
Masterarbeit

# Bewegungserfassung mittels Tiefensensoren

-

Andreas Weiß

07. Oktober 2015

Betreuer:

Prof. Dr. Raúl Rojas

Tobias Langner

Freie Universität Berlin
Fachbereich Mathematik und Informatik
Institut für Informatik

# Zusammenfassung

Motion Capturing ist aus der heutigen Unterhaltungsindustrie nicht mehr wegzudenken. In den großen Film- und Videospiel-Produktionen kommt das Verfahren immer wieder zum Einsatz, wenn etwa CG-Charaktere realitätsnah animiert werden sollen. Das Motion Capture-System erfasst hierzu den Bewegungsablauf eines echten Darstellers und überträgt diesen auf die virtuelle Figur. Mit Microsofts „Kinect“ existiert inzwischen eine kostengünstige Sensoreinheit, welche das Verfahren grundsätzlich auch einer breiten Masse zugänglich macht. Drehbewegungen oder Bewegungen am Boden (z.B. Hocke, Schneidersitz, usw.) führen einen einzelnen Tiefensensor jedoch schnell an seine Grenzen. Das Ziel der vorliegenden Arbeit besteht darin, diesen Problemen entgegenzuwirken und somit eine genauere Erkennung zu erzielen. Das Dreherkennungsproblem wird mithilfe eines zweiten Kinect-Sensors gelöst. Das Problem bodennaher Bewegungen wird hingegen als Klassifizierungsproblem behandelt. Das im Rahmen dieser Arbeit entwickelte Programm trainiert hierfür ein neuronales Netz unter Verwendung des Backpropagation-Algorithmus, wobei markante Erkennungsfehler der unterschiedlichen Posen die Klassifizierungsgrundlage bilden.

# Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit „Bewegungserfassung mittels Tiefensensoren“ selbstständig und ohne unerlaubte Hilfe verfasst habe. Desweiteren bestätige ich, dass ich ausschließlich auf die angegebenen Quellen zurückgegriffen habe und dass diese Arbeit keiner anderen Prüfungskommission vorgelegt wurde.

Berlin, 07.10.2015

_____

Andreas Weiß

# Inhaltsverzeichnis

# Abbildungsverzeichnis



e. Abb. = eigene Abbildung

# Tabellenverzeichnis

# 1. Einleitung

Motion Capturing (kurz: MoCap) bezeichnet das Verfahren, aus der Bewegungsabfolge eines Akteurs die Bewegungsinformationen zu extrahieren. Aus den gewonnenen Daten können anschließend die Positionsveränderungen bestimmter Körperteile (Hände, Arme, Kopf, Schultern, Beine, usw.), über die Zeit nachvollzogen werden. Die Einsatzmöglichkeiten sind hierbei mannigfaltig. Das Verfahren wird etwa in der Medizin zu Diagnosezwecken verwendet. *„So lassen sich zum Beispiel Gangzyklen und Gelenkbewegungen, Translationen oder Rotationen exakt untersuchen."* [14]. Weiter hat sich Motion Capturing als Standard in der heutigen Unterhaltungsindustrie etabliert. Gerade in Realfilmen werden CG-Charaktere nur selten per Hand animiert. Stattdessen schlüpfen echte Menschen in die Rollen der Figuren und animieren diese durch ihre eigenen Bewegungen. Das sogenannte Performance Capturing erweitert diesen Prozess sogar um die Erfassung komplexer Mimik, wodurch es z.B. Filmemachern ermöglicht wird, menschliche Emotionen auf das Gesicht des virtuellen Darstellers zu übertragen [4].

Für das Verfahren existieren verschiedene Implementierungen, welche sich in ihrem Detailgrad und ihrer Aufnahmetechnik unterscheiden. Härtel differenziert etwa elektromagnetische, elektromechanische und optische MoCap-Systeme [9]. Letztere sind gerade im kommerziellen Bereich weit verbreitet und zeichnen sich durch ihre Genauigkeit und die Bewegungsfreiheit des Darstellers aus [11]. Dieser trägt einen mit Markern versehenen Anzug und wird von mehreren Kameras im Aufnahmeraum gefilmt. Die Bilder werden anschließend durch eine Spezialsoftware analysiert, welche die erfassten Marker in Relation zu der bekannten Geometrie der jeweiligen Kamera setzt [9]. So werden die Ergebnisse der verschiedenen Kamerabilder zu einem Datensatz kombiniert. Laut Menache werden dabei üblicherweise nicht weniger als vier Kameras verwendet [11]. Eine bekannte Herausforderung optischer Verfahren ist das Verdeckungsproblem, bei welchem Marker nicht oder nur teilweise auf einem Kamerabild zu sehen sind, etwa, wenn diese durch den Darsteller selbst verborgen werden. Menache bezeichnet zudem die Notwendigkeit einer Nachbearbeitung sowie die kostspielige Hardware als Nachteile des Systems [11].

In meiner Bachelorarbeit habe ich mich mit den genannten Nachteilen auseinandergesetzt. Ziel war es, ein Motion Capture-Programm zu entwickeln, welches sowohl durch niedrige Anschaffungskosten als auch eine intuitive Handhabung dem Normalverbraucher zugänglich sein sollte. Die Implementierung arbeitete nach dem optischen Prinzip unter Verwendung zweier Webcams. Die erste Kamera sollte hierbei die Bewegungen in der XY-Ebene nachvollziehen, wohingegen die zweite Webcam die dazugehörigen Tiefenwerte liefern sollte. Der Algorithmus funktionierte zwar für einfache Bewegungsfolgen, komplexere Aktionen aber gingen mit einem hohen Nachbearbeitungsaufwand einher. Die vorliegende Arbeit soll nun mit der Verwendung von Tiefensensoren einen anderen Ansatz verfolgen. Es soll ein Algorithmus entwickelt werden, welcher es dem Nutzer erlaubt, mithilfe zweier Kinects einen Bewegungsablauf aufzuzeichnen und auf einen dreidimensionalen Darsteller zu übertragen. Dem Anwender soll es so ermöglicht werden, innerhalb kurzer Zeit eigene Motion Capture-Filmszenen zu erstellen. Jedoch gehen mit der Nutzung von Kinect bestimmte Einschränkungen einher, welche im nachfolgenden Kapitel näher erläutert werden. Ein wesentliches Ziel dieser Arbeit ist es, diesen entgegenzuwirken.

Im ersten Abschnitt sollen dem Leser die Grundlagen der Arbeit vermittelt werden. Es wird erklärt, wie der Kinect-Sensor funktioniert und welche Herausforderungen zu bewältigen sind. Weiter werden verwandte Arbeiten vorgestellt, die sich bereits mit der Thematik auseinandergesetzt haben.

Danach wird das im Rahmen dieser Arbeit entwickelte Programm behandelt. Hierbei werden die einzelnen Verarbeitungsschritte des Algorithmus näher erklärt (Ausgangsdaten, Kalibrierung; Extraktion der Bewegungsdatensätze; Problemlösungsstrategien, Übertragung auf das 3D-Modell, usw.).

Ein abschließendes Experiment soll zeigen, inwiefern der beschriebene Algorithmus eine Verbesserung für die besagten Einschränkungen darstellt. Hierzu werden die Komponenten des Programms sowohl einzeln als auch im Zusammenspiel in verschiedenen Versuchsreihen getestet. Eine Diskussion der Ergebnisse soll Aufschluss darüber geben, wo eventuelle Schwächen des Algorithmus liegen und wie diese zu erklären sind.

# 2. Grundlagen

Tiefensensorbasierte Motion Capture-Systeme funktionieren nach einem gänzlich anderen Prinzip als jene, welche auf dem optischen Verfahren beruhen. Optische Systeme extrahieren die Bewegungsinformationen eines Darstellers, indem sie die zweidimensionalen Marker-Positionen verschiedener Kamerabilder zu einem dreidimensionalen Datensatz kombinieren. Kinect arbeitet hingegen mit einem Tiefenbild des Raumes. Die Pose wird direkt aus einem dreidimensionalen Abbild des Akteurs hergeleitet. Als Einführung in die Thematik sollen im Folgenden der Kinect-Sensor und seine Funktionsweise als Grundlage für den später entwickelten Algorithmus vorgestellt werden.

## 2.1 Kinect

Die vorliegende Arbeit bezieht sich auf Kinect v1 unter Verwendung des Kinect SDK v1.7. Verweise auf Beobachtungen bei der Nutzung des SDKs werden im Folgenden auch mit [KSDK] gekennzeichnet. Bei Microsofts „Kinect“ handelt es sich um eine Sensoreinheit, welche im Wesentlichen aus einem Tiefensensor, einer Farbbildkamera sowie vier Mikrofonen besteht [17]. Mittels dieser Technologien stellt das System eine Reihe von Funktionalitäten zur Verfügung. So kann Kinect etwa 3D-Scans der näheren Umgebung erstellen, einfache Sprachkommandos verarbeiten, sowie komplexe menschliche Ganzkörper-Bewegungen in Echtzeit erfassen [KSDK]. Darüber hinaus ermöglicht der Sensor sogar die Erfassung von Mimik. Dabei beschränkt sich die Funktion nicht nur auf das Finden von Gesichtern im Bild, sondern erkennt beispielsweise Mund- und Augenbrauenbewegungen der Nutzer [KSDK]. Folglich kann das System als Grundlage für Performance Capturing-Anwendungen verwendet werden. Aufgrund der Relevanz für diese Arbeit soll im Folgenden jedoch verstärkt auf die Bewegungserfassung als eine der Kernfunktionen eingegangen werden.

### 2.1.1 Entwicklung

Die Sensoreinheit wurde in Zusammenarbeit mit der Firma PrimeSense entwickelt. Das Unternehmen stellte für „Project Natal“, so der damalige Codename von Kinect, *„seine hochmoderne 3D-Sensor Technologie“* [3] zur Verfügung [3]. Microsoft veröffentlichte Kinect am 4. November 2010 in Nordamerika als Steuerinterface für seine Spielekonsole Xbox 360 [1]. Das Gerät ermöglicht dem Spieler, die Konsole mittels Gesten- und Sprachsteuerung zu bedienen sowie Charaktere in Spielen mittels ech-

ter Bewegungen zu kontrollieren. Im Gegensatz zu professionellen Motion Capture-Systemen sind hierfür weder Marker noch Sensoren am Körper des Darstellers/Spielers nötig. Dieses Anwendungsgebiet stellt natürlich hohe Ansprüche an die Hardware bezüglich Robustheit, Echtzeitfähigkeit, Nutzerfreundlichkeit und Kosten. So muss der Sensor in einer konstanten Geschwindigkeit die *„volle Bandbreite an menschlichen Körperformen und Größen“* [15] in der Bewegung erfassen können. Das Ergebnis sollte dabei invariant gegenüber Umgebungseinflüssen, wie etwa Beleuchtung, Möbel, Einrichtungsgegenständen oder Haustieren sein [17]. Nicht zuletzt muss all dies auf einer Hardware realisiert werden, die für den Normalverbraucher erschwinglich ist. Nach Shotton et al. sei bis zur Einführung von Kinect jedoch keines der existierenden Systeme in der Lage gewesen, diese Ansprüche zu erfüllen [15]. Doch auch in anderen Bereichen wurde man auf die Sensoreinheit aufmerksam. *„Die Bedeutung von Kinect entwickelte sich weit über die Spieleindustrie hinaus.“* [17]. Laut Craig Eisler (General Manager, Kinect for Windows) habe man in nahezu allen erdenklichen Bereichen Forschungsarbeit mit Kinect beobachten können, etwa bei Künstlern oder in der Physiotherapie. Die *„erstaunliche und kreative Art und Weise“* [8], wie Kinect außerhalb der Spielebranche eingesetzt wird, bezeichnet Microsoft als den „Kinect Effect“ [8]. Zeng führt diese Entwicklung auf die hohe Verfügbarkeit sowie die geringen Anschaffungskosten des Sensors zurück [17].

### 2.1.2 Funktionsweise

Doch wie funktionieren das System und insbesondere dessen Bewegungserfassung? Vereinfacht zeichnet Kinect ein dreidimensionales Bild des Akteurs auf und klassifiziert dessen Pose, um die Lage aller Körperteile im Raum zu ermitteln. Realisiert wird dies durch einen Tiefensensor, bestehend aus einem Infrarot (im Folgenden IR)-Projektor sowie einer IR-Kamera. Bei der IR-Kamera handelt es sich um einen monochromen CMOS-Sensor [17]. Laut Shotton et al. haben Tiefenkameras einige Vorteile gegenüber traditionellen Intensitätssensoren, da sie auch bei schwacher Beleuchtung funktionieren, farb- und texturinvariant sind sowie das Problem mehrdeutiger Silhouetten lösen [15]. Laut Zeng basiert die Tiefenerkennung auf dem Prinzip des strukturierten Lichtes [17]. Der IR-Projektor projiziert dabei ein Punktmuster in den Raum, welches von der IR-Kamera aufgezeichnet wird. Sowohl das Punktmuster als auch die relative Geometrie zwischen dem Projektor und der Kamera sind bekannt [17]. Können nun die aufgenommenen Punkte erfolgreich denen des

Projektionsmusters zugeordnet werden, können mittels Triangulation die dazugehörigen Tiefenwerte berechnet werden [17]. So entsteht ein Tiefenbild des Raumes, wie in Abbildung 1 veranschaulicht, mit einer Auflösung von 640x480 Pixel und 30 Bildern pro Sekunde [KSDK].

**Abb. 1 Tiefenbild eines Kinect-Sensors (e. Abb.)**

Daraus muss im Folgeschritt die Körperhaltung des Darstellers extrahiert werden. Jede Pose wird durch ein Skelettmodell repräsentiert, bestehend aus den 3D-Koordinaten ausgewählter Körperteile, wie etwa Kopf, Schultern, Ellbogen, Knie, usw. *„Das Ziel ist, all diese 3D-Parameter in Echtzeit zu bestimmen, um eine flüssige Interaktion mit den begrenzten Rechenmitteln der Xbox 360 zu erreichen, ohne dabei die Spiel-Performance einzuschränken.“* [17]. Realisiert wird dies durch einen Random Forest-Klassifikator. Dieser wird mit einer großen Motion Capture-Datenbank trainiert, bestehend aus nahezu 500.000 Bildern [15]. In diesen werden die verschiedensten Aktionen ausgeführt, etwa *„Fahren, Tanzen, Treten, Rennen, Navigieren im Menü, etc.“* [15]. Basierend auf dieser Datenbank ordnet der Klassifikator jeden Pixel im Eingangsbild einem Körperteil zu [15]. Die daraus resultierenden Körperteil-Pixel werden auf einen gemeinsamen Mittelpunkt reduziert [17]. So liefert der Klassifikator gewichtete Vorschläge zu den 3D-Koordinaten aller Körperteile im Raum [15] und berechnet so das zur Pose des Akteurs passende Skelettmodell. Dieser Extraktionsprozess wird im Folgenden auch als Skeletal Tracking [2] bezeichnet. Laut Shotton et al. erreichen einige Systeme eine höhere Geschwindigkeit, indem einmal gefundene

Segmente von Frame zu Frame verfolgt werden. Diese seien jedoch nicht stabil, da ihre Leistung bei der Re-Initialisierung schnell nachlässt [15], etwa, wenn kurzzeitig verdeckte Körperteile wiedergefunden werden müssen. Kinect benötigt dagegen keine zeitlichen Informationen [15]. Shotton beschreibt die Veränderungen zwischen zwei Bildern als häufig so klein, dass sie unbedeutend sind [15]. Die Pixelklassifizierung wird folglich nicht durch frühere Bilder beeinflusst [15].

### 2.1.3 Herausforderungen

Das System erkennt zahlreiche Bewegungen, ist invariant gegenüber wechselnden Umgebungseinflüssen und ist weit verbreitet. Zur Erkennung sind weder Marker vonnöten, noch muss eine manuelle Kalibrierung des Sensors vorgenommen werden [KSDK]. Kinect stellt somit für Entwickler ein kostengünstiges Werkzeug für Motion Capture-Anwendungen dar. Dennoch bleiben einige Herausforderungen bestehen [12]. Newcombe et al. verweist hierbei insbesondere auf *„zahlreiche Löcher“* [12] im Bild, in denen keine Tiefenmessung möglich ist. Als mögliche Ursachen nennt er z.B. Materialien oder Strukturen, welche kein IR-Licht reflektieren [12]. Ebenso kann die Tiefe von verborgenen Elementen nicht bestimmt werden, wodurch auch verdeckte Bewegungen nicht nachvollzogen werden können [KSDK]. Dies deckt sich mit den Aussagen von Zeng, wonach der IR-Projektor einen Schatten wirft (siehe: Abbildung 2), in welchem die Tiefenwerte nicht exakt berechnet werden können [17]. Nach Newcombe führe ebenso Bewegungsunschärfe zu fehlenden Daten [12].

**Abb. 2 Tiefenbild mit verdeckten Elementen (e. Abb.)**

Eine weitere Herausforderung ergibt sich aus der Bandbreite menschlicher Bewegungen, für welche der Sensor nicht ausgelegt ist. Der Klassifikator wurde mit einer Reihe verschiedener Bewegungsfolgen trainiert. Laut Shotton et al. war es das Ziel, *„die Vielfalt an Posen abzudecken, welche Menschen in einem Unterhaltungs-Szenario einnehmen."* [15]. Eine 360°-Erkennung sei dabei nicht vorgesehen [2], auch wenn damit experimentiert wurde. Man habe demnach mit 900.000 Bildern trainiert und eine mittlere Erkennungsgenauigkeit von 0.655 pro Gelenk erreicht [15]. Die Bewegungserfassung wurde jedoch für Nutzer konzipiert, welche dem Sensor zugewandt stehen [2]. So werden abgewandte Personen seitenverkehrt interpretiert – die Labels der linken und rechten Seite werden vertauscht [KSDK]. Eine weitere Einschränkung zeigt sich in der Erkennung bodennaher Bewegungen (im Folgenden auch Bodenbewegungen genannt). Dies spiegelt sich in einem kleinen Versuch wider. Ein Akteur führt hierzu unterschiedliche Bodenbewegungen vor einem Kinect-Sensor aus, welcher in zwei Metern Entfernung in einer Höhe von ca. 50 cm aufgestellt wird. Jede Bewegung (im Detail: Schneidersitz, Knien und Sitzen mit ausgestreckten Beinen) wird 20 Mal ausgeführt. Ein optischer Vergleich der ausgeführten Posen mit den erfassten Koordinaten zeigt deutliche Erkennungsschwierigkeiten. So konnte keine der Posen zufriedenstellend erfasst werden. In allen Fällen wich mindestens eine der Bein-Koordinaten gravierend von ihrer tatsächlichen Position ab und/oder sprang sichtbar umher (> 10 cm). Besagte Schwierigkeiten traten auch zwischen den einzelnen Posen auf. Diese Beobachtungen decken sich mit den Aussagen Sirignanos (MSFT), der zufolge die Trainingsdaten auf stehenden Nutzern basieren [16]. Schwierig sei für Kinect hingegen die Trennung von Fläche und Mensch, z.B. wenn der Akteur nahe einer Wand steht oder auf dem Boden liegt [16].

In der vorliegenden Arbeit sollen Lösungsansätze für die genannten Herausforderungen erarbeitet werden. Einige Erkennungsfehler gehen mit dem eingeschränkten Sichtbarkeitsbereich des Kinect-Sensors einher, etwa, wenn eine Hand hinter dem Rücken nicht erfasst werden kann. Hier soll durch den Einsatz einer weiteren Kinect eine Verbesserung erzielt werden. Die zweite Sensoreinheit soll ebenso die Grundlage für eine 360°-Erkennung darstellen. Die Fehlinterpretation bodennaher Bewegungen wird als eigenständige Herausforderung behandelt. Im Versuch traten zwar gravierende, teils jedoch sehr markante Erkennungsfehler auf. So ergab beispielsweise ein Schneidersitz andere Bein-Koordinaten als ein Knien. Ein Lernalgorithmus soll

mit den gegebenen Daten dahingehend trainiert werden, die Fehlinterpretationen, insbesondere die der Beine, neu zu interpretieren.

## 2.2 Verwandte Arbeiten

Eisler geht bereits auf die enorme Bedeutung von Kinect außerhalb der Spielebranche ein und betont hierbei die kreative und vielfältige Nutzung des Sensors, welche in nahezu allen erdenklichen Bereichen zu beobachten sei [8]. Laut Zeng habe es schon einen Monat nach der Veröffentlichung Beschreibungen zu annähernd 90 Kinect-Projekten gegeben [17]. So ist es kaum verwunderlich, dass auch andere das Potential des Sensors für Motion Capture-Anwendungen erkannt und sich mit den besagten Einschränkungen (siehe Abschnitt 2.1.3) auseinandergesetzt haben. Exemplarisch werden im Folgenden zwei dieser Projekte näher erläutert. Dabei soll herausgestellt werden, inwiefern diese eine Lösung für die genannten Probleme darstellen und welche Vor- und Nachteile damit einhergehen.

Dem Problem des eingeschränkten Sensor-Sichtfeldes wirkt Asteriadis et al. mit seinem Algorithmus entgegen [7]. Seine Arbeit befasst sich mit der Fusionierung der Skelettmodelle von theoretisch beliebig vielen Kinects. Durch die zusätzlichen Perspektiven sollen eine Verbesserung der Sensorleistung erzielt und Verdeckungen, etwa durch einen Heimtrainer, aufgelöst werden [7]. Der Algorithmus verfolgt einen wahrscheinlichkeitsbasierten Ansatz. Zunächst werden die Sensoreinheiten über den Torso des Darstellers kalibriert. Hierbei werden die unterschiedlichen Koordinatensysteme zueinander in Bezug gesetzt [7]. Für jede neu erfasste Gelenkkoordinate wird eine Reihe von Kandidatenpunkten in der unmittelbaren Nachbarschaft generiert. Jedem Kandidaten wird dann ein Vertrauenswert zugeordnet [7]. Dieser richtet sich nach verschiedenen Faktoren: Wie stark ist die räumliche Nähe zur erfassten Koordinate? Ist der Körperteil verdeckt? Spiegelt der Kandidat in Relation zum Elternelement einen natürlichen Gelenkwinkel wider (Freiheitsgrade)? usw. Zudem wird die zeitliche Komponente berücksichtigt. So steigt die Wahrscheinlichkeit eines Kandidaten etwa, wenn sich dieser in eine fließende Bewegung einfügt [7]. Die finale Koordinate berechnet sich dann über jene Kandidatenpunkte, für welche das Vertrauen aller Kinects maximal wird [7]. Asteriadis et al. stellt somit einen funktionierenden Ansatz zur Auflösung von Verdeckungen vor, welcher sich zwar nicht im Konkre-

ten mit den zuvor genannten Herausforderungen befasst, jedoch einige grundsätzliche Probleme einer Mehr-Kinect-Erfassung behandelt.

Der Problematik der Bodenbewegungserkennung haben sich meinen Recherchen zufolge weitaus weniger Entwickler gewidmet. Kawatsu, Li und Chung befassen sich jedoch mit einem artverwandten Problem. Sie haben einen Algorithmus zur Erkennung von Stürzen entwickelt, welcher sich u.a. auf den Abstand zum Boden, sowie auf die Bewegungsgeschwindigkeit des Akteurs stützt [10]. Dieser Ansatz verfolgt zwar nicht das Ziel, Fehlerkennungen neu zu interpretieren, zeigt jedoch eine Möglichkeit zur grundlegenden Unterscheidung von Stand- und Bodenbewegungen.

Im vorangegangenen Kapitel wurden einige Grundlagen zu dieser Arbeit vermittelt. Mit Kinect wurde dem Leser das Eingabegerät vorgestellt, auf welchem der im Nachfolgenden entwickelte Motion Capture-Algorithmus basiert. Gleichzeitig wurden die Schwierigkeiten erläutert, welche es im Zusammenhang mit Kinect-basierten MoCap-Anwendungen zu bewältigen gilt und wie in verwandten Arbeiten damit umgegangen wurde.

# 3. Motion Capture-Programm

Das nun folgende Kapitel thematisiert das im Rahmen dieser Arbeit entwickelte Programm, beziehungsweise seinen MoCap-Algorithmus. Bei der Anwendung handelt es sich in erster Linie um ein Filmprogramm, welches dem Nutzer die Möglichkeit bieten soll, 3D-Charaktere mittels Motion Capture zu animieren. Die Bewegungen werden hierbei in Echtzeit übertragen. Zwar werden die Daten in einer automatischen Nachbearbeitung bereinigt, allerdings soll diese nur wenige Augenblicke beanspruchen. Der Anwender soll das Aufgenommene als Szene arrangieren, aber auch mit Effekten versehen und als Videodatei exportieren können. Im Kontext der Arbeit wird jedoch lediglich auf den Motion Capture-Algorithmus eingegangen. Dieser soll sowohl das Bodenbewegungsproblem lösen als auch eine echtzeitfähige 360°-Erfassung des Darstellers ermöglichen. Nachfolgend werden die einzelnen Verarbeitungsschritte, aus denen sich der Algorithmus zusammensetzt, näher erläutert. Abbildung 3 veranschaulicht den Grundaufbau der Methoden.


**Abb. 3 MoCap-Pipeline (e. Abb.)**

Die hier gezeigte Funktionskette wird im Folgenden auch als MoCap-Pipeline bezeichnet. Der Algorithmus bekommt zwei Eingaben (je eine pro Sensoreinheit), welche anschließend zu einem Datensatz vereinigt werden. Daraus werden darauffolgend die Bewegungsinformationen extrahiert. Die so gewonnenen Datensätze werden abschließend in einen Animation-Controller geschrieben und durch spezielle Nachbearbeitungsfunktionen (später mehr) behandelt. Der vorliegende Algorithmus ist nur für einen Akteur konzipiert. Grundsätzlich ist es mit Kinect (SDK v1.7) jedoch möglich, für bis zu zwei Akteure gleichzeitig ein Skeletal Tracking durchzuführen [2]. In Zukunft ließe sich die Anwendung folglich um ein paralleles Zwei-Personen-Tracking erweitern.

## 3.1 Ausgangsdaten

Zwei Kinects bilden zusammen die erste Komponente der MoCap-Pipeline. Die Sensoren führen in diesem Arbeitsschritt das Skeletal Tracking durch, extrahieren also pro Sekunde jeweils ~30 Skelettmodelle des agierenden Darstellers. Jedes Skelettmodell besteht aus insgesamt 20 kartesischen Koordinaten im dreidimensionalen Raum. Diese repräsentieren ausgewählte Körperteile des Nutzers und spiegeln seine Pose zum Zeitpunkt der Aufnahme wider, wie in Abbildung 4 veranschaulicht. Diese Modelle dienen dem Algorithmus als Eingabe. Das Skeletal Tracking an sich wird durch das Kinect SDK 1.7 implementiert.

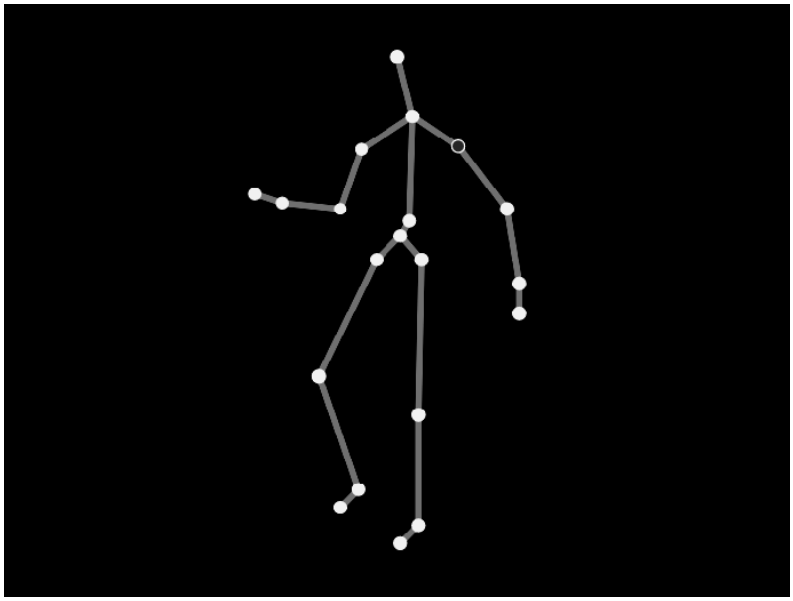

**Abb. 4 Skelettmodell aus dem Skeletal Tracking (e. Abb.)**

Alle generierten Skelettmodelle werden vom Algorithmus verarbeitet. Schließlich dient das Programm dazu, ganze Bewegungssequenzen des Nutzers aufzuzeichnen. Die nachfolgenden Abschnitte thematisieren jedoch lediglich die Verarbeitung einer einzelnen Momentaufnahme. Diese setzt sich allerdings aus zwei Skelettmodellen zusammen, denn wie zuvor beschrieben, ist die 360°-Erfassung des Akteurs ein wesentliches Ziel dieses Algorithmus. Beide Modelle zeigen den gleichen Augenblick, jedoch aus zwei verschiedenen Blickwinkeln. Zu diesem Zweck wird eine zweite Kinect im Raum platziert. Die Anordnung der Kameras ist hierbei von entscheidender Bedeutung. Sie werden gegenüberliegend voneinander aufgestellt und sollen den Nutzer, wie in Abbildung 5 veranschaulicht, aus zwei möglichst gegensätzlichen Perspektiven beobachten.

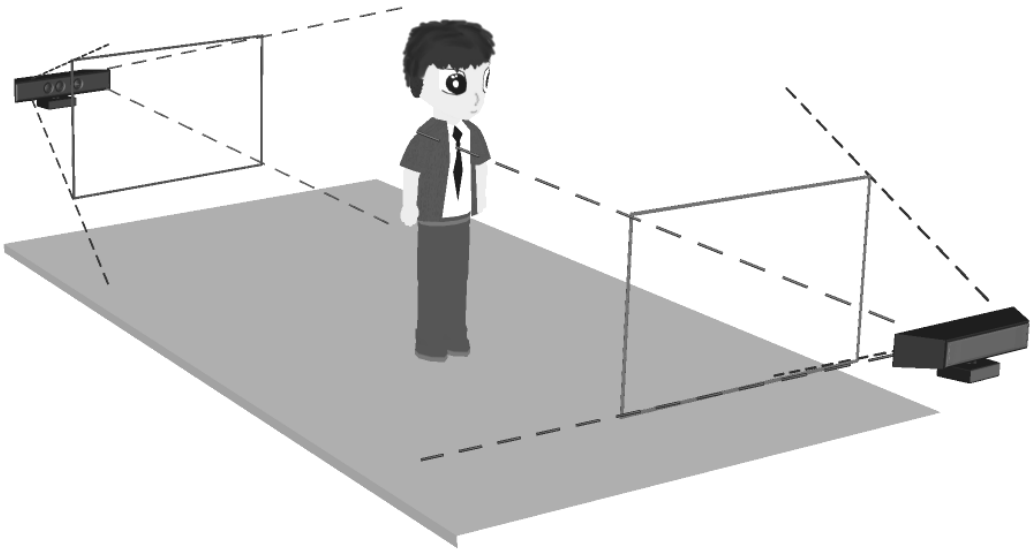

**Abb. 5 Grundaufbau des Aufnahmeraumes (e. Abb.)**

Um Interferenzen durch die Umgebung zu reduzieren, sollten möglichst alle Einrichtungs-Gegenstände aus dem nahen Umfeld des Darstellers entfernt werden. Weiter dürfen sich während der Aufnahme neben dem Akteur keine weiteren Personen im Sichtfeld der Sensoren befinden, da die hieraus resultierenden Datensätze das Ergebnis verfälschen. Ebenso erwiesen sich in der Praxis etwa Zimmerpflanzen und Lampen im Hintergrund als mögliche Störquellen, da sie von Kinect als Mensch interpretiert werden können. Beide Sensoren werden jeweils in einem separaten Thread behandelt. Nicht immer können beim Skeletal Tracking alle Punkte exakt bestimmt werden. Unbekannte Koordinaten werden meist aus vorigen Bildern, sowie anhand bekannter anderer Gelenkkoordinaten und dem Wissen über Skelett-Geometrie abgeleitet [5]. Neben den Koordinaten speichert die Datenstruktur zu jedem Punkt einen Zustands-Tag, genannt Tracking-State [5]. Für den Programmierer ist hierdurch sichtbar, ob ein Punkt getrackt wurde und wenn ja, ob es sich um eine erfasste oder lediglich um eine hergeleitete Koordinate handelt [5]. Ein ähnlicher Tag wird auch für das Skelettmodell selbst gesetzt [6]. Hintergrund ist die Fähigkeit des Sensors, über die Ganzkörpererkennung zweier Darsteller hinaus noch die Positionen von bis zu vier weiteren Personen im Raum zu bestimmen [2]. Entsprechend ist aus dem Tag ersichtlich, ob es sich um verwertbare Daten oder nur um ein Position-only-Tracking handelt [6]. Für die Weiterverarbeitung wird jeder nutzbare Datensatz mitsamt einer eindeutigen Sensor-ID an die TwoEyes-Struktur weitergeleitet, welche im nachfolgenden Abschnitt näher erläutert wird.

## 3.2 Erzeugung eines gemeinsamen Datensatzes

In diesem Abschnitt werden mit K1 und K2 die beiden Kinect-Sensoren bezeichnet. Noch handelt es sich bei den berechneten Skelettmodellen um zwei voneinander unabhängige Aufnahmen desselben Momentes aus zwei verschiedenen Perspektiven. Beide Modelle bewegen sich folglich in unterschiedlichen Räumen. Die Koordinatenachsen des ersten Sensors sind weder deckungsgleich mit denen des zweiten Sensors, noch sind sie an derselben Richtung orientiert. Da die Kinects gegenüberliegend zueinander aufgebaut werden, sind in den Ergebnisbildern sowohl links und rechts als auch vorne und hinten jeweils vertauscht (siehe: Abbildung 6). Zugleich gehen beide Kameras davon aus, die Vorderseite des Darstellers zu sehen. Dementsprechend gibt es noch keine Basis, eines der Modelle stärker zu gewichten als das andere. Ziel dieses Arbeitsschrittes ist es, beide Momentaufnahmen zu einem gemeinsamen Skelettmodell zu kombinieren. Das Ergebnis soll sich in seiner Struktur nicht von dem Datensatz eines einzelnen Sensors unterscheiden, sondern lediglich um die Informationen einer zweiten Kinect erweitert werden. Hierzu werden beide Skelettmodelle in einen gemeinsamen Raum transformieren und je nach Ausrichtung des Darstellers unterschiedlich stark gewichtet miteinander verrechnet.

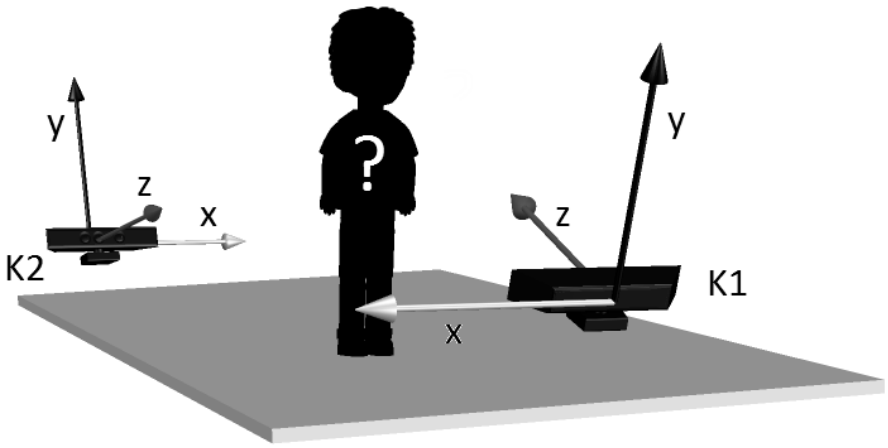


**Abb. 6 Logische Sensorkonfiguration vor der Kalibrierung (e. Abb.)**

Dies soll durch die TwoEyes-Struktur realisiert werden. Beide Kinects halten einen Zeiger auf dasselbe TwoEyes-Objekt und leiten, wie im vorigen Abschnitt beschrieben, alle verwertbaren Ergebnisdaten an dieses Objekt weiter. Über die mitgelieferte Sensornummer werden dabei die Eingaben beider Kameras auseinandergehalten. Der Algorithmus arbeitet im Folgenden immer nur jeweils mit dem aktuellsten Skelettmodell von K1 und K2. Diese werden in zwei Membervariablen zwischengespei-

chert. Mit jedem eingehenden Datensatz wird die jeweilige Variable unter gegenseitigem Ausschluss aktualisiert. Der nun vorgestellte Algorithmus zur Vereinigung der beiden aktuellsten Skelettmodelle basiert auf einer simplen Annahme: Bewegt sich eine Koordinate im K1-Modell x Schritte nach links, so bewegt sich die dazugehörige Koordinate im K2-Modell x Schritte nach rechts. Wandert sie z Schritte nach hinten, wandert die andere z Schritte nach vorne. Einzig die y-Verschiebungen stimmen aufgrund der identischen Ausrichtung der y-Achsen beider Sensoren überein.

### 3.2.1 Kalibrierung

Diese Annahme setzt jedoch eine Achsenparallelität der beiden Kinects voraus. Da diese aber nur bei einer absolut exakten Sensorkonfiguration gegeben ist, muss vor der Weiterverarbeitung der Skelettmodelle eine Kalibrierung des Aufnahmeraumes vorgenommen werden. Nach dieser werden die übertragenen Datensätze der beiden Kinects so transformiert, als seien sie aus einer perfekt justierten Position heraus aufgenommen worden. Einen ebenen Untergrund vorausgesetzt, müssen die Modelle hierfür lediglich um die x- und y-Achse herum rotiert werden. Als x-Wert wird einfach der Neigungswinkel der jeweiligen Kamera genutzt. Die y-Rotation wird hingegen mithilfe einer Konfigurationshaltung ermittelt. Der Nutzer nimmt zu diesem Zweck eine gut erkennbare Idle-Pose ein, das Gesicht zu K1 gewandt, welche von beiden Sensoren stabil erkannt wird (Skelettmodelle enthalten keine hergeleiteten Koordinaten (siehe Abschnitt 3.1: Tracking-State)). Der y-Rotationswert für das entsprechende Modell berechnet sich über den Vektor von einer Schulter zur anderen. Transformiert ergeben sich zwei achsenparallele Datensätze (siehe: Abbildung 7).

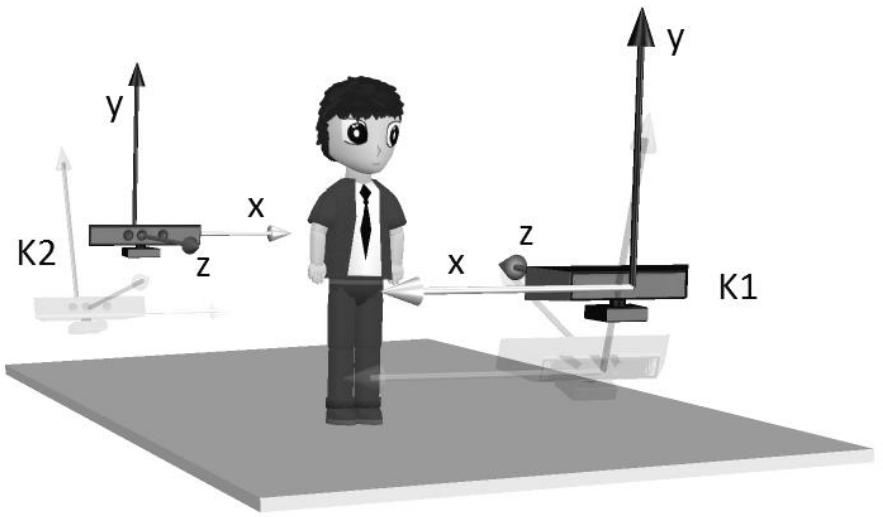


**Abb. 7 Logische Sensorkonfiguration nach der Kalibrierung (e. Abb.)**

Um nun beide Modelle gemäß der obigen Annahme kombinieren zu können, speichert das Programm die zuvor durchgeführte Idle-Pose transformiert für K1 und K2 als Kalibrierungsmodelle ab. Der Verwendungszweck wird im Folgeabschnitt erläutert. Gleichzeitig soll das Vorderseiten-Problem gelöst werden. Im Idle-Modell von K2 werden alle linken und rechten Koordinaten (linke/rechte Schulter, linker/rechter Ellbogen, usw.) vertauscht. Initial wird also angenommen, dass der Darsteller mit dem Gesicht zu K1 steht. Abschließend wird der Abstand zwischen den Schultern festgehalten und der Akteur wird in den Koordinatenursprung zurückgesetzt. Letzteres erleichtert die Nachbearbeitung, ist für den vorliegenden Algorithmus jedoch nicht relevant. Der Schulterabstand wird hingegen später benötigt, um die Gewichtungen der Modelle neu zu bestimmen (hierzu später mehr).

### 3.2.2 Transformierung

Für das vereinte Modell dient das Koordinatensystem von K1 als Ausgangsraum. Es muss also lediglich das K2-Modell in das Koordinatensystem des ersten Sensors transformiert werden. Sei $S$ das gegenwärtige Skelettmodell von Kinect Nr. 2. Basierend auf der oben gemachten Annahme wird daraus das in den K1-Raum transformierte Modell $S_t$ berechnet. $I_1$ und $I_2$ bezeichnen hierbei die Kalibrierungsmodelle von K1 und K2. So gilt für alle Körperelemente $i$:

$$S_t[i].x = I_1[i].x - (S[i].x - I_2[i].x)$$
$$S_t[i].y = I_1[i].y + (S[i].y - I_2[i].y)$$
$$S_t[i].z = I_1[i].z - (S[i].z - I_2[i].z)$$

Für Element i wird also einfach der Verschiebungsvektor (x, y, z) von $I_2$ zum neuen K2-Skelett errechnet und horizontal gespiegelt (-x, y, -z) zum Idle-Modell 1 addiert.

### 3.2.3 360°-Erkennung

Die bisherigen Überlegungen setzen allerdings voraus, dass die Blickrichtung des Darstellers auch während der Aufnahme bekannt ist. Da beide Sensoren grundsätzlich davon ausgehen, die Vorderseite des Akteurs zu sehen, werden sie folglich zwei gegensätzliche Datensätze liefern. Um diesem Problem entgegenzuwirken, muss einer der beiden Datensätze logisch gespiegelt werden (Bezeichner für links und rechts vertauscht). Hierfür dient ein einfacher Dreherkennungs-Algorithmus.

Für ein neues Skelettmodell berechnet das Programm zunächst die euklidischen Abstände der beiden Schultern zu den jeweils letzten bekannten Schulterkoordinaten im finalen Modell. Die vier resultierenden Längen werden dann einer Fallunterscheidung unterzogen. Seien $ll, lr, rl$ und $rr$ die vier euklidischen Abstände mit $l$ = linke Schulter und $r$ = rechte Schulter, wobei das erste $l/r$ für die letzte bekannte Koordinate steht und das zweite $l/r$ für die Koordinate im neuen Modell. Der Algorithmus geht genau dann von einer Drehung des Darstellers aus, wenn mindestens eine der drei nachfolgenden Bedingungen erfüllt ist:

1. $ll + rr > lr + rl$
2. $rl < rr \wedge rl < ll$
3. $lr < rr \wedge lr < ll$

Ist der Akteur, im Vergleich zur vorigen Aufnahme, demselben Sensor zugewandt, so gilt in der Regel keine der drei Bedingungen. Wechselt er hingegen die Blickrichtung, so interpretiert Kinect das Modell seitenverkehrt und geht demzufolge von einer schlagartigen (~) 180°-Drehung aus. Idealerweise sind dann alle drei Fälle erfüllt. Die Logik der Dreherkennung soll in Abbildung 8 veranschaulicht werden.

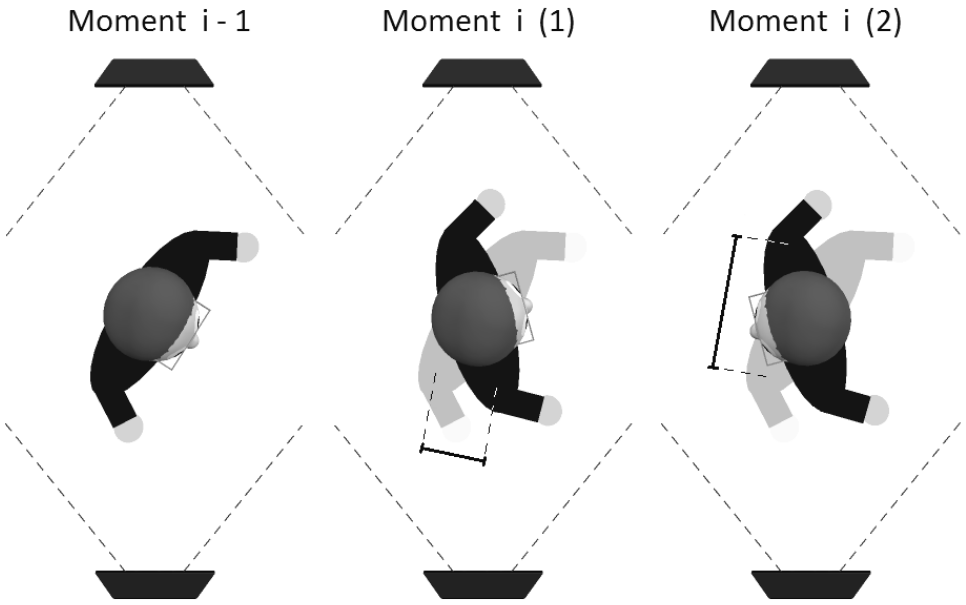


**Abb. 8 Dreherkennungs-Logik (e. Abb.)**

Für Moment i interpretiert der (im Bild) untere Sensor den Darsteller so, wie in i (2) veranschaulicht. Der obere Sensor bevorzugt Interpretation i (1). Der Vergleich der Schulterabstände zu Moment i – 1 entscheidet für i (1).

Ein früherer Ansatz stützte sich auf Annahmen bezüglich der Körperhaltung. Der Grundgedanke war, dass bestimmte Gelenkwinkel im Skelettmodell nur dann möglich, bzw. wahrscheinlich sind, wenn der Darsteller falsch herum erfasst wird, etwa nach vorne gedrehte Unterschenkel. In der Praxis erwies sich jedoch gerade der Übergang vom einen zum anderen Sensor als Schwachstelle, da in diesem Grenzbereich vergleichsweise unsaubere Datensätze generiert werden.

### 3.2.4 Gewichtung

Funktionieren die bisherigen Komponenten ordnungsgemäß, so liegen jetzt zwei theoretisch deckungsgleiche Datensätze vor, aufgenommen aus zwei verschiedenen Perspektiven. Praktisch gesehen sind die Skelettmodelle allerdings keineswegs immer deckungsgleich. Zu viele Faktoren fließen in die Berechnung eines Modells ein: Hergeleitete Koordinaten, Kalibrierungs-Ungenauigkeiten, Umgebungseinflüsse und nicht zuletzt die Tatsache, dass der Posen-Klassifikator von Kinect eben nicht für eine Rückseitenerkennung ausgelegt ist. Die Modelle sollten also nicht einfach zu gleichen Teilen miteinander verrechnet werden. Stattdessen soll eine Gewichtungsfunktion über die Relevanz der Koordinaten für das finale Modell entscheiden.

Diese basiert im Grunde auf der Schulterausrichtung des Akteurs. Das Koordinatensystem wird in vier Sektoren eingeteilt (siehe: Abbildung 9). Die Funktion vergleicht die letzten bekannten Positionen beider Schultern zueinander und entscheidet hierüber, in welchem Sektor sich der Darsteller momentan befindet. Je nach Sektor werden die linke und rechte Körperhälfte in beiden Kamerabildern unterschiedlich stark gewichtet. Mittige Körperteile, z.B. der Kopf, werden gleichwertig aus beiden Datensätzen hergeleitet. Für einen fließenden Übergang zwischen den Sektoren benötigt der Algorithmus den x-Anteil des aktuellen Schulterabstandes am Gesamtschulterabstand. Das Programm verwendet als Referenz jedoch nicht den momentanen Gesamtabstand, da dieser etwa bei Drehungen teils gravierend vom tatsächlichen Abstand abweicht. Stattdessen greift die Funktion auf die Länge aus der Kalibrierung zurück. Die Gewichtungsfaktoren werden anhand der folgenden Formeln bestimmt ($L/R$ = letzte bekannte Koordinate der linken/rechten Schulter; $G_{l/r}(K_i)$ = Gewicht der linken/rechten Körperhälfte für Kinect Nummer i):

$$xAnteil = \frac{aktueller\ Schulterabstand\ (x - Achse)}{Gesamtschulterabstand}$$

$$G_r(K_1) \begin{cases} 1 & ,wenn\ R_z < L_z\ und\ R_x \geq L_x \\ 1 - xAnteil & ,wenn\ R_z < L_z\ und\ R_x < L_x \\ 0 + xAnteil & ,wenn\ R_z \geq L_z\ und\ R_x \geq L_x \\ 0 & ,wenn\ R_z \geq L_z\ und\ R_x < L_x \end{cases} \qquad G_l(K_1) \begin{cases} 0 + xAnteil & ,wenn\ R_z < L_z\ und\ R_x \geq L_x \\ 0 & ,wenn\ R_z < L_z\ und\ R_x < L_x \\ 1 & ,wenn\ R_z \geq L_z\ und\ R_x \geq L_x \\ 1 - xAnteil & ,wenn\ R_z \geq L_z\ und\ R_x < L_x \end{cases}$$

$$G_r(K_2) = 1 - G_r(K_1) \qquad G_l(K_2) = 1 - G_l(K_1)$$

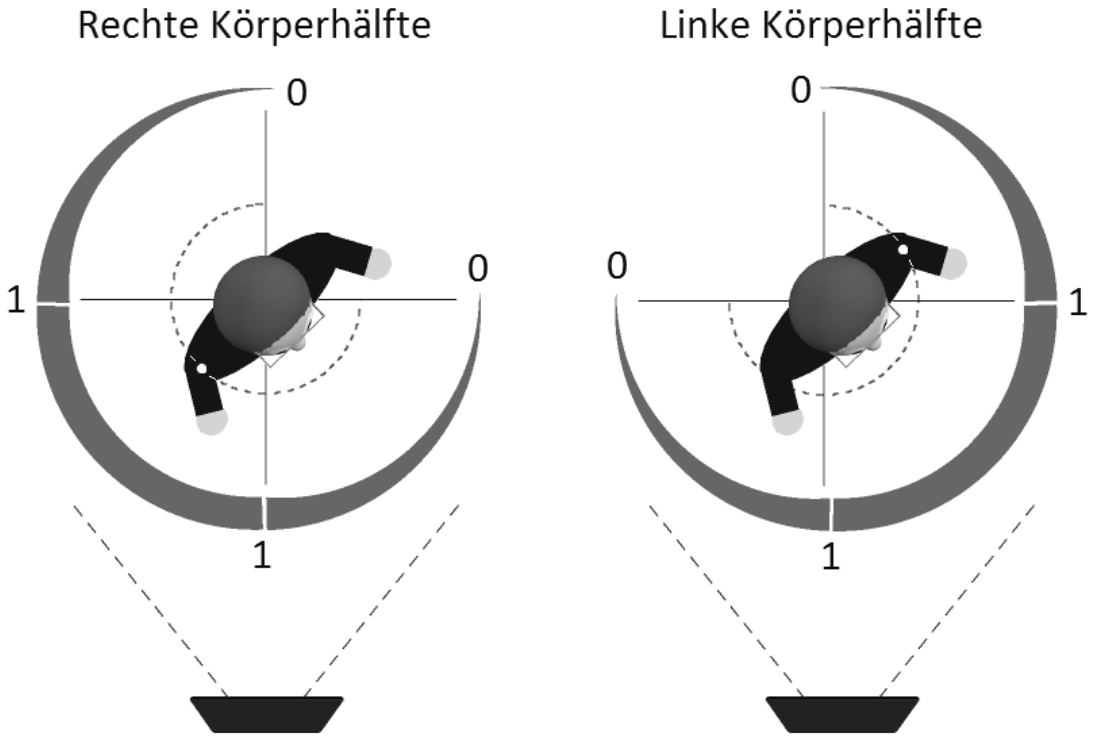


**Abb. 9 Gewichtungsberechnung (e. Abb.)**

Je nach Sektor variieren die Gewichtungen beider Körperhälften zwischen 0 und 1. Im hier abgebildeten Beispiel ergibt sich für die rechte Körperhälfte ein Gewicht von 1. Die linke Körperhälfte erhält ein Gewicht knapp über 0.5. Für den zweiten (hier nicht dargestellten) Sensor ergeben sich demnach Gewichte von 0 (rechte Körperhälfte) und einem Wert knapp unter 0.5.

Die soeben errechneten Gewichte werden genau dann angewandt, wenn beide Sensoren ein Körperelement i gleich sicher/unsicher erfasst, also für i den gleichen Tracking-State vergeben haben. Bei einem unterschiedlichen Tracking-State wird hingegen immer der sicherere vorgezogen. Hat K2 beispielsweise den linken Fuß erfasst, während K1 die Position nur herleiten kann, so fließt nur die K2-Koordinate in das Gesamtergebnis ein. Ebenso wird der Tracking-State an das finale Modell vererbt.

## 3.3 Extraktion der Bewegungsdaten

Im vorhergehenden Arbeitsschritt wurde ein neues Skelettmodell generiert, zusammengesetzt aus den Informationen zweier Kinects. Letztendlich soll mit diesem Datensatz ein menschliches 3D-Modell animiert werden. Folglich besteht der nächste Schritt der MoCap-Pipeline darin, aus dem Skelett die Daten zu extrahieren, welche für die Animation benötigt werden. Es soll erläutert werden, um was für Daten es sich im Konkreten handelt und wie diese aus dem Modell berechnet werden.

### 3.3.1 DirectX_Character

Die im Rahmen dieser Arbeit entwickelte Anwendung nutzt zur 3D-Visualisierung DirectX 9. Bei DirectX_Character handelt es sich um die Datenstruktur, welche die 3D-Ressourcen der zu animierenden Figur verwaltet. Weiter verfügt die Struktur über einen Animation-Controller, welcher sich um die Speicherung und Wiedergabe von Animationen sowie um die Interpolation fehlender Animationsmomente kümmert. Das eigentliche Modell liegt als X-Objekt vor und wird beim Laden hierarchisch geordnet. Hierbei stellt der Torso das Wurzelelement dar. Der Kopf, sowie alle oberen Gliedmaßen, sind direkte Unterelemente des Torsos. Es folgen als weitere Unterelementebenen die unteren Gliedmaßen und darunterliegend die Hände, bzw. Füße (siehe auch: Abbildung 10). Durch den hierarchischen Aufbau wird es z.B. ermöglicht, eine Rotation des Oberarmes direkt auf den Unterarm und somit auch auf die Hand zu übertragen. Die unterschiedlichen Körperteile sind dabei nicht an ein festes Skelett gekoppelt, sondern der Nutzer arbeitet unmittelbar mit den Rotationswerten der Elemente selbst. Diese gilt es aus dem Ergebnismodell des vorigen Arbeitsschrittes zu extrahieren.

### 3.3.2 Kinematische Kette

Die Drehwinkel der einzelnen Körperkomponenten werden mithilfe des jeweiligen Vektors im Skelettmodell und der Achsen des Koordinatensystems berechnet. Der Algorithmus wandert hierfür hierarchisch durch das Modell, beginnend beim Torso (Wurzelelement). Dieser bestimmt die Ausrichtung des gesamten Modells. Die z-Rotation berechnet sich aus dem Winkel zwischen der globalen xz-Ebene und der Linie von der linken zur rechten Schulter des Darstellers. Auch die Blickrichtung (y-Rotation) wird anhand der Schultern bestimmt. Für die x-Rotation wird hingegen die Verbindungslinie von der Hüfte bis zum Hals als Referenzvektor herangezogen. Für

alle weiteren Elemente muss jedoch der hierarchische Aufbau des 3D-Modells berücksichtigt werden.

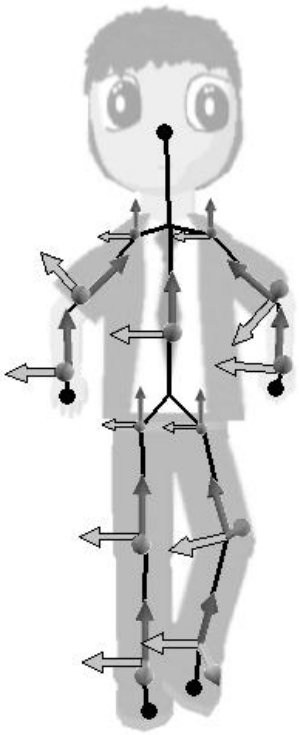

**Abb. 10 Kinematische Kette (e. Abb.)**

Unterelemente kopieren die Bewegungen ihrer Oberobjekte. Es entsteht eine kinematische Kette. Würden sich alle Winkel am globalen Koordinatensystem orientieren, so würde den Berechnungen der Bezug zu den Oberelementen fehlen. Die Folge wäre, dass sich die Winkel von oben nach unten (hierarchisch) zu einem fehlerhaften Wert aufsummieren. Hier reicht es nicht, einfach die übergeordneten Rotationswerte abzuziehen. Stattdessen müssen die Rotationen, wie in Abbildung 10 veranschaulicht, anhand der lokalen Koordinatensysteme der jeweiligen Oberelemente berechnet werden. Das lokale Koordinatensystem des Torsos basiert auf den Schulterkoordinaten und den oben berechneten Rotationswerten. Für alle weiteren Elemente wird nach dem folgenden Schema verfahren: Sei $(v_x, v_y, v_z)$ der normalisierte Vektor im Skelettmodell, welcher das zu behandelnde Unterobjekt beschreibt, beispielsweise der Vektor vom Ellbogen zur Hand, wenn die Winkel für den Unterarm extrahiert werden sollen. Weiter bezeichnen $x_{i-1}$, $y_{i-1}$ und $z_{i-1}$ die lokalen Koordinatenachsen des Elternelementes. Dann werden die dazugehörigen Rotationswerte $x$ und $z$ gemäß der folgenden Abbildung berechnet:

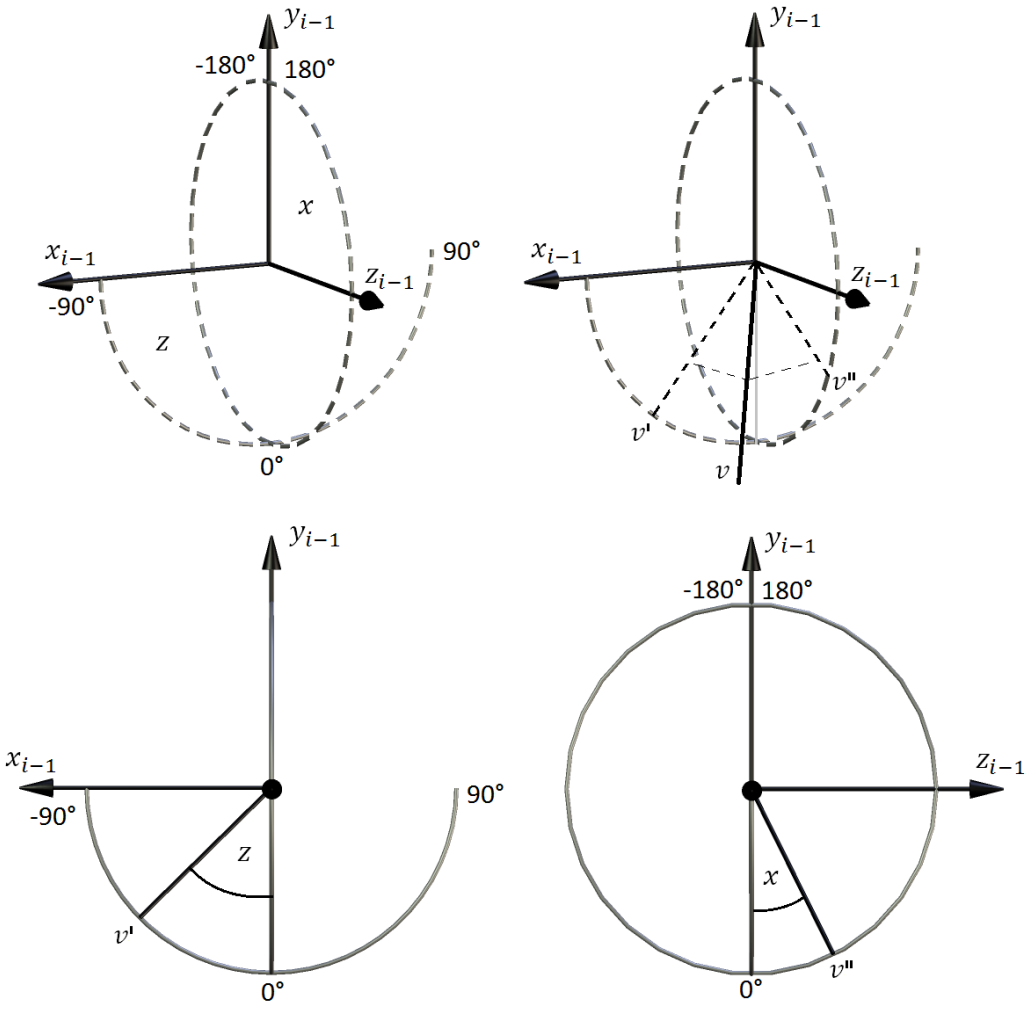


**Abb. 11 Berechnung der Rotationswerte: Kinematische Kette (e. Abb.)**

Für die z-Achsen-Rotation wird $v$ in die $x_{i-1}y_{i-1}$-Ebene projiziert → $v'$

Für die x-Achsen-Rotation wird $v$ in die $y_{i-1}z_{i-1}$-Ebene projiziert → $v''$

Anmerkung zu z: Für $v' = (v_x, v_y)$ mit positivem $v_y$ (oberhalb der $x_{i-1}$-Achse) ergibt sich der gleiche z-Wert wie für $(v_x, -v_y)$. Die Rotation über die $x_{i-1}$-Achse wird über den x-Wert erreicht ($v''$)

*y*-Rotationswerte werden nicht erfasst, da Drehungen entlang der Längsachse aus dem Skelettmodell nicht hervorgehen. Anschließend wird aus den Achsen $x_{i-1}$, $y_{i-1}$ und $z_{i-1}$ das lokale Koordinatensystem ($x_i$, $y_i$, $z_i$) für das Objekt selbst hergeleitet. Hierzu werden die Achsen mithilfe der soeben errechneten Werte transformiert. Ggf. wird nun mit Kind-Elementen fortgefahren. Für eine saubere Darstellung sollte jeweils der Übergang zwischen Ober- und Unterelement das Rotationszentrum des Unterelementes bilden. Weiter sollte das Modell in der Idle-Haltung (alle Rotationswerte auf Null) aufrecht stehen, mit herunterhängenden Armen.

Zu Beginn hat der Algorithmus zwei Aufnahmen desselben Momentes aus unterschiedlichen Blickwinkeln als Eingabe bekommen. Beide Kinects haben ein Skelettmodell generiert, welches die Pose des Darstellers aus ihrer jeweiligen Perspektive zeigt. Diese wurden im letzten Arbeitsschritt zu einem Datensatz vereint. Das kombinierte Modell enthält idealerweise mehr Informationen als die Ausgangsdaten eines einzelnen Sensors und spiegelt 360°-Drehungen des Akteurs wider. Daraus wurden nun die entsprechenden Rotationswerte für das Charakter-3D-Modell abgeleitet. Theoretisch lässt sich die Figur jetzt in Echtzeit durch Körperbewegungen animieren. Damit ist jedoch noch nicht das Problem der Bodenbewegungserkennung gelöst, wonach bei bodennahen Bewegungsabläufen mit fehlerhaften Ergebnissen zu rechnen ist. Tritt ein solcher Fehler auf, soll der vorliegende Algorithmus dies erkennen und die falschen Werte mithilfe von Musterposen aus einer Datenbank korrigieren. Wie im Abschnitt 2.1.3 beschrieben, führen unterschiedliche Bodenbewegungen zu teils sehr charakteristischen Fehlerkennungen. Somit lässt sich die Aufgabe grundsätzlich als Klassifizierungsproblem behandeln.

## 3.4 Fehlerkorrektur

Um falsch erkannte Bodenbewegungen neu zu interpretieren, wird ein Klassifikator benötigt, welcher die fehlerhaften Winkel als Eingabe bekommt und anhand ihrer Charakteristiken erkennt, um welche Pose es sich handelt. Dabei sollte der Klassifikator verschiedenen Anforderungen genügen. Zunächst muss er beliebig erweiterbar sein, sollte sich eine bestimmte Bewegung nicht in der Datenbank befinden. Hieraus ergibt sich ein weiterer Anspruch. Im Gegensatz zu ggf. vordefinierten Posen kennt der Klassifikator neue Bewegungen noch nicht. Dem Nutzer muss es möglich sein, die dazugehörige Trainingsmenge aufzuzeichnen. Es wird also ein lernfähiger Klassifikator benötigt.

### 3.4.1 Neuronales Netz

Dies soll durch ein neuronales Netz realisiert werden (siehe: Abbildung 12). Ein neuronales Netz besteht aus mehreren Recheneinheiten, welche zusammen eine Netzfunktion bilden. Diese bekommt einen Vektor als Eingabe und transformiert diesen in einen Ausgabevektor [13]. Erreicht wird dies durch unterschiedliche Schichten, auch Layer genannt [13]. Der Eingabevektor stellt den Input-Layer dar und beinhaltet in diesem Fall die zu klassifizierenden Daten. Diese werden im ersten Schritt transformiert und gelangen so in den Hidden-Layer [13]. Prinzipiell kann ein neuronales Netz

über beliebig viele Hidden-Layer verfügen [13]. Das hier beschriebene Programm nutzt aber lediglich eine Schicht. Durch eine weitere Transformation wird der Ausgabevektor erzeugt (Output-Layer) [13]. Im Grunde kann dieser willkürliche Daten enthalten. In einem Klassifikationsszenario sollte es sich hierbei jedoch um Werte handeln, aus denen die zur Eingabe passende Klasse ersichtlich ist.


**Abb. 12 Abstrakter Aufbau eines neuronalen Netzes (e. Abb.)**

Links: Input-Layer; Mitte: Hidden-Layer; Rechts: Output-Layer

Konkret implementiert soll die Netzfunktion die Rotationswinkel der Ober- und Unterschenkel (ohne Längsachsenrotation) als Eingabevektor entgegennehmen, da genau diese von den besagten Erkennungsschwierigkeiten betroffen sind. Die Eingabe soll in einen Ausgabevektor transformiert werden, welcher die zu den Beinwinkeln passende Bewegungsklasse repräsentiert. Sei $i$ der Index dieser Klasse in einer Liste aus $m$ bekannten Bewegungsklassen. Dann soll der Ausgabevektor die Länge $m$ haben, an der Stellen $i$ eine Eins enthalten und ansonsten nur Nullen. Für einen Schneidersitz könnte die Netzfunktion etwa die Ausgabe [1; 0; 0; …] liefern, wogegen z.B. ein Hinknien den Rückgabewert [0; 1; 0; …] erzeugt, usw. Die Ein- und Ausgabe wird in Abbildung 13 dargestellt:

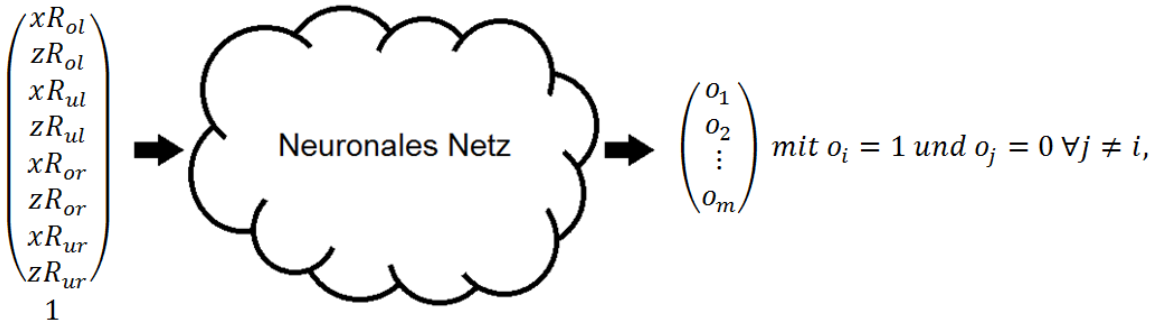


**Abb. 13 Ein- und Ausgabeformat: Neuronales Netz (e. Abb.)**

wenn $i$ der Index der zur Eingabe passenden Bewegungsklasse ist.

$xR$ = x-Rotation; $zR$ = z-Rotation; $o$ = Oberschenkel; $u$ = Unterschenkel; $l$ = links; $r$ = rechts

### 3.4.2 Backpropagation

Was fehlt, ist die Definition der Netzfunktion. Die Transformation innerhalb des Netzes geschieht über Gewichtungsmatrizen. Initial sind alle Werte in den Matrizen zufällig gewählt. Auch wenn für einen Eingabevektor $x$ die Ausgabe $t$ erwartet wird, so wird das Ergebnis der Funktion vorerst nicht $t$ entsprechen [13]. Das neuronale Netz muss also erst trainiert werden. An dieser Stelle kommt der Backpropagation-Algorithmus ins Spiel. Hierbei handelt es sich um einen populären Lernalgorithmus [13]. *„Das Lernproblem besteht darin, die optimale Kombination an Gewichtungen zu finden, sodass sich die Netzfunktion $\varphi$ möglichst nahe einer gegebenen Funktion $f$ annähert."* [13]. $f$ ist jedoch nicht explizit vorgegeben [13]. Lediglich die gewünschten Ein- und Ausgaben von $f$ sind bekannt (siehe: voriger Abschnitt) [13]. Wie nahe sich $\varphi$ schlussendlich an $f$ angenähert hat, wird über die Fehlerfunktion $E$ beurteilt. Sei $\{(x_1, t_1), \dots, (x_p, t_p)\}$ eine Trainingsmenge aus $p$ Eingabevektoren sowie den dazu erwarteten Ausgabevektoren. Sei zudem $o_i = \varphi(x_i)$ die gegenwärtige Ausgabe der Netzfunktion für die Eingabe $x_i$ mit $(1 \leq i \leq p)$ [13]. Dann gilt:

$$E = \frac{1}{2}\sum_{i=1}^{p} \|o_i - t_i\|^2 \qquad [13]$$

Der Backpropagation-Algorithmus soll mittels Gradientenabstieg das Minimum von $E$ finden [13], also die Gewichtungsmatrizen des neuronales Netzes so verändern, dass die Eingabevektoren zur gewünschten Ausgabe transformiert werden. Zu diesem Zweck wird für eine Eingabe der momentane Fehler der Funktion durch das Netz zurückpropagiert, mit dem Ziel, Korrekturwerte für die Gewichtungsmatrizen zu berechnen [13]. Hier sei zu erwähnen, dass die Netzfunktion $\varphi$ nur Schrittweise an $f$ angenähert werden kann. Um $\varphi$ möglichst genau an $f$ anzugleichen, muss der Korrekturprozess mit zahlreichen Ein- und Ausgabepaaren $(x_i, t_i)$ wiederholt werden. Ebenso wird eine Schrittweite festgelegt, mit welcher die Korrekturwerte multipliziert werden. Diese muss jedoch gut gewählt sein. Ist sie zu hoch, so werden möglicherweise optimale Gewichtungen übersprungen. Ist sie zu niedrig, so tritt das neuronale Netz auf der Stelle. Die Korrektur kann entweder sofort erfolgen (online) oder die Korrekturen werden erst zusammenaddiert (offline) [13]. Der vorliegende Algorithmus nutzt die online-Variante.

Im Falle der Bodenbewegungserkennung muss der Nutzer demnach manuell Bewegungsklassen definieren und für diese eine ausreichend große Trainingsmenge aufzeichnen, bestehend aus einer Vielzahl von Beinrotationswinkeln sowie den dazugehörigen Klassenvektoren. Wie groß diese Menge sein muss, richtet sich vor allem nach der Anzahl und Unterscheidbarkeit der Bewegungsklassen (u.a. Thema des nachfolgenden Experimentes). Nach dem Training sollte das Netz idealerweise in der Lage sein, neu eingegebene Winkel korrekt einer Pose zuzuordnen. Soll der Klassifikator um eine zusätzliche Haltung erweitert werden, so muss der Anwender lediglich die entsprechenden Ein- und Ausgabepaare (Beispielvektoren, Klassenvektor) der Trainingsmenge hinzufügen.

Mit dem neuronalen Netz verfügt das Programm jetzt über ein wesentliches Werkzeug, um zumindest einige fehlerhafte MoCap-Datensätze zu berichtigen. Im letzten Schritt der MoCap-Pipeline sollen nun alle bisherigen Bausteine, einschließlich des soeben definierten Klassifikators, zu einem Gesamtergebnis zusammengefügt werden. Angenommen wird hierzu eine optimal trainierte Netzfunktion, welche jede der ausgeführten Bodenbewegungen erkennt.

## 3.5 Animation

Die bisherigen Arbeitsschritte beschäftigen sich alle mit einzelnen Momentaufnahmen. Der folgende Abschnitt soll nun auch die zeitliche Komponente berücksichtigen. Es wird Zeit, eine komplette Bewegungssequenz aufzuzeichnen. Eine Animation ist genau genommen nur eine Folge verschiedener Posen. So ist auch diese Animation nichts anderes als eine Abfolge mehrerer Rotationsdatensätze, wie sie in den letzten Schritten der MoCap-Pipeline berechnet wurden. Im Grunde würde es zur Animation der 3D-Figur ausreichen, alle eingehenden Daten in ihren Animation-Controller zu schreiben. Für ein zufriedenstellendes Ergebnis ist jedoch eine Nachbearbeitung erforderlich. Diese soll unmittelbar nach der Aufnahme automatisch erfolgen und nicht viel Zeit in Anspruch nehmen.

### 3.5.1 Interpretation

Ein wesentlicher Teil der Nachbearbeitung besteht in der Korrektur von Erkennungsfehlern, wofür im vorangegangenen Schritt der Klassifikator implementiert wurde. Für die Neuinterpretation werden einfach alle Bodenbewegungen durch die Netzfunktion klassifiziert. Wird hierbei eine der trainierten Bewegungen erkannt, so müssen ledig-

lich die falschen Beinwinkel durch korrekte Werte ersetzt werden. Zu diesem Zweck verfügt das Programm über eine Bodenbewegungsliste. In ihr sind die optimalen Rotationswinkel für sämtliche Bodenbewegungen hinterlegt, für welche das neuronale Netz trainiert wurde. Dabei kann es sich nur dann um eine Bodenpose handeln, wenn vom Darsteller ein bestimmter Schwellwert auf der y-Achse unterschritten wird. Damit soll vermieden werden, dass fälschlicherweise Standposen durch den Klassifikator neu interpretiert werden. Gleichzeitig soll bei grenzwertigen Entscheidungen ein ständiges Hin- und Her zwischen den einzelnen Posen verhindert werden (siehe: Abbildung 14). Dies erfordert jedoch eine Betrachtung der umliegenden Animationsmomente. Das Programm iteriert hierzu über die Animation und wartet auf den ersten interpretierbaren Datensatz, also auf die erste vom Klassifikator erkannte Bodenbewegung. An dieser Stelle setzt der Algorithmus einen Startmarker und beginnt damit, die Klassifizierungsergebnisse in eine anfangs leere Liste zu schreiben. Hiermit wird so lange fortgefahren, bis entweder das Ende der Bodenbewegung (erkennbar durch den y-Schwellwert) oder das Ende der Gesamtanimation erreicht ist. Ist dies der Fall, so sucht die Funktion nach der letzten klassifizierten Pose und setzt an eben jener Stelle einen Zielmarker. Anschließend wird zwischen den beiden Markern per Mehrheit entschieden, welche Bodenbewegung letztendlich ausgeführt wurde. Im gesamten Bereich zwischen den Markern werden dann die fehlerhaften Beinwinkel mit den Musterwinkeln der entsprechenden Pose überschrieben. Die Funktionsweise soll exemplarisch durch Abbildung 14 veranschaulicht werden:

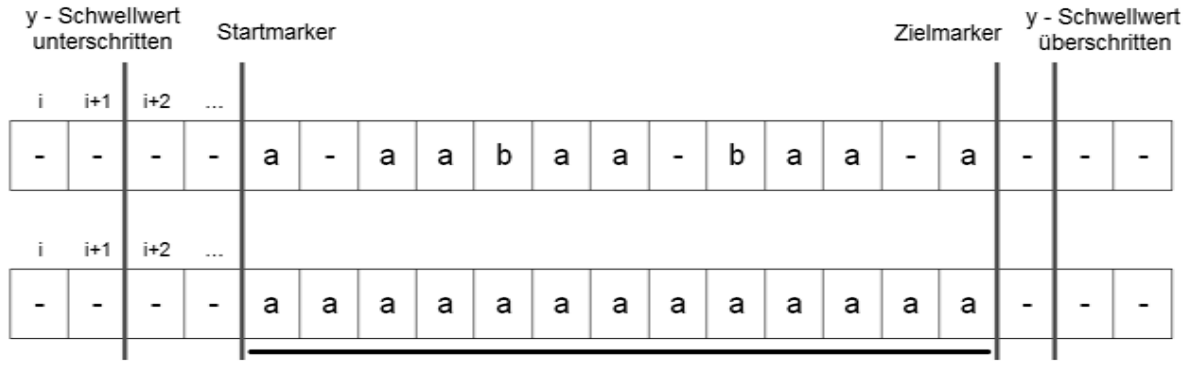


**Abb. 14 Interpretations-Logik (e. Abb.)**

Jedes Feld steht für einen Animationsmoment. Der Wert im Feld spiegelt das Klassifikationsergebnis des neuronalen Netzes für die Pose im jeweiligen Moment wider. a und b stehen für zwei verschiedene Bodenbewegungen. '-' bedeutet, dass der Moment keiner bekannten Pose zugeordnet werden konnte. In der ursprünglichen Animation (oben) wurden acht Momente als Pose a interpretiert, weitere zwei als Pose b und drei Momente wurden nicht erkannt. Nach Mehrheitsentscheidung werden alle Momente im markierten Bereich mit Pose a überschrieben (unten)

### 3.5.2 Verdrehungen auflösen

Idealerweise spiegelt die Animation nun korrekt die tatsächliche Bewegungsausführung wider. Jedoch bringt die soeben durchgeführte Korrektur ggf. ein Problem mit sich. Während die Rotationswerte augenscheinlich mit den Bewegungen des Darstellers übereinstimmen, können sich in den Winkeln einige Verdreher verstecken. 180° und 540° beispielsweise erscheinen im 3D-Modell gleich. Bei einer Interpolation mit einem gleichbleibenden Winkel, etwa, wenn eine Zeitlupe simuliert werden soll, führen die Werte aber zu zwei völlig unterschiedlichen Ergebnissen. Dasselbe Problem ist auch in nicht korrigierten Datensätzen vorhanden. Dreht sich z.B. der Akteur mehrfach um seine eigene Längsachse, so werden sich die y-Rotationswinkel in den Rohdaten stets wiederholen. Auch hier würde eine Winkelinterpolation zu ungewollten Ergebnissen führen. Daher müssen alle Verdrehungen dieser Art in der gesamten Animation aufgelöst werden. Das Programm iteriert hierzu über die Animationsmomente und rechnet alle Winkel so um, dass sie sich im Vergleich zum vorigen Moment um maximal 180° unterscheiden (siehe Beispiel: Abbildung 15).

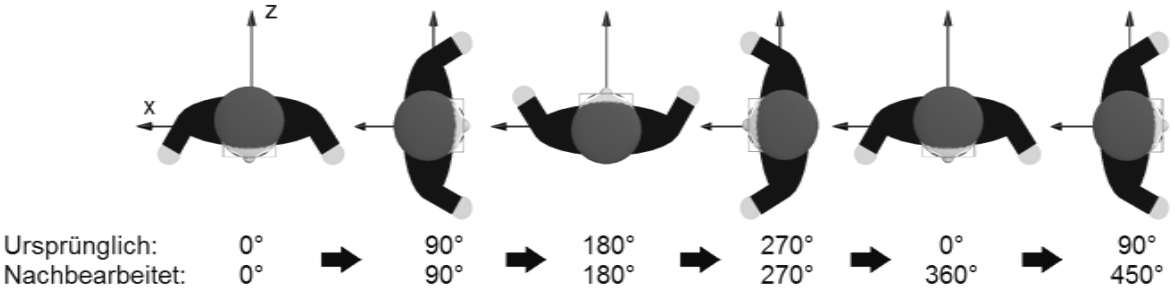


**Abb. 15 Auflösen von Verdrehungen (e. Abb.)**

Das beschriebene Problem ließe sich relativ einfach durch den Einsatz eines anderen Interpolationsverfahrens umgehen, etwa durch Vektor- oder Matrix-Interpolation. Der obige Zwischenschritt wäre somit überflüssig, aber es würden auch Drehungsinformationen verloren gehen. Entfernt der Nutzer etwa zur Glättung die Animationsmomente zwei, drei und vier in der vorangegangenen Abbildung, so wird die durchgeführte Drehung bei einer Vektor- oder Matrix-Interpolation gänzlich verschluckt. Lediglich bei einer Winkelinterpolation in der zweiten Zeile würde die Drehung erhalten bleiben. Dies gilt ebenso für manuell eingegebene Rotationswerte, z.B., wenn der Animation nachträglich eine Pirouette hinzugefügt werden soll, definiert durch einen Start- und einen Ziel-Winkel.

### 3.5.3 Glättung

Ein Makel zeigt sich vor allem in der praktischen Anwendung. Selbst wenn die Einzelposen der Figur optisch keine Fehler enthalten, so wirkt das Gesamtergebnis in der Bewegung oftmals zittrig und unruhig. Dieser Effekt entsteht in erster Linie dann, wenn bestimmte Koordinaten im Skelettmodell von Moment zu Moment umher zucken. Er wird aber auch durch die obige Korrektur der Beinwinkel hervorgerufen. Wechselt die Figur in einem Animationsmoment in eine womöglich komplett andere Beinhaltung, so verliert die Bewegung an Natürlichkeit. Um diesen Effekt abzuschwächen, wird ein Glättungsfilter auf die Animation angewandt. Hierbei wird jeder Winkel $w'$ des Modells zum Zeitpunkt $i$ $(w'_i)$ aus den umliegenden Animationsmomenten, mit abnehmender Gewichtung, nach folgender Formel hergeleitet:

$$g_j = \frac{2^k}{2^j} \quad mit \quad 0 \leq j \leq k \quad und \quad k \geq 1$$

$$w'_i = \frac{w_{i-k}\, g_k + \cdots + w_{i-1}\, g_1 + w_i\, g_0 + w_{i+1}\, g_1 + \cdots + w_{i+k}\, g_k}{g_0 + 2(g_1 + g_2 + \cdots + g_k)}$$

$w_i$ ist hierbei der jeweilige Winkel zum Zeitpunkt $i$ in der nicht geglätteten Animation. Je größer $k$ gewählt wird, desto mehr der benachbarten Momente werden in der Glättung berücksichtigt. Hierdurch wird die Bewegung zwar weicher, jedoch sei darauf zu achten, dass mit zunehmender Größe von $k$ auch ein beabsichtigtes Zucken des Darstellers gefiltert werden kann. Es gilt also, ein gutes Gleichgewicht für $k$ zu finden.

### 3.5.4 Übertragung

Die Animation ist nun komplett und wird abschließend auf das zu animierende 3D-Modell übertragen. Hierzu werden die zuvor erhobenen Rotationsdatensätze Bild für Bild in den Animation-Controller der Figur geschrieben. Über eindeutige Bezeichner werden dabei die entsprechenden Körperteile des 3D-Charakters ausfindig gemacht. Damit die Bewegung auch zeitlich korrekt wiedergegeben wird, muss der Übertragungsalgorithmus mögliche Frameratenschwankungen seitens Kinect kompensieren. Zu diesem Zweck besitzt jeder Datensatz einen Zeitstempel, welcher bei der Übertragung berücksichtigt wird.

In diesem Kapitel wurde dem Leser die für diese Arbeit entwickelte Anwendung vorgestellt. In den einzelnen Abschnitten wurden die Verarbeitungsschritte des MoCap-Algorithmus näher erläutert, angefangen bei den Eingabedaten und wie daraus der finale Datensatz errechnet wird. Zudem wurden mögliche Lösungen für die zuvor genannten Einschränkungen präsentiert, mit welchen in einem Kinect-basierten Motion Capture-Szenario zu rechnen ist. Im nächsten Kapitel wird die Anwendung verschiedenen praktischen Tests unterzogen.

# 4. Experiment

Die nun folgenden Versuche sollen offenlegen, inwiefern die Ziele dieser Arbeit erfüllt werden konnten. In diesem Kontext sind natürlich die Dreherkennungsfunktion und der Bodenbewegungsklassifikator von besonderem Interesse. Diese werden, zunächst unabhängig von den restlichen Funktionalitäten der MoCap-Pipeline, in zwei Versuchsreihen getestet. Hierbei werden schlicht die Fehlerraten der beiden Funktionen diskutiert. Ein dritter Test soll die Tauglichkeit der gesamten Pipeline in einem praktischen Anwendungsszenario überprüfen. Zu diesem Zweck wird mehrmals eine Motion Capture-Animation aufgezeichnet. Dabei werden erneut die Fehlerraten der einzelnen Komponenten beobachtet. Aber auch die Bearbeitungszeit sowie der qualitative Vergleich zwischen der unbehandelten und fertigen Aufnahme sind relevante Ergebnisse.

## 4.1 Versuchsaufbau

Alle Teilversuche werden unter den gleichen Bedingungen durchgeführt. Der Versuchsaufbau ist demzufolge in allen drei Fällen identisch. Für das Experiment werden die folgenden Hardware-Komponenten verwendet: Zwei Kinects v1 inklusive Stromzuführung sowie ein Rechner (Windows 8 (32 Bit); 2.66 GHz Prozessor; 4 GB RAM; DirectX 9 kompatible Grafikkarte; USB 2.0). Genutzte Software: Das im vorangegangenen Kapitel entwickelte Programm sowie die Laufzeitumgebungen für Kinect (in diesem Fall v1.7) und DirectX 9.

Das Experiment findet in einem Raum mit einer freien Fläche von mindestens Fünf mal Zwei Metern statt. Vor der eigentlichen Durchführung wird der Aufnahmeraum vorbereitet. Zunächst werden beide Kinects fünf Meter voneinander entfernt in einer Höhe von ca. 50 cm aufgestellt und jeweils an eine eigene Stromquelle angeschlossen. Beide Sensoren (nachfolgend auch K1 und K2 genannt) werden frontal aufeinander ausgerichtet und mit dem Computer verbunden. Dieser steht mittig an der Längsseite des Aufnahmebereiches. Ein USB-Verlängerungskabel wird nicht benötigt. Die Kameras werden mit einem Nickwinkel (pitch) von etwa Acht Grad justiert. Abbildung 16 zeigt die beschriebene Sensorkonfiguration.

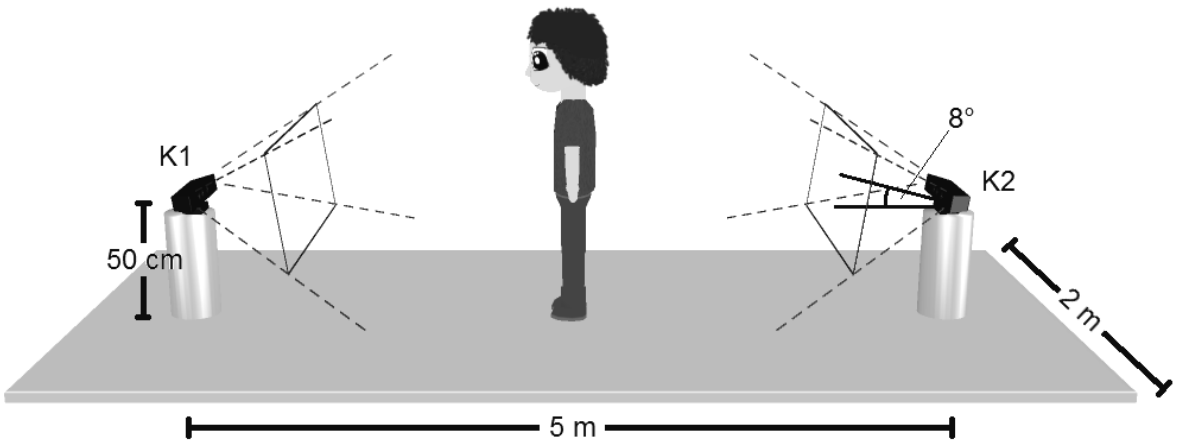


**Abb. 16 Versuchsaufbau: Sensorkonfiguration (e. Abb.)**

Um die Kapazität der USB-Schnittstelle nicht zu belasten, werden während des Versuches alle USB-Geräte, Maus und Tastatur ausgenommen, entfernt. Weiter sollen Interferenzen durch die Umgebung reduziert werden. Zu diesem Zweck werden alle Gegenstände aus den Sichtfeldern der Sensoren entfernt, welche von ihnen als Person interpretiert werden könnten oder gar den Darsteller verdecken oder ihn behindern könnten. Desweiteren dürfen sich keine zusätzlichen Infrarot-Lichtquellen im Bild befinden, welche die Tiefenmessung des Raumes stören könnten. Abgesehen hiervon wird bezüglich der Lichtverhältnisse lediglich darauf geachtet, dass keiner der Sensoren direkt von der Sonne angestrahlt wird. Um den Kinects die Erkennung zu erleichtern, trägt der Akteur normale nicht zu weite Kleidung. Vor der Ausführung wird das System gemäß Abschnitt 3.2.1 kalibriert. Der Darsteller nimmt demnach zentral zwischen den Kameras eine normale aufrechte Haltung ein und steht mit dem Gesicht zu K1 gewandt. Sobald die Idle-Posen (je eine für K1 und K2) erfasst werden, ist das System konfiguriert. Eine Neujustierung ist nur im Falle eines Programmneustarts notwendig oder wenn eine der Kinects verschoben wird, da hierdurch die Idle-Koordinaten verfälscht werden.

## 4.2 Versuchsdurchführung

Das Experiment besteht aus drei Versuchen, welche die unterschiedlichen Komponenten des Algorithmus sowohl separat als auch im Zusammenspiel testen sollen.

Zunächst wird die Drehungserkennung des Systems überprüft. Der Akteur stellt sich hierfür mittig in den Aufnahmebereich und dreht sich in einer fließenden Bewegung

jeweils 50 Mal langsam im- und gegen den Uhrzeigersinn. Das Tempo wird hierbei so gewählt, dass sich der Darsteller in ca. zwei Sekunden um 180° dreht. Für einen zweiten Durchlauf werden die Drehungen innerhalb einer halben Sekunde vollzogen. Die Erkennung soll so auch unter erschwerten Bedingungen getestet werden. Tritt eine Fehlerkennung auf, so wird der Versuch unterbrochen, bis die Ausrichtungen des Modells und des Darstellers wieder übereinstimmen. Gezählt werden die korrekt erfassten 180°-Drehungen.

Es folgt die Überprüfung des Lernalgorithmus. Hierfür wird ein neuronales Netz mit verschiedenen Bewegungen trainiert. Im Detail handelt es sich hierbei um einen Schneidersitz, Hinknien und Sitzen mit ausgestreckten Beinen. Für jede der Posen werden 200 Bilder als Trainingsmenge aufgezeichnet, sowie Weitere durch Verrauschen der Originaldaten simuliert. Dabei werden einfach die Winkel von echten Datensätzen um einen zufälligen Wert zwischen $-r$ und $r$ verändert, wobei $r$ ein durch den Nutzer festgelegter Rauschwert ist. In einer vereinfachten Variante wird erst die Erkennung sehr verschiedener Bewegungen geprüft. Der Klassifikator soll zwischen Sitzen mit ausgestreckten Beinen und Knien unterscheiden sowie stehende Posen ignorieren. Hierzu werden 300 zusätzliche Trainingsbilder generiert. Die Korrekturschrittweite beträgt 3.3 und $r$ ist gleich 0.01. Nachfolgend wird der Schwierigkeitsgrad erhöht. Es gilt nun, Knien und den Schneidersitz gegeneinander abzugrenzen (zusätzliche Trainingsbilder: 800; Schrittweite: 3.0; $r$: 0.01). Für repräsentative Ergebnisse wird jede Haltung 100 Mal klassifiziert.

Der abschließende Versuch soll offenlegen, wie gut der Algorithmus im Ganzen mit einer Bewegungsfolge zurechtkommt. Hierzu führt der Darsteller zehn Mal für jeweils 15 Sekunden unterschiedliche Posen aus, ändert seine Blickrichtung und wechselt mehrmals zwischen Stand- und Bodenbewegung. Zu diesem Zweck wird ein neuronales Netz zur Unterscheidung von Knien und Hocken trainiert. Neben der automatischen Nachbearbeitung wird die Szene noch, sofern dies erforderlich ist, per Hand nachkorrigiert. Ebenso wird der zeitliche Aufwand jedes Durchlaufes gemessen. Eine Gegenüberstellung der unbehandelten und fertigen Animation soll Aufschluss darüber geben, inwiefern der Posen-Klassifikator die Qualität des Ergebnisses beeinflusst und wo eventuelle Schwächen des Algorithmus liegen.

## 4.3 Ergebnisse

Im Folgenden werden alle Resultate und Beobachtungen der Versuchsreihe dokumentiert. Als Erstes werden die Ergebnisse des Dreherkennungstests aufgeführt, gefolgt von denen des Posen-Klassifikators. Beide Versuche stützen sich vorwiegend auf Messdaten. Demgegenüber stehen die Ergebnisse des Gesamttests. Hierbei wird vor allem auf Beobachtungen eingegangen. Die Auswertung der Ergebnisse erfolgt in Kapitel 4.4.

### 4.3.1 Drehungstest

Im ersten Versuch wurde der Algorithmus zur Erkennung von Ganzkörperrotationen (siehe Abschnitt 3.2.3) einem Test unterzogen. In der nachfolgenden Tabelle wird aufgeführt, für welche Drehrichtung bei welcher Geschwindigkeit welche Resultate erzielt wurden (*erfasste 180°-Rotationen / durchgeführte Rotationen*).

| | Langsam | Schnell |
|---|---|---|
| Im Uhrzeigersinn | 43/50 | 49/50 |
| Gegen den Uhrzeigersinn | 44/50 | 48/50 |

**Tab. 1 Ergebnisse: Dreherkennung**

Auffällig ist die vergleichsweise niedrige Erkennungsrate der langsamen Drehungen in Relation zur schnellen Ausführung. Entgegen der Vermutung aus dem vorigen Abschnitt, wonach die schnellen Drehungen eine zusätzliche Schwierigkeit darstellen sollten, wurden hier die besseren Ergebnisse erzielt. So steigerte sich die Erkennungsrate bei Drehungen im Uhrzeigersinn von 86% (langsam) auf 98% (schnell). Auch bei Drehungen im Uhrzeigersinn gab es einen Anstieg um 8% (von 88% auf 96%).

### 4.3.2 Posen-Erkennung

Im zweiten Test wurde die Funktionstüchtigkeit des Posen-Klassifikators überprüft. Hierzu wurden zwei Versuchsdurchläufe unterschiedlichen Schwierigkeitsgrades ausgeführt. In den Tabellen 2 und 3 werden die Resultate beider Testreihen zusammengefasst. Die Zeilen zeigen die ausgeführten Posen, während die Spalten die dazugehörigen Klassifikationsergebnisse enthalten.

| | Sitzen (a. B.) | Knien | Kein Ergebnis |
|---|---|---|---|
| Sitzen (ausgestreckte Beine) | 100 | 0 | 0 |
| Knien | 0 | 95 | 5 |
| Stehen | 0 | 0 | 100 |

**Tab. 2 Ergebnisse: Posen-Erkennung – Versuch 1**

Die dritte Spalte trägt den Titel „Kein Ergebnis“. Grund: Der Posen-Klassifikator wird nur mit bodennahen Bewegungen trainiert. Entsprechend erkennt der Klassifikator Stehen nicht als Stehen, sondern erkennt lediglich keine Bodenpose.

Gemäß der Annahme, dass es sich hierbei um sehr verschiedene und somit gut unterscheidbare Bewegungen handelt, wurden im ersten Test relativ hohe Erkennungsraten erreicht. So wurde beim Sitzen mit ausgestreckten Beinen eine Rate von 100% erzielt. Auch wurde keine der Standposen fälschlicherweise einer Bodenbewegung zugeordnet. Lediglich jedes zwanzigste Knien wurde nicht als solches erkannt. Hierbei sei zu erwähnen, dass in keinem der fünf Fälle die andere Bodenbewegung angenommen wurde. Somit ergibt sich für den ersten Durchlauf eine Gesamterkennungsrate von $98,\overline{3}\%$.

Demgegenüber steht der zweite Testlauf, bei welchem es zwei Posen mit angezogenen Beinen zu unterscheiden galt. Der erwartet höhere Schwierigkeitsgrad spiegelt sich in den Versuchsergebnissen wider.

| | Knien | Schneidersitz | Kein Ergebnis |
|---|---|---|---|
| Knien | 77 | 23 | 0 |
| Schneidersitz | 11 | 89 | 0 |

**Tab. 3 Ergebnisse: Posen-Erkennung – Versuch 2**

Knapp ein Viertel der Knien-Testdaten wurde fälschlicherweise als Schneidersitz interpretiert. Der Schneidersitz konnte hingegen in 89 von 100 Fällen korrekt klassifiziert werden. Im Vergleich zum vorigen Durchlauf fällt auf, dass das neuronale Netz jede Eingabe einer Pose zugeordnet hat. Jede Fehlerkennung entfällt auf die jeweils andere Bodenbewegung. Die Gesamterkennungsrate beträgt hier 83%.

### 4.3.3 Komplettversuch

Zuletzt wurde die Tauglichkeit der MoCap-Pipeline als Gesamtfunktion untersucht. Es wurde eine 15 Sekunden lange Animation aufgezeichnet. Im Detail wechselte der Akteur hierbei von einer normalen Standpose, das Gesicht zu K1 gewandt, in ein Knien, stand wieder auf, drehte sich zügig gegen den Uhrzeigersinn zu K2, dann zurück zu K1, gefolgt von einer Hocke. Tabelle 4 veranschaulicht für jede der zehn Aufnahmen, welche Aspekte der Animation nachträgliche per-Hand-Korrekturen erforderten und wie viel Zeit (inklusive Materialsichtung) diese in Anspruch nahmen. Die Zeile „Sonstiges“ umfasst dabei alle nicht aufgeführten Probleme, wie etwa Zittern oder Verdreher.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Knien | | | | | X | X | | X | | |
| Drehung zu K2 | | X | | | | | | | | |
| Drehung zu K1 | | | | | | | X | | | |
| Hocke | | | | | | | | | | |
| Sonstiges | X | | | | | | X | | X | X |
| Zeit (min.) | 1:15 | 1:25 | 0:29 | 0:24 | 1:16 | 1:29 | 1:35 | 1:16 | 1:26 | 0:54 |

**Tab. 4 Ergebnisse: Komplettversuch**

Das Hinknien zu Beginn der Animation stellte sich als größte Fehlerquelle des Versuches heraus. In drei von zehn Fällen wurde die Bewegung fälschlicherweise als Hocke interpretiert, was mit einem durchschnittlichen Nachbearbeitungsaufwand von ~1:20 Minuten einherging. Die Hocke konnte hingegen in allen Aufnahmen korrekt klassifiziert werden. Bei den Bodenbewegungen wurden somit Erkennungsraten von 70%, beziehungsweise 100% erreicht. Die Gesamtrate von 85% spiegelt somit in etwa die Ergebnisse des Knien-Schneidersitz-Versuches wider. Fehler in der Dreherfassung zeigten sich in den Animationen zwei und sieben. In beiden Durchläufen wurde je eine der Umdrehungen nicht erkannt. Im Mittel entstand für die Aufnahmen ein Bearbeitungsaufwand von 1:30 Minuten. Hierbei sei jedoch zu berücksichtigen, dass die 1:35 Minuten Nachbearbeitung für Aufnahme sieben die Korrektur zweier Aspekte umfasst (Drehfehler und Sonstiges) (*). Die reine Korrektur des Drehungs-

fehlers liegt demzufolge unter 1:35 Minuten. Am häufigsten traten hingegen die „sonstigen Probleme“ auf. Mit durchschnittlichen 77,5 Sekunden (unter Beachtung von (*)) wurden diese aber auch am schnellsten ausgebessert. So trat in der ersten Animation ein Verdreher des linken Unterschenkels auf, während in den anderen Animationen jeweils kurzzeitiges Zucken entfernt werden musste.

In den Versuchen drei und vier traten hingegen keine Fehler auf. Hier war lediglich eine Sichtung der Animation erforderlich. Insgesamt ließ sich somit in allen Durchläufen die gewünschte Animation extrahieren. Hierfür waren im Mittel 68,9 Sekunden Nachbearbeitungszeit nötig.

Die oben beschriebenen Ergebnisse sollen im nachfolgenden Abschnitt diskutiert und einer qualitativen Analyse unterzogen werden. Ebenso sollen mögliche Ursachen für aufgetretene Fehler gefunden werden.

## 4.4 Auswertung

Das vorangegangene Experiment hat eine Vielzahl von Daten geliefert. Eine Diskussion der Ergebnisse soll nun zeigen, welche Komponenten des Programms ordnungsgemäß funktionieren, in welchen Situationen Fehler beobachtet wurden und wie diese möglicherweise zu erklären sind.

Zunächst wurde die Dreherkennung einem Test unterzogen. Der Versuch hat verdeutlicht, dass der Algorithmus durchaus in der Lage ist, die 360°-Rotationen eines Menschen um seine eigene Achse nachzuvollziehen. So wurden insgesamt 92% der durchgeführten Drehungen erfolgreich vom Programm erkannt. Überraschend ist hingegen die Verteilung der Fehler auf die beiden Versuchsdurchläufe. Lediglich drei der 16 nicht erkannten Rotationen entfallen auf den vermeintlich schwierigeren zweiten Testlauf, bei welchem sich der Darsteller schneller umgedreht hat. Eine mögliche Erklärung für diese Diskrepanz findet sich im Abschnitt 3.2.3. Beim Übergang vom einen zum anderen Sensor steht der Nutzer etwa im 90°-Winkel zu beiden Kameras, wie in Abbildung 17 veranschaulicht. Er bewegt sich somit in einem Grenzwertbereich, für welchen die Bewegungserfassung von Kinect nicht konzipiert wurde [2].

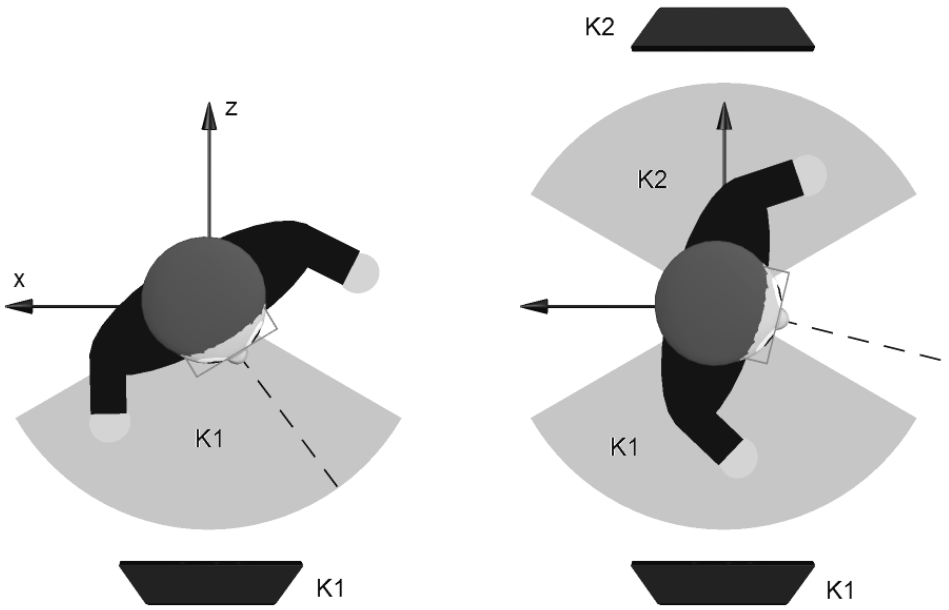


**Abb. 17 Stabiler Erkennungsbereich (e. Abb.)**

Die grau markierten Bereiche spiegeln für den jeweiligen Sensor in etwa den Winkelbereich wider, innerhalb dessen der Darsteller am stabilsten und genauesten von Kinect erfasst werden kann. In der zweiten Bildhälfte bewegt sich der Akteur (seine y-Rotation) außerhalb beider stabilen Erkennungsbereiche.

Während der Erzeugung eines gemeinsamen Datensatzes fiel auf, dass gerade zu diesem Zeitpunkt mit unsauberen Skelettmodellen zu rechnen ist, etwa in Form von stark zuckenden sowie falsch platzierten Körperteilkoordinaten (siehe: Abbildung 18), die Schultern eingeschlossen.

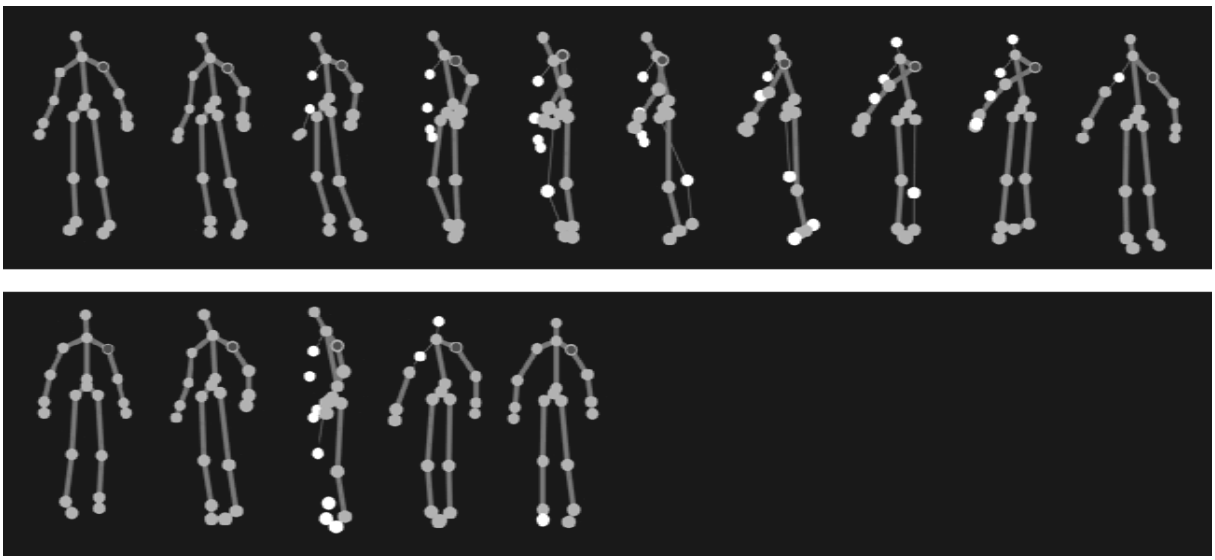

**Abb. 18 Skeletal Tracking-Drehungen (e. Abb.)**

oben: langsame Drehung

unten: schnelle Drehung → weniger Bilder als oben

Dies birgt ein Fehlerpotential, da die für die Dreherkennung relevanten Koordinaten nicht immer exakt bestimmt werden können. Die langsamen Drehungen begünstigen diesen Fehler, da einfach mehr Datensätze in den besagten Grenzwertbereich fallen.

Der Posenerkennungs-Versuch lieferte erwartungsgemäße Ergebnisse. Während im ersten Versuchsteil nahezu alle Bewegungen korrekt verarbeitet wurden, stellte der zweite Teil eine größere Herausforderung dar. Ein Blick auf die Trainingsdaten (ohne Rauschen) der beiden Testläufe soll Aufschluss darüber geben, wie die Differenzen in den Erkennungsraten zu erklären sind. Dargestellt werden jeweils die x- und z-Rotationsdaten in Grad für die vier Körperelemente, mit welchen das neuronale Netz trainiert wurde. In Abbildung 19 werden die Datensätze für Knien (Punkte) und Sitzen mit ausgestreckten Beinen (Sternchen) gegenübergestellt.

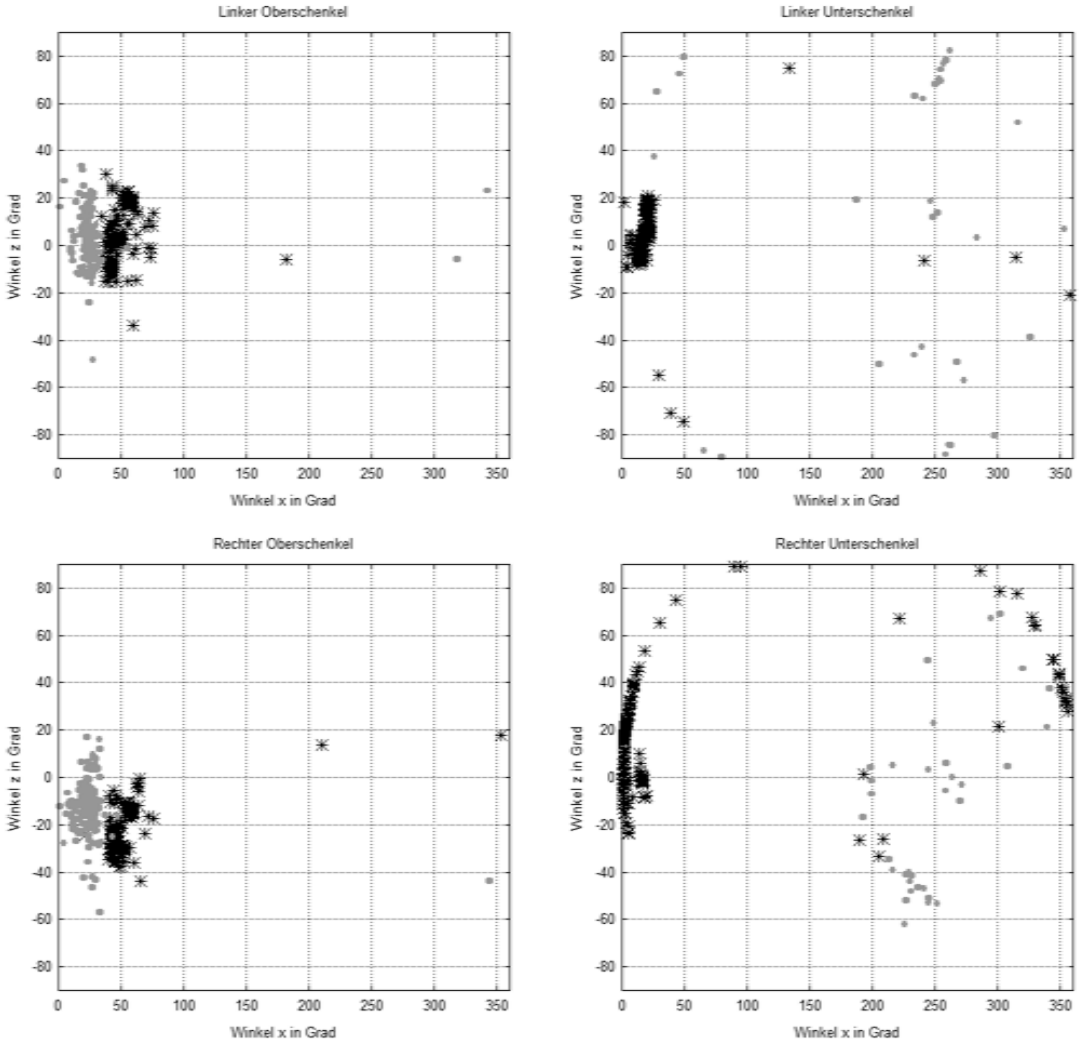


**Abb. 19 Trainingsdaten: Posen-Erkennung 1 (e. Abb.)**

Ein Vergleich der beiden Bewegungsklassen in den Oberschenkelwinkeln zeigt zwar, dass sich die Rotationswerte prinzipiell ähneln, trotzdem ergeben sich jeweils zwei relativ gut trennbare Punktwolken. Hier stellt die x-Komponente den wesentlichen Unterschied dar. So liegen die Oberschenkel-Durchschnittsrotationswerte für die x-Achse bei 50.7182° und 55.0842° (Sitzen mit ausgestreckten Beinen) gegenüber 26.7743° und 26.0639° (Knien). Für eine bessere Unterscheidbarkeit sorgen dagegen die Unterschenkel. Beim Knien sind diese tendenziell verdeckt, weshalb der Algorithmus hier augenscheinlich sehr willkürliche Winkel liefert. Für die andere Bewegungsklasse ergeben sich hingegen sehr charakteristische Werte. Hier seien insbesondere die mittleren x-Rotationswerte der beiden Unterschenkel hervorzuheben. Diese grenzen sich mit Durchschnittswerten von 33.2888° und 76.1060° deutlich gegen die Winkel fürs Knien ab (263.0105° und 287.8425°). Im nächsten Diagramm wird der zweite Testlauf behandelt. Die Knien-Trainingsdaten werden weiter als Punkte veranschaulicht, während die Sternchen den Schneidersitz repräsentieren.

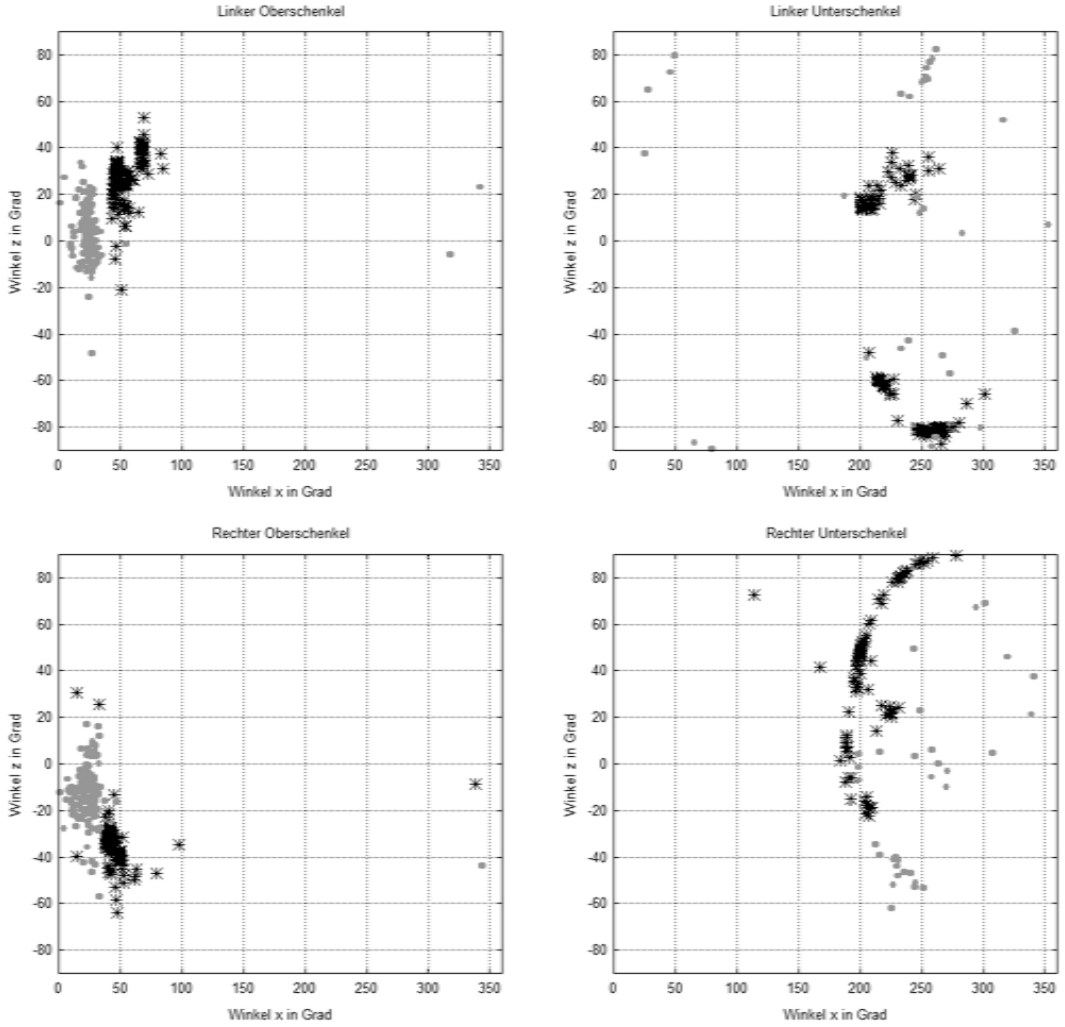


**Abb. 20 Trainingsdaten: Posen-Erkennung 2 (e. Abb.)**

Für die Oberschenkel ergeben sich erneut je zwei gut unterscheidbare Punktwolken. Gerade die Winkel des linken Oberschenkels trennen die beiden Bewegungsklassen gut voneinander ab. Dennoch liegt die Erkennungsrate des zweiten Versuchsteils deutlich unter der des ersten Versuchs. Die Ursache hierfür liegt womöglich in den Trainingsdaten der Unterschenkel, in welchen sich die beiden Bewegungsklassen im Schnitt stärker ähneln als im ersten Durchlauf. So liegen die mittleren x-Rotationswerte für den Schneidersitz (Unterschenkel) bei 235.5086°, bzw. 204.5975° gegenüber 263.0105° und 287.8425° fürs Knien. Die größere Ähnlichkeit der beiden Bewegungsklassen spiegelt sich auch in der nachfolgenden Tabelle wider. Gezeigt werden für die drei Bewegungsklassen die durchschnittlichen Rotationswinkel der Ober- und Unterschenkel aus den Trainingsdaten in Grad.

| | LO x | LO z | LU x | LU z | RO x | RO z | RU x | RU z |
|---|---|---|---|---|---|---|---|---|
| Ausgestreckte Beine | 50.7182 | 4.4060 | 33.2888 | 9.2876 | 55.0842 | -21.2625 | 76.1060 | 16.3350 |
| Knien | 26.7743 | 3.2716 | 263.0105 | -15.4641 | 26.0639 | -11.8029 | 287.8425 | 26.0753 |
| Schneidersitz | 53.3939 | 27.7827 | 235.5086 | -43.2411 | 50.3573 | -36.5490 | 204.5975 | 33.2717 |

**Tab. 5 Trainingsdatenvergleich der Bewegungsklassen**

L = Linker; R = Rechter; O = Oberschenkel; U = Unterschenkel

Im Vergleich unterscheiden sich die mittleren Rotationswinkel beim Sitzen mit ausgestreckten Beinen von denen des Kniens um durchschnittlich 67.4371°. Beim Schneidersitz-Knien-Vergleich sind es hingegen nur 30.7392°.

Abschließend folgt die Auswertung des Komplettversuchs. Hierbei werden zunächst die unbehandelten- den teils automatisch nachbearbeiteten Animationen in einer qualitativen Analyse gegenübergestellt. Im direkten Vergleich beider Sequenzen innerhalb derselben Aufnahme bringt die Nachbearbeitung deutliche Verbesserungen mit sich. Vereinzelt konnten zitternde Gliedmaßen nicht vom Glättungsfilter bereinigt werden. Durch die manuelle Korrektur erscheint die Animation in den entsprechenden Momenten wesentlich ruhiger im Kontrast zur unfertigen Aufnahme. Ein klarer Unterschied zeigt sich auch bei den Drehungen. In den Testläufen zwei und sieben wurde jeweils eine der Drehungen nicht erkannt. Hier spiegelt die nicht korrigierte Sequenz schlicht und einfach nicht den durchgeführten Bewegungsablauf vollständig

wider. Den größten Gegensatz stellen jedoch die Bodenposen dar. Erwartungsgemäß enthält das Rohmaterial an diesen Stellen zahlreiche Erkennungsfehler. Durch die teils stark umherspringenden Ober- und Unterschenkelwinkel ist die eigentliche Beinbewegung nicht mehr erkennbar und die Animation verliert viel an natürlichem Eindruck. Durch die Nachbearbeitung nimmt die 3D-Figur nun eine passende Ober- und Unterschenkel-Haltung ein. Der Kontrast zwischen zwei Sequenzen wird exemplarisch in Abbildung 21 dargestellt.

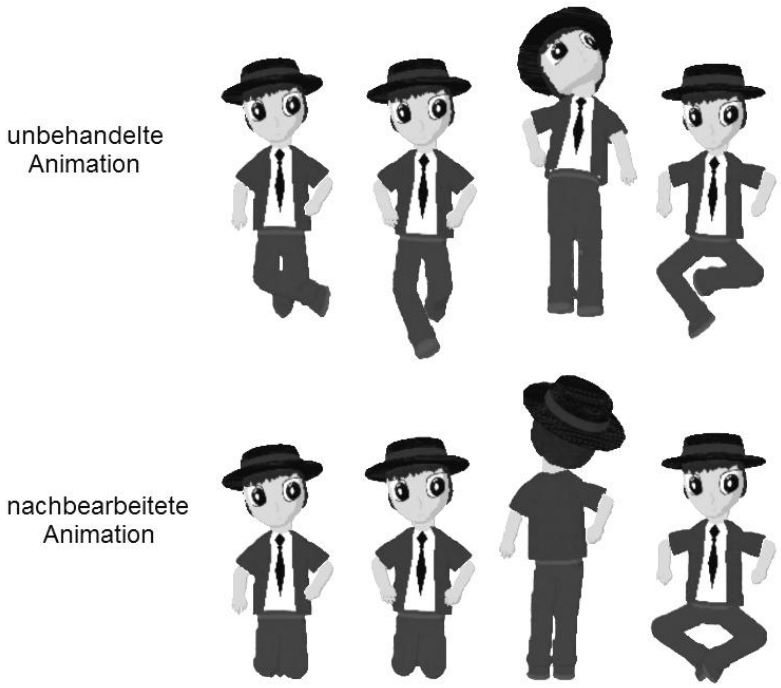


**Abb. 21 Animationsvergleich: Unbehandelt – Nachbearbeitet (e. Abb.)**

Die fertigen Sequenzen geben den Bewegungsablauf des Darstellers augenscheinlich korrekt wieder. Der Übergangsbereich zwischen den Kinects stellt jedoch eine Schwachstelle dar. Die bereits zuvor erwähnten unsauberen Skelettmodelle, welche in diesem Grenzwertbereich erfasst werden, gehen mit Erkennungsschwierigkeiten einher. So werden manche Bewegungen im Übergangsbereich nicht richtig erfasst oder das Modell zittert so stark, dass die Bewegung vom Glättungsfilter verschluckt wird. Entsprechend groß gestaltet sich der Nachbearbeitungsaufwand, sollten bei einer Drehung die besagten Probleme auftreten.

Die messbaren Ergebnisse des Komplettversuches geben im Grunde die Resultate der vorangegangenen Versuche wieder. So wurden neun von zehn Drehungen korrekt erfasst, was in etwa den Erkennungsraten aus dem ersten Test entspricht.

Ebenso konnte der Posen-Klassifikator 85% der Bodenbewegungen automatisch korrigieren. Dies deckt sich zwar grundsätzlich mit den Ergebnissen des Knien-Schneidersitz-Versuches, ist jedoch insofern überraschend, dass der Fehler in der Bewegung kleiner sein sollte. Der Grund für diese Annahme ist, dass die Bodenposen per Mehrheitsentscheid klassifiziert werden. Liegt die Erkennungsrate für einen Schneidersitz beispielsweise bei 80%, so liegt das neuronale Netz in jedem fünften Fall falsch. Werden jedoch 40 von 50 Bodenbewegungsmomenten als Schneidersitz interpretiert, so wird die Pose innerhalb der Animation dennoch korrekt klassifiziert.

Trotz allem wurde in drei Testläufen das Knien als Hocke erkannt. Eine mögliche Erklärung ist die Dauer der Bodenpose in Kombination mit dem Übergang von der Standpose zum Hinknien. Sowohl beim Hinsetzen als auch beim Aufstehen unterhalb des y-Schwellwertes (siehe Abschnitt 3.5.1) ähnelt die Körperhaltung tendenziell eher einer Hocke als einem Knien. Hinzu kommt, dass der Darsteller relativ schnell wieder aufstand. Dies lässt die Vermutung zu, dass der prozentuale Anteil des Hinsetzens, bzw. Aufstehens an der Gesamtdauer der Bodenbewegung so groß war, dass mehrheitlich eine Hocke angenommen wurde (siehe: Abbildung 22).

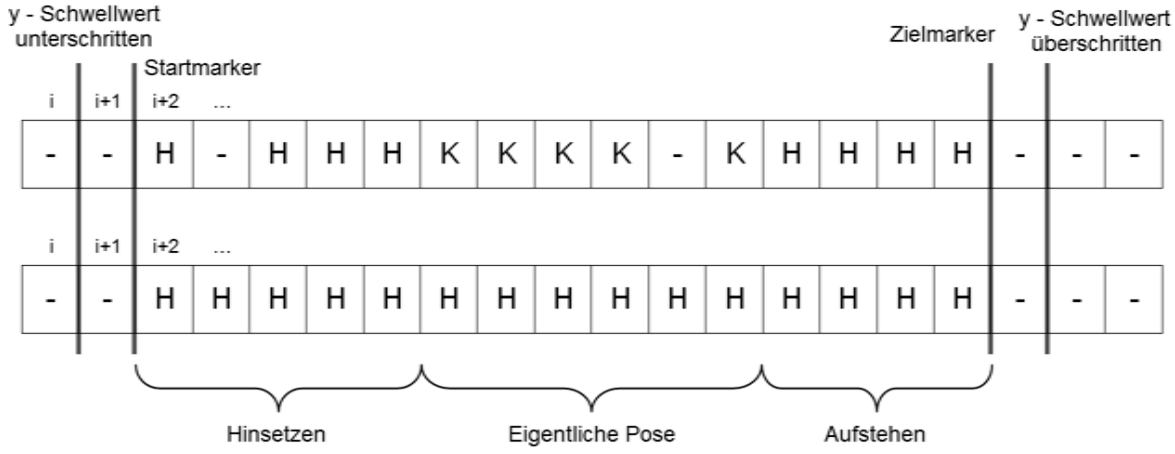


**Abb. 22 Beispiel: Interpretationsergebnisse (e. Abb.)**

H = Hocke; K = Knien; - = Kein Ergebnis

Trotz einer erforderlichen manuellen Nachbearbeitung hat der Versuch gezeigt, dass es mit dem Algorithmus möglich ist, aus dem Bewegungsablauf eines Akteurs innerhalb von kurzer Zeit eine Motion Capture-Animation zu extrahieren.

# 5. Fazit

Die Einleitung sowie das erste Kapitel dienten als Einführung in die Thematik und sollten dem Leser das Ziel dieser Arbeit vermitteln. Zunächst wurde das Motion Capturing an sich behandelt. Hierbei wurde auf die Bedeutung des Verfahrens hingewiesen und der Leser lernte mit der optischen Umsetzung einen bekannten Vertreter der MoCap-Systeme kennen. Das folgende Kapitel befasste sich sowohl mit Microsofts Kinect-Sensor und seiner Funktionsweise als auch mit der zu lösenden Ausgangsproblematik dieser Arbeit. Ebenso wurden verwandte Arbeiten vorgestellt, welche sich mit besagten oder ähnlichen Schwierigkeiten auseinandergesetzt haben.

Das darauffolgende Kapitel thematisierte den für diese Arbeit entwickelten MoCap-Algorithmus in seinen einzelnen Verarbeitungsschritten. Die Eingaben zweier Kinects werden in einen gemeinsamen Raum transformiert, zur 360°-Erkennung einer Analyse unterzogen und zu einem gemeinsamen Modell kombiniert. Nach der Extraktion der animationsrelevanten Rotationswinkel wurden die neuronalen Netze und der Backpropagation-Algorithmus als Basis für die Bodenbewegungserkennung erläutert. Es folgte ein Abschnitt über die Animationsaufzeichnung und die Nachbearbeitung mithilfe des zuvor beschriebenen Klassifikators.

Zuletzt sollte in einem Experiment die Funktionstüchtigkeit des Algorithmus überprüft werden. Dem Leser wurden der Versuchsaufbau, die eigentliche Durchführung sowie die dazugehörigen Ergebnisse präsentiert. Weiter wurden mögliche Erklärungen für die gemachten Beobachtungen gesucht und diskutiert.

Das Experiment hat gezeigt, dass der Algorithmus grundsätzlich eine Verbesserung für die genannten Herausforderungen darstellt. Durch den Einsatz einer zweiten Kinect stehen dem Programm, im Vergleich zu einem einzelnen Sensor, zusätzliche Bewegungsinformationen zur Verfügung, wodurch insbesondere zuvor verdeckte Bewegungen häufiger erfasst werden können. Weiter kann der Algorithmus Umdrehungen des Nutzers als solche erkennen und das Modell entsprechend drehen. Hier bieten sich aber noch verschiedene Verbesserungsmöglichkeiten an, welche im nächsten Abschnitt näher beschrieben werden. Durch den Bodenbewegungsklassifikator können zudem einige fehlerhafte Beinrotationswinkel korrigiert werden. Erwar-

tungsgemäß werden hier für gut unterscheidbare Bodenposen höhere Erkennungsraten erreicht, als es bei vergleichsweise untereinander ähnlichen Bewegungen der Fall ist.

## 5.1 Ausblick

Die Erkennungsraten ließen sich in Zukunft durch verschiedene Ansätze weiter erhöhen. Im Versuch wurden nicht alle Drehungen des Darstellers korrekt erkannt. Als mögliche Ursache hierfür wurden unsaubere Datensätze aufgeführt. Diese traten insbesondere dann auf, wenn der Akteur im ~90°-Winkel zu den Blickachsen der beiden Sensoren stand, wie es bei einer Umdrehung zwangsläufig zeitweise der Fall ist. Durch weitere Kinects ließe sich diesem Problem entgegenwirken, da bei entsprechender Sensorkonfiguration die besagten Grenzwertbereiche geschlossen werden würden, wie in Abbildung 23 veranschaulicht.

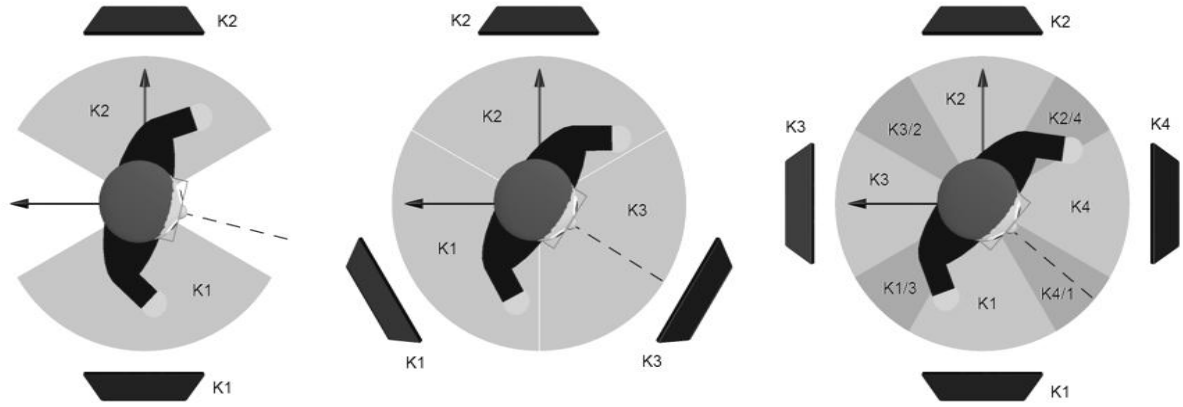


**Abb. 23 Stabiler Erkennungsbereich mit weiteren Sensoren (e. Abb.)**

Der Darsteller würde folglich zu jedem Zeitpunkt der Aufnahme von mindestens einem Sensor stabil erfasst werden. Jedoch steigen mit zunehmender Sensoranzahl natürlich auch die Anforderungen an die Hardware. Alternativ könnte die Zwei-Kinect-Lösung dieser Arbeit um zusätzliche Erkennungsparameter erweitert werden. So könnten etwa, ergänzend zum Positionsvergleich der beiden Schultern, auch die Koordinaten der Hände, Ellbogen, Knie, Füße und Hüften in die Berechnung mit einbezogen werden. Aber auch der Bodenbewegungsklassifikator könnte von dem Einsatz weiterer Sensoren profitieren. Durch die zusätzlichen Perspektiven ergeben sich ggf. deutlichere Charakteristiken für die unterschiedlichen Posen, sodass diese bes-

ser zu unterscheiden sind. Ebenso könnten weitere bewegungsspezifische Parameter in die Trainingsmenge des neuronalen Netzes aufgenommen werden. So liegt etwa die y-Achsen-Koordinate des Körpers bei einem Schneidersitz tendenziell tiefer als bei einem Hinknien oder der Neigungswinkel des Rückens ist bei einigen Bewegungen aufrechter als bei anderen, usw. Der Interpretationsschritt (3.5.1) ließe sich ggf. durch den Einsatz von Gewichten verbessern, z.B. durch eine stärkere Gewichtung jener Klassifikationsergebnisse, welche zeitlich in der Mitte der Bodenbewegung liegen. Mögliche Fehlerkennungen während des Hinsetzens und Aufstehens würden folglich weniger stark in die Interpretation einfließen.

Desweiteren ließe sich der Umfang des Programms im Allgemeinen vergrößern. Wie im Abschnitt 2.1 erwähnt, bringt das Kinect SDK v1.7 die Funktionalität der Gesichtserfassung mit sich. So können aus dem Kamerabild etwa Mund- und Augenbrauenbewegungen des Nutzers nachvollzogen werden. Dies bietet die Grundlage für eine Performance Capturing-Erweiterung, welche es dem Anwender ermöglicht, auch die Mimik des virtuellen Darstellers realitätsnahe zu animieren. Konkreter könnte mit den erfassten Daten das 3D-Modell des Gesichtes verformt oder die Gesichtstextur angepasst werden.

# Literaturverzeichnis

[1] (kein Datum). Abgerufen am 6. Mai 2015 von Imagine Games Network: http://www.ign.com/games/kinect/xbox-360-14357198

[2] (kein Datum). Abgerufen am 25. Mai 2015 von MSDN: https://msdn.microsoft.com/en-us/library/hh973074.aspx

[3] (31. März 2010). Abgerufen am 7. Mai 2015 von Microsoft: http://news.microsoft.com/2010/03/31/primesense-supplies-3-d-sensing-technology-to-project-natal-for-xbox-360/

[4] (18. März 2015). Abgerufen am 16. Juni 2015 von Wikipedia: https://en.wikipedia.org/wiki/Facial_motion_capture

[5] (2015). Abgerufen am 24. Juni 2015 von Microsoft: https://msdn.microsoft.com/en-us/library/nuiskeleton.nui_skeleton_position_tracking_state.aspx

[6] (2015). Abgerufen am 24. Juni 2015 von Microsoft: https://msdn.microsoft.com/en-us/library/nuiskeleton.nui_skeleton_tracking_state.aspx

[7] Asteriadis, S., Chatzitofis, A., Zarpalas, D., Alexiadis, D. S., & Daras, P. (2013). *Estimating human motion from multiple Kinect Sensors* .

[8] Eisler, C. (9. November 2011). Abgerufen am 13. Mai 2015 von MSDN: http://blogs.msdn.com/b/kinectforwindows/archive/2011/11/09/the-kinect-effect.aspx

[9] Härtel, S. E. (2011). *Motion Capture vs. Animation: in theoretischer Gegenüberstellung und praktischer Anwendung.* VDM Verlag Dr. Müller.

[10] Kawatsu, C., Li, J., & Chung, C. (25. April 2012). *Development of a Fall Detection System with Microsoft Kinect* . Southfield, Michigan, USA: Lawrence Technological University.

[11] Menache, A. (1999). *Understanding Motion Capture for Computer Animation and Video Games.* Morgan Kaufmann.

[12] Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., et al. (Oktober 2011). *KinectFusion: Real-Time Dense Surface Mapping and Tracking* . IEEE.

[13] Rojas, R. (1996). *Neural Networks.* Berlin: Springer-Verlag.

[14] Roman, J.-P. (2010). Bildverarbeitung in der Medizin. *Digitalkameras etablieren sich in immer mehr medizinischen Applikationen* . Weinheim: Wiley-VCH Verlag GmbH & Co. KGaA.

[15] Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., et al. (Juni 2011). *Real-Time Human Pose Recognition in Parts from Single Depth Images* . IEEE.

[16] Sirignano, C. (7. Februar 2014). Abgerufen am 25. Mai 2015 von MSDN: https://social.msdn.microsoft.com/Forums/en-US/02d2bd26-9685-4bb5-8d30-bb25d530e3e0/kinect-not-detecting-while-lying-on-floor?forum=kinectsdk

[17] Zeng, W. (2012). Multimedia at Work. *Microsoft Kinect Sensor and Its Effect* . IEEE Computer Society.