Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Nabil Toumi

Modellierung eines Prototyps für die Echtzeit-Ergonomie zur Ableitung eindeutiger Bewegungsmuster aus körpernahen Sensordaten am Beispiel von Hand-Arm-Bewegungen.

Fakultät Technik und Informatik
Department Maschinenbau und Produktion

Faculty of Engineering and Computer Science
Department of Mechanical Engineering and Production Management

Nabil Toumi

Modellierung eines Prototyps für die Echtzeit-Ergonomie zur Ableitung eindeutiger Bewegungsmuster aus körpernahen Sensordaten am Beispiel von Hand-Arm-Bewegungen.

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Produktionstechnik und -management
am Department Maschinenbau und Produktion
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

in Zusammenarbeit mit:

der HAW Hamburg
Berliner Tor 21
20099 Hamburg

Erstprüfer: Prof. Dr.-Ing. Henner Gärtner
Zweitprüfer: Prof. Dr. Patrick Lorer

Abgabedatum: 29.12.2021

Zusammenfassung

Nabil Toumi

Thema der Masterarbeit

Modellierung eines Prototyps für die Echtzeit-Ergonomie zur Ableitung eindeutiger Bewegungsmuster aus körpernahen Sensordaten am Beispiel von Hand-Arm-Bewegungen.

Stichworte

Echtzeit-Ergonomie, Bewertungsmodell, Ansatzpunkte, Ergonomische Bewertungsverfahren, Einflussfaktoren, Screening-Methoden, Erholzeit

Kurzzusammenfassung

Diese Masterarbeit ist ein Teil eines Projektes, bei dem Mit Hilfe von in Bekleidung eingebrachten Sensoren, Körperbewegungen aufgenommen und die daraus resultierenden Belastungen bewertet werden sollen, mit dem Ziel bei drohender Überbelastung dem Träger ein direktes Feedback geben zu können.
In diesem Teil des Projektes sollen aus Sensordaten, welche durch eingebrachte Sensoren auf einem Oberbekleidungsstück aufgenommen werden, eindeutige Bewegungsmuster erkannt werden. Hierzu wird ein Versuchsaufbau entwickelt, um am Beispiel von Hand-Arm-Bewegungen (Überkopfarbeit) Sensordaten, sogenannte Trainingsdaten, aufzunehmen. Diese Trainingsdaten sollen Mithilfe von Machine-Learning-Algorithmen analysiert und interpretiert werden. Hierzu muss ein geeigneter Machine-Learning-Algorythmus ausgewählt und neben der Aufnahme der Trainingsdaten zudem auch die Echtzeit-Daten der Bewegung (ground truth) erfasst werden. Für die Echtzeit-Bewegungserfassung werden unterschiedliche Systeme betrachtet und eine Auswahl getroffen. Die über den Versuch so aufgenommenen Trainngsdaten und die Echzeit-Daten werden anschließend genutzt um den Machine-Learning-Algorythmus zu trainieren und Bewegungsmuster der Überkopfarbeit zu erkennen.

Nabil Toumi

Master thesis title

Modeling of a prototype for real-time ergonomics to derive unique motion patterns from near-body sensor data using hand-arm movements as an example.

Keywords

Real-time ergonomics, assessment model, starting points, ergonomic assessment methods, influencing factors, screening methods, recovery time.

Abstract

This master thesis is part of a project in which sensors integrated into clothing are used to record body movements and evaluate the resulting loads, with the aim of providing direct feedback to the wearer in the event of imminent overload. In this part of the project, clear movement patterns are to be recognized from sensor data recorded by sensors inserted on an outer garment. For this purpose, an experimental setup will be developed to record sensor data, so-called training data, using the example of hand-arm movements (overhead work). These training data are to be analyzed and interpreted with the help of machine learning algorithms. For this purpose, a suitable machine learning algorithm must be selected and, in addition to recording the training data, the real-time data of the movement (ground truth) must also be recorded. For the real-time motion acquisition, different systems are considered and a selection is made. The training-data recorded during the experiment and the real-time data are then used to train the machine learning algorithm and to recognize movement patterns of the overhead work.

Inhaltsverzeichnis

I. Abblidungsverzeichnis

II. Tabellenverzeichnis

1 Einleitung

In dem fakultätsübergreifenden Projekt „Echtzeit-Ergonomie“ soll eine App entwickelt werden, die Mithilfe von Sensoren Bewegungen erfassen, bewerten und bei drohender Überbelastung warnen kann. Der Hintergrund zu diesem Forschungsvorhaben basiert zum einen auf den fortlaufenden Wandel in der Produktion von Unternehmen, bei dem die Arbeitsprozesse flexibler und die Tätigkeiten variabler werden. Zum anderen hält der Trend an, dass viele Tätigkeiten in selbständigen Arbeitsverhältnissen durchgeführt werden. Dies wird sich durch die zunehmende Digitalisierung in Zukunft noch forcieren. Eine Studie der Bertelsmann Stiftung geht davon aus, dass bis 2030 die Selbstständigkeit die häufigste Beschäftigungsform sein wird [1, S. 44]. Ein weiterer beachtenswerter Aspekt, ist die Verlängerung der Lebensarbeitszeit. Die Rente ab 67 Jahren wurde schon beschlossen und die Diskussion über eine weitere Verschiebung des Renteneintrittsalters ist in vollem Gange. Diese Rahmenbedingungen zeigen auf wie wichtig es für den Einzelnen aber auch für die Gesellschaft ist, die Arbeitskraft zu erhalten und präventiv Gesundheitsschutz zu betreiben. Allerdings wird auch klar, dass die bisherigen meist in Großkonzernen eingesetzten Verfahren zur Analyse von Belastungen in unterschiedlichen Tätigkeiten zukünftig nicht mehr ausreichend sein werden, um einen Großteil der arbeitenden Bevölkerung abzubilden [2].

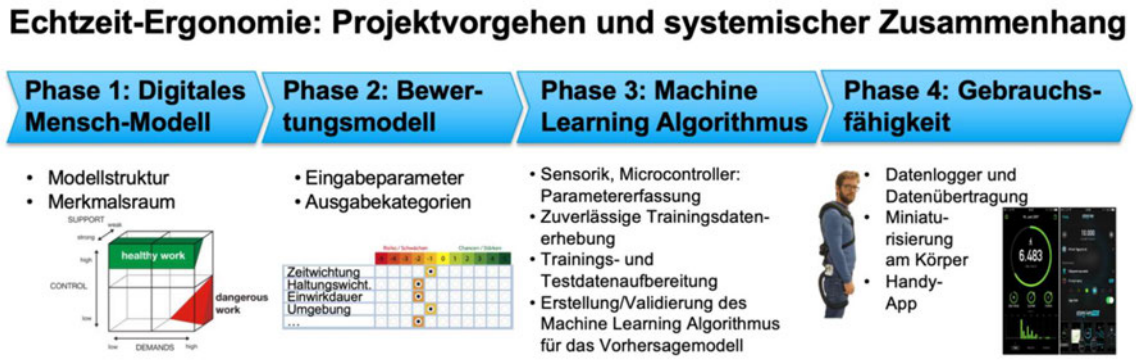


Abbildung 1: Echtzeit-Ergonomie [3]

1.1 Ausgangssituation und Problemstellung

Um zukünftig einem Großteil, in unterschiedlichsten Tätigkeitsfeldern und Vertragsverhältnissen tätigen Menschen, eine Analyse ihrer Belastungen zu ermöglichen, wurde das Projekt „Echtzeit-Ergonomie“ gestartet. Die grundsätzliche Idee besteht darin ein System, bestehend aus Bekleidungsstücken mit eingebrachten Sensoren und einer App für die Analyse und Bewertung, zu entwickeln. Vor Aufnahme der Arbeit kann z.B. ein Shirt und eine Hose mit eingenähter Sensorik angezogen werden und während der Arbeit erfolgt die Belastungsanalyse in Echtzeit. Erkennt das System eine Überlastung wird der Träger gewarnt und Vorschläge zur Regeneration werden übermittelt. Für ein solches System ergeben sich zwei Herausforderungen, zum einen muss eine Echtzeit-Belastungsanalyse entwickelt werden, zum anderen müssen die Sensordaten der Bekleidungsstücke interpretiert werden. Die Bekleidungsstücke werden bei unterschiedlichen Menschen unterschiedlich sitzen und auch bei Bewegungen verrutschen. Trotzdem sollen mit den aufgenommenen Daten klare Bewegungsmuster erkannt werden und das ohne eine aufwendige Kalibrierung vor Aufnahme der Arbeit. Es werden also verrauschte Sensordaten aufgenommen und es muss eine Möglichkeit zur Interpretation der Bewegungsmuster entwickelt werden.

In den zwei vorhergegangenen Phasen des Projektes „Echtzeit-Ergonomie“ wurden, im Rahmen einer Studien- bzw. Bachelorarbeit, digitale Menschen-Modelle und Bewertungsmodelle erforscht. Hierbei wurden insbesondere für den Anwendungsfall Überkopfarbeit, Sensorpositionen ermittelt, um Belastungen aufzunehmen und einen Ansatz zu einem ergonomischen Bewertungsmodell zu erstellen, um diesen dann zu bewerten und Rückmeldungen geben zu können [4] [5].

In der nun dritten Projektphase soll ein Prototyp eines Sensorsystems für die Bewegungserfassung gestaltet werden. Im Rahmen einer Bachelorarbeit wurde zunächst, mit unterschiedlichen Inertialmesssensoren, ein System für Gelenkwinkelmessungen umgesetzt. Ziel war es in Versuchen die Genauigkeit, Stabilität und Reproduzierbarkeit der Ergebnisse der Gelenkswinkelmessungen zu untersuchen [6].

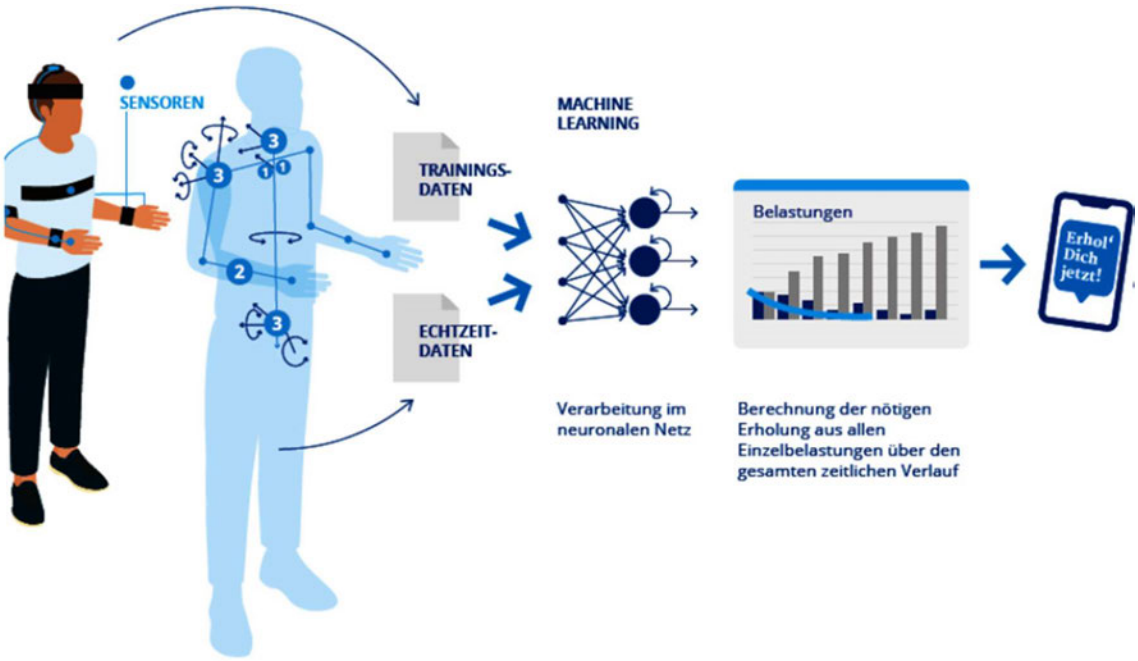


Abbildung 2: Projektphase 3 Echtzeit-Ergonomie [2]

Mit diesen Vorerfahrungen wird ein Prototyp zur Aufnahme von Sensordaten einer Hand-Arm-Bewegung konzipiert. Diese aufgenommenen Sensordaten sollen als Trainingsdaten für einen Machine-Learning-Algorithmus genutzt werden. Um dem Algorithmus das Lernen zu ermöglichen, wird ein weiterer Echtzeitdatensatz zum Abgleich benötigt. Diese beiden Systeme, das Sensorsystem und das Referenzsystem, müssen aufeinander eingestellt werden, damit die jeweiligen Positionsdaten den gleichen Bezug haben. Die Entwicklung eines stabilen Prototypen welcher die Trainingsdaten liefert, die Auswahl eines Referenzsystems und die Kopplung auf eine gleiche Basis, um eine Vergleichbarkeit der Datensätze zu erzielen, ist die Herausforderung dieser Arbeit.

1.2 Motivation und Ziel der Arbeit

Das Ziel der Masterarbeit ist es für ein eindeutiges menschliches Bewegungsmuster Sensordaten zu erfassen und diese mit Echtzeit-Daten abgleichen zu können. Am Beispiel der Hand-Arm-Bewegung Überkopfarbeit soll ein Sensorsystem Sensordaten, sogenannte Trainingsdaten, aufnehmen. Unterschiedliche Systeme der Echtzeit-Bewegungserfassung sollen untersucht und ein geeignetes zur Aufnahme der Echtzeit-Datensätze ausgewählt

werden. Zur Erhebung der Daten wird ein Versuchsaufbau definiert und zunächst für das Bewegungsmuster Überkopfarbeit ausgerichtet. Die Rahmenbedingungen für die Versuchsdurchführung, wie z.B. definierter Bewegungsablauf, Einstellkriterien für die Probanden, Anzahl der Messreihen, werden festgelegt. Die in den Versuchen aufgenommenen Trainings- und Echtzeit-Daten sollen später durch einen Machine-Learning-Algorithmus analysiert und das Bewegungsmuster der Überkopfarbeit erkannt werden. Geeignete Machine-Learning-Algorithmen sollen betrachtet werden.

1.3 Vorgehensweise

Die Masterarbeit beginnt mit der Einleitung in das Thema, hierbei werden die Ausgangssituation und Problemstellung sowie die Motivation und das Ziel der Arbeit erläutert. Im zweiten Kapitel werden die theoretischen Grundlagen zu Motion Capture Systemen und Inertialsensoren erläutert. Diese sind die Basis für das später gewählte Referenzsystem als auch das zu entwickelnde Sensorsystem.

Im dritten Kapitel wird das Sensorsystem betrachtet. Zuerst wird auf die Hardware, die Auswahl der Sensoren, des Microcontrollers und die nötige Verkabelung eingegangen. Dann wird aufgezeigt mit welcher Software das Sensorsystem programmiert und eingestellt wird.

Im vierten Kapitel erfolgt der Vergleich möglicher Referenzsysteme und die Festlegung auf eines. Das folgende Kapitel fünf soll aufzeigen wie die Kopplung der beiden Systeme erfolgt. Zunächst wird dazu die Spiele-Engine Unity vorgestellt über die eine Verbindung gelingt. Anschließen wird das Thema Koordinatentransformation betrachtet, welches eine/die Hauptaufgabe für eine gelungene Umsetzung ist. Die unterschiedlichen Versuche ein Körpermodell zu gestalten, welche Herausforderungen sich ergaben und welche Lösung umgesetzt werden konnte werden daraufhin geschildert und abschließend wird schließlich erläutert, wie und in welcher Form die beiden Positionsdatensätze generiert werden.

Nachdem die Entwicklung und Koppelung der beiden Sensorsysteme gelungen ist, betrachten wir im sechsten und siebten Kapitel einen möglichen Versuchsaufbau zur Erfassung der Trainings- und Echt-Zeitdaten, sowie mögliche Machine-Learning-Algorithmen zur Analyse dieser. Abgeschlossen wird die Masterarbeit mit dem Fazit zum Umgesetzten und einem Ausblick, wie im Forschungsprojekt mit dem Ergebnis weitergearbeitet werden könnte.

2 Theoretische Grundlagen Motion Capture und Inertialsensoren

In diesem Kapitel werden die Grundlagen zum Thema Motion Capture und Inertialsensoren erläutert. Für die Masterarbeit sind diese beiden Aspekte insofern relevant, da es sich bei den zu entwickelnden Systemen, Sensorsystem und Referenzssystem, jeweils um ein Motion Capture System handelt und die Inertialsensoren liefern die jeweilige Technik zur Aufnahme der Bewegungsdaten. Dabei steht der Oberbegriff „Motion Capture“ im Allgemeinen für das Erfassen von Bewegungsdaten. Hierbei werden verschiedenste einsetzbare Technologien integriert. Auf die Technologien zur Aufnahme von Bewegungsdaten mithilfe von Inertialsensoren, die in dem angestrebten System eingesetzt werden soll, wird sich fokussiert.

2.1 Historie von Motion Capture

Die Idee Bewegungsabläufe aufzunehmen und zu analysieren begann mit der Fragestellung: „Ob ein galoppierendes Pferd immer mindestens einen Huf am Boden hat?“. Der Unternehmer Leland Stanford beauftragte den Fotografen Eaweard Muybridge im Jahre 1872 diese Frage zu beantworten. Dies gelang Muybridge indem er den Bewegungsablauf eines Pferdes, durch einen Versuchsaufbau mit zwölf Kameras und einer deutlichen Reduzierung der Belichtungszeit, in einer Bilderreihe aufnahm [7, S. 2].

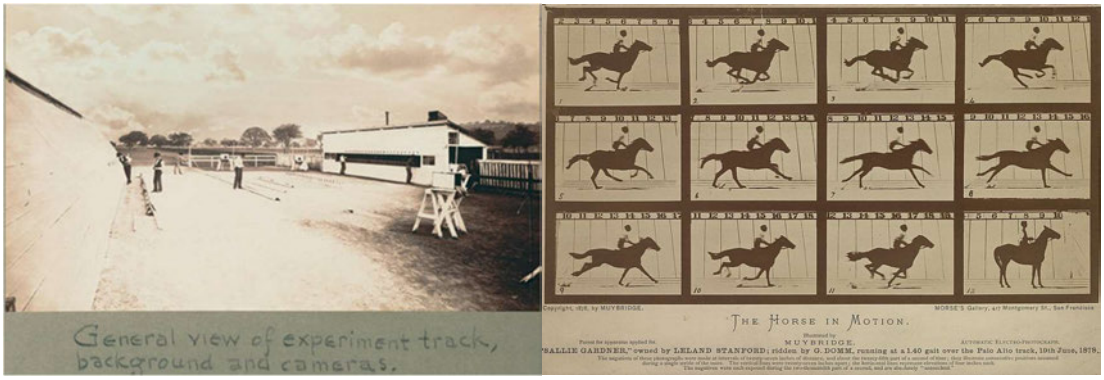


Abbildung 3: Versuchstand zur Aufnahme des Bewegungsablaufes eines Pferdes (links) und die Bilderreihe „Pferd in Bewegung“ (rechts) [8].

In dem späteren Werk „Animal Locomotion“ dokumentierte Muybdrige dann auch die ersten Bewegungsstudien von Menschen.

Der nächste Entwicklungsschritt folgte durch den Trickfilm. Walt Disney wollte 1937 für die Produktion von „Schneewittchen“ die Bewegungen der Zeichentrickfiguren möglichst realistisch umsetzen. Hierzu ließ er alle Szenen durch Schauspieler spielen und filmen. Dieses Filmmaterial wurde anschließend Bild für Bild bearbeitet und die Zeichentrickfiguren auf die jeweilige Bewegung der Schauspieler animiert. Dieses Verfahren nannte sich Rotoscoping und war sozusagen ein zweidimensionales Motion Capture Verfahren [9].

Das erste Motion Capture System wurde in den 1980ern entwickelt. Tom Calvert, Professor an der Simon Fraser University, brachte Potentiometer an Menschen an, um computeranimierte Figuren für choreografische Studien und klinische Bewegungsanalysen umzusetzen. Die analoge Ausgabe der Potentiometer wurde in eine digitale Form umgewandelt und einem Computeranimationssystem zugeführt. Das Animationssystem verwendete die Motion-Capture-Vorrichtung zusammen mit Labanotation, ein System zur Aufzeichnung und Analyse menschlicher Bewegung, um die Bewegungen zu erfassen und auszuwerten [10].

Mit der Weiterentwicklung der Computertechnik in den 1980er Jahren wurden unterschiedliche Motion Capture Systeme auf Basis von optischen, elektromagnetischen und elektromechanischen Bewegungsaufnahmesystemen entwickelt. 1988 gelang dann dem Animationsstudio Pacific Data Images die erste Umsetzung einer Echtzeitanimation. Durch Einsatz eines Exoskelettes gelang es Daten in Echtzeit bereitzustellen und damit zum ersten Mal eine Roboterpuppe zu steuern. Mithilfe dieser Technik wurde z.B. im Film „The Muppet Movie“ die Puppe „Kermit“ in Echtzeit durch einen Akteur gesteuert [10].

Im Jahr 1992 wurde von SimGraphics das System “facial waldo” entwickelt. Hierbei wurde mittels elektromagnetischer Sensoren, welche am Kinn, den Lippen, Augenbrauen und Wangen angebracht wurden, Bewegungen des Gesichts analysiert und auf virtuelle Gesichter übertragen. Durch diese Innovation konnte neben den Bewegungen von Schauspielern auch deren Mimik auf eine virtuelle Person projiziert werden. Mit Hilfe dieses Systems wurde zum Beispiel das beliebte Computerspiel Mario von Nintendo kreiert [10].

Im Laufe der folgenden Jahre wurden die Motion Capture Systeme stetig weiterentwickelt, vor allem im Bereich des Motion Tracking.

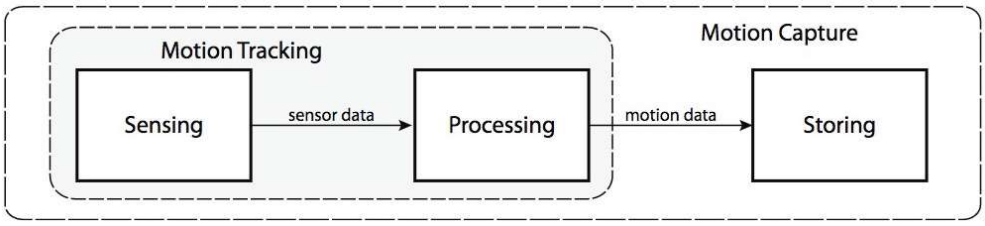


Abbildung 4: Motion Capture und Teilbereich Motion Tracking [11].

Das Motion Tracking umfasst, wie Abbildung 4 zeigt, den schwierigen Teil des Erfassens und Verarbeitens der Bewegung. Es beinhaltet zusätzlich die Speicherung der Daten. Grundsätzlich können die von Sensoren oder Kameras bereitgestellten Daten direkt gespeichert werden, aber sie sind selten in ihrem rohen Zustand zu gebrauchen. Widerstand, Ströme oder Farbpixel, unabhängig von der Ausgabe von Sensoren oder Kameras, müssen verarbeitet werden, um als Bewegungsinformationen nützlich zu sein. Diese Verbesserungen führten zu einem breiteren Einsatz in unterschiedlichen Anwendungsgebieten und natürlich zu einer höheren Qualität in der Umsetzung [12, S.12-14].

2.2 Anwendungsbereiche von Motion Capture

Wie im Kapitel „Historie“ zu erkennen ist, spielte der Einsatz von Motion Capture für die damalige Entwicklung der Filmindustrie eine genauso herausragende Rolle wie sie auch heute noch hat. Wurde die Technik zunächst in Trickfilmen eingesetzt, so ging es später darum einzelne Figuren zu animieren oder dann auch komplett animierte Produktionen zu zeigen. Heutzutage kommt kaum eine größere Produktion ohne den Einsatz von einer Art von Motion Capture aus. Einige Beispiele für Figuren und Produktionen sind: Die Figur „Gollum“ aus „Der Herr der Ringe“ die vollständig in Bewegung, Mimik und Gestik umgesetzt wurde. Der Film Avatar der mit Hilfe von MoCap die außerirdischen Figuren zum Leben erweckte. Und die Marvel Verfilmungen in denen durch den Einsatz der MoCap-Technologie das Comic-Universum in den Film übertragen werden konnte [9].

Neben der Filmbranche war auch in der Gamingbranche der Einsatz von Motion Capture früh verbreitet und heute ist eine Umsetzung von großen Titeln ohne diese Technologie undenkbar. Zusätzlich zum Einsatz, für die Charakteranimation werden in Spielen auch Bewegungen anderer dynamischer Objekte, z.B. von Flugkörpern aufgenommen und animiert.

Seit einigen Jahren wurde Motion Capture aber auch als Steuerungstool für den Spieler entdeckt, um eine neue Art von Spieleerlebnis zu vermitteln. Vorreiter war hierbei die Umsetzung der Bewegungssteuerung durch die Nintendo Wii. Mittlerweile sind solche Bewegungssteuerungssysteme bei jedem größeren Hersteller von Spielekonsolen Stand der Technik. Die nächste Entwicklung, VR-Spiele, ist ebenfalls schon im vollen Gange und hier ist sowohl in der Produktion als auch in der Steuerung der Spiele das Thema Motion Capture unabdingbar.

Die Einsatzmöglichkeiten haben sich auch in der Industrie vielfältig entwickelt. Beginnend von direkter Nutzung von Bewegungsdatensätzen z.B. fürs Anlernen komplexerer Bewegungen speziell für humanoide Roboter, über Ergonomieanalysen für neue Maschinen und Produkte, bis hin zu Simulationen in virtueller Realität für Produkte, Anlagen und Dienstleistungen sind die Anwendungsmöglichkeiten sehr weit gefächert.

Weitere Gebiete in denen die Bewegungsanalyse durch Motion Capturing Systeme Verbesserungen hervorrufen sind die Sportmedizin, der Hochleistungssport und die medizinische Rehabilitation: Durch die individuelle Analyse können z.B. angepasste Therapieprogramme und verbessertes Training angeboten und umgesetzt werden [12, S. 37-41].

2.3 Überblick unterschiedlicher Motion Capture Systeme

Motion Capture Systeme lassen sich in zwei Kategorien einteilen, optische und nicht-optische Systeme. In der Abbildung 5 sind die unterschiedlichen Systeme aufgeführt. Die optischen Motion Capture Systeme werden in die Typen markerbasiert und markerlos unterteilt. Die nicht-optischen Motion Capture Systeme sind sensorbasiert und werden in die Typen mechanische, magnetische und inertiale Systeme unterteilt [11, S. 674].

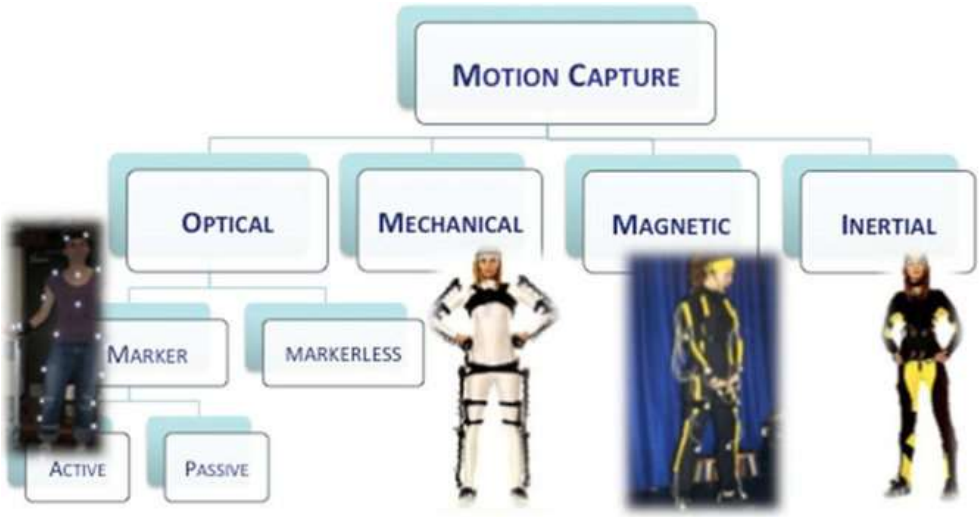


Abbildung 5: Unterschiedliche Motion Capture Systeme [11]

Bei markerbasierten Motion Capture Systemen trägt eine Person einen Ganzkörperanzug mit integrierten reflektierenden Markern oder die Marker z.B. direkt ins Gesicht geklebt. Während sich die Person bewegt oder eine Haltung einnimmt, werden die Positionen der Marker durch mehrere im Raum positionierte Kameras fixiert, das reflektierte Licht aufgenommen und die Bilder an einen Computer weitergegeben. Dieser fasst die Bilder in einem dreidimensionalen Modell zusammen, so dass die Bewegungen in Echtzeit aktiv wiedergegeben werden können. Zum Einsatz können aktive oder passive Marker kommen. Aktive Marker sind selbst Lichtquellen, meist LED's. Das Leuchten wird von den Kameras besser erfasst und es können kleinere Bewegungen oder Mimik detailliert aufgenommen werden. Nachteil der aktiven Marker ist eine große Anzahl von Drähten, die getragen werden müssen und einschränken können. Passive Marker bestehen aus reflektierendem Material, welches die Lichtquelle reflektieren kann. Der Nachteil von passiven Markern ist, dass bei schnelleren Bewegungen oder dicht beieinanderliegenden Markern die Motion Capture Software diese verwechseln kann. Dies kann zur ungenauen Aufnahme von Bewegungen führen [14].

Mechanische Motion Capture Systeme folgen, durch den Einsatz von Potentiometern, den Biegungen der Gelenke. Dem Akteur wird ein spezielles mechanisches MoCap Exoskelett aufgelegt, das alle seine Bewegungen mitmacht. Im Computer werden Daten über die jeweilige Position der Gelenkwinkel übertragen und es wird ein Echtzeit-Bewegungsmodell erstellt. Diese Systeme sind kostengünstig und können autonom genutzt werden. Allerdings

müssen die Systeme zeitaufwendig auf den Benutzer eingestellt werden und die Bewegungsfreiheit ist etwas eingeschränkt [13, S.167-168].

In magnetischen Motion Capture Systemen sind Marker-Magnete auf dem Akteur verteilt die ein magnetisches Feld aussenden und im Raum eine Transmittereinheit, die ein Magnetfeld aufspannt. Das System erfasst die Bewegung, indem eine Messeinheit die Positionen der Magneten durch Verzerrungen des magnetischen Flusses ertastet. Diese Sensoren, von denen jeder einzelne die Position und Rotation des entsprechenden Gelenks misst, senden die Bewegungsdaten an einen Computer. Der Einsatzbereich ist durch die Transmitter und die erzeugte Größe des Magnetfeldes begrenzt. Zudem können magnetische und elektrische Störungen die Genauigkeit der Messungen beeinträchtigen [13, S.169].

Abschließend betrachten werden die für diese Arbeit vielversprechend erscheinenden inertialen Motion Capture Systeme beschrieben. Hierbei werden Inertialsensoren für die Aufnahme von Bewegungen eingesetzt. Das Grundprinzip der Messung basiert auf translatorischen und rotatorischen Beschleunigungen. Die genaue Funktionsweise von Inertialsensoren wird im folgenden Kapitel genauer betrachtet. Die Akteure tragen zur Bewegungsmessung entweder ganze Sensoranzüge oder für Teilbewegungen werden Sensoren an notwendigen Stellen angebracht. Die Daten werden auf einen Computer übertragen und die Berechnung und Umsetzung der jeweiligen Bewegung erfolgt in Echtzeit. Diese Systeme ermöglichen eine hohe Bewegungsfreiheit und gute Echtzeitergebnisse dank einer hohen Präzision und einen geringen Kalibrierungsaufwand. Sie sind allerdings aufgrund ihrer Anfälligkeit für Driftfehler schlecht für längere Einsätze ausgerichtet/geeignet. Eine regelmäßige Kalibrierung ist für gute Ergebnisse notwendig [12, S. 28-29].

Wie in der Einleitung beschrieben soll für die Umsetzung des Protoyps für die Echtzeitmessung ein Motion Capture System basierend auf Inertialsensoren gestaltet werden. Diese wurden in Vorarbeite in dem Projekt „Echtzeit-Ergonomie“ ausgewählt. In der Umsetzung des Projektes ist das Ziel ein Bekleidungsstück mit integrierten Sensoren zu schaffen, welches Messungen mit einem geringen Kalibrierungsaufwand, hoher Flexibilität, Mobilität und hoher Präzision in Echtzeit ermöglicht. Aus der Vorstellung der unterschiedlichen Motion Capture Systeme wird deutlich, dass sich hierfür das Inertial Motion Capture am besten eignet.

2.4 Funktionsweise von Inertialsensoren

Im Normalfall bestehen Inertialsensoren aus einer Kombination von mehreren Messeinheiten. Am häufigsten werden Beschleunigungssensoren, Gyroskope und Magnetometer zu einer Messeinheit kombiniert. Hiervon hängt ab, wieviel Freiheitsgrade (degrees of freedom DOF) der Inertialsensor hat. Die Freiheitsgrade beziehen sich auf die verschiedenen Möglichkeiten, wie sich ein Objekt im dreidimensionalen Raum bewegen kann. Beispielsweise beinhaltet ein Inertialsensor mit neun Freiheitsgraden, eine 3-Achsen-Translationsbewegung entlang jeder Achse (vorne/hinten, rechts/links, oben/unten) auf einer Ebene und eine 3-Achsen-Drehbewegung auf jeder der X-, Y-, und Z-Achsen und die 3-Achsen-Magnetfeldstärke der x-, y- und z-Achsen [15, S. 112-113].

Die vom Inertialsensor gesammelten Rohdaten liefern eine Vielzahl unterschiedlicher Informationen der drei unterschiedlichen Sensoren. Die Sensorfusion wird hierbei eingesetzt, um die Daten jedes Sensors zu kombinieren und auf diese Weise die Geräteausrichtung und -position zu erhalten [16].

Beschleunigungsmesser (Accelerometer)

Der am häufigsten verwendete Bewegungssensor ist der Beschleunigungsmesser [17, S. 60]. Die vom Beschleunigungsmesser gemessene Beschleunigung ist die Änderungsrate der Geschwindigkeit eines Gerätes im Verhältnis zu der Zeit. Der Beschleunigungsmesser misst die Änderung der Beschleunigung auf einer Achse, dabei folgt er dem sensorinternen Trägheitskoordinatensystem. Das heißt, befindet sich der Sensor im freien Fall, ist die Beschleunigung in Abwärtsrichtung 0 m/s². Liegt der Sensor auf einer Ebene, ist die Beschleunigung in Aufwärtsrichtung gleich der Erdanziehungskraft, d. h. g = 9,8 m/s², und er misst die Beschleunigung des nach oben drückenden Gerätes [17, S. 60-65]].

Drehratensensor (Gyroskop)

Das Gyroskop misst die Winkelgeschwindigkeit einer Masse bezüglich eines inertial ausgerichteten Koordinatensystems. Es kann die Winkelgeschwindigkeit relativ zu sich selbst messen; es misst also seine Rotation, indem es die Trägheitskraft, die Coriolis-Kraft, nutzt. Das Gyroskop misst die Winkelbeschleunigung in drei Achsen: Nicken (x-Achse), Rollen (y-Achse) und Gieren (z-Achse) [17, S. 66-68].

Magnetfeldsensor (Magnetometer)

Das Magnetometer misst den das den Magnetismus. In dem es die magnetische Fluss-dichte im Raums misst, kann es die Ausrichtung des Erdmagnetfelds feststellen. Der Sensor kann also das Erdmagnetfeld durch diese Schwankungen wahrnehmen. Hierdurch kann der Sensor seine Ausrichtung nach Norden messen. Diese Ausrichtung wird genutzt, um mit den kombinierten Beschleunigungsmesser- und Gyroskopdaten die absolute Lage im Raum zu bestimmen [16].

3 Entwicklung des Sensorsystems

Das voranstehende Kapitel 2 ordnet die unterschiedlichen Techniken von Motion Capture Systemen ein und beschreibt die grundsätzliche Funktionsweise von Inertialsensoren. Dieses Kapitel entwickelt ein Sensorsystem zur Erfassung einer Hand-Arm-Bewegung. Das Ziel ist, Echtzeitdatensätze aufnehmen zu können, die es erlauben die Bewegung „Überkopfarbeit“ zu erkennen. Hierzu ist ein System mit mindestens zwei Inertialsensoren nötig. Welche weiteren Hard- und Softwarekomponenten eingesetzt und wie das System konfiguriert wird, wird im Folgenden beschrieben.

3.1 Hard- und Software des Sensorsystem

Die hier beschriebene Hard- und Software wird im ersten Schritt genutzt, um die Sensoren einzustellen und die Stabilität der Bewegung mit zwei Sensoren zu überprüfen. Die Berechnung der Positionsdaten und Speicherung des Echtzeitdatensatzes wird in Kapitel 5 erläutert.

3.1.1 Arduino

Für die Programmierung der Sensoren und das Auslesen von Daten wird ein Ein-/Ausgabeboard mit einem Microcontroller benötigt. Ausgewählt wurde hierzu das Arduinosystem.

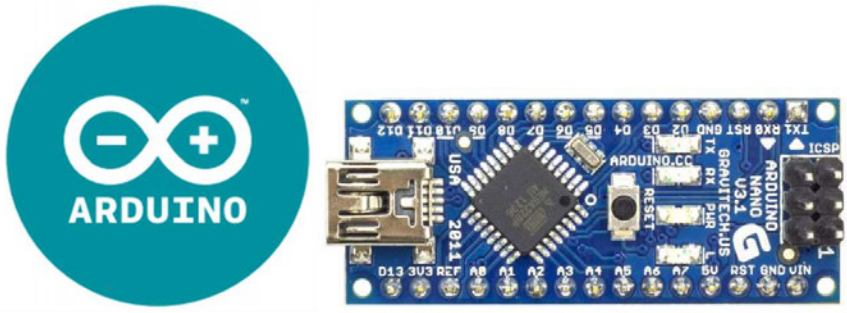


Abbildung 6: Arduino IDE Software (links) und Arduino Nano (rechts) [18].

Der Vorteil bei Arduino ist, dass es unterschiedliche Boards und die dazugehörige Arduino-Software (IDE) jeweils in Open-Source gibt. Über die Software kann in C/C++ Programmcode geschrieben werden und z.B. über eine USB-Schnittstelle auf das Board geladen werden. Über den Microcontroller können Eingaben an angeschlossene Hardware, in unserem

Fall die Inertialsensoren, ausgeführt werden und Daten übernommen und abgespeichert werden. Ein Arduino-Board enthält eine Reihe von Pins in zwei Varianten: analoge Pins, die eine Reihe von Werten lesen können, und digitale Pins, die einen einzelnen Zustand lesen und schreiben können. Auf diese Weise können verschiedenen Sensoren, Schalter, LEDs o.ä. angeschlossen und programmiert werden. Der Microkontroller verfügt über einen seriellen Monitor, über diesen können von ihm empfangene Ausgaben angezeigt werden [19].

In Abbildung 6 ist das ausgewählte Board für dieses Projekt zu sehen, der Arduino Nano. Das Hauptargument für diese Board ist die Baugröße und daraus folgend die Möglichkeit es in Kombination mit einem Inertialsensor auf einer Steckplatine zu befestigen. Hierdurch soll der Prototyp möglichst kompakt und praktikabel in der Praxis sein. Zudem bringt die kleinere Baugröße im Vergleich zu anderen Boards keine größeren Leistungsnachteile mit sich. Das Kernelement des Boards ist der ATmega328-Mikrocontroller mit einer Taktrate von 16 MHz, wie bei der Großzahl der Arduinoboards. Auf der Platine befinden sich 14 Ein-/Ausgabe Pins, 6 Digitale PMW Pins und 8 analoge Pins, über diese kann weitere Hardware angeschlossen und die Kommunikation der Programme erfolgen. Der Arduino Nano verfügt über eine Mini-USB-Schnittstelle über die zum einen die Stromversorgung geleistet und zum anderen das Übersenden der Programmierung aus der Arduino IDE Software erfolgt. In Tabelle 1 sind die technischen Daten des Arduino Nano dargestellt. Die Schaltung und Programmierung des Systems wir in Kapitel 3.1.4 erläutert [19].

Tabelle 1: Technische Daten Arduino Nano [18].

Mikrocontroller	ATmega328
Taktrate	16 MHz
Flash-Speicher	32 KB
SRAM	2 KB
EEPROM	1 KB
I/O Pins	6 PWM + 8 analoge Eingänge
Betriebsspannung	5V
Anschluss	MiniUSB
Größe	1,85 cm x 4,3 cm
Maximaler Strom pro I/O-Pin	40 mA
Belastbarkeit des 3.3-V-Ausgangs	50 mA

3.1.2 Inertialsensor Adafruit BNO055

Für den Prototyp werden zwei Adafruit BNO055 Sensoren eingesetzt. Hierbei handelt es sich um einen 9-DOF-Sensor. Im Gegensatz zu anderen 9-DOF-Sensoren ist der Adafruit BNO055 mit einem integrierten Algorithmus zur Sensordatenfusion ausgestattet. Üblicherweise müssen Sensordaten von Beschleunigungsmesser, Gyroskop und Magnetometer in eine tatsächliche "3D-Raumorientierung" umgewandelt werden, hierfür Algorithmen zu erstellen und die Berechnungen durchzuführen ist aufwendig.

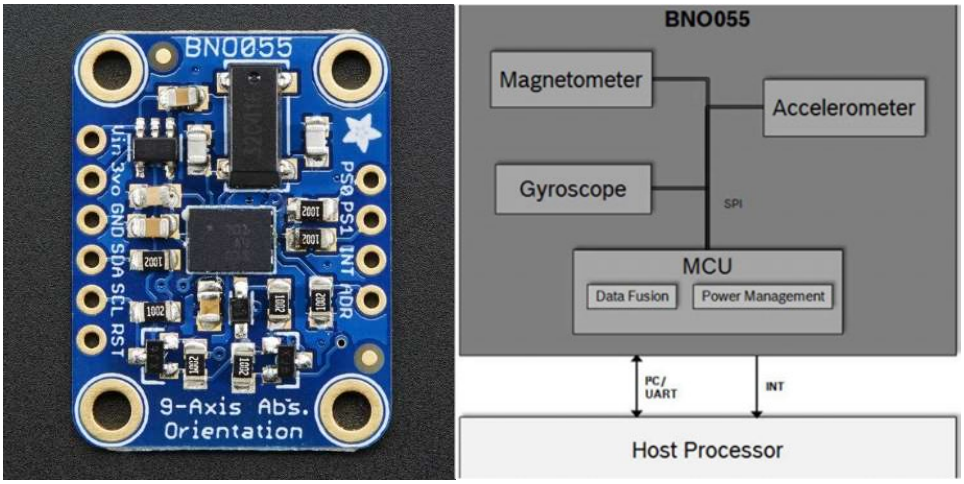


Abbildung 7: Adafruit BNO055 (rechts) und Systemarchitektur (links) [20], [21].

Abbildung 7 zeigt die Systemarchitektur des BNO055. Der MEMS-Beschleunigungsmesser, das Magnetometer und das Gyroskop sind zusammen auf einem Chip mit einem ARM Cortex-M0-basierten Prozessor untergebracht, der alle Sensordaten verarbeitet, die Sensorfusion und Echtzeitanforderungen abstrahiert und die Daten an den Host Prozessor übermittelt. Auf diese Weise können die Sensordaten in absoluter Orientierung in Euler-Vektoren (100 Hz) und in Quaternion (100 Hz) direkt ausgegeben werden. Ein Hauptargument dafür diese Sensoren zu nutzen. Zusätzlich können Winkelgeschwindigkeits-, Beschleunigungs-, Gravitations- und magnetischer Feldstärkenvektor ausgegeben werden. Über die vorhandene IC2-Schnittstelle ist die Verbindung und das Programmieren über den Arduino Nano möglich [20].

3.1.3 Software zur Visualisierung der Daten

Um die Sensoren zu kalibrieren, die Programmierung zu überprüfen und die Bewegung zu visualisieren, wird weitere Software genutzt. In Arduino IDE ist ein serieller Plotter integriert, dieser kann sowohl die übermittelten Sensordaten in nummerischer als auch in Diagrammen aufzeigen. Aufgrund der besseren Einstellbarkeit und Übersichtlichkeit wird für die Ausgabe der Sensordaten das Programm SerialPlot genutzt [22].

Um die Sensordaten zu visualisieren und die 3D-Bewegung optisch überprüfen zu können, wurde zunächst die Software Processing genutzt. Bei Processing gibt es sowohl die Möglichkeit frei verfügbare Simulationen zu nutzen als auch eine Anwendung selbst zu programmieren. Leider war dieser Weg nicht von Erfolg gekrönt. Es konnte weder eine passende Simulation eines Objektes (z.B. Würfel) welches sich mit 3D-Echtzeit-Daten steuern lässt, noch ein solche Anwendung mit der neu zu erlernenden Programmiersprache selbst gestaltet werden. Zum einen lag dies an den sehr vielen Beispielanwendungen, die frei zur Verfügung standen, zum anderen war in den gefundenen Tutorials das künstlerische animieren Thema und das Erlernen der Programmierung von 3D-Anwendungen schwierig [23].

Der zweite Versuch erfolgte in der Software IDLE, dies ist die Entwicklungsumgebung von Python. Über dieses Tool konnte eine Visualisierung gestaltet werden. Zum einen ist die Programmiersprache ähnlich wie die mir etwas bekannten Programmiersprachen C/C++ und die Tutorials waren mir leichter verständlich. Diese wird im Folgenden Kapitel beschrieben [24].

3.1.4 Gestaltung des Systems mit einem Sensor

Der erste Schritt ist das Anbringen des Arduino Nano und des BNO055 Sensors auf eine Steckplatine. Dann wird der BNO055 Sensor an den Arduino Nano angeschlossen. Die Stromversorgung wird über den 5V-Pin des Arduino Nano an den Vin-Pin des BNO055. Die Erdung erfolgt über den jeweiligen Ground-Pin (GND). Der I2C-Data-Pin (SDA) des BNO055 wird mit dem A4-Kanal des Nano und der I2C-Takt-Pin (SCL) mit dem A5-Kanal des Nano verbunden. Über die I2C-Schnittstelle kommuniziert das Programm mit dem BNO055 und die Sensordaten werden über eine UART-Schnittstelle an den Computer übertragen. Abbildung 8 zeigt die vorgenommene Verkabelung [20].

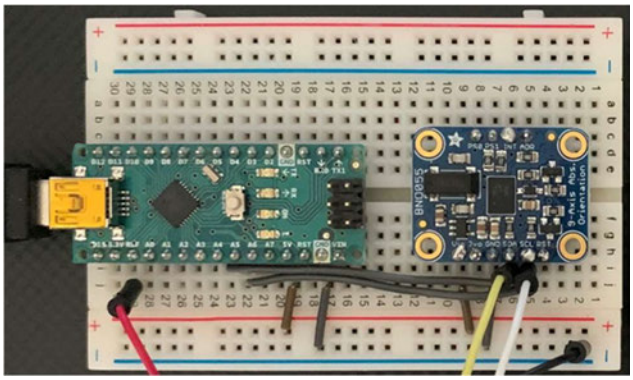


Abbildung 8: Verkabelung eines BNO055 Sensors mit dem Arduino Nano [25].

Bevor in Arduino IDE die Programmierung erfolgen kann, werden noch die Arduino- und Adafruit BNO055 Bibliotheken bzw. Treiber geladen. Die Bibliotheken bieten die Möglichkeit, Code wie Gerätetreiber oder häufig verwendete Hilfsfunktionen gemeinsam zu nutzen.

In einem ersten kleinen Programm werden die Beschleunigungs-, Gyroskop- und Magnometerwerte in X-, Y- und Z-Achse ausgelesen und die Werte bzw. Diagramme in Serial-Plot angezeigt. Für die Übertragung der Daten wird eine Baudrate von 115200 bei einer Übertragung in 8 Bit eingestellt, da es sich um eine erhöhte Datenmenge handelt. Die Abtastrate ist mit 100 Hz eingestellt. Nachdem die Übertragung der Daten und die Darstellung funktioniert, erfolgt als nächstes die Kalibrierung des Sensors.

Kalibrierung des Adafruit BNO055

Beim Start bzw. Restart der Programmierung über den Nano auf den Sensor BNO055 wird die Startposition definiert. Die gemessen Daten ergeben sich später auf Basis der jeweiligen Startposition. Vor allem ist dies später wichtig in Bezug auf die zu generierenden Winkeldaten, um stabile und vergleichbare Werte zu erhalten muss die Startposition fest definiert werden.

Die Kalibrierung des BNO055 erfolgt über die FunktiongetCalibration aus der Adafruit_BNO055-Bibliothek. Hiermit kann der jeweilige Kalibrierungsstatus jedes Sensors abgerufen und eingestellt werden. Obwohl der BNO055 interne Algorithmen zur fortlaufenden Kalibrierung von Gyroskop, Accelerometer und Magnetometer enthält, ist die genaue Art des Kalibrierungsprozesses unbekannt und daher eine manuelle Kalibrierung vor Aufnahme von Messdaten erforderlich.

Die vier Kalibrierungsregister werden mit der Funktion getCalibration(&system, &gyro, &accel, &mg) aufgerufen - ein Gesamtsystemkalibrierungsstatus sowie einzelne Gyroskop-, Magnetometer- und Beschleunigungsmesserwerte – werden mit den Werten 0 bis 3 zurückgemeldet. 0 steht für nicht kalibrierte und 3 vollständig kalibriert Daten. Die Systemkalibrierung ist vollständig erfolgt, wenn der Gesamtkalibrierungsstatus mit 3 angegeben wird.

Auf folgende Weise erfolgt die Kalibrierung:

- Das Gyroskop wird kalibriert, indem das Gerät in einer Position stillsteht.
- Das Magnetometer kalibriert, indem Bewegungen im Raum ausgeführt werden, wie z.B. ausführen einer 8 in der Luft.
- Accelerometer wird kalibriert indem, der Sensor um 45 Grad zu jeder Achse geneigt und für ca. 5 Sekunden gehalten wird. Dies sollte sowohl in der Normalposition des Sensors als auch beim Drehen des Sensors überkopf erfolgen.

Abbildung 9 zeigt die Änderung der Kalibrierungsstatus nach erfolgter Kalibrierung [25].

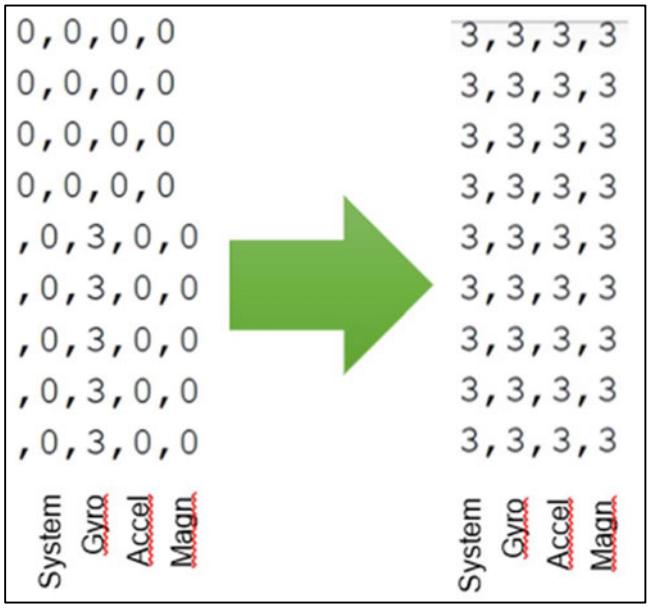


Abbildung 9: Kalibrierungsstatus nach erfolgter Kalibrierung [31].

Abbildung 9 wurde erstellt aus zwei vorher-/nachher-Ausschnitten der geplotteten Kalibrierungswerte. Für die Übersichtlichkeit wurden die vor den Kalibrierungswerten stehenden Werte ausgeschnitten. Auf der linken Seite ist der Kalibrieungsbeginn mit allen Werte auf 0

zusehen. Der Wert der am schnellsten kalibriert ist der Gyroskopwert. In der dritten Zeile springt dieser auf 3. Nach der erfolgten Durchführungen der oben beschriebenen Bewegungen zur Kalibrierung, ist auf der rechten Seite das Ergebnis zu sehen. Das Gesamtsystem und die drei Sensoren zeigen den Wert 3 an und sind somit vollständig kalibriert.

Gestaltung der Rotationsdaten vom BNO055

Der BNO055 wird im Folgenden so programmiert, dass Rotationsdaten im 3D-Raum generiert werden. Diese sollen dann in der Visualisierungssoftware einem 3D-Objekt übertragen werden und die Genauigkeit der Bewegung überprüft werden. Die Rotationen sollen zunächst in Euler-Winkeln gestaltet werden.

Euler-Winkel sind drei Winkel, die zur Angabe der Ausrichtung oder der Änderung der Ausrichtung eines Objekts im dreidimensionalen Raum verwendet werden. Jeder der drei Winkel gibt eine elementare Drehung um eine der Achsen in einem dreidimensionalen kartesischen Koordinatensystem an. Wichtig ist hierbei zu beachten, dass die Rotationsreihenfolge eine entscheidende Rolle spielt. Eine unterschiedliche Reihenfolge von Euler-Winkeln würde für jede Berechnung unterschiedliche Orientierungen ergeben. Daher wird im Einsatz von Euler-Winkel mit sogenannten Konventionen gearbeitet. Die in der Luftfahrt übliche Konvention „Yaw-Pitch-Roll“-Konvention wird in diesem Projekt umgesetzt [27].

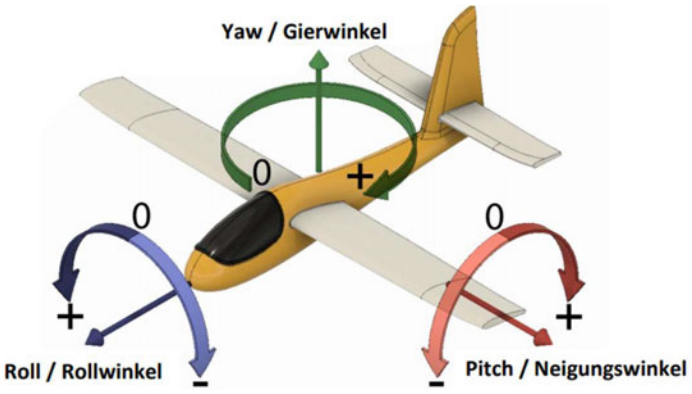


Abbildung 10: Darstellung der drei Euler-Winkel in der Yaw, Pitch und Roll [26].

Bei der „Yaw-Pitch-Roll“-Konvention sind die Euler-Winkel wie folgt beschrieben (siehe Abbildung 10):

• Der Gierwinkel **Ψ** (Yaw) zeigt die Rotation um die Z-Achse des global Koordinatensystems.

• Der Nickwinkel **Θ** (Pitch) zeigt die Rotation um die Y-Achse des global Koordinatensystems.

• Der Rollwinkel **Φ** (Roll) zeigt die Rotation um die X-Achse des globalen Koordinatensystems.

Da die Verknüpfung dieser Rotationen wiederum mittels Matrixmultiplikation erfolgt, ergibt sich folgende Beziehung:

$$R_{xyz}=R_X(\Phi)*R_Y(\Theta)*R_Z(\Psi) \qquad (1)$$

Zu beachten ist dabei, dass die Reihenfolge der Rotationen von rechts nach links gelesen wird. Das heißt, es wird zuerst um die Z-Achse gedreht, dann um die Y-Achse und zuletzt um die X-Achse. Nachdem die Konvention festgelegt ist, können mit den Messwerten vom Accelerometer, Gyroskop und Magnetometer die Gier-, Nick- und Rollwinkel errechnet werden [27].

Mit den Roll-, Nick- und Gier-Rotationsmatrizen

$$R_x(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix} \qquad (2)$$

$$R_y(\Theta) = \begin{pmatrix} \cos\Theta & 0 & -\sin\Theta \\ 0 & 1 & 0 \\ \sin\Theta & 0 & \cos\Theta \end{pmatrix} \qquad (3)$$

$$R_z(\psi) = \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ 0 - \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad (4)$$

kann die Formel (1) nach dem Nick- und Rollwinkel gelöst werden und ergibt:

$$\text{Nickwinkel (Pitch): } \tan(\theta) = \frac{a_y}{a_z} \rightarrow \theta = arctan\left(\frac{a_y}{a_z}\right) \qquad (5)$$

$$\text{Rollwinkel (Roll): } \tan(\phi) = -\frac{a_x}{\sqrt{(a_y^2+a_z^2)}} \rightarrow \phi = arctan\left(-\frac{a_x}{\sqrt{(a_y^2+a_z^2)}}\right) \quad (6).$$

Bei den Variablen a_x, a_y und a_z handelt es sich um die durch das Accelerometer gemessenen Beschleunigungen auf der X-, Y-, und Z-Achse des Sensors. Das Accelerometer misst

die Beschleunigung und die Schwerkraft, so kann die Ausrichtung relativ zur Schwerkraft errechnet werden [28].

In Arduino IDE entspricht die Berechnung folgendem Code (Ausgabe in Grad):

```
theta=-atan2(acc.y()/9.8,acc.z()/9.8)/2/3.141592654*360;
phi=-atan2(-acc.x()/9.8,sqrt(acc.y()*acc.y()+acc.z()*acc.z())/9.8))/2/3.141592654*360;
```

Als nächstes wird der Gierwinkel berechnet. Mit dem Accelerometer allein lässt sich dieser nicht bestimmen. Das liegt daran, dass wenn die z-Achse auf den Boden gerichtet ist, ist es egal, welchen Gierwinkel man um die z-Achse dreht, der Sensor immer a_z = 1g und a_x = a_y = 0 misst. Daher wird ein weiterer Sensor zur Berechnung des Gierwinkels benötigt, das Magnetometer. Wenn sich das Magnetometer auf einer ebenen Fläche befindet, die von keinem anderen elektromagnetischen Feld umgeben ist, wird der Gierwinkel wie folgt berechnet:

$$\text{Gierwinkel (Yaw): } \tan(\psi) = \frac{m_y}{m_x} \rightarrow \psi = arctan\left(\frac{m_y}{m_x}\right) \quad (7)$$

In Arduino IDE entspricht die Berechnung folgendem Code (Ausgabe in Grad):

```
psi = atan2(mgt.y(), mgt.x()) / 2 / 3.142 * 360;
```

Steht das Magnetometer senkrecht zur Schwerkraft kann mit den Werten m_y und m_x der Gierwinkel in der Horizontalen berechnet werden. Allerdings ist zu beachten, dass diese Formel nicht korrekt ist, wenn der Sensor geneigt ist. In dem Fall muss eine Neigungskompensation mithilfe des Accelerometers erfolgen. Hierfür werden die Roll- Nickwerte aus der Berechnung des Accelerometers in die Formel zur Berechnung des Gierwinkles mit Kompensation eingefügt:

$$M_y = m_y * \cos(\phi) + m_z * \sin(\phi) \quad (8)$$

$$M_x = m_x * \cos(\theta) + m_y * \sin(\phi) * \cos(\theta) + m_z * \cos(\phi) * \sin(\theta) \quad (9)$$

$$\text{Gierwinkel (Yaw) mit Kompensation: } \tan(\psi) = \frac{M_y}{M_x} \quad (10)$$

In Arduino IDE entspricht die Berechnung folgendem Code (Ausgabe in Grad):

```
Xm=mag.x()*cos(thetaRad)+mag.y()*sin(phiRad)*sin(thetaRad)+mag.z()*cos(phiRad)*sin(thetaRad);
Ym=mag.y()*cos(phiRad)+mag.z()*sin(phiRad);

psi=atan2(Ym,Xm)/(2*3.14)*360;
```

Sind die drei Rotationswinkel programmiert, erfolgt die Übertragung über den Arduino Nano auf den BNO055 Sensor, die dann resultierenden Sensordaten für die drei Winkel werden im SerialPlot als Diagramme betrachtet. Dabei ist zu sehen, dass bei langsamen Bewegungen die Winkelwerte gut wiedergegeben werden. Bei schnellen Bewegungen allerdings kann ein Überschlagen über die Werte erfolgen und die Ausgabe ist ungenau. Zudem ist zu sehen, dass kurze Vibrationen mit dem Sensor ebenfalls zu falschen Winkelwerten führen. Dies liegt daran, dass die bisher genutzten Accelerometer und Magnetometer fehleranfällig sind bei kurzen schnellen Bewegungen bzw. Vibrationen, da sie alle Kräfte die auf sie wirken sofort messen. Der weitere Sensor, das Gyroskop, hat dieses Problem nicht. Es misst Winkel und keine lineare Bewegung und ist daher nicht empfindlich. Außerdem sind Accelerometer und Magnetometerauf lange Sicht zuverlässiger, während Gyroskope ihre besten Ergebnisse kurz nach dem Einschalten erzielen. Die Sensoren haben also unterschiedliche Fehlerquellen, sie zu kombinieren, um diese auszugleichen würde eine bessere Schätzung für die resultierenden Winkel ergeben. Dies wird durch Einsatz eines Komplementärfilter erreicht. Der Komplementärfilter ist einfach und leicht zu verwenden, er enthält einen festen Wert für die Gewichtung von Tiefpassfilter und Hochpassfilter. Kurzfristig verwenden wir die Daten des Gyroskop (gry_x, $gyr_{y)}$ und auf lange Sicht die Daten des Beschleunigungs- und Magnetometersensors. Die Formel für den Komplementärfilter ist [29]:

$$K\_Pitch: \theta_k = (\theta_k + gyr_y(y)* dt) * 0.95 + \theta_m * 0{,}05 \qquad (11)$$

$$K\_Roll: \phi_k = (\phi_k + gyr_x(x)* dt) * 0.95 + \phi_m * 0{,}05 \qquad (12)$$

In Arduino IDE entspricht die Berechnung folgendem Code (Ausgabe in Grad):

```
dt=(millis()-millisOld)/1000.;
millisOld=millis();
thetaK=(theta+gyr.y()*dt)*.95+thetaM*.05; // Komp.filter theta
phiK=(phi-gyr.x()*dt)*.95+ phiM*.05;      // Komp.filter phi
thetaG=thetaG+gyr.y()*dt;                 // Gyro theta
phiG=phiG-gyr.x()*dt;                     // Gyro Phi

phiRad=phi/360*(2*3.14);
thetaRad=theta/360*(2*3.14);

//psi = atan2(mgt.y(), mgt.x()) / 2 / 3.142 * 360; nur 2D

Xm=mag.x()*cos(thetaRad)+mag.y()*sin(phiRad)*sin(thetaRad)+mag.z()*cos(phiRad)*sin(thetaRad);
Ym=mag.y()*cos(phiRad)+mag.z()*sin(phiRad);

psi=atan2(Ym,Xm)/(2*3.14)*360;
```

Bei jedem Messzyklus wird im Hochpassfilter (HPF) 95 % des vorherigen gefilterten Winkelwertes mit dem addierten Winkel aus den Gyroskopdaten berechnet und 5 % aus dem Tiefpassfilter (TPF) mit den Accelerometer- und Magnetometerdaten. Erfahrungsgemäß resultieren die besten Werte bei Einstellungen des HPF von 98 - 95 % und TPF von 2 - 5 %. In der Überprüfung der Winkelwerte über die Anzeige der Diagramme in SerialPlot zeigt sich, dass der Komplementärfilter seine Aufgabe erfüllt. Die Werte sind auch bei schnellen Bewegungen und Vibrationen stabil und es ergeben sich keine auffälligen Ausschläge.

Zur weiteren Überprüfung wird in Python bzw. der Animationssoftware VPython ein Board animiert und die berechneten Winkelwerte über die serielle Schnittstelle in Python eingelesen, um die Bewegung in 3D zu visualisieren [24]. Zusätzlich werden zwei Koordinatensysteme animiert. Eines soll auf dem Board das körpereigene Koordinatensystem darstellen, hierzu werden drei senkrecht aufeinander stehende Pfeile animiert. Das zweite Koordinatensystem soll als globales Koordinatensystem dienen und feststehen. Die Bewegung des Boards und des körpereigenen Koordinatensystems werden mithilfe von Vektorberechnung bzw. der Rodrigues Rotationsformel berechnet. Die angewendeten Berechnungsformeln sind im Python Code zu finden. In Abbildung 11 ist das Ergebnis der Programmierung zu sehen.

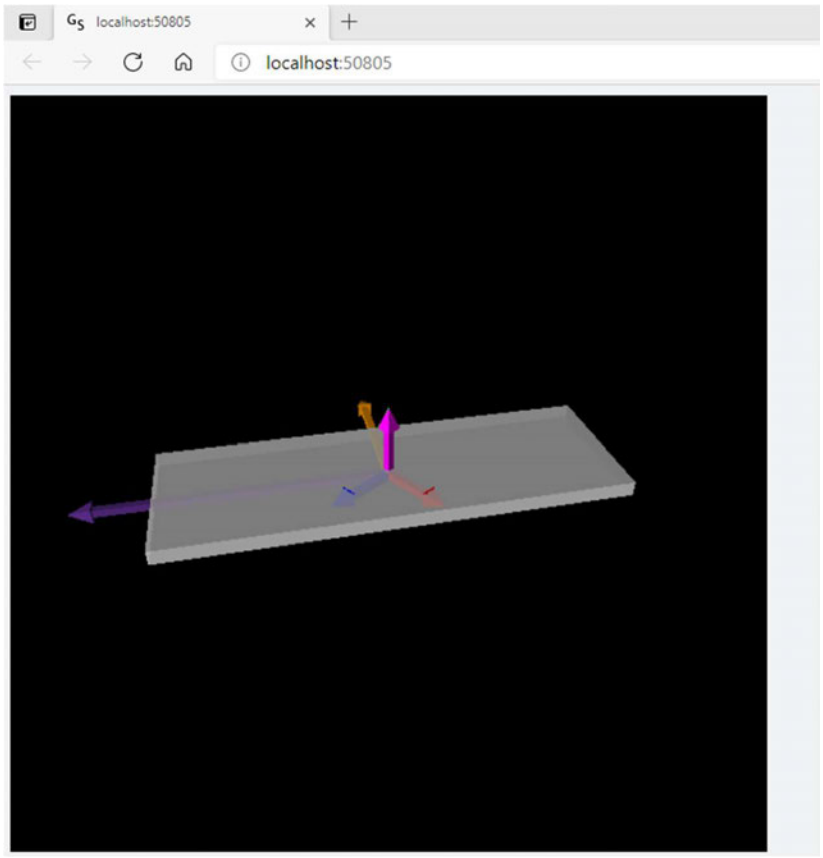


Abbildung 11: Darstellung Board und Koordinatensystem in VPython programmiert [25].

Auf diesem Wege ist die erste Gestaltung eines einfachen Echtzeit Motion Capture Systems gelungen. Die Ausgeführten Bewegungen mit dem Board werden durch den Sensor erfasst, die Winkeldaten werden aufgenommen und zur Berechnung der visualisierten Bewegung in Python übertragen und zum Schluss wird die Bewegung mit dem animierten Board ausgeführt. Beim Austesten ist grundsätzlich die Bewegung flüssig und entspricht der reellen Bewegung mit dem Board. Allerdings fällt auf, dass beim Rotieren um 90° Probleme auftauchen. Hierbei ergeben sich häufiger Sprünge, das Board rotiert plötzlich um 180°. Dieses Problem resultiert aus dem Einsatz von Euler-Winkeln. Beschriebene Bewegungen in Euler-Winkeln können mehrdeutig sein. Wird das Board z.B. zuerst mit einem Yaw-Winkel von 45° und danach mit einem Pitch-Winkel von 90° rotiert, dann ergibt sich dieselbe Orientierung wie bei einer Pitch-Rotation um 90° mit folgender Roll-Rotation um 45°. Dabei entsteht das Problem, dass eine Rotation von Pitch um ±90°, die Längsneigung mit dem Gierwinkel zusammenfallen lässt. Daraus folgt, dass die Orientierung auf die Y-Achse eingeschränkt

ist. So verliert das System einen Freiheitsgrad. Dieser Effekt ist bekannt als „Gimbal lock“. In Abbildung 12 wird dieser Effekt nochmal veranschaulicht.

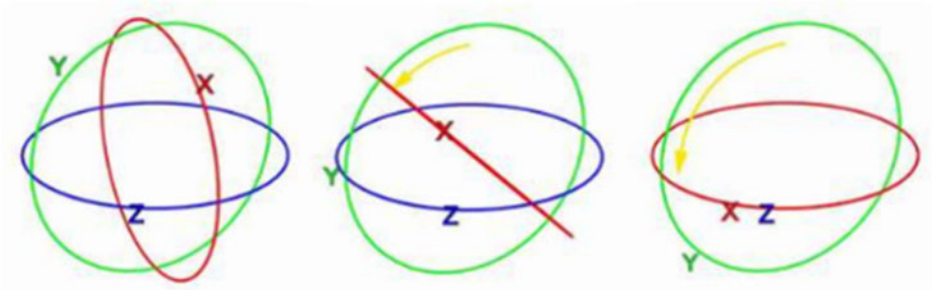


Abbildung 12: Gimbal lock. Wenn sich die Pitch (Y)-Achse um 90 Grad dreht, liegen die Roll (X)- und Yaw (Z)-Achse auf einer Ebene und ein Freiheitsgrad geht verloren. [38]

Dieses Problem muss gelöst werden, um fehlerfreie und korrekt beschriebene Bewegungen zu erhalten. Die Lösung hierfür wurde schon mit der Auswahl des Sensors getroffen, wie beschrieben ist der BNO055 in der Lage seine Daten sowohl in Euler-Vektoren als auch in Quaternionen direkt auszugeben. Quaternionen sind ein Zahlenbereich, der den Zahlenbereich der reellen Zahlen erweitert. Quaternionen ermöglichen es das Gimbal lock Problem zu umgehen.

Quaternionen sind in der Lage Orientierungen mit 4 Werten auszudrücken. Sie sind eine Erweiterung der reellen Zahlen, ähnlich den komplexen Zahlen. Sie werden durch eine reale und drei imaginäre Komponenten beschrieben. Das heißt, sie werden durch ein Quadrupel gebildet:

$$q = w + ix + jy + kz \quad (13)$$

$$\text{mit } i^2 + i^2 + k^2 = -1 \quad (14)$$

Die Komponenten i, j und k sind jeweils Einheitsquaternionen und entsprechend demnach den Einheitsvektoren in einem Hauptachsensystem. Sie unterliegen jedoch anderen Kombinationsregeln:

$$i*j = k, j*i = -k, k*j = -i, j*k = i\,, k*i = j, i*k = -j \quad (15)$$

Die Multiplikation sind nicht kommutativ, das bedeutet das Produkt aus zwei Quaternionen $q1$ und $q2$ zwei verschiedene Ergebnisse liefert:

$$q1*q2 \neq q2*q1 \quad (16)$$

So sind Quaternionen eine vierdimensionale Darstellung zur genauen Beschreibung von Rotationen im dreidimensionalen Raum. Die Rotation wird direkt um die gewünschte Drehachse ausgeführt, dies beseitigt den Gimbal lock. Zudem entfällt die Mehrdeutigkeit von Rotationen, jede Quaternion stellt genau eine Repräsentation dar.

Der BNO055 liefert über den Befehl .getQuat() die Daten direkt in Quaternionen:

```
imu::Quaternion quat=myIMU.getQuat();

Serial.print(quat.w());
Serial.print(",");
Serial.print(quat.x());
Serial.print(",");
Serial.print(quat.y());
Serial.print(",");
Serial.print(quat.z());
```

Um in VPython die Bewegung des Boards zu animieren müssen die Quaternion wieder in Euler-Winkel umgerechnet werden, da die Berechnung in Python auf Vektoren basiert. Dies erfolgt über die Formeln [30]:

$$\text{Nickwinkel (Pitch): } \theta = arctan\left(\frac{2(p_1p_3-p_0p_2)}{4(p_0p_3+p_1p_2)^2+(1-2(p_0^2+p_1^2))^2}\right) \quad (17)$$

$$\text{Rollwinkel (Roll): } \phi = arctan\left(\frac{2(p_0p_3+p_1p_2}{1-2(p_0^2+p_1^2}\right) \quad (18)$$

$$\text{Gierwinkel (Yaw): } \psi = arctan\left(\frac{2(p_0p_1+p_2p_3}{1-2(p_2^2+p_3^2}\right) \quad (19)$$

Hierbei sind p_0 = w, p_1 = i, p_2 = j und p_3 = k.

Mit den in Quaterionen übergeben und umgerechneten Daten lässt sich in VPython die Bewegung unter den neuen Bedingungen überprüfen. Es zeigt sich, dass die 3D-Rotationen ohne Sprünge und Fehler flüssig in 360° abbildbar sind. Damit ist die Umsetzung des ersten vereinfachten Echtzeit Motion Capture System geglückt. Als nächstes wird dieses um den zweiten Sensor erweitert.

3.1.5 Erweiterung des Sensor Systems um einen zweiten Sensor

Der zweite der BNO055 Sensor wird mit demselben Arduino Nano angeschlossen. Die Stromversorgung erfolgt allerdings über den weiteren 3,3V-Pin des Arduino Nano an den Vin-Pin des BNO055. Die Erdung GND, I2C-Data-Pin SDA und der I2C-Takt-Pin SCL

werden ebenfalls mit den A4 und A5-Pins verbunden. Zusätzlich wird beim zweiten BNO055 der ADR-Pin ebenfalls angeschlossen. Der ADR ändert die Standard-I2C-Adresse so kommt es zu keinen Konflikten mit dem ersten Sensor. Das neue Schaltbild ist in Abbildung 13 zu sehen [20]

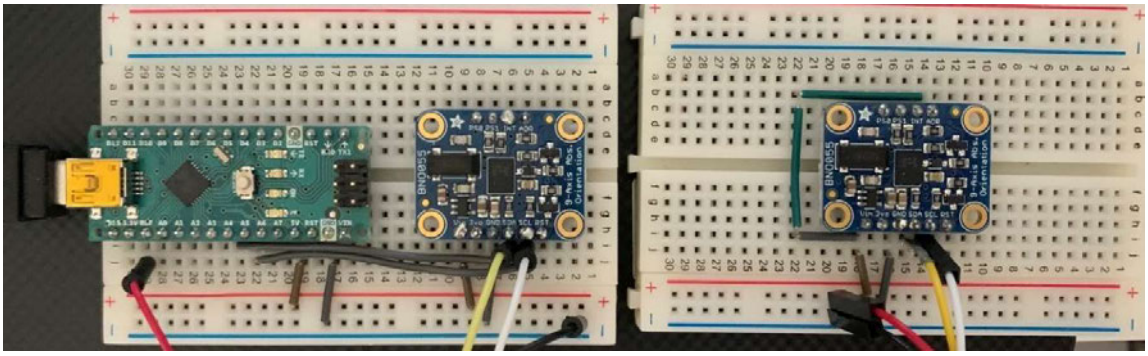


Abbildung 13: Schaltbild Arduino Nano mit zwei BNO055 [25].

Als nächstes erfolgen Anpassungen in der Programmierung. Zuerst müssen den Sensoren feste Adressen zugewiesen werden. Für den Sensor1 ist dies die Adresse Adafruit_BNO055(55,0x28) und für den Sensor2 Adafruit_BNO055(55,0x29). Dann erfolgt die Erweiterung im Code, sowohl im Arduino IDE als auch in Python, auf den zweiten Sensor. Im Anschluss wird das zweite Board für die Visualisierung modelliert. Die neue Programmierung ist in der Dokumentation zu finden. Abbildung 14 zeigt die Umsetzung mit zwei Boards bei der die Echtzeit 3D-Bewegung überprüft wird:

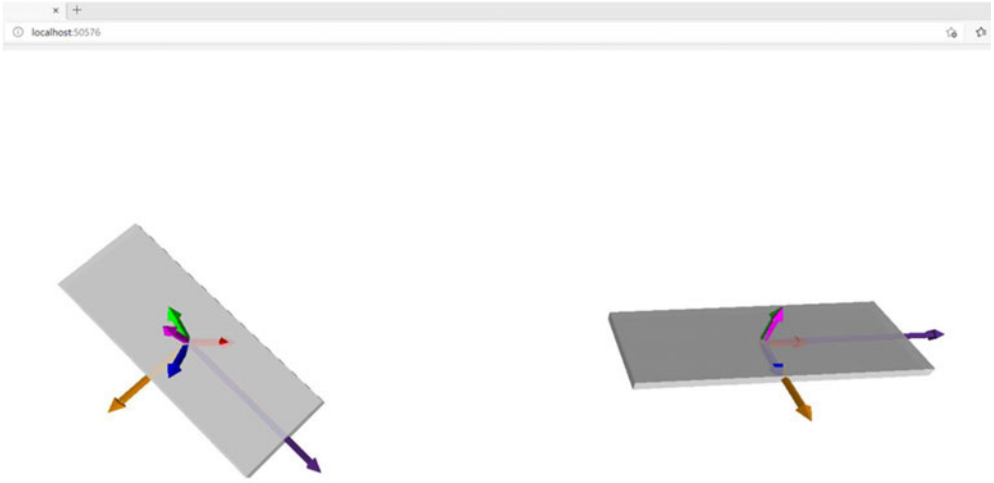


Abbildung 14: Python zwei Boards angesteuert über zwei unterschiedliche Sensordaten[31].

Die Boards können unabhängig voneinander über den jeweiligen Sensor gesteuert werden. Wie bei der Umsetzung mit einem Sensor ist es gelungen die Bewegungen in Echtzeit fehlerfrei und stabil zu übertragen. Weiterhin ist zu beachten, dass vor dem Anwenden der Sensoren die Kalibrierung nötig ist. Nach erfolgter Kalibrierung und bei den angedachten kürzeren Messreihen (evtl. über mehrere Minuten) werden gute Ergebnisse geliefert. Die Problematik des Driftens verursacht durch die Gyroskope ist nicht zu beobachten. Ob dies an dem automatischen internen Kalibrierungsprozess des BNO055 für die Sensoren oder des Einsatzes des Komplementärfilters liegt, ist nicht klar. Dies kann über spätere Versuchsreihen ermittelt und auch über den Einsatz weiterer Filter wie z.B. Kalman- oder Madgwick-Filter verbessert werden. Für den späteren Versuch bzw. die zu generierenden Datensätze für den Machine Learning Algorithmus, ist die 100 % Genauigkeit der Messwerte sowieso nicht erwünscht, vielmehr geht es darum mit verrauschten Datensätzen bestimmte Muster zu erkennen und diese als spezifische Bewegung zu deklarieren.

Nachdem nun das Sensorsystem für die Echtzeitmessung von Rotationsdaten von zwei Sensoren erfolgreich gestaltet wurde, folgt im nächsten Kapitel die Recherche und der Abgleich zwischen möglichen Referenzsystemen, die den Echtzeitdatensatz für das spätere Erlernen der Bewegungen liefern soll.

4 Auswahl des Referenzsystems

In Kapitel 2.3 wurden die unterschiedlichen Typen von Motion Capture Systemen vorgestellt. Ein Vergleich all dieser unterschiedlichen Systeme könnte als Orientierung dienen, würde aber in der Praxis keinen Vorteil bringen. Vor allem da die qualitativ hochwertigen Systeme zum einen hochpreisig sind und zum anderen die Installation z.T. einen sehr hohen Aufwand bedeuten würde. Der praktikablere und hier umgesetzte Ansatz, ist die möglichen Ressourcen zu betrachten und eine Auswahl innerhalb dieser zu treffen. Im Falle der Referenzsysteme bedeutet dies, es stehen drei Möglichkeiten zur Verfügung. Über die Verwaltungs-Berufsgenossenschaft besteht die Möglichkeit ein CUELA-Messsytem zu nutzen. In der Fakultät Design, Medin und Information der HAW steht ein HTC Vive System für Studentenprojekte zur Verfügung. Die dritte Möglichkeit besteht in der Nutzung der Open Source Software Kinovea zur Bewegungsanalyse. Diese drei Systeme werden im Folgenden beschrieben und verglichen.

4.1 Das CUELA-Messsystem

Das Messsystem CUELA (Computer-Unterstützte Erfassung und Langzeit-Analyse von Belastungen des Muskel-Skelett-Systems) wurde im Institut für Arbeitsschutz der DGUV entwickelt. Es handelt sich um ein personengebundenes Messsystem zur Messung von Belastungen des Muskel-Skelett-Systems. CUELA erfasst Körper-/Gelenkbewegungen mithilfe von Bewegungssensoren (Goniometer und 3D-Inertialsensoren), die auf der Kleidung der Arbeitsperson angebracht werden. Alle Messdaten werden in einem Datenspeicher und auf handelsüblichen Compact-Flash-Karten abgespeichert. Ein sehr geringer Energieverbrauch ermöglicht zusammen mit den verwendeten kompakten Lithium-Ionen-Akkus einen mehrstündigen Betrieb des Systems im Feld, auch an nicht stationären Arbeitsplätzen. Der Datenspeicher verfügt über einen Online-Aufzeichnungsmodus, bei dem sich die Messdaten über eine Bluetooth-Verbindung direkt an einen Computer weiterleiten und in Echtzeit visualisieren lassen. Es bestehen individuelle, auf Körperumfang und -größe bezogene Einstellmöglichkeiten. Die Bewegungen werden mit einer Abtastfrequenz von 50 Hz digitalisiert. Die zugehörige Software WIDAAN erlaubt eine automatisierte Auswertung der Messdaten nach arbeitswissenschaftlichen und biomechanischen Bewertungskriterien [31]. Die Abbildung 15 zeigt verschiedene Varianten des CUELA-Messsystems.

Abbildung 15: Varianten des CUELA-Messsystems: Basissystem, Erweiterung Kopf und Schulter-Arm-Bereich und Sitzsystem [31].

Für das Referenzsystem wäre die Variante mit der Erweiterung um Kopf, Schulter und Arm-Bereich relevant. Für die Messung der Armbewegung sind jeweils weitere Sensoren an Ober- und Unterarm angebracht. Über die Analysesoftware WIDAAN besteht die Möglichkeit sich die Winkeldaten für die Armbewegung anzeigen zu lassen. Hierbei können für die Arm-Bewegung folgende Winkel erfasst werden: Am Schultergelenk die Oberarm-Adduktion/-Abduktion (zum Körper hin, vom Körper weg); am Schultergelenk die Oberarm-Flexion/-Extension (nach vorne, nach hinten); am Schultergelenk die Oberarm-Rotation (nach innen, nach außen); Am Ellenbogengelenk die Unterarm-Flexion/-Extension (Beugung/Streckung des Unterarms); Am Ellenbogengelenk die Unterarm-Pronation/-Supination (Handfläche nach unten/oben). Die jeweiligen Winkeldaten können über die WIDAAN-Software in einer csv-Datei abgespeichert werden [32]. Abbildung 16 zeigt bespielhaft die Analysesoftware WIDAAN mit der Anzeige von Winkeldaten in grafischer Form.

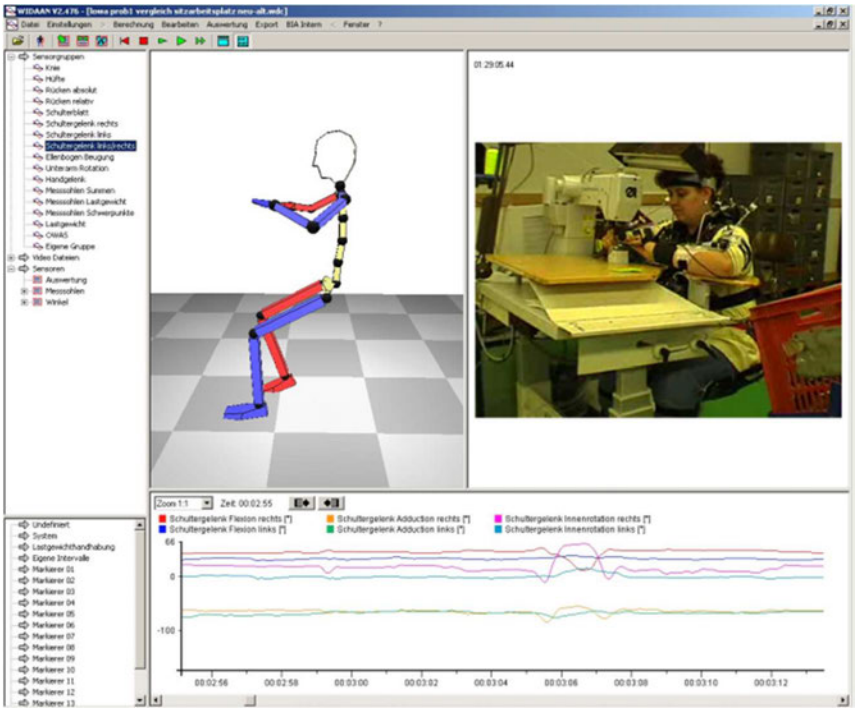


Abbildung 16: Anzeige der Winkeldaten in der Software WIDAAN [32].

Vor dem Einsatz des CUELA-Systems muss das System überprüft und die Messgeräte kalibriert werden. Dann müssen die Elektroden und Messgeräte an den definierten Positionen an der Versuchsperson angebracht werden. Hierbei ist zu beachten, dass die Sensoren festsitzen und nicht verrutschen können. Die Verkabelung muss so angebracht werden, dass kein verhaken möglich ist. Zudem muss die Kalibrierung und Erprobung der Signale bezogen auf die jeweilige Versuchsperson nach Anbringung des Messsystems personenbezogen fertiggestellt werden. Anschließend ist die Versuchsdurchführung möglich [32].

4.2 Das Kinovea-System

Kinovea ist eine Viedoanalysesoftware, die für die Sportanalyse entwickelt wurde und open source zur Verfügung steht. Kinovea bietet unterschiedliche Features zum Erfassen, Verlangsamen, Vergleichen, Kommentieren und Messen von Bewegungen in Videos [34]. Beim Messen von Bewegungen ist die Kinovea-Software in der Lage Objekte, Strukturen oder Körperteile auf Videoaufnahmen zu erkennen und diesen zu folgen. Hierzu sind entweder markante Punkte an den zu verfolgenden Objekten oder Körperteilen nötig oder es können passive Marker eingesetzt werden. Eine bessere Definition der zu verfolgenden

Punkte und bessere Ergebnisse werden durch den Einsatz von passiven Markern erzielt. Die Kinovea-Software bietet dabei den großen Vorteil, dass das visuelle Tracking in den Videos durch den Einsatz von Algorithmen automatisch erfolgen kann. Zwei Varianten des Tracking stehen dem Nutzer dabei zur Verfügung, die Trackability- oder die Trajectories-Variante. Bei der Trackability-Variante werden Linien, Winkel, Modelle o.ä., welche vom Nutzer im Video eingesetzt werden, getrackt. Bei der Trajectories-Variante werden Bewegungen über einen längeren Zeitraum verfolgt und die Trackinginformationen abgespeichert. In beiden Fällen ist dies über den Einsatz von Algorithmen von OpenCV, einer freien Programmbibliothek mit Algorithmen für die Bildverarbeitung und Computer Vision, möglich. Für den Anwendungsfall, das Tracken einer Arm-Bewegung, ist die Trajectories-Variante relevant. Das automatische Tracking funktioniert, in dem in einem Bildausschnitt ein kleiner Bereich als Template (Schablone) definiert wird. Durch dieses Template wird die zu verfolgende Stelle definiert (z.B. ein Marker). Nach dieser Konfiguration wird anschließend in jedem weiteren Bild des Videos nach einer Übereinstimmung zu dem Template gesucht und die Position gespeichert. Es ergibt sich die Bewegung über die Zeit als Trackingpfad [35]. In Abbildung 17 ist auf der rechten Seite die Konfiguration des Templates und auf der linken Seite der resultierende Trackingpfad zu sehen.

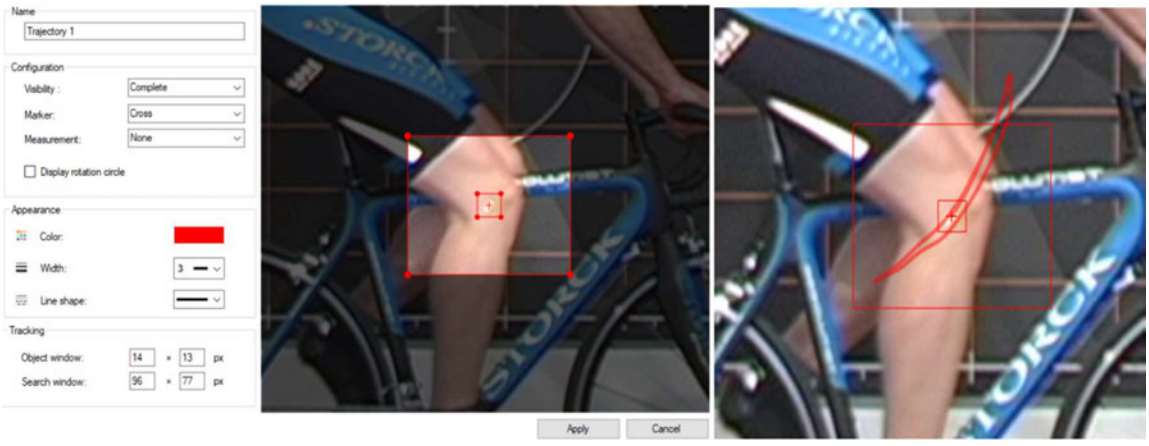


Abbildung 17: Automatischer Trackingprozess in der Kinovea-Software [43].

Das Tracken mehrerer Punkte zur gleichen Zeit ist möglich, allerdings ergeben sich bei Überlagerungen Trackingprobleme die zu Fehlern führen können [35, S. 64]. Aus den jeweiligen Trackingpfaden lassen sich die Daten in Diagrammen oder csv-Dateien abspeichern bzw. anzeigen.

Für die Aufnahme und Analyse der Arm-Bewegung im Raum sind für die Erstellung der Videoaufnahmen zwei Full-HD-Kameras (min 50 fps) mit Stativ nötig. Die Aufnahmen

müssten zeitgleich frontal und seitlich (90° zur Frontkamera) erfolgen, um über die beiden resultierenden Datensätze die Bewegung im dreidimensionalen Raum zu interpolieren. Die Einstellung des jeweiligen Versuchsaufbaus zur Aufnahme der Bewegung und die Vorbereitung der Versuchspersonen ist nicht aufwendig. Die Analyse der erstellten Videos und das generieren der Datensätze könnte, je nachdem wie gut das automatische Tracking funktioniert, zeitintensiv ausfallen.

4.3 Das HTC Vive System

Das dritte System das zur Verfügung steht, ist das HTC Vive System. Es handelt sich hierbei um ein Virtual Reality System (VR-System) auf Basis eines Head Mounted Displays (HMD). Das HTC Vive System wurde in Zusammenarbeit zwischen dem Hardwarehersteller HTC und dem Spieleentwickler Valve entwickelt. Es ist dazu konzipiert, Spiele oder andere Anwendungen in virtueller Realität, also computergenerierter dreidimensionaler Umgebung, über das Head Mounted Display zu übertragen. In Abhängigkeit der Bewegung des HMD, der Blickrichtung des Anwenders, wird fortlaufend das dreidimensionale Bild angepasst und so eine 360° Umgebung simuliert. Dies ist möglich, da das HTC Vive System die dazugehörige Hardware wie z.B. das Head Mounted Display, die Controller oder Tracker durch ein sogenanntes room-scale Tracking erfassen kann. Wie die funktioniert wird im Folgenden erläutert.

In Abbildung 18 ist das Basis-System der HTC Vive zu sehen.

Abbildung 18: HTC Vive System [35].

Dieses besteht aus dem Head Mounted Display (mitte), zwei Motion Controller (vorne links und rechts) und den zwei Basisstationen (hinten links und rechts). In der Abbildung 19 ist der Trackingprozess der HTC Vive visualisiert.

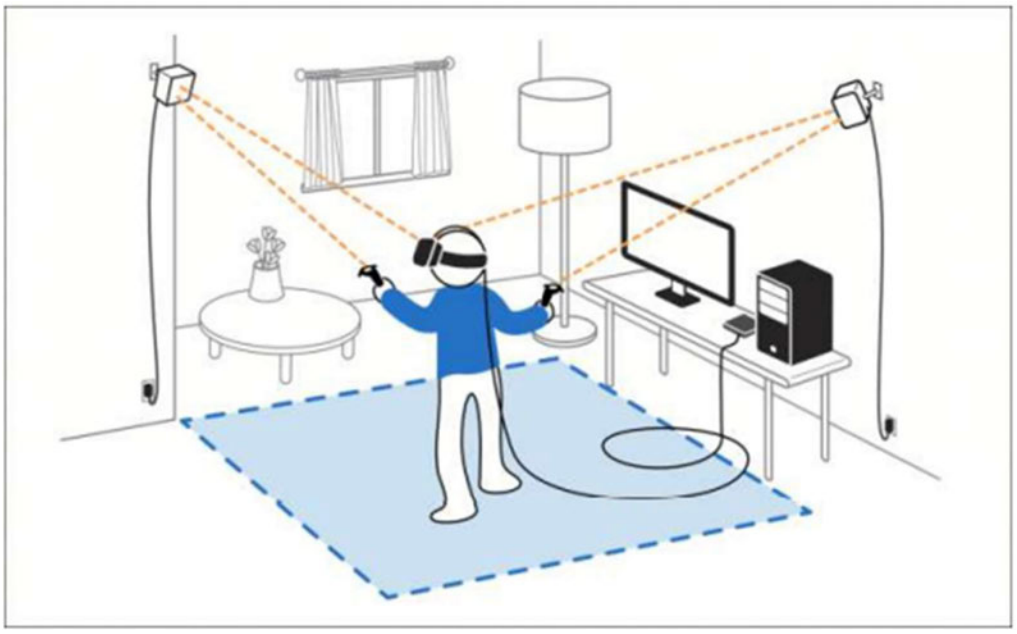


Abbildung 19: HTC Vive Tracking mit den Basisstationen [35].

Das verwendete Prinzip des room-scale Tracking basiert darauf zwischen zwei externen Messstationen einen Raum aufzuspannen, in dem Objekte erfasst werden können. In diesem Fall erfolgt das Tracken mit den zwei Basisstationen, die diagonal in einem Abstand von max. 5 m und einer Höhe von mehr als 2 m angebracht werden. Die Abtastung des Raumes erfolgt durch in den Basisstationen eingebaute Infrarot-LEDs und Infrarot-Laser. Die Infrarot-LEDs leuchten den Raum in einer 60 Hz-Frequenz aus. Die beiden Infrarot-Laser senden abwechselnd horizontale und vertikale Infrarot-Laserstrahlen. Auf der Oberfläche des Headsets und der Controller der Vive befinden sich Fotodioden, mit einer vorkonfigurierten Anordnung. Durch diese Fotodioden wird der zeitliche Abstand des Aufleuchtens der Infrarot-LEDs und der Zeitpunkt wann anschließend der Infrarot-Laser auf eine Fotodiode getroffen ist gemessen. Der Unterschied in Zeitpunkt und Reihenfolge, zu dem die verschiedenen Fotodioden vom Laser getroffen werden, ermöglicht die Wiederherstellung der Position und Ausrichtung des Headsets und der Controller. Zusätzlich sind das Headset und die Controller mit Trägheitssensoren zur Messung der Bewegung ausgestattet. Die auf diesem Wege aufgenommen Positionsdaten werden durch eine Sensorfusion mit Positionsdaten des room-scale-Verfahrens kombiniert. Hierdurch werden eine bessere

Stabilität und Genauigkeit der Positionsdaten erzielt. Die Genauigkeit ist vom Hersteller im Submilimeterbereich angegeben und wurde in unterschiedlichen Versuchen mit einer Abweichung von 2 – 3 mm nachgewiesen.

Um das HTC Vive System nutzen zu können, muss die Vive, die Software SteamVR auf einen Rechner installiert und die Hardware angeschlossen werden. Anschließend erfolgt die Ausrichtung des zu trackenden Bereiches mittels der Basisstationen und die Kalibrierung über SteamVR. Ist dies erfolgt kann das System über einen längeren Zeitraum präzise die Hardware tracken. Die Kalibrierung muss beim Wechsel der Versuchsperson nicht angepasst werden, da sie personenunabhängig ist.

4.4 Vergleich der drei Systeme

Die drei Systeme wurden in den vorangegangenen Abschnitten vorgestellt und erläutert. In diesem Abschnitt sollen die für die Aufnahme von Bewegungsdaten einer Arm-Bewegung wichtigen Kriterien, definiert und verglichen werden. Der Aspekt, dass das Sensorsystem und das Referenzsystem Datensätze im Rahmen eines Versuchsaufbaus generieren sollen, wird dabei mitbetrachtet. Die ersten beiden wichtigen und zu überprüfenden Kriterien sind das Messprinzip und daraus resultierend die Möglichkeit der Datenerfassung in Echtzeit und das Datenformat. Des Weiteren soll der Kalibrierprozess unter den Aspekten, Aufwand der Kalibrierung und Fehleranfälligkeit betrachtet werden. Aufgrund des späteren Einsatzes in Versuchsreihen, sind die Kriterien Aufwand der Umrüstung zwischen zwei Versuchspersonen und daraus folgende Fehlermöglichkeiten , relevant. Der abschließende Vergleichsfaktor ist die Genauigkeit der Messdaten der jeweiligen Systeme. In Tabelle 2 sind die genannten Kriterien zusammengefasst und zeigen die jeweiligen Ergebnisse für die überprüften Systeme.

Tabelle 2: Vergleich der möglichen Referenzsysteme [25], [33], [37].

System	Messprinzip	Echtzeitdaten und Datenformat	Aufwand und Fehleranfälligkeit der Kalibrierung	Aufwand bei Umrüstung und Fehlermöglichkeiten	Genauigkeit der Messdaten
CUELA	Goniometer und Inertialsensoren	Echtzeitdaten; Angabe unterschiedlicher Winkel in Grad	Kalibrierungsprozess aufwendig; Kalibrierung in zwei Schritten: 1. Der jeweiligen Sensoren im System 2. Das System auf den jeweiligen Probanden.	Hoher Aufwand, die Kalibrierung auf die Person muss erneut durchgeführt werden. Fehlermöglichkeit hoch.	Die Messgenauigkeit beträgt ±1,7°.
Kinovea	Kameraaufnahmen und Analyse über Software	Keine Echtzeitdaten; Winkeldaten in 2D über Umrechnung 3D Winkeldaten oder Positionsdaten möglich.	Positionierung der Kameras einfach. Wählen der Templates einfach. Bei Überlagerungen evtl. Probleme beim automatischen Tracking und hoher Aufwand durch Nachbearbeitung.	Geringer Aufwand bei der Aufnahme der Bewegung. Aufwand bei der Analyse bleibt bestehen. Fehlermöglichkeit gering bei der Aufnahme, hoch bei der Analyse.	Angaben zur Messgenauigkeit bei Bewegungen sind nicht vorhanden. Aufgrund des Analyseverfahrens, der nachträglichen Bildanalyse, der möglichen Überlagerungsfehler und einer evtl. nötigen manuellen Nachbearbeitung, ist von eine schlechteren Genauigkeit im Vergleich zu den anderen Verfahren auszugehen.
HTC Vive	Infrarot-LEDs, -laser und Trägheitssensoren	Echtzeitdaten; Positionsdaten im erfassten Raum.	Ausrichten Basisstationen einfach. Kalibrierung erfolgt per Anleitung über die Software steamVR.	Aufwand gering. Keine Kalibrierung nötig. Hardware Headset,Controller u.ä. müssen übergeben werden.	Vom Hersteller ist die Genauigkeit im Submillimertebereich angegeben. In Test liegt eine Genauigkeit im Bereich von 2 -3 mm vor.

Der Vergleich zeigt, dass das Kinovea-System keine Echtzeitdaten bietet und die Kalibrierung einfach ist. Allerdings ist der Prozess der Datenanalyse aufwendig und könnte zu Fehlern führen. Angaben über die Messgenauigkeit sind nicht vorhanden. Das CUELA-System liefert Echtzeitdaten, der Kalibrierungsprozess ist aufwendig und muss bei jeder Person individuell angepasst werden. Die Messgenauigkeit ist angegeben mit ± 1,7° angegeben. Beim HTC Vive System werden die Daten ebenfalls in Echtzeit erfasst aber im Gegensatz zu den anderen Systemen als Positionsdaten. Die Kalibrierung ist einfach und erfolgt mit Softwareunterstützung. Der Aufwand beim Umrüsten auf eine neue Testperson ist gering und die Messgenauigkeit ist mit einer Abweichung von 2-3 mm angegeben.

Als Ergebnis aus dem Vergleich der drei Systeme wird die Auswahl auf das HTC Vive System getroffen. Neben der hohen Genauigkeit sind vor allem die gute Handhabbarkeit im Versuchsprozess und die einfache Kalibrierung die Hauptargumente hierfür. Als neue Herausforderung ergibt sich das Datenformat in Positionsdaten. Das bisher gestaltete Sensorsystem erfasst die Daten als Winkeldaten. Um die Datensätze vergleichen zu können müsste dementsprechend eine Angleichung erfolgen. Dies und wie das Referenzsystem die Armbewegung Überkopfarbeit erfasst wird im Kap. 5 dargestellt.

5 Koppelung Referenzsystem und Sensorsystem in Unity

In Kapitel 5 wird zunächst die Software SteamVR und Unity zur Einstellung des Referenzsystems erläutert. Die Umsetzung bezogen auf die Armbewegung Überkopfarbeit wird aufgezeigt. In einem weiteren Abschnitt wird die Herausforderung der Angleichung der Datenformate beschrieben und die Umsetzung mit dem Einbinden des Sensorsystems in Unity dargelegt. Abschließend wird die Idee, beide Systeme über die Spiele-Engine Unity zu koppeln erläutert und die daraus resultierenden Vorteile benannt.

5.1 Unity und SteamVR

Unity ist eine leistungsstarke, plattformübergreifende 2D/3D-Spiele-Engine mit einer benutzerfreundlichen Entwicklungsumgebung. Unity dient Entwicklern dazu die Erstellung von Spielen und Anwendungen für Mobiles, Desktop, Web und Konsolen umzusetzen. Unity wurde 2005 vor allem für die Entwicklung von Videospielen veröffentlicht. Durch Weiterentwicklung und Anpassung eignet sie sich aber auch für die Erstellung anderer Arten von interaktiven Inhalten, wie Animationen, Simulationen oder 3D-Visualisierungen. Die Nutzung der Software ist im studentischen Umfeld zudem kostenlos.

Unity vereinfacht und unterstützt mithilfe von verschiedenen Tools die Entwicklung von 3D-Inhalten. Im eigenen Asset-Store können z.B. Assets wie Plugins und 3D-Modelle heruntergeladen werden. So können mit unterschiedlichen Assets Szenen und Umgebungen zusammengestellt, die Physik und Animationen hinzufügt und die Anwendung erprobt werden. Ein weiter großer Vorteil ist, dass unterschiedliche Skripte hochgeladen oder in den Programmiersprachen C++ und Java selber erstellt werden können.

Im Folgenden wird die Unity-Oberfläche kurz erläutert. Abbildung 20 zeigt die Unity-Benutzeroberfläche im Standardformat.

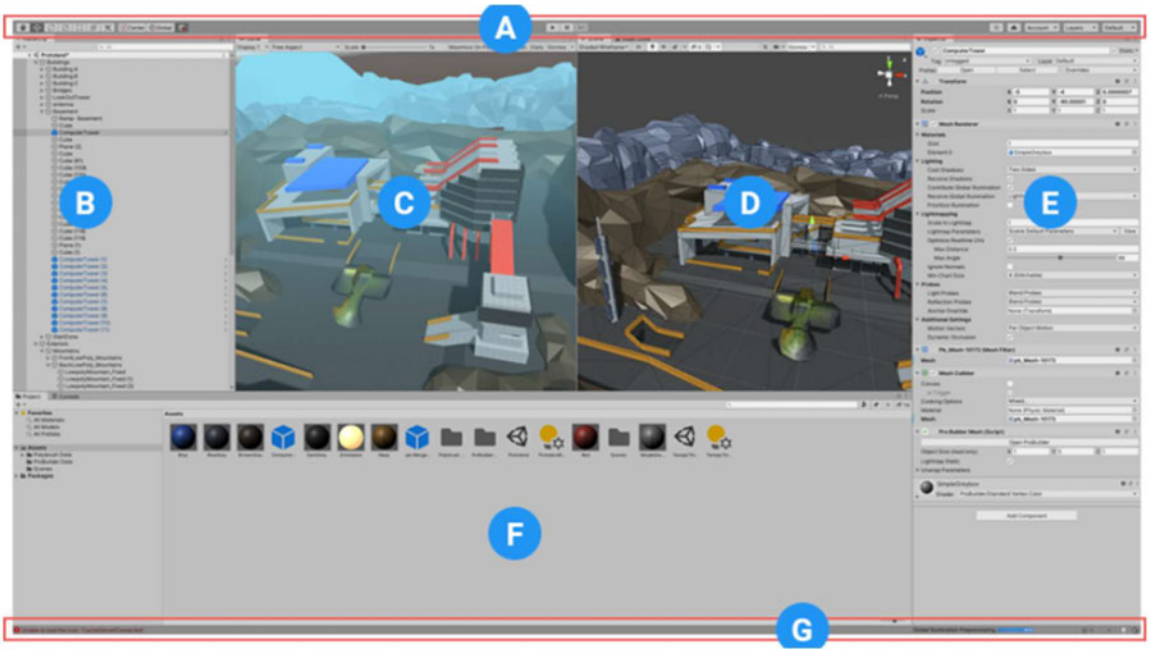


Abbildung 20: Unity-Benutzeroberfläche [38].

A zeigt die Symbolleiste. In C und D sind die Spielperspektive und die Szenenperspektive zu sehen. Hier kann das Endprodukt betrachtet oder in der Szene die jeweiligen Teilelemente bearbeitet werden. Im Inspektorfenster E können alle Eigenschaften des jeweiligen Objektes angezeigt und bearbeitet werden. F ist das Projektfenster, hier können Assets geladen und genutzt werden. Das Hierarchiefenster B zeigt jedes Objekt der Szene und über dieses wird die Hierarchie und damit auch eine evtl. Verkettung festgelegt. Abschließend ist unter G die Statusleiste, über die Benachrichtigungen der Prozesse angezeigt werden, zu sehen.

Ein Asset welches genutzt wird, um die Armbewegung über Unity zu tracken ist das SteamVR Plugin. Das SteamVR Plugin ermöglicht eine Kommunikation zwischen der SteamVR-Software und Unity. Durch das Plugin kann auf die für die Entwicklung nötigten Elemente des VR-Systems zugegriffen werden. Die Verarbeitung der Controllereingaben, sowie das Berechnen, welche Positionierung die Controller und weitere Hardware einnehmen und das Laden der 3D-Modelle von Controllern und Hardware, wird ermöglicht [39].

SteamVR ist eine Software entwickelt von der Valve Corporation, welches die Nutzung von VR-Anwendungen ermöglicht. Unterstützt werden dabei neben dem eigenen HTC Vive System andere Systeme, wie die Oculus Rift und das Windows Mixed Reality Headset. SteamVR bietet, neben der Möglichkeit VR-Anwendungen auszuführen, auch die

Kalibrierung der Hardware. Hierzu ist eine Übersicht über den Status der VR-Hardware, wie das Head Mounted Display und die Controller, eingebunden. Im Kalibrierungsprozess wird die Hardware erkannt und mit dieser ein Bewegungsbereich während der Nutzung von VR-Anwendungen festgelegt. In diesem Bewegungsbereich wird die Hardware präzise erfasst und kann dann für alle VR-Anwendungen genutzt werden, die mit SteamVR gestartet werden. Kommt der Nutzer während einer VR-Anwendung an den Rand dieses Bereiches, wird dies durch die Einblendung eines Gitternetzes in der virtuellen Umgebung signalisiert. Je nach verfügbarem Platz oder der gewünschten Anwendung, kann für den Bewegungsbereich eine sitzende oder stehende Position eingerichtet werden.

5.2 Einstellung des Referenzsystems

Für die Einstellung des Referenzsystems wird zunächst die im oberen Abschnitt beschriebene Software, Unity und SteamVR auf das Notebook installiert. Dann erfolgt die Ersteinrichtung der HTC Vive, dazu muss das Headset mit dem Computer verbunden werden, die Basisstationen werden diagonal in 2 m Höhe platziert und fixiert, sie sollen sowohl das Headset als auch die Controller erkennen. Bei der ersten Initialisierung von SteamVR werden alle Hardware-Komponenten überprüft. Werden alle Hardwarekomponenten des Geräts erkannt, kann die Kalibrierung beginnen. Abbildung 21 zeigt die beiden Möglichkeiten der Raumvermessung.

Abbildung 21: Kalibrierung der HTC Vive über SteamVR [38].

In SteamVR gibt es zwei Optionen, „Nur Stehposition“ und „Raumfüllende VR“. Die Option „Nur Stehposition“ hat keine Mindestanforderungen an den Raum und ist leicht einzurichten. Die Option „Raumfüllende VR“ nimmt mehr Zeit in Anspruch, im dem nach Vorgabe mit dem Controller eine quadratische Fläche zugewiesen werden muss. Für das Referenzsystem wird die Option „Raumfüllende VR“ umgesetzt. Nachdem der Raum eingestellt ist, wird der Mittelpunkt des Raumes und der Boden kalibriert, indem das Headset auf den Boden gelegt wird und manuell die Eingabe der Höhe des Headsets erfolgt. Ist der Kalibrierungsprozess erfolgreich beendet, wird anschließend die Position und Ausrichtung des Headsets und der Controller, von den Basisstationen ohne weitere Eingriffe genau verfolgt.

Um die Armbewegung besser tracken zu können, kommen zwei HTC Vive Tracker zum Einsatz, diese können genauso wie die Controller getrackt werden, bieten aber den Vorteil einfacher und genauer an Ober- und Unterarm positioniert werden zu können. Nach dem Aktivieren der HTC Vive Tracker müssen diese in SteamVR initialisiert werden, um dann in Unity einstellbar und somit die Positionsdaten abgreifbar sind. Abbildung 22 zeigt auf der linken Seite einen Vive Tracker mit angebautem Klettband und rechts das Einstellen der Tracker in SteamVR.

Abbildung 22: HTC Vive Tracker (links) [25] und SteamVR Device manager (rechts) [40].

Abbildung 23: HTC Vive Tracker Positionierung für das Tracken einer Armbewegung [25].

Nach dem Aktivieren der Tracker werden diese in Unity erkannt. Die Abbildung 23 zeigt, wie die Tracker an Ober- und Unterarm angebracht werden sollen, um die Armbewegung zu tracken. Als letzter Schritt müssen im erstellten Unity-Projekt, welches auf 3D eingestellt ist, zwei GameObjects eingefügt werden. GameObjects sind alle Objekte die in eine Szene eingefügt werden und erscheinen im Hierarchiefenster. Das Einfügen dieser Objekte erfolgt, um mithilfe der GameObjects die Positionsdaten über Unity anzeigen zu können. Einfachheitshalber werden zwei Cubes in Unity modelliert und erscheinen in der Szene. Dann werden die Tracker, über das Skript Steam VR_Tracked Object, den Cubes zugewiesen. Das Skript Steam VR_Tracked Object wird über SteamVR bereitgestellt, wie weitere vorinstallierte Skripte, und ermöglicht es die Daten der Controller, Tracker oder weiterer Hardware auf Objekte in Unity zu übertragen. Auf diese Weise können animierte Objekte in Unity über die HTC Vive gesteuert bzw. die Position erfasst werden. Ist die Verknüpfung des Objektes mit dem Tracker erfolgt, kann das Programm ausgeführt werden. Wenn alles richtig eingestellt ist, ist zu sehen das beide Cubes über die HTC Tracker bewegt werden. Die Positionsdaten können im Inspektorfenster für den jeweiligen Cube bzw. Tracker angezeigt werden (siehe Abb. 24). Das Referenzsystem ist damit funktionsfähig.

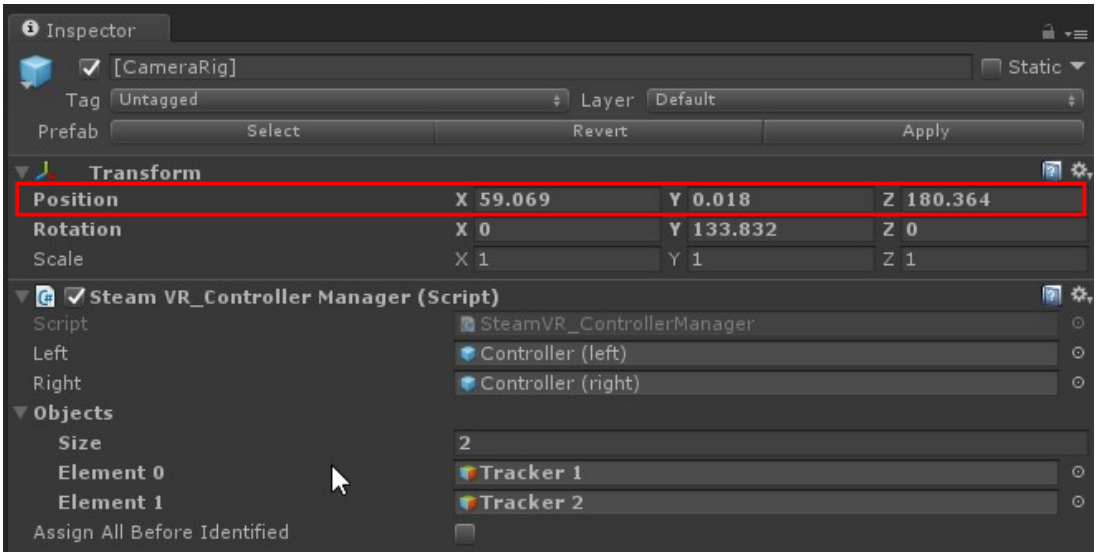


Abbildung 24: Positionsdaten für einen HTC Vive Tracker [25].

Wie beschrieben gelingt es nun mit zwei HTC Vive Trackern die Armbewegung zu erfassen. Für beide Tracker werden in Unity fortlaufend die Positionsdaten im Raum angezeigt. Allerdings muss noch näher definiert werden in welchem Koordinatensystem dies geschieht. Einstellbar sind in diesem Unity-Projekt, das globale Koordinatensystem in Unity oder über das durch SteamVR eingesetzte „CameraRig“, ein Koordinatensystem der Vive. Das CameraRig ist eine Standard-Kamera-Einstellung mit der SteamVR_Camera Komponente und einigen weiteren ergänzenden Komponenten. Durch das CameraRig werden auch alle von der HTC Vive angeschlossenen VR-Komponenten (z. B. Controllern, Basisstationen

und Tracker) zur Verfügung gestellt. Die Koordinatensysteme, die über das CameraRig eingestellt werden können, sind das Koordinatensystem der Basisstation1 und das Koordinatensystem des Head Mounted Displays. Zu bedenken ist bei der Auswahl des Koordinatensystems, dass sich je nach Wahl unterschiedliche Positionsdaten ergeben und das weitere Festlegungen gemacht werden müssen, um bei den Versuchen vergleichbare Daten zu generieren. Bei der Auswahl des globale Unity Koordinatensystems sind die Positionsdaten der Tracker abhängig davon wie das CameraRig zum Koordinatensystem positioniert ist. Die Tracker bewegen sich innerhalb des CameraRigs, dieses kann in der Szene beliebig positioniert werden. Es ergeben sich also ganz unterschiedliche Werte, wenn das Koordinatensystem mittig im CameraRig, an einer Ecke, oberhalb oder in X-, Y- und Z-Achse zum CameraRig versetzt ist. Wird die Basisstation1 als Koordinatensystem gewählt, würden die Positionsdaten innerhalb des CameraRig aus der vorderen oberen oder hinteren oberen Ecke gemessen. Das Koordinatensystem des HMD wäre ebenfalls innerhalb des CameraRig, allerdings ist das HDM beweglich und so könnte je nach Standort das Koordinatensystem variieren. Des Weiteren hängen die Positionsdaten von dem Versuchsaufbau im realen Raum ab. Ist der Aufbau, die Position der Testpersonen, die Anbringung der Tracker, des HDM immer gleich? All diese Variablen sind zu bedenken vor dem Treffen einer Auswahl. Zudem komm noch eine weitere hinzu. Die Gestaltung aus dem Sensorsystem. Vor allem ist dies wichtig, da die Daten für einen Abgleich genutzt werden sollen und daher eine Vergleichbarkeit gegeben sein muss. Daher erfolgt zunächst keine Festlegung des Koordinatensystems für das Referenzsystem. Das Sensorsystem wird betrachtet, um eine geeignete Lösung für beide Systeme zu finden.

5.3 Umstellung des Sensorsystem auf Positionsdaten

In Kapitel 3 wurde die Gestaltung des Sensorsystems beschrieben. Das Sensorsystem wurde mit dem Microkontroller Arduino Nano und zwei Inertialsensoren BNO055 so programmiert, dass die Rotationen beider Sensoren in Quaternion-Daten erfasst und die Rotationsbewegung in VPython visualisiert wurde. Dabei sind die Quaternion-Daten, die die Rotation beschreiben, in vier Zahlenwerten (w-, x-, y-, und z-Anteil) die in Summe die Zahl 1 ergeben dargestellt. Eine Beschreibung einer Armbewegung ist mit diesen Einstellungen noch nicht erzielt. Als weitere Schritte zur Beschreiung der Armbewegung, war angedacht die Quaternion-Daten über das Berechnungsprogramm Matlab in Winkeldaten der X-, Y-

und Z-Achse des Sensors umzurechnen. Das Programm Matlab bietet hierzu Beispielberechnungsformeln an [41]. Die Sensoren haben außerdem die Möglichkeit den erfassten Daten einen Zeitstempel hinzuzufügen. Auf diese Weise könnte die Angabe der jeweiligen Winkelwerte der Sensoren zu einem bestimmten Zeitpunkt dargestellt werden. Eine Aussage zu der Bewegung eines Armes zu treffen, ist ausschließlich auf Basis der Winkeldaten nicht möglich. Um dies zu gestalten ist der Arm als Mehrkörpersystem zu betrachten, bestehend aus Gelenken (Schulter- und Ellenbogengelenk) und verbundenen starren Körpern (Ober- und Unterarm) [49, Kap 1]. Mit festgelegter Position der Sensoren an Ober- und Unterarm, kann die Distanz zum Schulter- und Ellenbogengelenk gemessen werden, es ergibt sich zu den Winkeldaten jeweils ein Vektor. Auf diese Weise könnten die Bewegung des Oberarmes bezogen auf das Koordinatensystem „Schultergelenk“ und die Bewegung des Unterarmes bezogen auf das Koordinatensystem „Ellenbogengelenk“ berechnet und in Positionsdaten der jeweiligen Koordinatensysteme wiedergegeben werden (veranschaulicht in Abb. 25) [42, Kap.3].

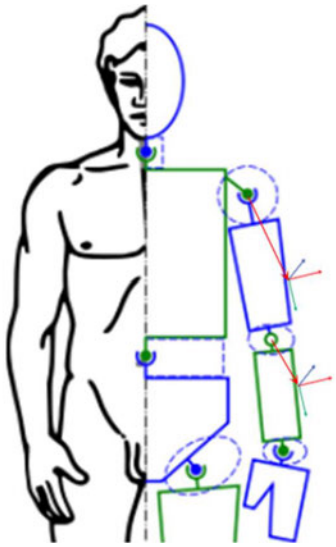


Abbildung 25: Mehrkörpermodell des Armes [31].

Die Beschreibung der gesamten Armbewegung ergibt sich so nicht. Dazu muss entweder ein globales oder ein geeignetes körperfestes Koordinatensystem festgelegt werden, von dem aus die beiden Sensorpositionen berechnet werden. Ein weiterer beachtenswerter-Faktor ist, dass die Position des Unterarmes abhängig ist von der Position des Oberarmes. Die beiden Strukturen sind verkettet. Um dies berechnen zu können muss zwischen den festgelegten Koordinatensystemen eine verkettete Transformationsberechnung erfolgen [50, S. 32-40]. Erfolgt eine Rotation des Schultergelenk-Koordinatensystems muss

die Rotation auch in das Ellenbogengelenk-Koordinatensystem umgerechnet werden. Die Positionsdaten des Sensors, bezogen auf globale oder ein festgelegtes Körperkoordinatensystem, würden über die jeweilige Transformationsberechnung in eines dieser Koordinatensysteme berechnet werden.

Wie beim Referenzsystem ist die Festlegung auf ein Hauptkoordinatensystem, von dem aus die Positionsdaten der Sensoren und damit die Armbewegung, gemessen wird noch nicht erfolgt. Zudem müssten für das Sensorsystem noch die Transformationsberechnungen implementiert werden, um Positionsdaten zu erhalten. Diese Umrechnungen erfolgen für das Referenzsystem automatisch in Unity über die Physik-Engine, die korrekte Kinematik von Objekten und weitere physikalische Gesetzmäßigkeiten sind hier vorinstalliert [43]. Dieser Vorteil soll und kann ebenfalls für das Sensorsystem genutzt werden, da Unity über die serielle Schnittstelle die Sensordaten über den Arduino Nano empfangen kann. Ein weiterer Vorteil der so genutzt werden soll, ist für beide Systeme das gleiche globale Koordinatensystem zu nutzen und so die gleiche Datenbasis zu erzielen. In den folgenden Abschnitten wird aufgezeigt, wie die Implementierung des Sensorsystems in Unity umgesetzt wurde.

5.3.1 Implementierung des Sensorsystems in Unity

Als erstes wird ein neues Projekt in Unity geöffnet. Die Projekteinstellungen müssen angepasst werden, um in Unity Zugriff auf den seriellen Port des Arduino Nano zu bekommen. Dazu werden im Unity Editor über „Bearbeiten“ die „Projekteinstellungen“ geöffnet und dann das Register „Player“ gewählt. Auf der rechten Seite sind die Eigenschaften der "Player Settings" zu sehen. Im Abschnitt "Andere Einstellungen" gibt es den Unterabschnitt "Konfiguration". Hier wird unter „API-Kompatibilitätsebene“ die Net.Referenz von „.NET 2.0“, der Standardeinstellung bei der nicht alle Net.Referenzen genutzt werden, auf „NET. 4.0“ eingestellt. In Abb. 26 ist die Einstellmöglichkeit zu sehen. Mit dieser Einstellung bekommt Unity Zugriff auf die serielle Schnittstelle [44].

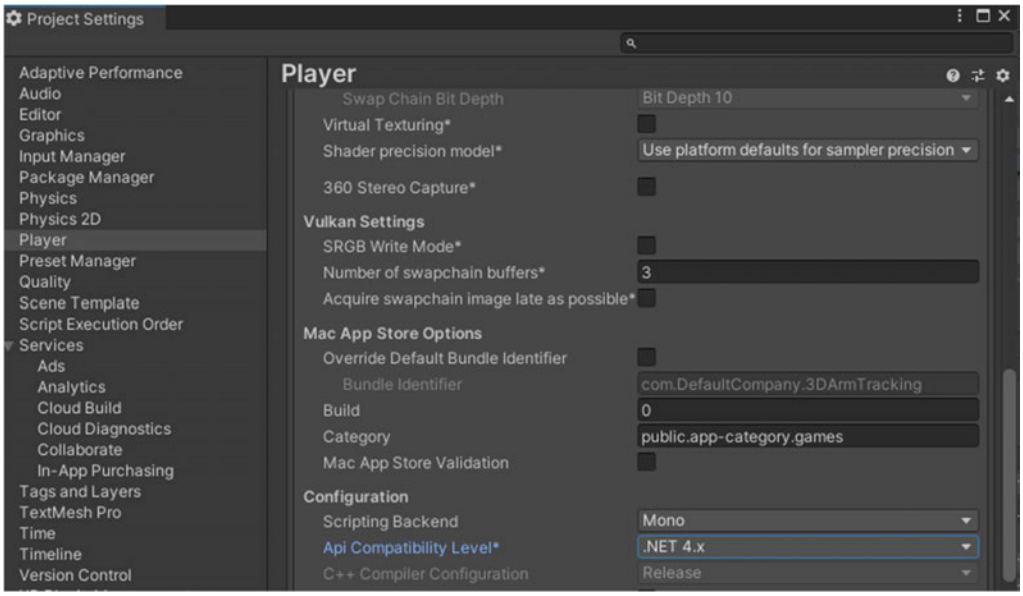


Abbildung 26: Einstellung des Zugriffs auf die serielle Schnittstelle [25].

Nachdem die serielle Schnittstelle lesbar ist, wird ein Skript erstellt um die Sensordaten an Unity zu übermitteln. Dazu wird im Hierarchiefenster auf ein Objekt (z.B. Main Camera) geklickt, rechts wird nun das Inspektorfenster zum Objekt sichtbar. Ganz unten ist die Schaltfläche "Add Component", diese wird angewählt. Es erscheinen Optionen, bei denen "Neues Skript" ausgewählt und benannt wird, z.B. in „arduinoSerialCommunication“. In diesem Skript wird der Serial Port (in diesem Fall COM3) mit der eingestellten Baudrate 115200 eingestellt (siehe Abb. 27) und die Sensordaten die übertragen werden sollen, können nachfolgend programmiert werden.

```
using UnityEngine;
using System.Collections;
using System.IO.Ports;

No asset usages
public class arduinoSerialCommunication : MonoBehaviour {

    public GameObject eins;   Unchanged
    public GameObject zwei;   Unchanged

    public string sendToArduino = "1"; SerialPort stream = new SerialPort("COM3", baudRate: 115200);
```

Abbildung 27: Skript zum Einstellen der seriellen Schnittstelle und zum Programmieren der zu übertragenden Daten [25].

Ob und welche Daten ankommen, kann über den Befehl „Debog.Log()“ im Fensterbereich „Console“ angezeigt werden.

Erster Versuch „Einfaches Arm-Modell“

Nachdem einrichten des Skriptes, wurde die Übertragung der Quaternion-Daten von den Sensoren nach Unity programmiert. In der „Console“ ist nach ausführen des Programmes der Datenstrom ersichtlich. Zum Testen, ob mit diesen Daten auch ein Objekt in Unity gesteuert werden kann, wird ein Testobjekt modelliert (GameObject_Test, ein Würfel, siehe Abb. 28). Im Inspektorfenster des Testobjekts wird das Skript hinzugefügt und dem im Skript definierten „GameObjekt eins“ wird das Objekt „GameObject_Test“ zugewiesen. Jetzt werden die Rotationsdaten des Sensors1 als Quaternion, beim Ausführen des Programmes, auf den Würfel „GameObject_Test“ übertragen und dieser lässt sich in Unity steuern (siehe Dokumentation Skript „arduinoSerialCommunications“).

Für die Nachbildung der Armbewegung wird ein einfaches Arm-Modell erstellt auf das die Sensordaten übertragen werden sollen. Abbildung 28 zeigt das Arm-Modell und das dazu gehörende Hierarchiefenster.

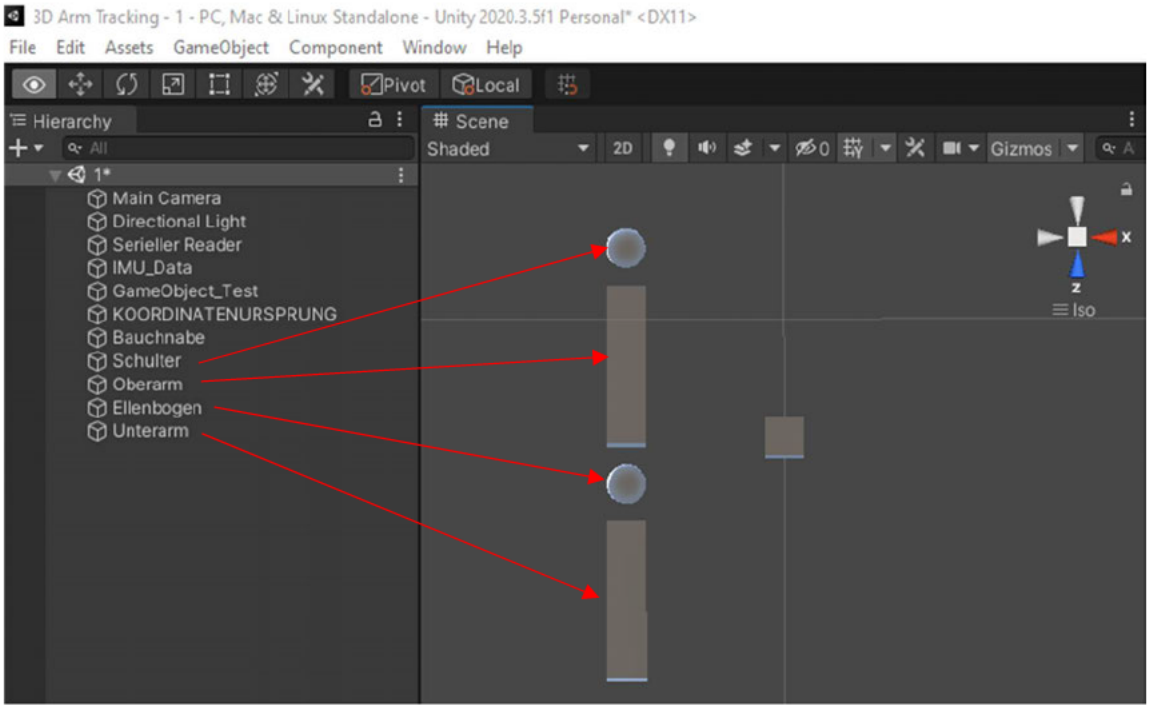


Abbildung 28: Arm-Modell in Unity [25].

Das Arm-Modell besteht aus zwei Kugeln, die die Schulter und den Ellenbogen repräsentieren sollen. Auf diese werden über das Skript die Rotationsdaten der beiden Sensoren übertragen. Dazu sind zwei Quader modelliert worden, die als Modell für den Ober- und Unterarm dienen. Beim Ausführen des Programmes ist zunächst keine Bewegung zu

sehen. Bei näherer Betrachtung, durch anwählen der Objekte Schulter und Ellenbogen, werden die jeweiligen Koordinatensysteme der Objekte sichtbar. Es ist zu sehen, dass die Kugeln rotieren, wenn die Sensoren bewegt werden. Eine Weitergabe der Daten auf die Objekte Ober- und Unterarm erfolgt allerdings nicht. Um dies zu erzielen, müssen in Unity die Objekte „geparentet“ werden, d.h. im Hierarchiefenster in eine „Eltern-Kind-Beziehung“ gesetzt werden. Auf diese Weise werden alle Positions- und Rotationswerte vom Elternobjekt auf das Kindobjekt weitergegeben und das Kindobjekt führt alle seine Bewegungsänderungen in Bezug auf das Elternobjekt und nicht auf die globale Koordinate aus. Abbildung 29 zeigt zwei genutzte Parenting-Varianten.

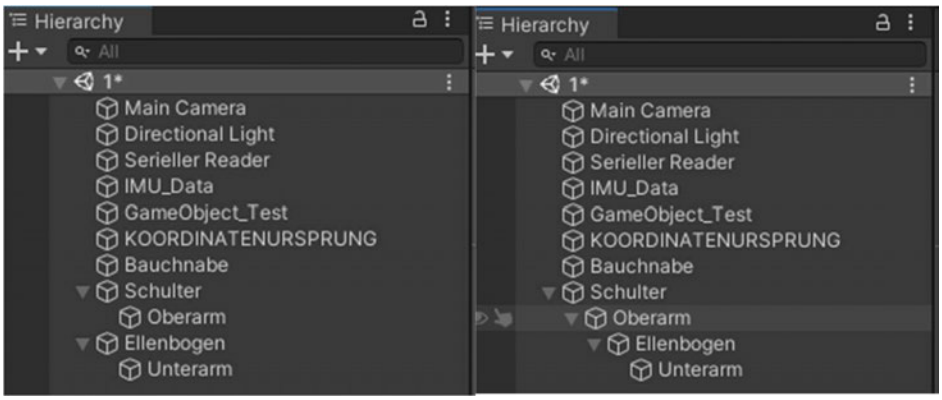


Abbildung 29: Zuweisung „Eltern-Kind-Beziehung“ für das Arm-Modell in Unity [25].

Auf der linken Seite ist zu sehen, dass die Zuweisung „Oberarm zu Schulter“ und „Unterarm zu Ellenbogen“ ist. Daraus ergibt sich eine Rotation des Oberarmes um die Schulter und des Unterarmes um den Ellenbogen aber noch keine Koppelung des Ellenbogens an den Oberarm. Auf der rechten Seite ist die Hierarchie so eingestellt, dass die Bewegung von der Schulter bis zum Unterarm gekoppelt ist. Beim Ausführen des Programmes, konnte die Bewegung des Arm-Modells erfolgen, allerdings ergab sich ein weiteres Problem (siehe Abb. 30).

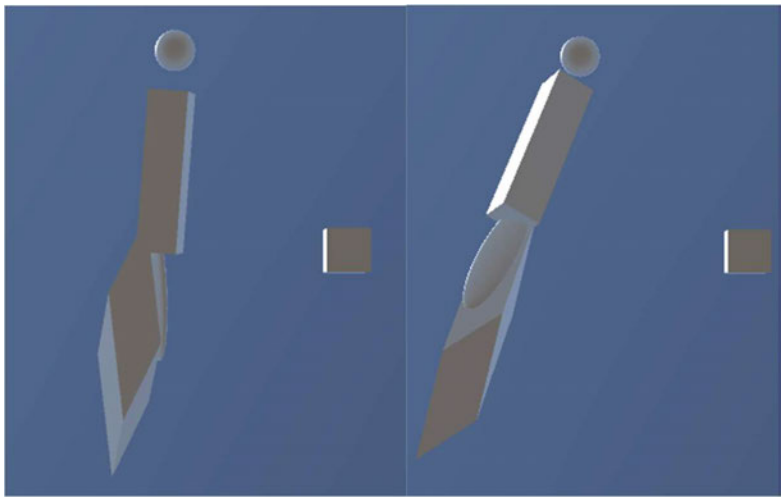


Abbildung 30: Verzerrung der Objekte Ellenbogen und Unterarm bei der Ausführung des Programmes und Übergabe der Sensordaten in Unity [25].

Die Verzerrungen die in Abbildung 30 zu sehen sind, ergaben sich, sobald die Rotation um den Ellenbogen erfolgte. Versuche durch Anpassungen des Parentings oder das Einbringen von Zwischenobjekten die Verzerrungen zu lösen, scheiterten. Es stellte sich heraus, dass beim Modellieren der Objekte die eingestellte Größe (im Transformfenster Scale) als Skalierung den Kind-Objekten weitergegeben wird und deshalb der Verzerrungsfehler entsteht. Um dies zu umgehen, müssen die Modelle in einem anderen Programm erstellt werden, dann in Unity als Asset hochgeladen werden und mit 1 skaliert sein. Im zweiten Versuch soll dieses Problem über den Einsatz eines 3D-Avatar umgangen werden.

Zweiter Versuch „3D-Avatar“

Ein 3D-Avatar ist ein 3D Model welches schon texturiert, gerigged und geskinnt ist. Der Avatar ist also so vorbereitet, dass er in Unity direkt genutzt und Bewegungen simuliert werden können. Es ergibt sich der Vorteil, dass die Posen des Avatars über das schon erstellte Rig (Skelett der Figur) angepasst werden können. Indem die Position und Orientierungen den einzelnen Knochen zugewiesen werden, kann dies erfolgen. Über die kostenlose Software Makehuman konnte ein 3D-Avatar eines Menschen generiert und als Asset in Unity geladen werden [. Im Hierarchiefenster ist für den Avatar die Objektstruktur zu sehen. Hier wurde rausgesucht über welche Rig-Objekte die Armbewegung ausgeführt werden kann. In diesem Fall waren dies „RightArm“ und „RightForeArm“. Im Skript erfolgte die Zuweisung der Rotationsdaten für Schulter und Ellenbogen an diese Rig-Objekte. Abbildung 31 zeigt links die Rig-Struktur und rechts den Avatar.

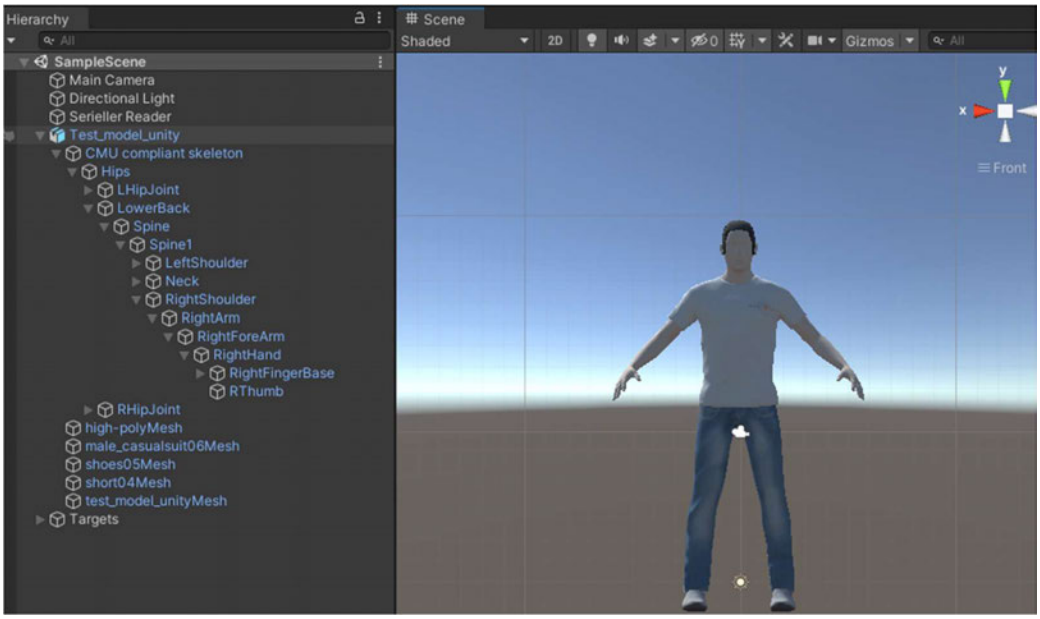


Abbildung 31: Makehuman Avatar und Rig-Struktur in Unity [25].

Beim Ausführen des Programmes konnten die Rotationsdaten auf die Rig-Objekte „RightArm“ und „RightForeArm“ übergeben und eine Armbewegung des rechten Armes umgesetzt werden. Hierbei sind ebenfalls Verzerrungen zu sehen, diese können aber auch nur das Mesh (Gitterstruktur des Avatars) betreffen. Allerdings fällt auf, dass die Rotationen, die in der Realität mit den Sensoren ausgeführt werden, in Unity nicht wiedergegeben werden. In Abbildung 32 ist der Avatar im Gamemodus bei durchgeführter Bewegung dargestellt.

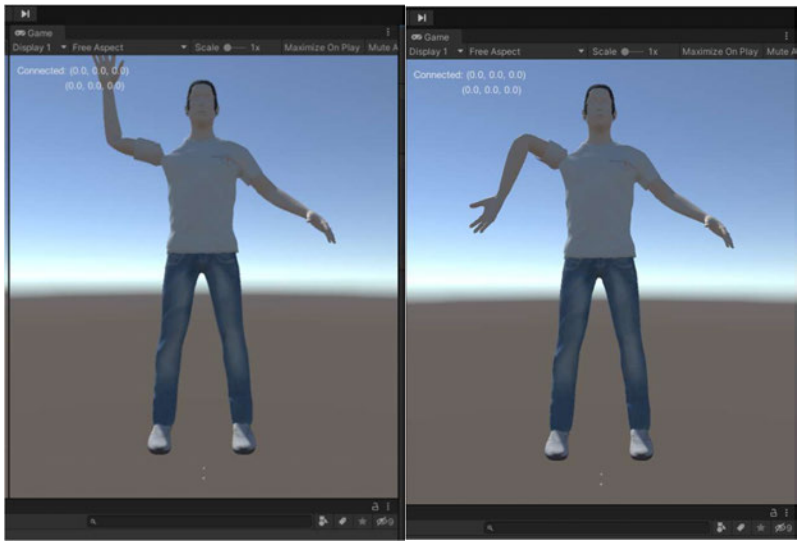


Abbildung 32: Makehuman Avatar und Rig-Struktur in Unity [25].

Um die Bewegung deutlicher nachvollziehen zu können, wurden die Sensoren in der Realität an Ober- und Unterarm angebracht und damit Bewegungen durchgeführt. Weiterhin waren die Bewegungen des Avatars nicht korrekt. Auch war eine Logik nicht erkennbar, denn bei gleicher Ausrichtung der Sensoren und gleicher Bewegung (z.B. Arm gerade von unten nach vorne bewegt) ergaben sich für den Arm des Avataren unterschiedliche Bewegungen für Ober- und Unterarm (z.B. bewegte sich der Oberarm zu Seite und der Unterarm rotierte). Um Fehler der Sensoren auszuschließen, wurde die Wiedergabe der Rotationsdaten in der Visualisierungssoftware VPython (siehe Kap. 3.1.5) überprüft. Hier wurden die Rotationen der Boards weiterhin richtig wiedergegeben. Auch bei Übergabe der Daten an den linken Arm des Avatars bestanden die Fehler fort. Beim Anwählen der einzelnen Rig-Objekte fiel auf, dass die Orientierung der jeweiligen Körperkoordinatensysteme unterschiedlich war, dies könnte ein Grund für die Fehler sein. Diese Ausrichtung konnte manuell für den Avatar nicht angepasst werden. Daher war die nächste Idee ein Modell mit eigenem Rig zu gestalten.

Dritter Versuch „Oberkörper-Modell mit Rig“

Für die Gestaltung eines Modelles mit integriertem Rig wird eine Animationssoftware benötigt. Die freiverfügbare 3D-Grafiksoftware Blender wurde hierzu genutzt. In Blender können Objekte modelliert, texturiert, animiert und weitere Bildbearbeitungen erfolgen. Über die Armature-Funktion kann eine „Knochenstruktur“ für ein Objekt erstellt werden. Diese „Knochenstruktur“ bildet das Rig und kann, nach der Übertragung des Modells in Unity, animiert werden [46]. Abbildung 33 zeigt das erstellte Oberkörper-Modell mit dem integrierten Rig.

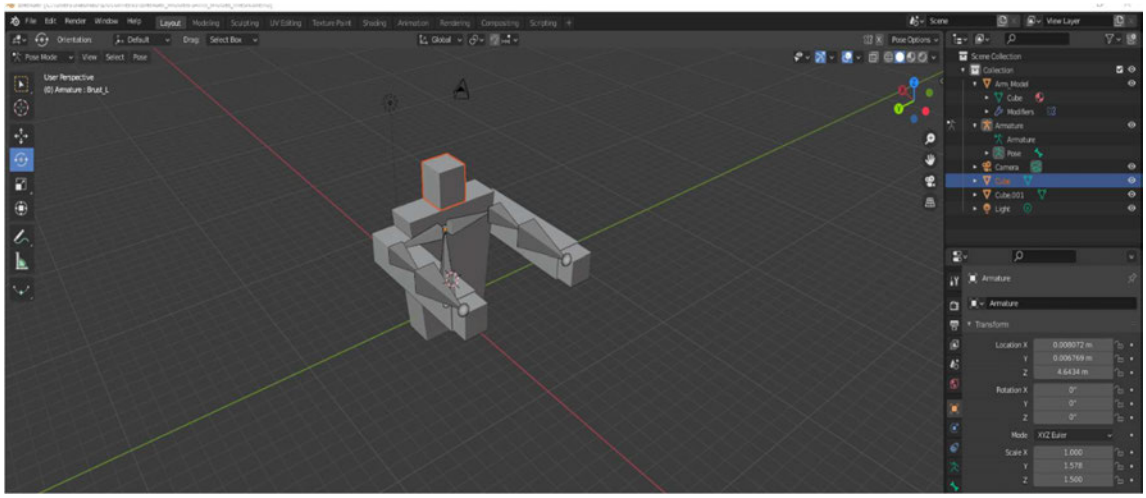


Abbildung 33: Makehuman Avatar und Rig-Struktur in Unity [25].

Beim Übertragen des Blendermodells nach Unity ist auf einige Aspekte zu achten. Wie oben erwähnt können falsch gesetzte Skalierungen beim Animieren von Objekten zu Verzerrungen führen. Um dies zu vermeiden wird das gesamte Oberkörper-Modell mit dem Faktor 1 skaliert. Ein weiterer viel wichtigerer Punkt ist bei diesem Thema aufgefallen. Die Koordinatensysteme, die die Programme Unity und Blender nutzen haben nicht die gleiche Orientierung. In Unity wird ein linkshändiges Koordinatensystem und in Blender ein rechtshändiges Koordinatensystem verwendet. Das heißt in Blender zeigt die Z-Achse nach oben, während sie in Unity nach vorne zeigt und in Unity die Y-Achse nach oben zeigt [47]. Dies könnte auch die Erklärung für die in den bisherigen Versuchen falsch wiedergegebenen Rotationen der Sensoren sein. Denn auch die Sensoren BNO055 nutzen im Gegensatz zu Unity ein rechtshändig orientiertes Koordinatensystem [21, S.24]. Die Anpassung der Koordinatensysteme für das Modell funktioniert über den Export als fbx-Datei. Hier wird die Option „Apply Transform“ aktiviert, so können die Transformationskoordinaten eingestellt werden, damit die Orientierung für Unity funktioniert [47].

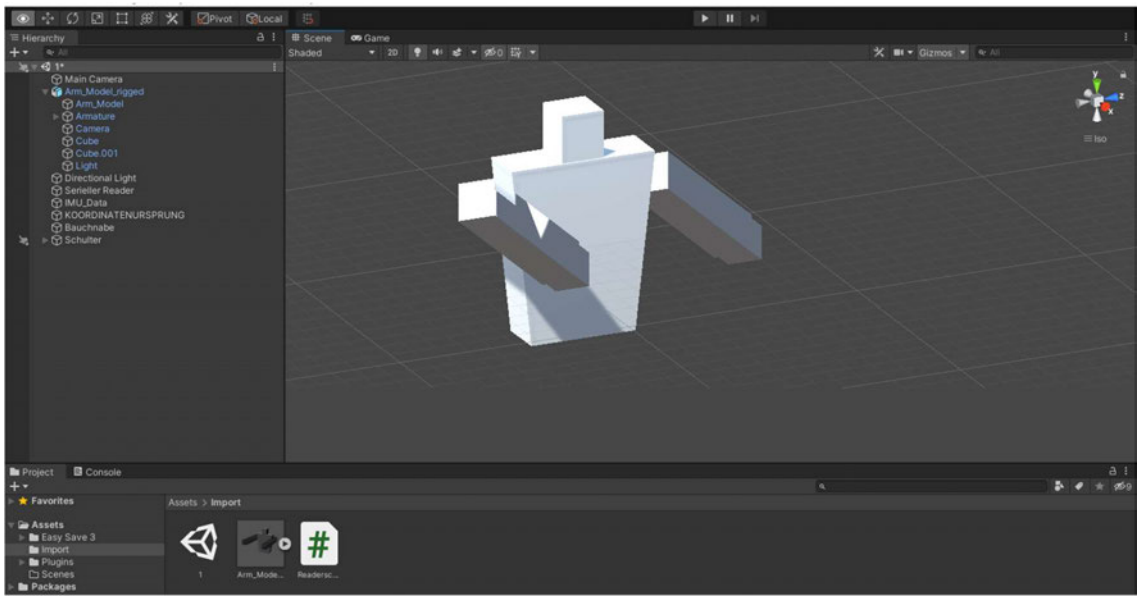


Abbildung 34: Oberkörper-Modell in Unity eingefügt [25].

Wie in Abbildung 34 zu sehen, konnte das Oberkörper-Modell mit den beschriebenen Einstellungen, als Asset in Unity eingefügt werden. Die Transformierung des Koordinatensystems war erfolgreich. Es erfolgte wieder die Übergabe der Sensordaten über das Skript auf die Rig-Objekte „Schulter“ und Ellenbogen“. Beim Ausführen des Programmes konnte der Arm mit den Sensordaten wieder bewegt werden. Aber wie bisher, spiegelte diese Bewegung nicht die reale Bewegung der Sensoren wider. Außerdem ergaben sich wieder

Verformungen des Arm-Meshes. Diese Verformungen würden bei der Positionsmessung, der auf den Arm aufgebrachten Sensor-Objekte, zu weiteren Ungenauigkeiten führen. Die Verformungen können über MeshFilter und MeshRenderer angepasst werden, dies müsste nochmal in Blender erfolgen [48, S. 115].

Bei der Betrachtung der Umsetzung mit dem Oberkörper-Modell sind einige Herausforderung aufgetaucht, die gelöst werden müssen. Die Wichtigste ist die Transformierung der Sensorkoordinatensysteme in Unity, um reale Armbewegungen auf das Modell zu übertragen. Des Weiteren muss für die Bestimmung der Positionsdaten der angebrachten Sensoren auf dem Arm, in Unity ebenfalls jeweils ein Sensor-Objekt integriert werden. Bisher könnte nur jeweils die Position der bestehenden lokalen Objekt -Koordinatensysteme genutzt werden. Je nachdem wo diese am Objekt eingestellt sind, wäre eine Bestimmung der Position möglich. Das bedeutet ein Unterarm-Objekt mit einem lokalen Koordinatensystem an der Stirnseite, würde entweder die Position am Ellenbogen oder am Handgelenk wiedergeben können aber keine mittig auf dem Unterarm. Zudem sind in Realität die Sensoren auf den Armen angebracht und nicht wie bei den Objekt-Koordinatensystemen mittig im Objekt. Hinzu kommt die Herausforderung, dass bei den Versuchen unterschiedliche Probanden zum Einsatz kommen. Die Anatomie dieser Personen wird unterschiedlich sein und auch hieraus soll ein später verwendeter Algorithmus lernen. Das Oberkörper-Modell spiegelt dies aber nicht wider bzw. müsste bei jedem neuen Probanden in Blender eine Anpassung des Modelles erfolgen. Dies wäre aufwendig und wenig praktikabel. Mit diesen Erkenntnissen folgte ein nächster Versuch zur Umsetzung der Armbewegung in Unity.

Vierter Versuch „das programmierte Arm-Modell“

In diesem Versuch soll ein einfaches einstellbares Arm-Modell entstehen. Einstellbar bedeutet, dass die Längen von Ober- und Unterarm auf einfache Weise geändert werden können. Außerdem soll die Position der Schulter bezogen auf die Körpermitte einstellbar sein. Ein Bestimmen der Positionsdaten auf den Bezug Körpermitte soll somit umgesetzt werden. Auf diese Weise kann im Versuch die Anpassung auf die Anatomie des einzelnen Probanden erfolgen und die Handhabung ist erleichtert. Es sollen zwei Sensoren modelliert und auf das Arm-Modell positioniert werden, um eine fehlerhafte Positionsbestimmung aufgrund der Modellierung zu vermeiden. Wie beschrieben, werden die Sensoren auf der Kleidung verrutschen und liefern verrauschte Daten, das Modell soll aber richtig eingestellt sein. Weiterhin muss die Koordinatentransformation von den Sensordaten nach Unity gelöst werden. Um die oben genannten Themen umzusetzen werden wesentliche Teile programmiert

in einem Skript. Dies erfolgt, da z.B. das „Parenting“ der Objekte klarer und direkter definiert werden oder auch um die Anpassung der Längen der Objekte dabei ermöglicht werden kann. Wichtig bleibt wie im vorherigen Abschnitt beschrieben, keine Skalierungsfehler zu begehen, daher werden die Objekte (Quader) in Blender erstellt mit der Skalierung 1 eingestellt und als fbx-Datei in das Projekt geladen.

Als erstes sollte die Koordinatentransformation richtig umgesetzt werden, hierzu wurden die Daten eines Sensors auf einen Quader in Unity übertragen. Mithilfe des Skriptes wurde zunächst versucht die Quaternion in Eulerwinkel wiederzugeben und die Vektoren dann in der jeweiligen X-, Y, und Z-Achse in Grad zu rotieren, um die richtige Orientierung des Koordinatensystems zu erzielen. Dies scheiterte in unzähligen Versuchen. Nach langer Recherche fiel auf, dass beim Senden des BNO055 einer rechtshändige Quaternion das Format (w, x, y, z) ist, in Unity ist eine linkshändige Quaternion aber im (x, y, z, w)-Format beschrieben. Es ergab sich, dass nicht nur die Reihenfolge der Quaternion-Werte andere sind, sondern dass auch eine Vorzeichenänderung nötig ist, um eine richtige Übertragung zu erzielen [49]. Abbildung 35 zeigt einen Ausschnitt der Codierung (gesamter Code in der Dokumentation) mit der die richtige Wiedergabe der Sensordaten erzielt werden konnte.

```
Quaternion(x: -float.Parse(serialData[3]), y: -float.Parse(serialData[1]), z: -float.Parse(serialData[2]), w: float.Parse(serialData[0]));
Quaternion(x: -float.Parse(serialData[7]), y: -float.Parse(serialData[5]), z: -float.Parse(serialData[6]), w: float.Parse(serialData[4]));
```

Abbildung 35: Lösung zur Transformation der Sensordaten [25].

Zu sehen ist die veränderte Reihenfolge der über die serielle Schnittstelle eingelesenen Quaternion-Werte. Die ursprüngliche Reihenfolge startet mit den serialData-Werten 0 bis 3 für die (w,x,y,z)-Komponente des ersten Quaternion von Sensor 1 und die zweite Reihenfolge lief von 4 bis 7 für die (w,x,y,z)-Komponente des zweiten Quaternion von Sensor 2. Die eigentliche Änderung auf das Unity-Format hätte dementsprechend die Reihenfolge (1,2,3,0) und (5,6,7,4) sein sollen. Aber diese ergab weiterhin eine falsche Darstellung. So wurde durch weitere Versuche die richtige Reihenfolge, welche in Unity dem Format (z,x,y,w) entsprach, gefunden. Die übertragenen Rotationen auf den Quader wurden jetzt korrekt wiedergegeben und konnten für das Arm-Modell in das Skript übernommen werden. Die weiteren oben beschriebenen Anforderungen wurden im Skript umgesetzt und über den Befehl .SetParent das „Parenting“ von Schulter zu Oberarm, zu Ellenbogen und Unterarm festgelegt (siehe Abb. 36)

```
Schulter.transform.position = SchulterPosition;
OberarmGO.transform.position = SchulterPosition;
OberarmGO.transform.SetParent(Schulter.transform);
OberarmGO.transform.localScale = new Vector3(OberarmGO.transform.localScale.x, y:OberArmLaenge /10f, OberarmGO.transform.localScale.z);
Vector3 tempPosition = new Vector3(EndPointGO.transform.position.x, y:EndPointGO.transform.position.y - OberArmLaenge, EndPointGO.transform.position.z);
EndPointGO.transform.position = tempPosition;
UnterArmGO.transform.localScale = new Vector3(UnterArmGO.transform.localScale.x, y:UnterArmLaenge /10f, UnterArmGO.transform.localScale.z);
Ellenbogen.transform.position = EndPointGO.transform.position;
Ellenbogen.transform.SetParent(Schulter.transform);
UnterArmGO.transform.position = EndPointGO.transform.position;
UnterArmGO.transform.SetParent(Ellenbogen.transform);
```

Abbildung 36: Skriptcode mit umgesetzten Anforderungen [25].

Die Einstellmöglichkeiten durch das Skript sind im Inspektorfenster umsetzbar. Die Schulterposition kann in X-,Y- und Z-Achse verändert werden. Die Armlängen können in den Feldern darunter eingetragen werden. Die Werte können beim Probanden abgemessen werden und im Skript in Metern eingegeben werden. Die Position „Sensoren am Probanden“ werden ebenfalls gemessen und bei den neu hinzugefügten Objekten „Oberarm-„ und „Unterarmsensor“ im Transformfenster eingetragen. Abbildung 37 zeigt die Eingabefelder und die erfolgten Zuweisungen im Inspektorfenster.

Abbildung 37: Inspektorfenster des Skriptes „Reader“ [25].

Außerdem wurde eine automatische Kalibrierung für das Sensorsystem programmiert. Da die Sensoren zu jedem Zeitpunkt Rotationsdaten liefern und diese zu Ungenauigkeiten

führen können. Wurden im Skript beim Ausführen, die ersten Rotationsdaten abgespeichert und im Folgenden jeweils von den neu aufgenommenen Rotationswerten abgezogen. Dadurch erhält die erste eingenommene Position beim Starten des Programmes die „Nullposition“ und Messung beginnt von dieser Position aus.

Im Hierarchiefenster ist die daraus resultierende Struktur und im Szenenfenster sind die eingefügten Objekte zu sehen (siehe Abb. 38).

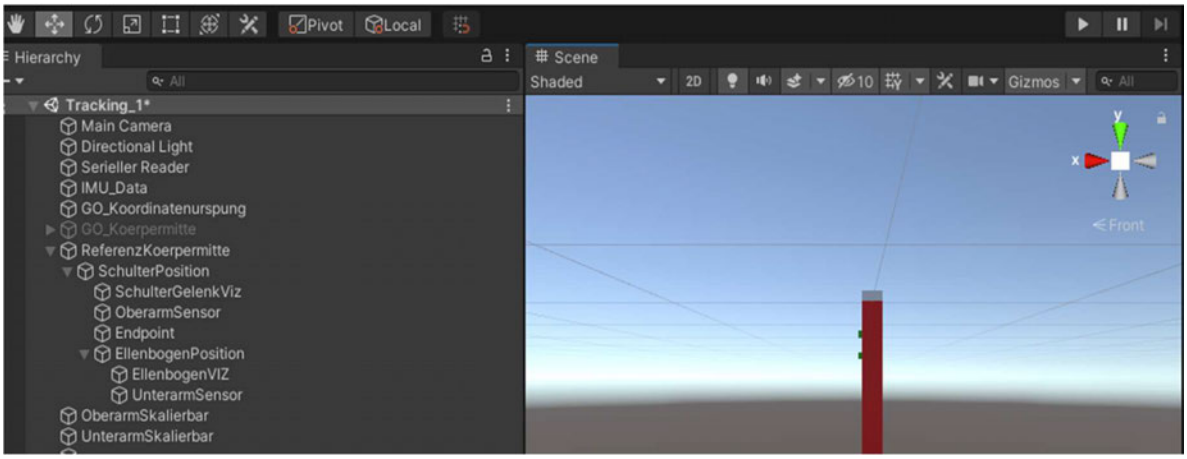


Abbildung 38: Ansicht Hierarchie- und Szenenfenster des Arm-Modells [25].

Die Ausrichtung der Objekte wurde auf das globale Unity Koordinatensystem gewählt. Als Referenzpunkt wurde das Objekt „ReferenzKoerpermitte“ modelliert und mit der Position (0,0,0) im Ursprung des globalen Koordinatensystems positioniert. Für die beiden Objekte „Oberarm-„ und „Unterarmsensor“, im Szenenfenster als kleinen grüne Würfel zu erkennen, können so die Positionsdaten erfasst werden. Durch das Anpassen des Arm-Modells auf den jeweiligen Probanden, ist eine Messung von der Körpermitte möglich. Die Ansicht des Arm-Modells im Szenenfenster wirkt fehlerhaft.Dies liegt daran, dass erst beim Ausführen des Skriptes die Objekte in die vorgesehenen Positionen gefügt werden. Abbildung 39 zeigt den Gamemode drei Folgebilder beim Bewegen des Armes.

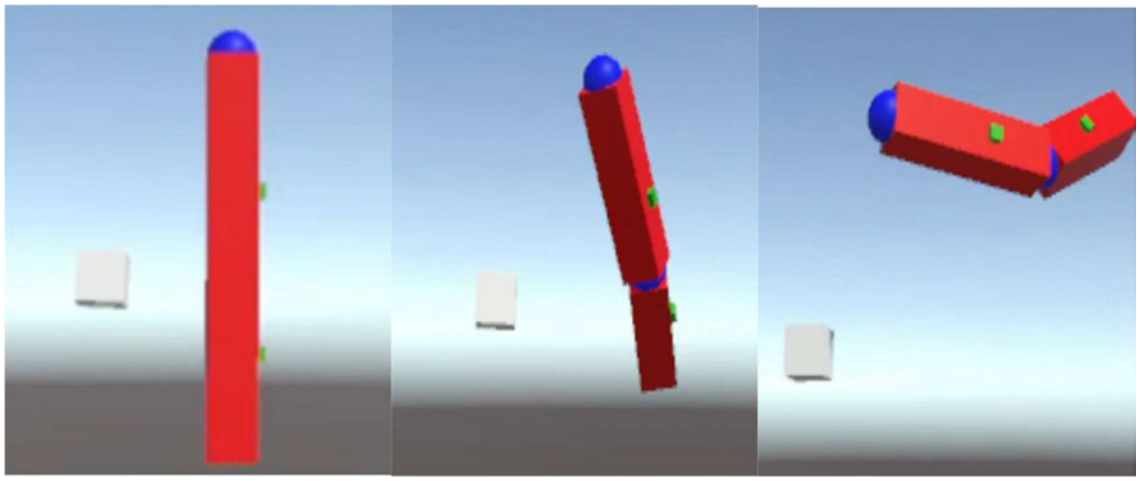


Abbildung 39: Arm-Modell in der 3D-Ansicht im Gamemode [25].

Für eine bessere Unterscheidung der Elemente des Arm-Modells wurden die Objekte Schulter und Ellenbogen blau gefärbt, die Objekte Ober- und Unterarm rot und die Objekte Oberarm- und Unterarmsensor grün.

Die Implementierung des Sensorsystems und die Gestaltung eines auf den Versuch anpassbaren Arm-Modells konnte umgesetzt werden. Im nächsten Abschnitt soll die Umsetzung zum Erfassen und Bereitstellen der Positionsdaten der Sensoren betrachtet werden.

5.3.2 Erfassen und speichern der Positionsdaten

Über das „Reader-Skript“ wurde die Ausgabe der Positionsdaten, für die Objekte Oberarm- und Unterarmsensor, im Gamemode programmiert. Beim Ausführen des Programmes laufen die aktuellen Positionsdaten der Sensoren in der Spielperspektive oben rechts mit. Diese sollen ausgelesen werden und in Tabellenform für eine spätere Analyse verfügbar gemacht werden. Neben den reinen Positionsdaten sind noch weitere Informationen für die weitere Nutzung wichtig. Zu jeder Positionserfassung muss ein Zeitstempel erfasst werden, über diesen ist zum einen die Reihenfolge der Positionssätze nachvollziehbar, zum anderen ermöglicht dieser Zeitstempel die Vergleichbarkeit mit dem Datensatz des Referenzsystems. Dazu muss für den jeweiligen Positionssatz nachvollziehbar sein, von welchem Sensor er stammt. Für die Umsetzung wird nach einer Recherche ein weiteres Plugin „Easy Save 3“ genutzt. Nach dem Einfügen von „Easy Save 3“, wird ein leeres Objekt (ES3Spreadsheet) in die Szene eingefügt, diesem wird das Skript „Save Data“ hinzugefügt. Im Skript „Save Data“ werden die oben benannten Anforderungen programmiert und über

das Objekt „ES3Spreadsheet“ können dem Skript die Objekte Oberarm- und Unteram-Sensor zugewiesen und ausgelesen werden. In Abbildung 40 ist die Umsetzung im Skript zu sehen.

```
void Update()
{
    //sheet.SetCell(0,0, " ");
    string timestamp = DateTime.Now.ToString( format: "MM/dd/yyyy hh:mm:ss.ffff");
    sheet.SetCell( col: 0, row: 0, timestamp);
    sheet.SetCell( col: 1, row: 0,  value: "Sensor1");
    sheet.SetCell( col: 2, row: 0,  value: transform.TransformPoint(Sensor1.transform.position).ToString( format: "n4"));
    sheet.SetCell( col: 3, row: 0,  value: "Sensor2");
    sheet.SetCell( col: 4, row: 0,  value: transform.TransformPoint(Sensor2.transform.position).ToString( format: "n4"));

    sheet.Save( filePath: "Sensordaten.csv",  append: true);
```

Abbildung 40: Skript Save Data [25].

Beim Ausführen des Programmes wird eine Tabelle, im Format „Sensordaten.csv“, parallel miterfasst und nach Beendigung des Programmes, in der letzten Version sowohl in den Assets als auch in einem vorgegebenen Speicherort abgespeichert. Der Datensatz für das Sensorsystem steht somit zur Verfügung.

5.4 Erfassen der Sensordaten für das Referenzsystem und die der Koppelung der Systeme

Nachdem das Sensorsystem fertiggestellt wurde, soll das Referenzsystem passend dazu eigestellt und die daraus resultierenden Sensordaten erfasst werden. Für das Speichern der Daten kann, wie im vorherigen Abschnitt erläutert, das Plugin „Easy Safe 3“ eingefügt und mit den Objekten der beiden Tracker gekoppelt werden. Eine Umbenennung der Objekte von „Sensor“ auf Tracker“ genügt, um den Code anzupassen und auf gleiche Weise nutzen zu können. Jetzt muss noch das gleiche Bezugssystem für die Messung der Positionsdaten eingestellt werden. Beim Sensorsystem wurde das globale Koordinatensystem von Unity mithilfe des Objektes „ReferenzKoerpermitte“ eingestellt. Über die Anpassung des Arm-Modells kann bei dem jeweiligen Probanden aus der Körpermitte (ist zu definieren) gemessen werden. Für das Referenzsystem wird demensprechend ebenfalls das globale Unity Koordinatensystem als Bezug für die Messung gesetzt. Ein Referenzobjekt ist dabei nicht notwendig, da das Objekt „CameraRig“ in dem die Tracker geladen und beweget werden (siehe Kap. 5.2) hierzu genutzt werden kann. Das lokale Koordinatensystem des

„CameraRig“ wird dementsprechend auf (0,0,0) den Ursprung im globalen Koordinatensystem gesetzt. Auf diese Weise haben das Sensor- und das Referenzsystem den gleichen Bezug. Das Referenzsystem ist somit eingestellt und die Positionsdaten der Sensoren können erfasst werden.

Mit diesen Einstellungen wurden die beiden Systeme parallel getestet. Dadurch dass die Programme nur nacheinander gestartet werden können, ergibt sich im Zeitstempel ein leichter Versatz im Hundertstelsekunden-Bereich. Schlimmer ist allerdings, dass Positionen der Sensoren und Tracker eine deutliche Verschiebung auf der Y-Achse (nach oben) aufwiesen. In der Nachbetrachtung fiel auf, dass der Höhenversatz daherkommt, dass bei der Kalibrierung des Referenzsystems die Höhe des Bodens festgelegt wird und das „CameraRig“ auf diese Höhe eingestellt ist. Das bedeutet bei der Messung werden die Positionsdaten der Tracker vom kalibrierten Boden zur Höhe des Armes in der Y-Achse gemessen. Das Arm-Modell misst dagegen von der eingestellten Körpermitte zu den Sensoren. Um diesen Unterschied zu korrigieren gibt es zwei Möglichkeiten: Entweder den kalibrieten Boden höher zu setzten, auf die Höhe der Köpermitte der Probanden oder den Messpunkt für das Arm-Modell auf den Boden zu setzen. Aufwendiger wäre es die Korrektur beim Referenzsystem zu vollziehen, da dies zu Folge hätte, dass bei jedem neuen Probanden die Kalibrierung der HTC Vive wiederholt und angepasst werden müsste. Die Anpassung der Höhe wäre dabei einfach, bei der Abfrage müsste dann das Headset vor die Körpermitte gehalten werden, anstatt es auf den Boden zu legen.

Um den zeitlichen Versatz der Messung zu korrigieren, entstand die Idee beide Projekte in eines zu verschieben. Damit würden die Programme zeitgleich ablaufen und die Aufnahme der Zeiten zur selben Zeit durchgeführt. So wurden alle Objekte und Assets aus dem Projekt des Referenzsystems in das Projekt des Sensorsystems eingefügt. Das Skript „Save Data“ wurde angepasst und die Positionsdaten der Sensoren und Tracker wurden in eine csv-Datei gespeichert. Mit Anpassung der Höhe der HTC Vive konnten so vergleichbare Positionsdaten generiert werden. Nachdem zusammenfügen der beiden Projekte entstand die weitere Idee die beiden Systeme über einen weiteren HTC Tracker zu koppeln. Auf diese Weise könnte die Einstellung der Höhe erspart werden. Der dritte eingesetzte Tracker sollte hierzu dem Objekt „ReferenzKoerpermitte“ zugewiesen werden. Dazu wurde dem Objekt „ReferenzKoerpermitte“ das Steam VR_Tracked Object Skript hinzugefügt, hier kann im Index der gewünschte Tracker ausgewählt werden. Mit Ausführen des Programmes würde das Objekt „ReferenzKoerpermitte“ dann die Position des dritten Trackers im Raum

einnehmen. Da alle weiteren Objekte im Sensorsystem über die „Eltern-Kinder-Beziehung“ an das Objekt „ReferenzKoerpermitte“ gekoppelt sind, würde das gesamte Arm-Modell die Position bezogen auf den dritten Tracker annehmen. Der letzte Schritt um dann für beide Systeme den gleichen Bezug herzustellen, ist es den dritten Tracker mit dem Objekt „CameraRig“ zu verbinden. Ist dies erfolgt, messen sowohl die Sensoren als auch die Tracker von dem neunen Bezug, dem dritten Tracker aus. Damit koppelt der dritte Tracker beide Systeme und muss bei den Versuchen den Probanden an die gewünschte Position Körpermitte angebracht werden. In Abb. 41 ist die Positionierung der Tacker und Sensoren bei einem Test zu sehen.

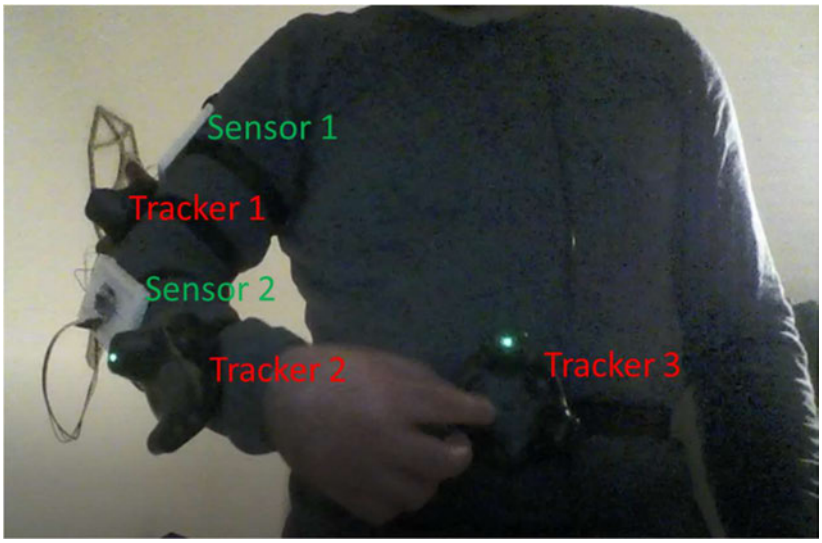
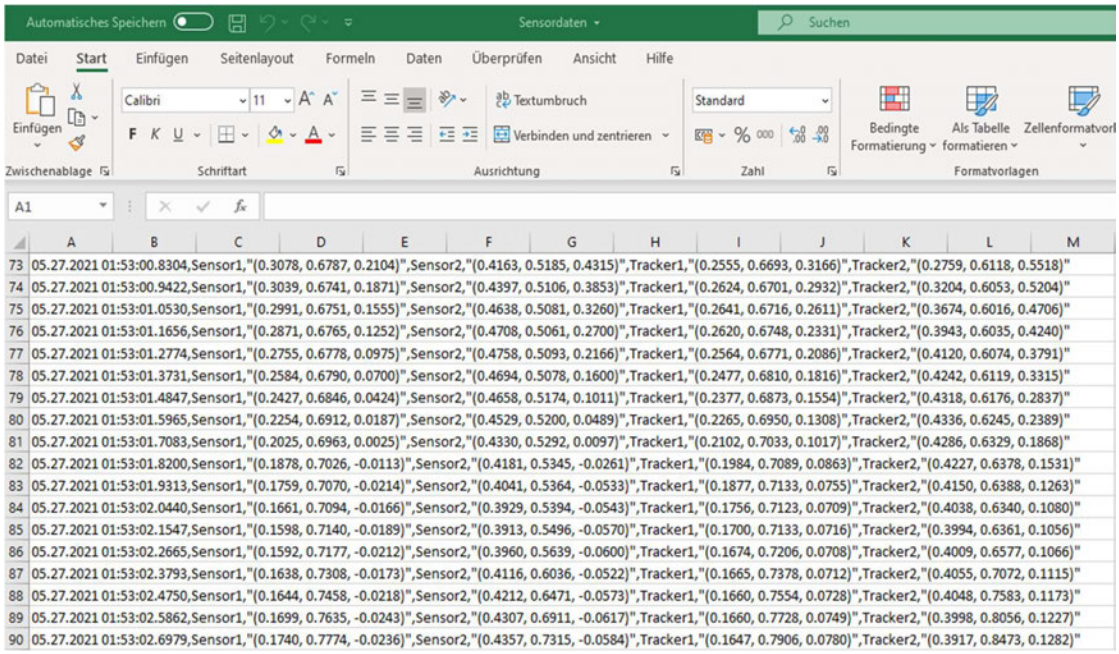


Abbildung 41: Anbringung der Sensoren und Tracker an eine Testperson [25].

Durch diese Lösung können in den Versuchen mit geringem Kalibrierungsaufwand zeitgleich Datensätze für das Sensor- und Referenzsystem generiert werden. Abbildung 42 zeigt einen solchen Datensatz.



	A	B	C	D	E	F	G	H	I	J	K	L	M
73	05.27.2021 01:53:00.8304,Sensor1,"(0.3078, 0.6787, 0.2104)",Sensor2,"(0.4163, 0.5185, 0.4315)",Tracker1,"(0.2555, 0.6693, 0.3166)",Tracker2,"(0.2759, 0.6118, 0.5518)"												
74	05.27.2021 01:53:00.9422,Sensor1,"(0.3039, 0.6741, 0.1871)",Sensor2,"(0.4397, 0.5106, 0.3853)",Tracker1,"(0.2624, 0.6701, 0.2932)",Tracker2,"(0.3204, 0.6053, 0.5204)"												
75	05.27.2021 01:53:01.0530,Sensor1,"(0.2991, 0.6751, 0.1555)",Sensor2,"(0.4638, 0.5081, 0.3260)",Tracker1,"(0.2641, 0.6716, 0.2611)",Tracker2,"(0.3674, 0.6016, 0.4706)"												
76	05.27.2021 01:53:01.1656,Sensor1,"(0.2871, 0.6765, 0.1252)",Sensor2,"(0.4708, 0.5061, 0.2700)",Tracker1,"(0.2620, 0.6748, 0.2331)",Tracker2,"(0.3943, 0.6035, 0.4240)"												
77	05.27.2021 01:53:01.2774,Sensor1,"(0.2755, 0.6778, 0.0975)",Sensor2,"(0.4758, 0.5093, 0.2166)",Tracker1,"(0.2564, 0.6771, 0.2086)",Tracker2,"(0.4120, 0.6074, 0.3791)"												
78	05.27.2021 01:53:01.3731,Sensor1,"(0.2584, 0.6790, 0.0700)",Sensor2,"(0.4694, 0.5078, 0.1600)",Tracker1,"(0.2477, 0.6810, 0.1816)",Tracker2,"(0.4242, 0.6119, 0.3315)"												
79	05.27.2021 01:53:01.4847,Sensor1,"(0.2427, 0.6846, 0.0424)",Sensor2,"(0.4658, 0.5174, 0.1011)",Tracker1,"(0.2377, 0.6873, 0.1554)",Tracker2,"(0.4318, 0.6176, 0.2837)"												
80	05.27.2021 01:53:01.5965,Sensor1,"(0.2254, 0.6912, 0.0187)",Sensor2,"(0.4529, 0.5200, 0.0489)",Tracker1,"(0.2265, 0.6950, 0.1308)",Tracker2,"(0.4336, 0.6245, 0.2389)"												
81	05.27.2021 01:53:01.7083,Sensor1,"(0.2025, 0.6963, 0.0025)",Sensor2,"(0.4330, 0.5292, 0.0097)",Tracker1,"(0.2102, 0.7033, 0.1017)",Tracker2,"(0.4286, 0.6329, 0.1868)"												
82	05.27.2021 01:53:01.8200,Sensor1,"(0.1878, 0.7026, -0.0113)",Sensor2,"(0.4181, 0.5345, -0.0261)",Tracker1,"(0.1984, 0.7089, 0.0863)",Tracker2,"(0.4227, 0.6378, 0.1531)"												
83	05.27.2021 01:53:01.9313,Sensor1,"(0.1759, 0.7070, -0.0214)",Sensor2,"(0.4041, 0.5364, -0.0533)",Tracker1,"(0.1877, 0.7133, 0.0755)",Tracker2,"(0.4150, 0.6388, 0.1263)"												
84	05.27.2021 01:53:02.0440,Sensor1,"(0.1661, 0.7094, -0.0166)",Sensor2,"(0.3929, 0.5394, -0.0543)",Tracker1,"(0.1756, 0.7123, 0.0709)",Tracker2,"(0.4038, 0.6340, 0.1080)"												
85	05.27.2021 01:53:02.1547,Sensor1,"(0.1598, 0.7140, -0.0189)",Sensor2,"(0.3913, 0.5496, -0.0570)",Tracker1,"(0.1700, 0.7133, 0.0716)",Tracker2,"(0.3994, 0.6361, 0.1056)"												
86	05.27.2021 01:53:02.2665,Sensor1,"(0.1592, 0.7177, -0.0212)",Sensor2,"(0.3960, 0.5639, -0.0600)",Tracker1,"(0.1674, 0.7206, 0.0708)",Tracker2,"(0.4009, 0.6577, 0.1066)"												
87	05.27.2021 01:53:02.3793,Sensor1,"(0.1638, 0.7308, -0.0173)",Sensor2,"(0.4116, 0.6036, -0.0522)",Tracker1,"(0.1665, 0.7378, 0.0712)",Tracker2,"(0.4055, 0.7072, 0.1115)"												
88	05.27.2021 01:53:02.4750,Sensor1,"(0.1644, 0.7458, -0.0218)",Sensor2,"(0.4212, 0.6471, -0.0573)",Tracker1,"(0.1660, 0.7554, 0.0728)",Tracker2,"(0.4048, 0.7583, 0.1173)"												
89	05.27.2021 01:53:02.5862,Sensor1,"(0.1699, 0.7635, -0.0243)",Sensor2,"(0.4307, 0.6911, -0.0617)",Tracker1,"(0.1660, 0.7728, 0.0749)",Tracker2,"(0.3998, 0.8056, 0.1227)"												
90	05.27.2021 01:53:02.6979,Sensor1,"(0.1740, 0.7774, -0.0236)",Sensor2,"(0.4357, 0.7315, -0.0584)",Tracker1,"(0.1647, 0.7906, 0.0780)",Tracker2,"(0.3917, 0.8473, 0.1282)"												

Abbildung 42: Sensordaten.csv generiert aus Unity [25].

6 Versuchsaufbau

Die Armbewegung Überkopfarbeit soll in Versuchsreihen mit unterschiedlichen Testpersonen ausgeführt und die Positionsdaten erfasst werden. Auf diese Weise soll für diese Bewegung ein Datenpool generiert werden. In diesem sind die Echtzeitdaten über das Referenzsystem und die verrauschten Sensordaten über das Sensorsystem abgespeichert. Durch die Analyse mit einem eingesetzten Machine Learning Algorithmus, soll durch den Abgleich der Datensätze, eine Identifizierung der Bewegung Überkopfarbeit über die verrauschten Sensordaten erzielt werden. Im Folgenden wird ein möglicher Versuchsaufbau zur Aufnahme der Datensätze skizziert und auf wichtige Rahmenbedingungen für das Referenz- und Sensorsystem hingewiesen.

In Abbildung 43 ist der Vorschlag für einen Versuchsstand dargestellt, indem der Aufbau der Systeme integriert ist.

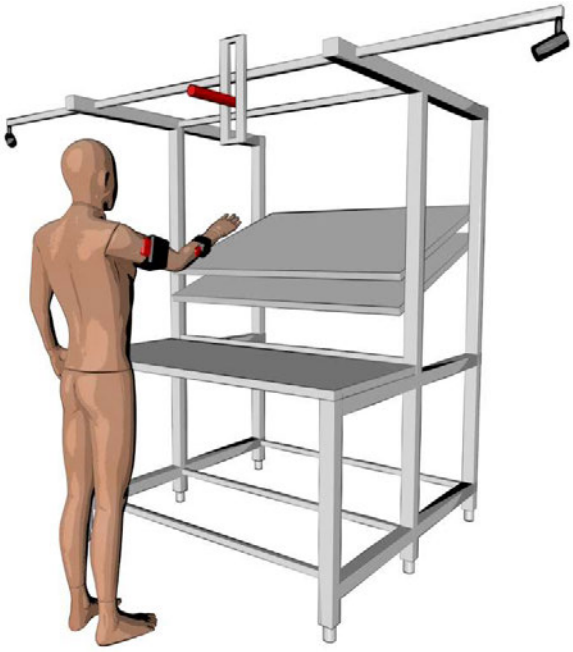


Abbildung 43: Versuchsstand zum Erfassen der Bewegung Überkopfarbeit [25].

Der Arbeitstisch ist multifunktional konzipiert, da im Laufe des Projektes „Echtzeit-Ergonomie“ weitere Bewegungsmuster erfasst werden müssen. Hierzu soll die Möglichkeit bestehen mit einfachen Umbauten arbeitstypische Hebe-, Trage- und Ausführungsbewegungen einzustellen und zu erfassen. Für die Bewegung Überkopfarbeit ist an einer oberen Querstrebe eine einstellbare Positionsstange vorgesehen. Diese kann dazu dienen den oberen Zielpunkt bei dem Bewegungsablauf zu definieren und ist auf die jeweilige Testperson

anpassbar. Zum einen kann hiermit, die für den Versuch noch festzulegende Beschreibung der Bewegung Überkopfarbeit (z.B. ab 30 cm über dem Kopf), eingestellt werden. Zum anderen sind Erweiterungen für die Bewegung, z.B. Schraub- oder Schiebebewegung umsetzbar. Für die Erfassung und Vergleichbarkeit der Positionsdaten ist eine definierte Start- und Endposition im Versuch von Vorteil.

Für das Sensorsystem, welches eine automatische Kalibrierung mit der Startpose erhält (siehe Kap. 5.3.1), muss diese für jeden Versuch festgelegt sein. Für die Erfassung der Überkopfarbeit ist die Startposition, mit Arm gerade nach unten gerichtet seitlich am Körper, festgelegt. Des Weiteren ist für die Sensoren des Sensorsystems ein regelmäßige Kalibrierung sinnvoll, um Driftfehler zu vermeiden. Hierzu es bietet sich an bei jedem Wechsel der Testperson den in Kap. 3.1.4 beschriebenen Kalibrierungsprozess zu durchlaufen. Dabei sollten auch die Steckverbindungen der Verkabelung und die Rückgabe der Quaternion im seriellen Plotter überprüft werden, um Fehler zu erkennen. Das Vermessen der Testpersonen, die Anbringung der Sensoren und das Anpassen des Arm-Modells in Unity erfolgt dann abschließend vor dem Versuchsbeginn.

Für das Referenzsystem müssen die Basisstationen installiert werden. Hierfür gibt es zwei Möglichkeiten, entweder wie in der Abb. 43 dargestellt seitlich am Arbeitstisch frontal zur Testperson oder in diagonaler Aufstellung. Bei der Aufstellung muss bedacht werden, dass die Tracker zu jedem Zeitpunkt der Bewegung durch die Basisstationen erfasst werden müssen. Sind die Basisstationen installiert folgt die Kalibrierung des Referenzsystems wie in Kap. 5.2 beschrieben. Bei erfolgreicher Kalibrierung kann und sollte das Referenzsystem mit diesen Einstellungen über den gesamten Versuchsablauf genutzt werden. Bei erneuter Kalibrierung wäre es schwer genau die exakt gleichen Abmaße des Messraumes zu erzielen (Abschreiten des Raumes mit Controllern). Daher könnte eine Abweichung in den Echtzeitdaten die Folge sein. Bei Anbringung der Tracker an die Testperson wird die Position der Tracker erfasst und in das Arm-Modell als Position für Sensor1 und Sensor2 eingetragen. Hierdurch spiegeln beide Systeme die gleichen Positionen für Tracker und Sensoren wider.

Grundsätzlich sollten die Rahmenbedingungen für den Versuchsablauf festgelegt und idealerweise über alle Versuchsreihen die gleichen Bedingungen herrschen (z.B. Temp., Luftfeuchte u.ä.). Zudem müssen Störquellen anderen magnetischer Felder (z.B. durch weitere elektrische Geräte) vermieden werden, da diese Einfluss auf die Messdaten der Inertialsensoren haben (Störung des Magnetometers).

7 Fazit und Ausblick

Das Gelingen der Umsetzung eines Echtzeit-Ergonomie-Systems, bestehend aus Bekleidungsstücken mit eingebrachten Sensoren und einer App zur Analyse und Bewertung der Belastungen, hängt insbesondere von der Möglichkeit ab aus verrauschten Sensordaten Bewegungsmuster zu erkennen. In dieser Projektphase soll die Umsetzung von der Aufnahme über die Analyse bis hin zur Bewertung der Sensordaten erfolgen.

In dieser Arbeit wurde ein Sensorsystem zur Aufnahme der verrauschten Daten und ein Referenzsystem zur Aufnahme der Echtzeitdaten gestaltet. Durch diese zwei Systeme können zwei Positionsdatensätze für eine Bewegung erfasst werden. Mit diesen Datensätzen sollen Bewegungsmuster aus den verrauschten ermittelt werden können.

Das Sensorsystem wurde mit der Auswahl von zwei BNO055 Inertialsensoren konzipiert. Die Programmierung und Erprobung der Sensoren erfolgte über einen Arduino Nano Mikrokontroller. Für das Visualisieren der erfolgten Bewegungen wurden in VPython 3D-Objekte erstellt und mit den Sensordaten animiert. Über die Visualisierung der Sensordaten konnten Fehler beim Einsatz von Eule-Winkeln erkannt werden. Die Programmierung der BNO055 Sensoren wurde angepasst auf die Wiedergabe der Sensordaten in Quaternion. Die Erfassung der Rotationen in Quaternion führte zu einer stabilen Erfassung der Rotationsbewegung.

Für die Auswahl des Referenzsystem erfolgte ein Vergleich der Systeme CUELA-Messystem, Kinovea-System und HTC Vive System. Die Wahl fiel auf das HTC Vive System. Es überzeugte mit hoher Genauigkeit, guter Handhabbarkeit und einem einfachen Kalibrierungsprozess.

Beim Vergleich der beiden ausgewählten Systeme unterschieden sich die breitgestellten Sensordaten. Das Sensorsystem lieferte Rotationsdaten in Quaternion und das Referenzsystem lieferte Positionsdaten. Die Vergleichbarkeit der aufgenommen Sensordaten wurde durch die Angleichung des Datenformates erzielt. Hierzu erfolgte eine Koppelung des Sensor- und des Referenzsystems über die Spiele-Engine Unity. Für das Sensorsystem wurde ein sich automatisch kalibrierendes, auf die Testperson einstellbares Arm-Modell programmiert. Über den Einsatz eines dritten HTC Vive Trackers, konnte für beide Systeme ein gleicher Bezugspunkt, als Körpermitte der Testperson festgelegt werden. Die aufgenommen Datensätze werden in einer CSV-Datei als Positionsdaten der jeweiligen Tracker und

Sensoren abgespeichert. Ein hinzugefügter Zeitstempel ermöglicht, bei der weiteren Verarbeitung der Daten, die Zuordnung.

Mit dem erfolgreich gestalteten Sensor- und Referenzsystem sollen, im Laufe des weiteren Projektes, Trainings- und Echtzeitdaten in Versuchen aufgenommen werden. Hierzu wird zunächst das Bewegungsmuster Überkopfarbeit betrachtet. Für die Realisierung des Echtzeit-Ergonomie-Systems müssen im Laufe des Projektes eine Vielzahl von weiteren Bewegungsmustern erfasst werden. Hierfür müssen, für die Abbildung von Ganzkörperbewegungen, die Sensorsysteme angepasst und erweitert werden.

Mit den gesammelten Trainings- und Echtzeitdaten, sollen mit dem Einsatz von Machine Learning Algorithmen, Bewegungsmuster aus verrauschten Datensätzen erkannt werden. Beim Machine Learning werden dabei die Daten durch automatische Entscheidungsprozesse gefiltert und der Algorithmus versucht Muster zu erkennen. Hierzu müsste ausgewählt werden über welche Art von Machine Learning die Erkennung der Bewegung erzielt werden soll. Da bei dem Datensatz die Echtzeitdaten zur Verfügung stehen, würde sich hier das „Supervised Learning“ (überwachtes Lernen) anbieten [50]. Hierbei würde der Algorithmus mit den bekannten Daten Muster erkennen und die als Bewegung Überkopfarbeit klassifizieren. Ist die Lernphase abgeschlossen, werden dem Algorithmus Datensätze mit unterschiedlichen Bewegung bereitgestellt, aus diesen muss dann die Unterscheidung in die Bewegung Überkopfarbeit oder nicht erfolgen. Hier kann überprüft werden wie genau die Klassifizierung läuft und ob gegebenenfalls nachgebessert werden muss. Mit welchem Algorithmus dabei gearbeitet wird, welche Vor- und Nachteile vorhanden sind, muss dabei erforscht werden. Häufig genutzte Machine Learning Algorithmen für menschliche Bewegungen sind z.B. Support Vector Machine, neuronale Netze und K-Nearest-Neighbor [50].

8 Literaturverzeichnis

[1] Heumann, St., Landmann, J.: Auf dem Weg zum Arbeitsmarkt 4.0?. Bertelsmann Stiftung, Gütersloh, 2016.

[2] HAW Hamburg: Entwicklung einer Gesundheits-App – Zu viel gebückt? “https://www.haw-hamburg.de/detail/news/news/show/zu-viel-gebueckt/“, Abruf am 27.12.2021.

[3] HAW Hamburg: Projektantrag – Echtzeit-Ergonomie. Hamburg, 2019.

[4] Hakim, M., Hasanov, E.: Echtzeit-Ergonomie. Studienarbeit. Hamburg, 12.06.2020.

[5] Stadel, A.: Ansatzpunkte für ein Bewertungsmodell in der Echtzeit-Ergonomie auf Basis einer Analyse herkömmlicher ergonomischer Bewertungsverfahren. Bachelorarbeit. Hamburg, 31.08.2020.

[6] Hakim, M., Hasanov, E.: Sensorbasiertes System zur Erfassung von ausgewählten Bewegungselementen im Rahmen eines Screening-Systems. Bachelorarbeit. Hamburg, 2021.

[7] Kitagawa, M., Windsor,B.: MoCap for Artists – Workflow and techniques for Motion Capture. Elsevier, Burlington, 2008.

[8] Muybridge, E.: Horse Annie G.galloping with rider. Wikimedia.org. „https://commons.wikimedia.org/wiki/File:Muybridge_race_horse_gallop.jpg#/media/File:Horse_Annie_G._galloping_with_rider_(rbm-QP301M8-1887-626).jpg“, Abruf am 27.12.2021.

[9] Gray, A.: A Brief History of Motion-Capture in the Movies. „https://www.ign.com/articles/2014/07/11/a-brief-history-of-motion-capture-in-the-movies“, Abruf am 27.12.2021.

[10] Sturmann, D.: History of Motion Capture for Computer Character Animation. „https://www6.uniovi.es/hypgraph/animation/character_animation/motion_capture/history1.htm“, Abruf am 27.12.2021.

[11] Colombo, G., Facoletti, G., Rizzi, C.: Virtual Testing Laboratory for Lower Limb Prosthesis, Computer-Aided Design and Applications, 2013.

[12] Menache, A.: Understanding Motoin Capture for Computer Animation. Elsevier, Burlington, 2011.

[13] Jackel, D., Neureither, St., Wagner, F: Methoden der Computeranimation. Springer Verlag, Berlin, 2006.

[14] Viscircle: Einsteigerguide: Was versteht man eigentlich unter Motion Capture?. „https://viscircle.de/einsteigerguide-was-versteht-man-eigentlich-unter-motion-capture-2/“, Abruf am 27.12.2021.

[15] Dörner, R., Broll, W., Grimm, P. Jung, B.: Virtual und Augmented Reality (VR / AR). Springer Verlag, Berlin, 2013.

[16] Generationrobots.com: IMU und Robotertechnik: Was Sie wissen sollten. „https://www.generationrobots.com/blog/de/imu-und-robotertechnik-was-sie-wissen-sollten-2/“, Abruf am 27.12.2021.

[17] Wild-Pfeifer, F.: Das Potential von MEMS-Inertialsensoren zur Anwendung in der Geodäsie und Navigation. C.H.Beck, München, 2015.

[18] Arduino: „https://store.arduino.cc/products/arduino-nano“, Abruf am 27.12.2021.

[19] Arduino: Nano guide. „https://docs.arduino.cc/hardware/nano“, Abruf am 27.12.2021.

[20] Adafruit: BNO055. „https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor“, Abruf am 27.12.2021.

[21] Datasheet BNO055: „https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bno055-ds000.pdf“, Abruf am 27.12.2021.

[22] SerialPlot: „https://hackaday.io/project/5334-serialplot-realtime-plotting-software“, Abruf am 27.12.2021.

[23] Processing: „https://processing.org/de/“, Abruf am 27.12.2021.

[24] IDLE Python: „https://www.python.org/“, Abruf am 27.12.2021.

[25] Eigne Abbildung, Nabil Toumi.

[26] Euler-Winkel: „https://iludis.de/wp-content/uploads/2019/05/IMU_Robotik.pdf“, Abruf am 27.12.2021.

[27] Rose,D.: Rotations in Three-Dimensions: Euler Angles and Rotation Matrices. „https://danceswithcode.net/engineeringnotes/rotations_in_3d/rotations_in_3d_part1.html“, Abruf am 27.12.2021.

[28] Pedley, M.: Tilt Sensing Using a Three-Axis Accelerometer. Freescale Semiconductor,2013.

[29] Esfandyari, JDe Nuccio, R, Xu,G.: Solutions for MEMS sensor fusion. „https://eu.mouser.com/applications/sensor_solutions_mems/“, Abruf am 27.12.2021.

[30] Lu, S., Zhang, X., Wang, J.: An IoT-Based Motion Tracking System for Next-Generation Foot-Related Sports Training and Talent Selection. „https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8253628/“, Abruf am 27.12.2021.

[31] Ellegast, R: Messung von Muskel-Skelett-Belastungen mit dem CUELA-Messsystem. IFA Nr. 0013. Deutsche Gesetzliche Unfallversicherung, Berlin, 2013.

[32] CUELA: „https://www.dguv.de/medien/ifa/de/pub/rep/pdf/rep04/biar0704/kap2.pdf“, Abruf am 27.12.2021.

[33] Messgenauigkeit CUELA: Ganzkörper-Vibrationen beim Fahren von Kompaktkehrmaschinen. IFA-Report 1/2017. Deutsche Gesetzliche Unfallversicherung, Berlin, 2017.

[34] Kinovea : „https://www.kinovea.org/“, Abruf am 27.12.2021.

[35] Sahling, G.: Adaptionen einer Open Source Tracking Software (Kinovea) zur Verbesserung der Anwendbarkeit im wissenschaftlichen Kontext. „https://utheses.univie.ac.at/detail/38134#“, Abruf am 27.12.2021.

[36] HTC Vive System: „https://www.vive.com/de/“, Abruf am 27.12.2021.

[37] Bauer,P,Lienhart, W., Jost, S.: Genauigkeitsuntersuchung eines VR Systems zur 3D-Koordinatenbestimmung. Fachbeitrag AVN, 2020.

[38] Unity: „https://docs.unity3d.com/Manual/UsingTheEditor.html“, Abruf am 27.12.2021.

[39] SteamVR: „https://www.steamvr.com/de/“, Abruf am 27.12.2021.

[40] HTC Vive Einrichten: „https://www.vive.com/de/support/vive/category_howto/resetting-the-play-area.html“, Abruf am 27.12.2021.

[41] Matlab Berechnung: „https://de.mathworks.com/help/nav/ref/imusensor-system-object.html“, Abruf am 27.12.2021.

[42] Woernle, Ch.: Mehrkörpersysteme - Eine Einführung in die Kinematik und Dynamik von Systemen starrer Körper. Springer Verlag, Berlin, 2016.

[43] Unity: „https://unity.com/de“, Abruf am 27.12.2021.

[44] Communication Unity and Arduino: „https://create.arduino.cc/projecthub/raisinga-wesome/unity-game-engine-and-arduino-serial-communication-12fdd5“, Abruf am 27.12.2021.

[45] Makehuman: „http://www.makehumancommunity.org/“, Abruf am 27.12.2021.

[46] Blender: „https://www.blender.org/“, Abruf am 27.12.2021

[47] How to Import Blender Models into Unity – Your One-Stop Guide: „https://gamede-vacademy.org/how-to-import-blender-models-into-unity-your-one-stop-guide/“, Abruf am 27.12.2021.

[48] Seifert, C. Wislaug, J.: Spiele entwickeln mit Unity 5: 2D- und 3D-Games mit Unity und C# für Desktop, Web & Mobile. Carl Hanser Verlag, 2017.

[49] Koordinatentransformation in Unity: „https://forums.adafruit.com/viewto-pic.php?t=81671“, Abruf am 27.12.2021.

[50] Wuttke, L: Machine Learning: Definition, Algorithmen, Methoden und Beispiele. „https://datasolut.com/was-ist-machine-learning/“, Abruf am 27.12.2021

9 Selbstständigkeitserklärung

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Dieses Blatt, mit der folgenden Erklärung, ist nach Fertigstellung der Abschlussarbeit durch den Studierenden auszufüllen und jeweils mit Originalunterschrift als letztes Blatt in das Prüfungsexemplar der Abschlussarbeit einzubinden.
Eine unrichtig abgegebene Erklärung kann -auch nachträglich- zur Ungültigkeit des Studienabschlusses führen.

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: Nabil

Vorname: Toumi

dass ich die vorliegende Masterarbeit ☐ bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Modellierung eines Prototyps für die Echtzeit-Ergonomie zur Ableitung eindeutiger Bewegungsmuster aus körper-nahen Sensordaten am Beispiel von Hand-Arm-Bewegungen.

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

- die folgende Aussage ist bei Gruppenarbeiten auszufüllen und entfällt bei Einzelarbeiten -

Die Kennzeichnung der von mir erstellten und verantworteten Teile der -bitte auswählen- ☐ ist erfolgt durch:

Hamburg	29.12.2021	[redacted]
Ort	Datum	Unterschrift im Original