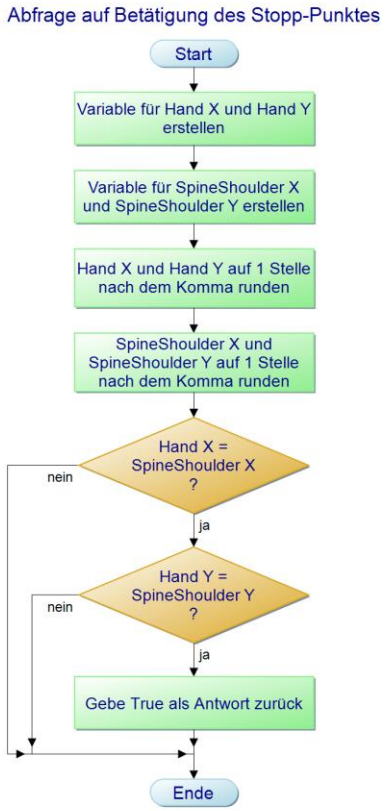


Abbildung 4.16 Abfrage auf Betätigung des Stopp-Punktes

In Abbildung 4.16 ist ein Programmablaufplan dieser Abfrage auf Betätigung des Stopp-Punktes sichtbar. Zu Beginn müssen die Variablen für die X- und Y-Komponente der Hand erstellt werden. Anschließend werden die Variablen *SpineShoulder* X und *SpineShoulder* Y generiert. Alle Variablen werden in den nächsten Arbeitsschritten auf die erste Stelle nach dem Komma gerundet. Abschließend wird eine zweifache boolesche Abfrage durchgeführt. Diese überprüft, ob die Koordinaten für Hand X und *SpineShoulder* X sowie Hand Y und *SpineShoulder* Y gleich sind. Ist dies der Fall, wird die Aufzeichnung der Körperdaten gestoppt und die Dateinamenserweiterung der erstellten *Textfile* mit den aufgezeichneten Daten aus dem Textfile Format in das TRC Datenformat gewechselt.

```
public void changeExtensions()
	{
		string txtFileName = @"Output/kinectSkeleton.txt";
		File.Move(txtFileName, Path.ChangeExtension(txtFileName, ".trc"));
	}
```

Quellcodeausschnitt 4.11 ChangeExtensions

In Quellcode 4.11 sichtbar wird hierfür die Methode `changeExtrensions` benötigt. Diese lädt in einem ersten Schritt die Daten für die erstellte `kinectSekelton.txt` File in den *string* `txtFileName` [msdn16j]. Anschließend wird mit Hilfe der Klasse `File` die Datennamenserweiterung geändert und aus *kinectSkeleton.txt* wird *kinectSkeleton.trc*.

4.3 OpenSim

In den folgenden Unterkapiteln soll auf den Teil des Projektes eingegangen werden, der unter OpenSim durchgeführt wird. Hierfür wird zu Beginn das ausgewählte Simulationsmodel erläutert. Anschließend werden die wichtigsten Eigenschaften eines Modells in OpenSim betrachtet. Abschließend soll das Einbinden der aufgezeichneten Daten in OpenSim beschrieben werden.

4.3.1 Modell

Um die aufgezeichneten Koordinaten der Joint Punkte in OpenSim zu verwenden, wird ein Modell des menschlichen Körpers benötigt. Diese Modelle bestehen aus einer Datei im *Osim* Datenformat und sind in der Programmiersprache XML gehalten [sic15b]. In diesem Projekt wird mit einem kompletten Körpermodell mit einfacher Armstruktur ohne Muskelstränge gearbeitet. Erhältlich ist dieses Modell über die offizielle Internetdomain von OpenSim [sim12]. Nach dem Herunterladen kann dieses Modell über die Schaltfläche File und der Auswahl Load Model in OpenSim eingefügt werden.

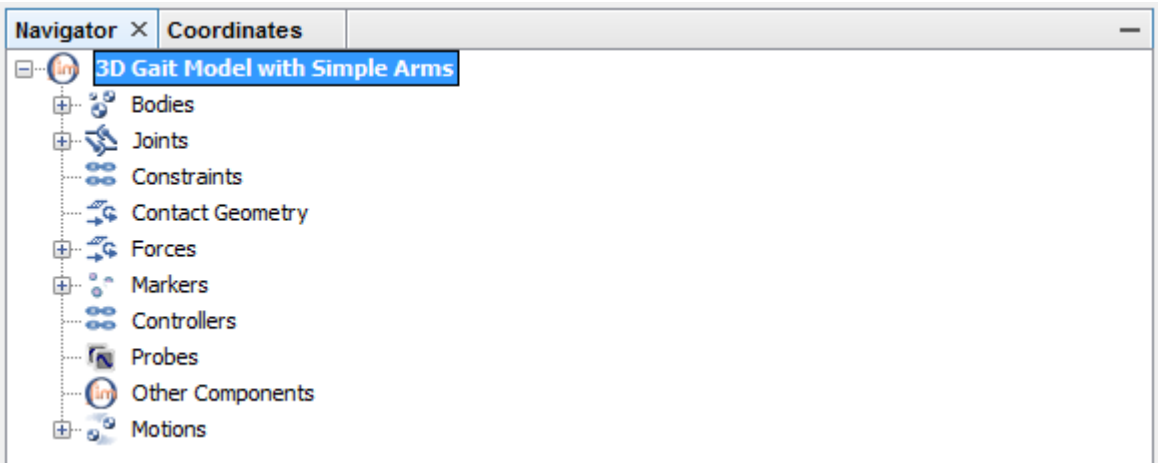
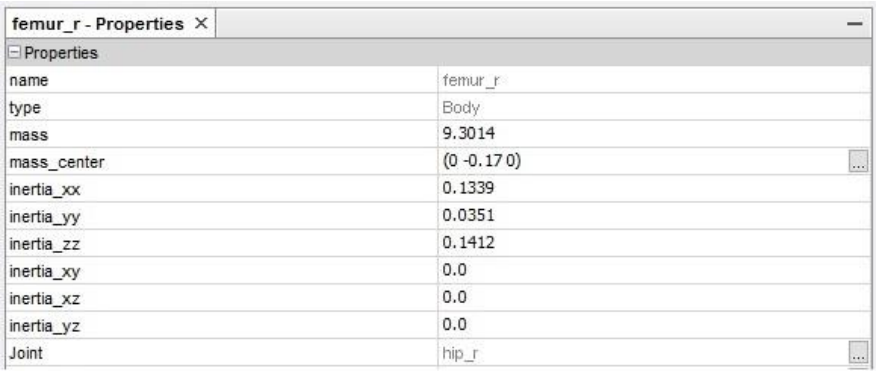


Abbildung 4.17 Model Übersicht

Anschließend öffnet sich im Navigator Fenster, wie in Abbildung 4.17 sichtbar, das Optionsmenü für das ausgewählte Modell. In den folgenden Unterkapiteln soll auf die wichtigsten Optionen eingegangen werden.

4.3.1.1 Bodies

Bei den sogenannten Bodies handelt es sich um die Bausteine aus denen sich das Modell zusammensetzt. Ein Body besteht aus einem Joint und einem oder mehreren Knochen.



femur_r - Properties	
Properties	
name	femur_r
type	Body
mass	9.3014
mass_center	(0 -0.17 0)
inertia_xx	0.1339
inertia_yy	0.0351
inertia_zz	0.1412
inertia_xy	0.0
inertia_xz	0.0
inertia_yz	0.0
Joint	hip_r

Abbildung 4.18 Model Bodies

Abbildung 4.18 soll am Beispiel von dem *Body femur_r* (Rechter Oberschenkelknochen) einen Einblick in die Eigenschaften eines Bodys geben. Nach dem festgelegten Namen und Datentyp wird die Masse des Bodys festgelegt. Anschließend kann das genaue Massezentrum ausgewählt werden. Des Weiteren besteht die Möglichkeit, die Inertia (Trägheit) zu verändern. Abschließend wird der zu dem Body zugehörige Joint aufgelistet.

4.3.1.2 Joints

Joints stellen die Verbindung zwischen den einzelnen Bodies dar. Bei dieser Verbindung spielen die in den Eigenschaften der Bodies definierten Trägheitswerte und Masseschwerpunkte eine entscheidende Rolle.

knee_r - Properties	
Properties	
name	knee_r
type	CustomJoint
location_in_parent	0 0 0
orientation_in_parent	0 0 0
location	0 0 0
orientation	0 0 0
CoordinateSet	
reverse	☐
SpatialTransform	
parent_body	femur_r

Abbildung 4.19 Model Joints

Abbildung 4.19 behandelt die Eigenschaften eines Joints. In diesem Fall handelt es sich um das Gelenk *knee_r* (Rechtes Knie). Der Joint Typ ist in der Datei festgelegt. Der Punkt *Location_in_parent* stellt die Koordinaten des Joints in einem Body dar. In diesem Fall befindet er sich im Ursprung des zugehörigen Bodys. Soll das Gelenk in eine gewünschte Richtung zeigen, kann dies über den Punkt

orientation_in_parent definiert werden. Die folgenden zwei Reiter, *location* und *orientation*, dienen als Koordinaten, wenn die Joints frei von Bodies funktionieren sollen. Bei einem menschlichen Körpermodell ist dies nie der Fall, es kann aber bei Modellationen von anderen Objekten notwendig sein. In dem folgenden Punkt *CoordinateSet* können alle in der Datei des Modells festgelegten Größen und Bewegungsradien des Joints eingesehen werden. Mit der Option *Reverse* besteht die Möglichkeit, den mit dem Gelenk verbundenen Body zu verändern. [sic15b].

4.3.1.3 Constraints

Ein *Constraint* (engl. für Einschränkungen) wird benötigt, wenn ein *Body* in seiner Bewegung eingeschränkt ist. In OpenSim gibt es eine Auswahl von drei Einschränkungen. Diese umfassen den punktuellen, den fixierte unnd den koordinatenabhängige *Constraint*. In folgender Tabelle 4.4 soll auf ihre jeweilige Funktion eingegangen werden [sic15b].

PointConstraint	Fixiert zwei Bodies an einem Punkt miteinander
WeldConstraint	Fixiert zwei Bodies an einer festgelegten Seite zusammen
CordinateCouplerConstraint	Fixiert einen Bodie in Abhängigkeit zu zwei weiteren Bodies

Tabelle 4.4 Constraint Typen

Wenn die Patella (Kniescheibe) eines menschlichen Körpers dargestellt werden soll, müssen beispielsweise *Contraints* benutzt werden. Da die Patella in Abhängigkeit von dem Knie funktioniert, müsste dies mit Hilfe eines *Constraints* realisiert werden. In dem Modell, welches für dieses Projekt ausgewählt wurde, existieren keine *Constraints*, da es sich um eine simple Darstellung des menschlichen Körpers handelt.

4.3.1.4 Forces

Bei Forces (engl. für Kräfte) in OpenSim handelt es sich um die Kräfte, die einen Einfluss auf das Skelett des Modells nehmen. Hierbei wird zwischen aktiven und passiven Kräften unterschieden. Passive Kräfte werden beispielsweise als Dämpfter zwischen einzelnen Bodies verwendet. Eine Patella (Kniescheibe) kann als eine passive Kraft agieren. In diesem Projekt finden die passiven Forces jedoch keine Verwendung. Bei den aktiven Kräften handelt es sich um die Muskeln des ausgewählten Modells.

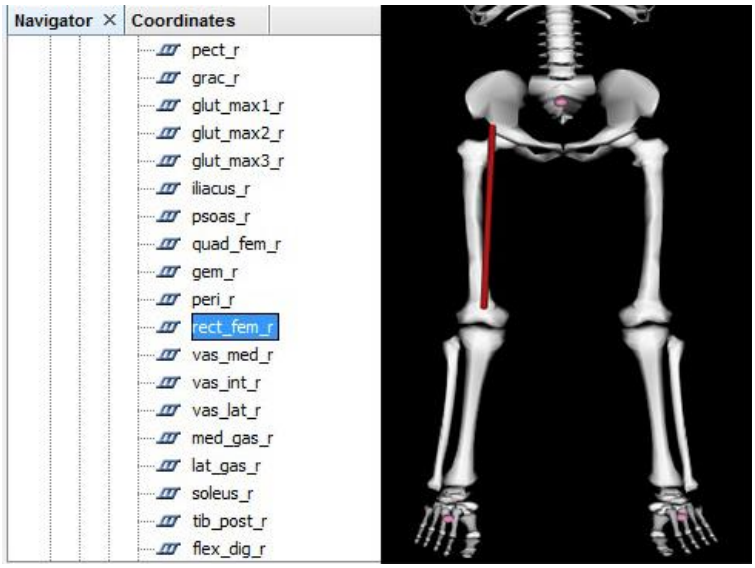


Abbildung 4.20 Model Forces

Exemplarisch wurde in Abbildung 4.20 der *Musculus rectus femur* (lat. für gerader Muskel des Oberschenkels) ausgewählt. Dieser ist auf der rechten Seite der Abbildung in Form eines roten Stranges sichtbar.

rect_fem_r - Properties

Properties	
name	rect_fem_r
type	Thelen2003Muscle
isDisabled	☐
min_control	0.0
max_control	1.0
GeometryPath	
max_isometric_force	1169.0
optimal_fiber_length	0.114
tendon_slack_length	0.31
pennation_angle_at_optimal	0.08726646
max_contraction_velocity	10.0
ignore_tendon_compliance	☐
ignore_activation_dynamics	☐
default_activation	0.05
default_fiber_length	0.1
activation_time_constant	0.01
deactivation_time_constant	0.04
FmaxTendonStrain	0.033
FmaxMuscleStrain	0.6
KshapeActive	0.5
KshapePassive	4.0
Af	0.3
Flen	1.8
fv_linear_extrap_threshold	0.95

Abbildung 4.21 Model Forces Properties

In Abbildung 4.21 werden die Eigenschaften des *Musculus rectus femur* dargestellt. Nach dem Namen des Muskels wird der ausgewählte Muskeltyp aufgezeigt. Hierbei handelt es sich um den Thelen2003Muscle Typ. Dieser Muskeltyp basiert auf der Publikation von Thelen aus dem Jahre 2003. Sie beinhaltet die mathematische Herleitung für die Kräfte, die in der Muskulatur wirken [sic15c].

Im nächsten Punkt besteht die Möglichkeit den Muskel manuell auszuschalten. Hierdurch hat er auch keinen Einfluss mehr auf die Bewegung des Modells. Anschließend werden alle nötigen Koordinaten und Werte für die Funktion des Muskels aufgelistet. Neben den Start- und End-Koordinaten, werden auch die Längen der Sehnen und Fasern dargestellt.

4.3.1.5 Modell-Marker

Mit Modell-Marker sind die Punkte am Simulationsmodel gemeint, in die experimentelle Daten eingebunden werden.

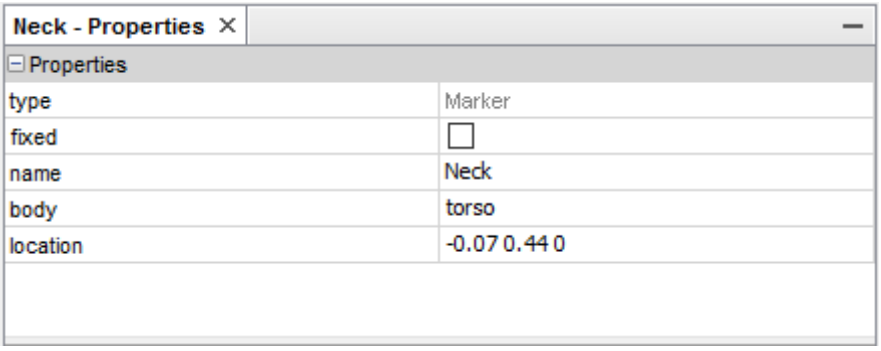


Abbildung 4.22 Modell Marker

In Abbildung 4.22 wurde als Beispiel der Marker Neck gewählt. In diesen Einstellungen befinden sich vier veränderbare Attribute. Mit der Option *fixed* kann der Marker fixiert werden, damit er auch bei einer Bewegungsanimation an der gleichen Position bleibt. In der nächsten Zeile wird der Name des Punktes definiert. Dies ist wichtig, da dieser Name mit dem Namen des in Kapitel 4.2.5.5 aufgezeichneten Kinect Punkten übereinstimmen muss. Anschließend muss die richtige Körperregion gefunden werden. Dies passiert über den ausgewählten Body [sic10]. Die Auswahl der möglichen Körperregion können in Tabelle 4.5 gefunden werden.

Torso	Torso und Kopf
Humerus Right/Left	Oberarmknochen
Ulna Right/Left	Elle
Radius Right/Left	Speiche
Hand Right/Left	Hand
Pelvis	Becken
Femur Right/Left	Oberschenkelknochen
Tibia Right/Left	Schienbeinknochen
Talus Right/Left	Sprungbein
Calcaneus Right/Left	Fersenbein
Toes Right/Left	Zehen

Tabelle 4.5 Vorhandene Bodies

Abschließend können die Koordinaten des Markers eingesetzt werden. Diese können durch Positionierung des Punktes mit Hilfe der Maus oder durch manuelle Eingabe der Werte erstellt werden. In diesem Projekt wurde die Positionierung der Marker Punkte denen der Kinect Joint Punkte nachempfunden.

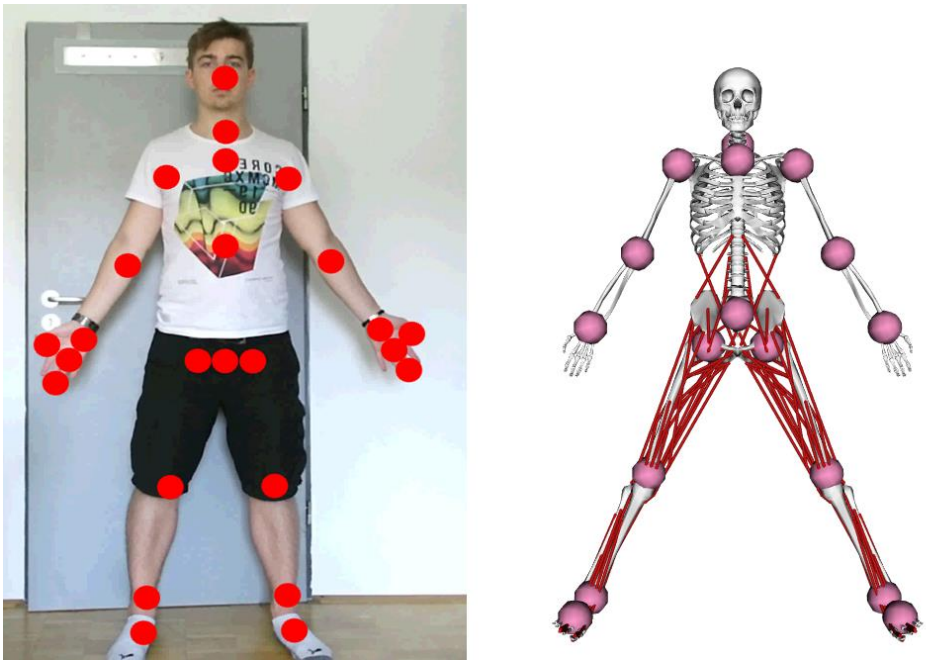


Abbildung 4.23 Marker Vergleich mit Kinect Jointpunkten

In Abbildung 4.23 befindet sich ein direkter Vergleich zwischen Kinect Joints (links) und den in OpenSim gesetzten Marker Punkten (rechts).

4.3.2 Inverse Kinematics Tool

Bei dem Inverse Kinematics Tool von OpenSim handelt es sich um eine Möglichkeit die erstellte *kinectSkeleton.trc* in OpenSim zu öffnen. Das IK Tool verarbeitet jeden aufgezeichneten Frame (Bild) einzeln und weist die aufgezeichneten Koordinaten bestmöglich den Markern am ausgewählten Model zu. Diese Zuweisung basiert auf der mathematischen Methode der kleinsten Quadrate. In den folgenden Unterkapiteln soll auf diese Grundlagen eingegangen werden und die Fehlermeldungen, die bei der Durchführung des Inverse Kinematics Tool auftreten können erklärt werden [sic16].

4.3.2.1 Methode der kleinsten Quadrate

Bei der Methode der kleinsten Quadrate handelt es sich um das mathematische Standartverfahren, um in einer Datenpunktwolke eine Kurve zu finden, die möglichst nahe an den Datenpunkten verläuft.

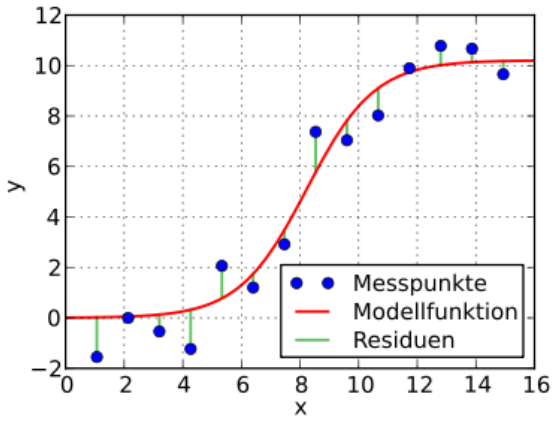


Abbildung 4.24 Methode der kleinsten Quadrate Ausgleichskurve [wik16b]

In Abbildung 4.24 ist eine Beispielkurve für eine Datenwolke sichtbar. Residuen stehen hierbei für die Entfernung zwischen einem Messpunkt und der Ausgleichsgrade.

Um die Werte für diese Kurve zu kalkulieren, verwendet OpenSim eine auf die Anforderung abgestimmte Formel für die Methode des kleinsten Quadrates.

$$\min_{q}\left[\sum_{m=1}^{\#\text{markers}} w_m \left\| x_m^{\exp} - x_m(q) \right\|^2 + \sum_{c=1}^{\#\text{coordinates}} \omega_c \left(q_c^{\exp} - q_c\right)^2\right]$$

Formel 4.3 Methode der kleinsten Quadrate [sic16]

Diese Formel setzt sich aus zwei Teilen zusammen. w_m und ω_c stehen hierbei für die Wertigkeit des ausgewählten Marker Punktes oder der ausgewählten Winkelkoordinate. Je nach Wichtigkeit kann diese erhöht werden, wodurch ein größerer Einfluss auf das Gesamtergebnis ermöglicht werden kann. Auf der linken Seite werden die Marker Punkte berücksichtigt. x_m^{exp} steht hierbei für die experimentellen Werte der Projektapplikation und $x_m(q)$ für die Marker Punkte auf dem Modell. Auf der rechten Seite befindet sich der Teil der Formel, der für experimentelle Winkelkoordinaten zuständig ist. q_c^{exp} steht hier für die experimentellen Winkeldaten und q_c für die vorhandenen Winkeldaten am Modell [sic16].

4.3.2.2 Marker Errors

Bei einem Marker Error handelt es sich um einen Fehler der auftritt, wenn die Distanz zwischen experimentellen Marker Positionen und den von Opensim berechneten Positionen im Modell zu groß sind.

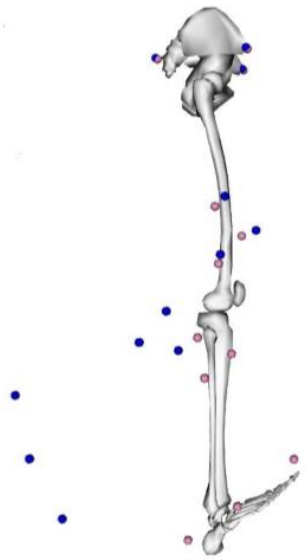


Abbildung 4.25 Marker Error [sic14]

Abbildung 4.25 soll die Entstehung eines Markers Errors bildlich darstellen. Die blauen Punkte stehen für die experimentell aufgezeichneten Werte und die roten Punkte für die festgelegten Marker Positionen am Modell [sic16]. Würden diese Punkte in die Formel für die Methode der kleinsten Quadrate aus Abbildung in Kapitel 4.3.2.1 eingesetzt, wäre eine große Differenz erkennbar. Diese Differenz würde in einem nächsten Schritt in OpenSim als Marker Error angezeigt werden.

4.3.2.3 Coordinate Errors

Ein Coordinate Error tritt beispielsweise auf, wenn der experimentell aufgezeichnete Winkel zwischen den Beinen nicht dem Winkel der Beine des OpenSim Modells entspricht [sic16]. Genau wie bei den Marker Errors basiert dieses Auftreten auf der Durchführung der Methode der kleinsten Quadrate. Da in diesem Projekt mit keinen experimentellen Winkeldaten gearbeitet wird, tritt dieser Error nicht auf.

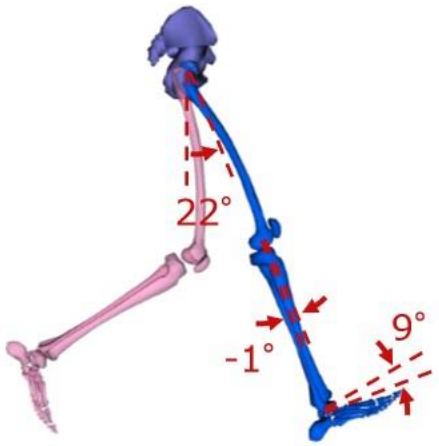


Abbildung 4.26 Coordinate Error [sic14]

Abbildung 4.26 soll schematisch die Entstehung eines Coordinate Errors erklären. Die eingezeichneten Gradzahlen zeigen die Winkel an, in denen sich die Bodies des Modells zueinander befinden. Wird nun ein experimenteller Winkel hinzugefügt, wird von OpenSim überprüft, ob die Differenz zwischen diesem und dem schon vorhandenen Winkel zu groß ist. Ist dies der Fall, wird ein Coordinate Error in der Applikation angezeigt. Berechnet wird der Coordinate Error nach dem rechten Teil der Formel für die Methode der kleinsten Quadrate aus Formel 4.3 in Kapitel 4.3.2.1.

4.3.2.4 Durchführung

Für die Durchführung des Inverse Kinematics Tool muss dieses in einem ersten Schritt gestartet werden. Es befindet sich unter dem Reiter Tools in der dritten Spalte. Nach dem Anklicken öffnet sich ein Fenster, welches in Abbildung 4.27 zu finden ist.

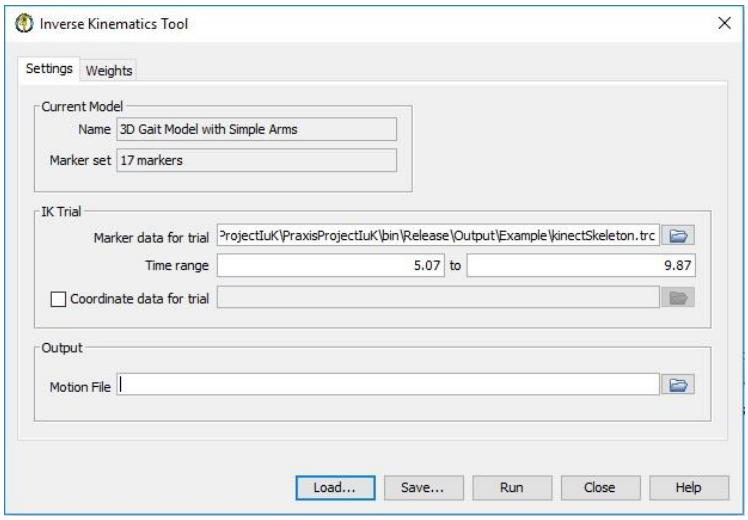


Abbildung 4.27 Inverse Kinematics Tool Settings

Begonnen werden soll mit dem Reiter Settings. Hier wird zu Beginn das ausgewählte Simulationsmodell namentlich genannt und die Anzahl der festgelegten Marker aufgezeigt. Anschließend folgt der Bereich des IK Trials. Hier besteht die Möglichkeit, die erstellte kinectskeleton.trc Datei auszuwählen. In der darauffolgenden Zeile kann der Zeitraum der aufgezeichneten Daten ausgewählt werden, der durch das Simulationsmodell nachgestellt wird. Darüber hinaus kann die *Checkbox Coordiante data for Trials* aktiviert werden. Dadurch besteht die Möglichkeit extra aufgenommene Winkelkoordinaten der Berechnung hinzuzufügen. Abschließend kann ausgewählt werden, ob die berechnete Bewegung als Motion File nach der Berechnung ausgegeben werden soll. Hierfür muss ein Name in das Feld eingegeben werden.

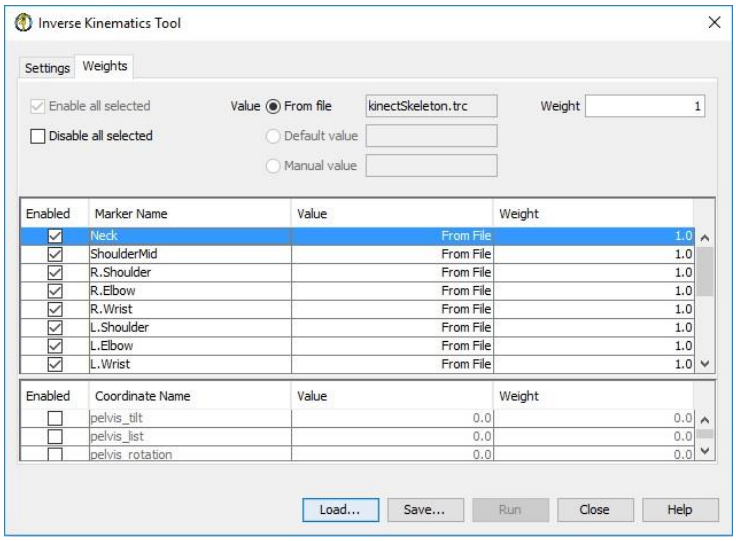


Abbildung 4.28 Inverse Kinematics Tool Weights

In Abbildung 4.28 ist *Weigths*, der zweite Reiter von dem Inverse Kinematics Tool, sichtbar. In diesem besteht die Möglichkeit, die Wertigkeit der einzelnen Marker Punkte in der Berechnung auszuwählen. Für dieses Projekt bleibt die Gewichtung von allen Markern gleich. Eine weitaus wichtigere Rolle spielt die Wertigkeit der Punkte bei einem Motion Capture System auf Leuchtpunkt Technologie. Hier gibt es eine signifikant größere Anzahl von Marker Punkten, die in ihrer Wertigkeit unterschieden werden müssen. Beispielsweise muss ein Schulter Marker eine größere Wertigkeit haben als der Marker des Schulterblattes, der keine radialen Bewegungen durchführt. [sic12].

Sind alle Wertigkeiten und Daten ausgewählt, kann der *Button Run* betätigt werden. Dieser befindet sich auf der unteren rechten Seite des Inverse Kinematics Tools Fensters. Nach Betätigung schließt sich das Fenster und in der unteren rechten Seite von OpenSim ist ein Balken mit dem Fortschritt der Berechnungen erkennbar. Entstehen während der Berechnung Marker- oder Coordinate Errors, werden diese im Messager Feld angezeigt.

Abbildung 4.29 Motion Slider

Sind die Berechnungen für den Bewegungsablauf beendet, kann dieser abgespielt werden. Über der Viewbox mit dem Model befindet sich der Motion Slider aus Abbildung 4.29. In diesem befindet sich links eine Box, in der die ausgewählte Bewegung namentlich genannt wird. Rechts daneben befindet sich die Time- und Speed-Anzeige. Hier besteht die Möglichkeit, die genaue Zeit im Ablauf der

aufgezeichneten Bewegung einzustellen und die Abspielgeschwindigkeit zu verändern. Abschließend ist rechts eine simple Oberfläche zum Abspielen und Pausieren der Bewegung vorhanden. Des Weiteren ist ein Balken oben rechts vorhanden, der ein manuelles Verschieben der Ablaufzeit ermöglicht.

5 Zusammenfassung

In diesem Kapitel soll in einem ersten Teil auf die bestehenden Probleme des Projektes eingegangen werden. Anschließend soll in einem Fazit die mögliche Weiterführung erläutert werden.

5.1 Probleme während der Entwicklung

Bei diesem Projekt kam es im Rahmen seiner Entwicklung zu Problemen. Direkt nach Beginn der Entwicklung wurde klar, dass die Kinect Kamera der XBOX ONE über eine neue Bibliothek verfügt. Dies bedeutet einen kompletten Neuaufbau der Applikation. Es konnte nicht auf das Programm des vorherigen Projektes aufgebaut werden. Dafür enstand im Rahmen des jetzigen Projektes eine simplere und übersichtlichere Verwendung der Daten von der Kinect Kamera.

Nach dem Einbinden der Datenströme von Farb- und Infrarotkamera, sollten diese ursprünglich zusammengefügt werden. Dies sollte mit Hilfe eines Skelettes als Overlay auf der Videoaufnahme realisiert werden. Die Gelenkpunkte werden wie gewünscht angezeigt. Die Knochen hingegen werden trotz intensiver Bemühung nicht auf dem Videosignal angezeigt.

Ein weiteres Problem bestand bei der Aufnahme der Bewegung. Diese muss zunächst erkannt werden, welches nur zwei Lösungswege erlaubt. Entweder wird der Datenstrom dauerhaft aufgezeichnet und nur gespeichert, wenn die gewollte Bewegung durchgeführt wurde, oder es wird ein Start- und Stopp-Punkt für die Aufnahmefunktion eingebunden. Die erste Lösungsvariante wurde mit Hilfe des KinectGestureBuilders von Microsoft durchgeführt, konnte aber kein gutes Ergebnis erzielen. Grund hierfür war zum Einem der Beta Status des GestureBuilders und zum Anderen die unverlässliche Einbindung der hiermit erstellen Gestenbibliothek. Nach weiterer Recherche wurde deutlich, dass von Seiten des Kinect Entwicklerteams auf eine fehlerhafte Einbindung der Gestenbibliothek hingewiesen wird. Auf Grund dieser Aussage wurde statt der Gestenerkennung ein Start- und Stopp-System für die Aufzeichnung der Körperdaten realisiert.

Ein weiteres großes Problem waren die unterschiedlichen Koordinatensysteme mit denen gearbeitet wurde. Die Umrechnung aus dem Kinect System in das von OpenSim benötigte Koordinatensystem stellte zunächst eine Herausforderung dar. Nach einiger Recherche konnte der richtige Weg zur Umrechnung gefunden werden. Trotz funktionierender Konvertierung der Punkte, besteht weiterhin das Problem des Winkels, in dem der Benutzer zum Boden steht. Der gewählte Lösungsweg funktioniert, doch könnten die Koordinaten genauer sein, wenn der Bodenwinkel in die Berechnung einfließen würde.

Abschließend muss auf das Problem des Koordinatenursprungs eingegangen werden. In der für dieses Projekt programmierten Applikation wird ein Ursprung mit Hilfe von Gelenkpunktkoordinaten gebildet. Bleibt der Benutzer auf der Stelle stehen und bewegt die Arme oder den Kopf ist hierbei jede Bewegung möglich. Bewegt sich die Person jedoch im Raum besteht das Problem, dass der Koordinatenursprung sich mitbewegt. Falls eine Bewegung im Raum nun aufgenommen werden soll, wird das Modell in OpenSim an der gleichen Stelle verweilen. Ein möglicher Lösungsansatz hierfür wäre eine festgelegte Höhe der Kamera, wordurch ein Koodinatenursprung in Abhängigkeit von diesem ausgemessenen Punkt aufgenommen werden könnte.

5.2 Fazit

Aus der anfangs definierten Zielsetzung ein Programm zu entwickeln, welches die Aufnahme einer Armbewegung und Weiterverarbeitung dieser in OpenSim beinhaltet, entstand eine Applikation, die weitaus mehr Funktionen besitzt. Nicht nur der Arm, 16 weitere Körperpunkte des Benutzers werden durch die Kinect aufgezeichnet. Des Weiteren entstand ein Konverter im Programm, der eine Umformung von aufgezeichneten Körperdaten in das für Bewegungsfunktionen standardisierte Datenformat TRC ermöglicht.

Insgesamt wurde ein Programm entwickelt, das als Grundlage für weitere Projekte dienen kann. Es ist leicht auf andere Anforderungsgebiete anpassbar. Beispielsweise wäre für die Zukunft eine Auswertung der aufgezeichneten Bewegungsmuster möglich. Ein anderer Arbeitsschwerpunkt wäre die Abfrage auf vorher definierte Bewegungen durch visuelle Hilfestellung. Es könnten Kreise eingezeichnet werden, denen mit der Hand gefolgt werden muss.

Literaturverzeichnis

[wik16a] Wikipedia Kinect. Version September 2016 https://en.wikipedia.org/wiki/Kinect_for_Xbox_One

[dev16] Kinect one specs. https://developer.microsoft.com/de-de/windows/kinect/hardware

[pte14] Vangos Pterneas kinect v2 overview : http://pterneas.com/2014/02/08/kinect-for-windows-version-2-overview/

[mir10] How the kinect works. https://mirror2image.wordpress.com/2010/11/30/how-kinect-works-stereo-triangulation/

[sig13] Kinect Microphone. http://users.dickinson.edu/~jmac/selected-talks/kinect.pdf

[ifi13] XBOX ONE Kinect Teardown. https://d3nevzfk7ii3be.cloudfront.net/igi/OqK2QmCEgF1StiHB.huge

[dna15] NET DNA. Infraredprojektor : http://1abxf1rh6g01lhm2riyrt55k.wpengine.netdna-cdn.com/wp-content/uploads/make-images/AJXWdfOgEnZkJYEN.jpg

[msdn15] MSDN, IR Camera with Projector : https://social.msdn.microsoft.com/Forums/getfile/171552

[ops16] OpenSim – See the work. http://opensim.stanford.edu/work/index.html

[sim15a] Simbody Multibody Physics API. https://simtk.org/projects/simbody/

[sic13a] OpenSim Capabilities. http://simtk-confluence.stanford.edu:8080/display/OpenSim/OpenSim%27s+Capabilities

[mic16] Kinect for Windows SDK 2.0. https://www.microsoft.com/en-us/download/details.aspx?id=44561

[thu16] Kinect Adapter https://www.thurrott.com/wp-content/uploads/2016/06/en-INTL-L-Microsoft-Kinect-for-Win-Plug-9J7-00001-RM3-mnco.jpg

[msdn16a] MultiSourceFrameReader Class. https://msdn.microsoft.com/en-us/library/microsoft.kinect.multisourceframereader.aspx

[msdn16b] FrameSourceTypes Enumeration. https://msdn.microsoft.com/en-us/library/microsoft.kinect.multisourceframereader.aspx

[msdn16c] BitmapSource Methode. https://msdn.microsoft.com/de-de/library/ms616045(v=vs.110).aspx

[msdn16d] JointType Enumeration. https://msdn.microsoft.com/en-us/library/microsoft.kinect.jointtype.aspx

[ggp16] Jointpunkte der Kinect : http://lh3.ggpht.com/-9AjhW77cUUE/Uy3uQvsTbgI/AAAAAAAAA88/L70ZVLewW-g/Body-Joints_thumb%25255B2%25255D.png?imgmax=800

[msdn16e] CoordinateMapper Class. https://msdn.microsoft.com/en-us/library/microsoft.kinect.coordinatemapper.aspx

[msdn16f] Kinect Coordinatesystem. https://msdn.microsoft.com/de-de/library/dn785530.aspx

[sic15] Coordinatesystem OpenSim. http://simtk-confluence.stanford.edu:8080/display/OpenSim/Coordinate+Systems

[bho16] Rotation um die Y-Achse. https://prof.beuth-hochschule.de/fileadmin/user/linnemann/PDF-Dateien/Robotertechnik/Roboter_Technik_Vorlesung_Teil_03.pdf

[sic13b] TRC Dataformat. http://simtk-confluence.stanford.edu:8080/display/OpenSim/Marker+(.trc)+Files

[par09] Parent, Rick : Computer Animation Complete : All-in-One: Learn Motion Capture, Characteristic, Point-Based, and Maya Winning Techniques. 2. Auflage, Morgan Kaufmann Verlag.

[msdn16g] Math Round Methode. https://msdn.microsoft.com/de-de/library/system.math.round(v=vs.110).aspx

[msdn16h] File WriteLine Tutorial. https://msdn.microsoft.com/de-de/library/8bh11f1k.aspx

[msdn16i] ChangeExtension Methode. https://msdn.microsoft.com/de-de/library/system.io.path.changeextension(v=vs.110).aspx

[sic15b] OpenSim Model. http://simtk-confluence.stanford.edu:8080/display/OpenSim/OpenSim+Models#OpenSimModels-Joints

[sim12] Used Model. https://simtk.org/projects/runningsim

[sic15c] Thelen2003 Muscle Type. http://simtk-confluence.stanford.edu:8080/display/OpenSim/Thelen+2003+Muscle+Model

[sic10] Model Marker. http://simtk-confluence.stanford.edu:8080/display/OpenSim/Marker+Editor

[sic16] Inverse Kinematics. http://simtk-confluence.stanford.edu:8080/display/OpenSim/How+Inverse+Kinematics+Works

[wik16b] Methode des kleinsten Quadrate. https://de.wikipedia.org/wiki/Methode_der_kleinsten_Quadrate

[sic14] Marker and Coordinate Error Image. http://simtk-confluence.stanford.edu:8080/display/OpenSim/OpenSim+Workshop+Bologna+2016?preview=/18776466/18776459/InverseKinematics.pdf

[sic12] Weigth. http://simtk-confluence.stanford.edu:8080/display/OpenSim/Simulation+with+OpenSim+-+Best+Practices

Abbildungsverzeichnis

Quellcodeverzeichnis

Tabellenverzeichnis

Anhang

MainWindow.xaml.cs

Daten auf der DVD

Der schriftlichen Dokumentation liegt eine DVD mit folgender Beschriftung bei.

IuK Praxisprojektarbeit
Kinect – OpenSim
Oliver Mayr
Matr.Nr. 7082091
10.10.2016

Auf der CD enthalten sind folgende Ordner.

- OpenSim_Model
- Visual_Studio
- Dokumentation

In dem Ordner OpenSim_Modell ist das verwendete Bewegungsmodell für die Software OpenSim zu finden.

- **ProjectIuK.osim**

Die Applikation, die das Aufnehmen der Körperdaten unter Visual Studio ermöglicht, befindet sich im Ordner Visual Studio. Zum Einsehen der Funktionsweise und Ausführen des Programmes muss die im Ordner befindliche Microsoft Visual Studio Solution gestartet werden.

- **PraxisProjektIuk.sln**

Des Weiteren befindet sich eine digitale Kopie der Dokumentation und aller diesbezüglichen Bilder und Literaturquellen in dem Ordner Dokumentation.

- **IuK_Praxisprojekt_Mayr.pdf**

Auch kann hier eine Anleitung für das Einrichten und Durchführen dieses Projektes in schriftlicher Form gefunden werden.

- **Aufbauanleitung.pdf**