

NaoVR – Nao Access Over Virtual Reality 2.0

Studienarbeit

Für die Prüfung zum

Bachelor of Science

des Studienganges Informatik

an der Dualen Hochschule Baden-Württemberg Karlsruhe

von

Vincenzo Ciullo, Dennis Wiebe

Abgabedatum	17.05.2021
Bearbeitungszeitraum	25 Wochen
Matrikelnummer	8683078 / 6735868
Kurs	TINF18B5
Ausbildungsfirma	IQS Software GmbH / Stadler + Schaaf Mess- und Regeltechnik GmbH
Betreuer	Prof. Dr. Marcus Strand
Gutachter der Dualen Hochschule	Prof. Dr. Marcus Strand

Eidesstattliche Erklärung

Gemäß § 5 (3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 29. September 2017.

Ich versichere hiermit, dass ich die Projektarbeit mit dem Titel

NaoVR – Nao Access Over Virtual Reality 2.0

Selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum

Dennis Wiebe

Ort, Datum

Vincenzo Ciullo

Abbildungsverzeichnis

Abbildung 1: VM Settings - Network Adapter.....	5
Abbildung 2: VM Settings - Memory	6
Abbildung 3: Beispiel für die Verwendung der HTC Vive Tracker, [8].....	12
Abbildung 4: Kaputte Kamerahalterung	15
Abbildung 5: Einnehmbare Posen des Nao-Roboters [9]	17
Abbildung 6: Community Aldebaran Webseite [10]	19
Abbildung 7: Developer Center Webseite.....	20
Abbildung 8: Turnigy nano Tech 300mAh 2s und Compact Charger.....	22
Abbildung 9: Erhalten der IP-Adresse der VM	28
Abbildung 10: Starten der Kernprogramme mit "roscore"	29
Abbildung 11: Ausschnitt zur Erstellung eines ROSBridge_Servers	30
Abbildung 12: Startet die Verbindung mit dem Nao mit der Python SDK	31
Abbildung 13: Alle Terminals zum Starten von ROS & Verknüpfung mit Nao	32
Abbildung 14: Ausschnitt aus Blender, 3D-Modell für Kamerahalterung	33
Abbildung 15: Vollständig reparierte Kamerahalterung	35
Abbildung 16: Point of View des Nao-Roboters.....	37
Abbildung 17: Ausschnitt aus Pythonskript.....	41
Abbildung 18: C# Methode zum Ausführen von Pythonskripten.....	42
Abbildung 19: Bild anhand, welches das Robotermodell eingefärbt wird.....	43
Abbildung 20: Prototyp der Anbringung der Tracker.....	49
Abbildung 21: Finale Anbringung der Tracker am Anwender	51
Abbildung 22: Code für das Laufen des Roboters	55
Abbildung 23: Code für das in die Hocke gehen des Roboters	57
Abbildung 24: Code für die Ermittlung der korrekten Rotationsrichtung	59
Abbildung 25: Ergebnis der Testanwendung SpeechRecognitionTest	60
Abbildung 26: Veränderte Bedienung der Controller der HTC Vive.....	63
Abbildung 27: Ausschnitte aus einem Video.....	67
Abbildung 28: Ausschnitt der Ausstattung der verschiedenen Nao Modelle	72

Abkürzungsverzeichnis

VM	Virtuelle Maschine
RAM	Random-Access Memory
ROS	Robot Operating System
VR	Virtual Reality
LED	light-emitting diode
SDK	Software Development Kit
IP	Internet Protocol
FPS	Frames per Second

Inhaltsverzeichnis

Abbildungsverzeichnis	III
Abkürzungsverzeichnis.....	IV
1 Einleitung.....	1
1.1 Motivation und Zielsetzung.....	1
1.2 Stand der Technik.....	3
1.3 Entwicklungsumgebung.....	4
1.4 Vorgehensweise	7
2 Grundlagen.....	8
2.1 Nao-Roboter	8
2.2 HTC VIVE Pro & Zubehör.....	10
3 Anforderungsanalyse	13
4 Probleme des aktuellen Stands	14
4.1 Kamerahalterung & Kameras	14
4.2 Funktionalität Greifen.....	15
4.3 Posen des Nao-Roboters	16
4.4 Community Aldebaran	18
4.5 Batterien für Kameras defekt.....	21
4.6 Roboter gebraucht.....	23
4.7 Laboraufenthalte & Corona	24

5	Umsetzung	26
5.1	Inbetriebnahme VM und Verbindung zu ROS.....	26
5.2	Kamerahalterung	33
5.3	Anpassung der Kameras	36
5.4	Funktionalität Greifen.....	38
5.5	Implementation der Posen des Nao-Roboters	40
5.6	Nao's Design in VR	43
5.7	Anpassung der States	44
5.8	Tracker in Steam VR	46
5.9	Tracker in Unity.....	46
5.10	Tracker am Benutzer	47
5.11	Auswertung Tracker.....	52
5.12	Synchronisierung der Bewegungen	52
5.13	Sprachfeature des Roboters.....	60
5.14	Veränderte Bedienung der Software	62
6	Evaluation.....	64
6.1	User Experience	64
6.2	Latenz in der Ausführung	66
7	Fazit.....	68
7.1	Zusammenfassung	68
7.2	Ausblick.....	69
	Literaturverzeichnis	VII

1 Einleitung

1.1 Motivation und Zielsetzung

Diese Studienarbeit ist ein Folgeprojekt zur Studienarbeit „NaoVR - Nao Access Over Virtual Reality“ von Janis Schneider und Dennis Jahnke. Das Projekt, auf dem diese Studienarbeit beruht, wurde an der dualen Hochschule Baden-Württemberg in Karlsruhe im Jahr 2019/2020 bearbeitet. Das Ziel der Arbeit war die Steuerung des Roboters durch ein HTC VIVE System und der passenden Controller. Des Weiteren wurde am Kopf des Roboters eine Halterung befestigt, in der zwei Kameras eingefasst waren, sodass der Anwender durch die Augen des Roboters sehen kann [1]. Dies stärkt die Immersion der Anwendung und gibt dem Anwender das Gefühl, tatsächlich im Cockpit des Roboters zu sitzen.

Die Umsetzung dieser Arbeit liegt nahe an der Vorgängerarbeit und hat den Zweck das bestehende System zu erweitern und somit zu verbessern. Die Idee ist, das System wiederherzustellen, ausgehende Probleme zu analysieren und Lösungsalternativen zu entwickeln. Dazu gehört zunächst die Greifmöglichkeiten des Nao durch den Controller zu verbessern sowie die Laufbewegungen und Beugebewegung nicht anhand von Betätigen eines Knopfes auf den HTC VIVE Controllern zu ermöglichen. Somit soll die generelle Bewegung des Nao-Roboters nicht nur mit den Armen des Benutzers wie bisher, sondern mit zusätzlichen HTC VIVE Trackern an den Beinen synchronisiert werden. Hierzu müssen die Bewegungen an den Trackern so aufgenommen werden, dass der Nao-Roboter nicht das Gleichgewicht beim Nachahmen der Bewegungen verliert und umfällt. Eine weitere Möglichkeit besteht darin mehr Interaktion zwischen Benutzer und

dem Nao mittels eines Mikrofons herzustellen, sodass der Nao etwas Gesprochenes aufnehmen und wiedergeben kann.

Sollte noch zusätzlich Zeit für Erweiterungen bestehen, können zusätzliche Funktionen in Erwägung gezogen werden. So kann eine neu entwickelte Künstliche Intelligenz (KI) oder eine bestehende KI verwendet werden, dass der Nao beispielsweise auf Fragen antworten kann oder anhand von Sprachkommandos Bewegungen ausführt.

1.2 Stand der Technik

Die vorgehende Studienarbeit „Nao VR – Nao Access Over Virtual Reality“ von Janis Schneider und Dennis Jahnke implementiert bereits eine Möglichkeit den Nao-Roboter mit Hilfe einer HTC VIVE Brille und Controllern zu bewegen. Durch das Wechseln in einen Status, in dem die Bewegungen der Controller nicht mehr interpretiert werden, ist es sogar möglich den Roboter durch Knopfdruck zum Laufen, Drehen oder in die Knie gehen zu bringen [1]. Jedoch nimmt diese Entscheidung, erst den Status wechseln zu müssen, die Immersion, tatsächlich im Cockpit eines Roboters zu sitzen und diesen zu steuern.

Des Weiteren sind bei der Instandsetzung der alten Version der Arbeit einige Probleme der Umsetzung erkannt worden, welche im Verlauf der Studienarbeit verbessert werden (siehe Punkt *Umsetzung*).

1.3 Entwicklungsumgebung

Die Entwicklungsumgebung für diese Studienarbeit besteht aus zwei Rechnern. Ein Rechner, auf dem ein Ubuntu-System gehostet wird und ein Rechner für die Programmierung der Studienarbeit. Auf beiden Rechnern läuft ein Windows 10 Betriebssystem. Auf dem Host-PC wird ein 64bit Ubuntu-System der Version 14.04 gehostet. Die Virtualisierung erfolgt über eine lizenzierte VM Workstation 12 pro [2] in der Version 12.5.9.

Für die Programmierung der Studienarbeit ist auf dem Entwicklungsrechner Unity in der Version 2019.3.7 installiert. Da die Programmierung in .NET erfolgt, ist des Weiteren auf dem Rechner Visual Studio 2019 installiert, welches mit GitHub verknüpft ist, sodass eine Versionsverwaltung der Codezeilen und Klassen gewährleistet ist. Da der gleiche Rechner bereits für die Vorarbeit dieser Studienarbeit verwendet wurde, sind die benötigten Plugins für das Arbeiten mit Unity und dem Roboter in der VR bereits installiert.

Wie bei dem vergangenen Projekt ist auf der virtuellen Ubuntu Maschine die ROS Distribution Indigo Igloo mit Schnittstellen zu ROS# installiert worden. Aufgrund der Tatsache, dass diesmal eine andere Virtualisierungssoftware verwendet wird, müssen vorher Einstellungen vorgenommen werden, damit sich die VM im gleichen Netzwerk wie der Nao-Roboter und dem System, auf dem Unity läuft, befindet. In den Einstellungen befindet sich die Kategorie „Network Adapter“ (siehe *Abbildung 1: VM Settings - Network Adapter*). Hier ist es wichtig zunächst den Haken bei „Connected“ zu setzen, damit sich die VM überhaupt in einem Netzwerk befinden kann. Bei der Network Connection sollte „Bridged“ gewählt

werden. Der Haken beim Replizieren des physikalischen Netzwerkstatus sollte nicht gesetzt sein. Mit diesen Einstellungen muss sich lediglich der lokale Rechner mit dem Nao Netzwerk verbinden, damit die VM eine IP-Adresse im richtigen Netzwerk erhält.

Da die Ubuntu VM einige Terminals und Programme während der Unity Anwendung ausführen muss, gibt es weitere Einstellungen, die vorgenommen werden können. Die Arbeitsspeicher-Kapazität sollte möglichst hoch gewählt werden. In unserem Fall steht der VM 8GB RAM zur Verfügung (*siehe Abbildung 2: VM Settings - Memory*).

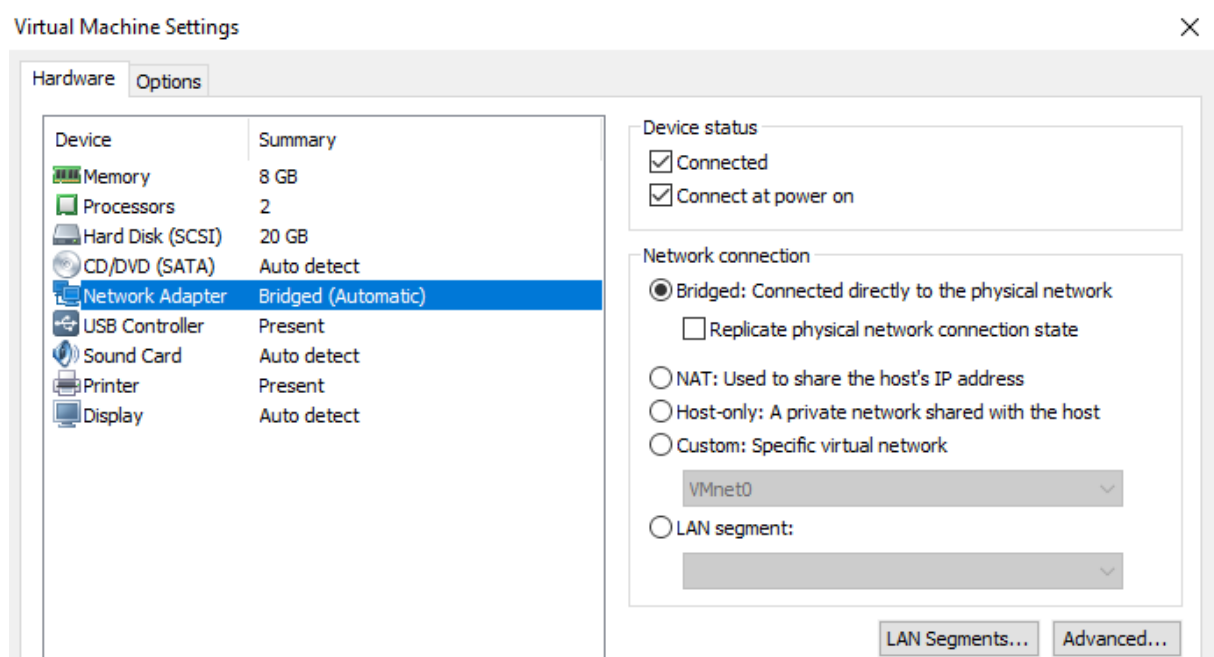


Abbildung 1: VM Settings - Network Adapter

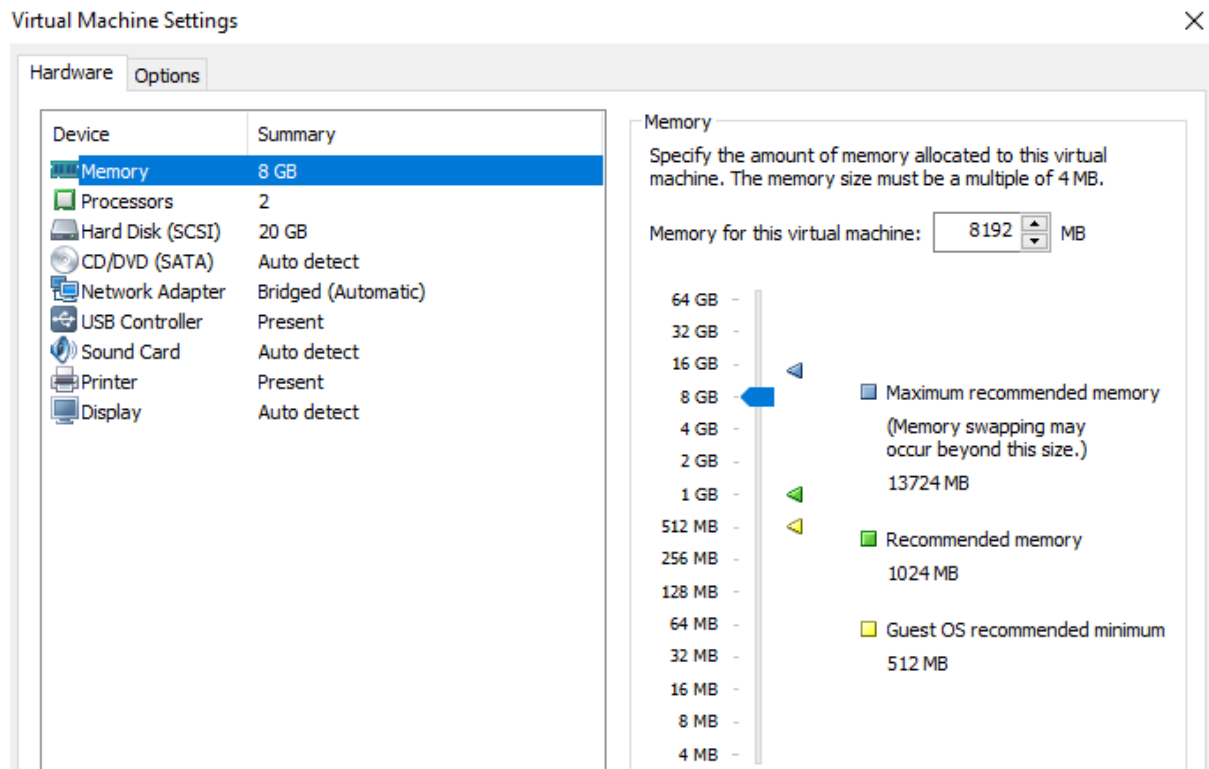


Abbildung 2: VM Settings - Memory

Der Nao-Roboter, der für diese Studienarbeit verwendet wird, ist aus der V4 Generation. Da die Kameras, die im Roboter verbaut sind, den Ansprüchen nicht genügen, erhält der Anwender das Bild in der VR durch eine zusätzliche Kamera. Hierbei wird für jedes Auge die gleiche Kamera verwendet. Dies wird in Punkt *Anpassung der Kameras* genauer erklärt. Die Kamera, die verwendet wird, ist eine „Foxxer Predator V4“ mit einer 2.5mm Linse. Der verbundene Sender ist vom Typ „Team Blacksheep UNIFY PRO 5G8 HV – RACE (SMA)“. Damit das Bild auch am Rechner ankommt, sind dort zwei Empfänger der Marke „FUAV Mini 5.8G FPV“ per USB angeschlossen. Dadurch werden die Foxxer Kameras am PC als Webcam erkannt. Diese werden auf dieselbe Frequenz gestellt, auf der die

Kamera das Bild sendet, sodass für beide Augen das gleiche Bild verwendet werden kann.

1.4 Vorgehensweise

Die Punkte für die Bearbeitung der Studienarbeit wurden zwischen den zwei Projektmitarbeitern aufgeteilt. Dies hat den Vorteil, dass die Arbeit schneller bearbeitet werden kann und jeder in seinen Fachbereichen sein Wissen verwenden kann, um die Studienarbeit zu einem erfolgreichen Abschluss bringen zu können.

Dennis Wiebe befasste sich mit den folgenden Aufgaben:

- Aufsetzen, Installieren und Warten der Ubuntu VM
- Wiederherstellen der ROS Installation sowie Verbindung zwischen Unity und ROS
- Kamera inklusive Sicht in der VR Brille
- Erstellen von 3D-Modellen und Ausdruck mit 3D-Drucker
- Weitere Funktionalitäten mit Hilfe von Python-Skripten

Vincenzo Ciullo bearbeitete folgende Punkte:

- Instandsetzung der vorigen Studienarbeit, nachdem die Verbindung zur Linux VM stand
- Korrektur von Programmfehlern der vorigen Studienarbeit
- Sprachfeature implementiert
- Tracker der HTC VIVE eingebunden und eine Möglichkeit geboten, diese als Steuerung des Roboters zu verwenden

2 Grundlagen

Dieses Kapitel befasst sich mit den Grundlagen, die für diese Arbeit verwendet werden. Es präsentiert und erklärt den Nao-Roboter und das HTC VIVE System, welche beide für die Umsetzung dieser Arbeit sehr wichtig sind.

2.1 Nao-Roboter

Nao ist ein humanoider Roboter, welcher von der französischen Firma Aldebaran Robotics hergestellt wurde. Der Roboter wurde zum ersten Mal 2006 präsentiert. Seit 2007 ist der Roboter der offizielle Nachfolger des Sony Aibo als Roboter im RoboCup. Dies ist ein Fußballturnier, welches mit Robotern durchgeführt wird. So wurde der Nao-Roboter das erste Mal 2008 in einem Fußballturnier eingesetzt und dient seitdem als Standardplattform für dieses Turnier.

Bis zum Jahr 2018 erschienen verschiedene Revisionen des Roboters, um diesen ständig zu verbessern. Das Modell V2 aus dem Jahre 2010 bietet überarbeitete Motoren, welcher bereits ein Jahr später durch den Nao Next Gen abgelöst wurde. Dieser ist die V4 der Nao-Roboter Produktreihe und wird für die Studienarbeit verwendet. Der Nao-Roboter V6 aus dem Jahr 2018, bietet im Gegensatz zu seinem Vorgängermodell, starke Verbesserungen [3].

Der für die Studienarbeit verwendete Nao-Roboter ist 58cm hoch und wiegt 5.2 Kilogramm. Normalerweise beträgt die Akkulaufzeit des Roboters mindestens 60 Minuten, jedoch ist die Kapazität des Akkus durch das Alter des Roboters stark eingeschränkt (siehe *4.6 Roboter gebraucht*). Die CPU des Roboters ist ein Intel Atom Z530. Des Weiteren sind 1 GB RAM und ein 8GB Micro SDHC verbaut.

Das Betriebssystem des Roboters ist NAOqi der Version 2.1. Wie diese auf der VM installiert wird, ist im Kapitel *5.1 Inbetriebnahme VM und Verbindung zu ROS* erläutert.

Zum Funktionsumfang des Roboters gehören unter anderen vier Ultraschallsensoren, ein Trägheitssensor, Drucksensoren in den Füßen, vier Mikrofone und zwei Lautsprecher. Die Text-to-Speech-Funktion beziehungsweise die Spracherkennung des Roboters unterstützen 20 verschiedene Sprachen. Die zwei im Kopf verbauten Kameras, sie bilden die Augen des Roboters, sind in der Lage Personen, Gesichter und Objekte zu erkennen. Die in den Augen und Ohren verbauten acht LEDs lassen sich einzeln ansteuern und unterschiedlich einstellen, was die Farben angeht.

Die Bewegung des Roboters ist in 25 Freiheitsgraden möglich. Des Weiteren gibt es andere Versionen von Nao-Robotern, die in den möglichen Bewegungen eingeschränkt sind [3]. Da für die Studienarbeit der H25 verwendet wird, sind die anderen Versionen des Roboters für diese Arbeit nicht relevant.

Eine normale, vereinfachte Programmierung des Roboters kann über die hauseigene Choreographie Suite erfolgen. In der Software ist es möglich, Blöcke anzuordnen. Diese werden abhängig ihrer Reihenfolge vom Roboter abgearbeitet, wenn der Ablauf gestartet wird. Auch eine parallele Ansteuerung einzelner Motoren oder Kameras ist neben der Erstellung eigener Blöcke mit C++ oder Python möglich. Die dadurch verfassten Programme können auf dem Roboter ausgeführt oder dauerhaft auf dem Nao gespeichert werden. Das Speichern auf

dem Roboter sollte so zum Beispiel genutzt werden, wenn man mit der Gesichtserkennung der Kamera arbeitet, oder einen Knopf betätigt.

Über die Choreographie Suite lassen sich ebenfalls Live-Bilder der Kameras des Roboters, der aktuelle Akkustand, der Verbindungsmanager und Diagnosetools betrachten. Über eine Bibliothek lassen sich die vom Roboter einnehmbaren Posen einsehen [4].

Wie ursprünglich geplant, dient der Nao-Roboter heute auch noch besonders an Universitäten, Schulen und anderen Bildungseinrichtungen als leicht bedienbares Forschungsobjekt. Einen Roboter V4 lässt sich heute nicht mehr käuflich erwerben, jedoch kann die neuste Version (V6) für 8644,89€ gekauft werden (Stand 2021, [5]; der Standardpreis beträgt 10500\$ und wurde zum tagesaktuellen Kurs in Euro umgerechnet, [6]).

2.2 HTC VIVE Pro & Zubehör

Für die Umsetzung dieser Arbeit wird des Weiteren ein HTC VIVE Pro System von der Dualen Hochschule zur Verfügung gestellt. Bei der HTC VIVE handelt es sich um ein sogenanntes Virtual Reality-Headset, welches von HTC in Kooperation mit Valve produziert wird. Es wurde 2015 vorgestellt und ist seitdem ebenfalls käuflich zu erwerben [7].

Bei der für diese Arbeit verwendeten Brille handelt es sich um eine HTC VIVE Pro HMD. Diese kann momentan für 1199€ käuflich erworben werden (Stand 2021; [8]). Im Kaufumfang sind die benötigten Sensoren zur Erkennung der Brille und der Controller in der VR ebenfalls vorhanden. Die Brille hat zwei AMOLED

Displays, welche jeweils eine Auflösung von 1440x1600 Pixel haben. Eine hohe Auflösung ist bei VR-Headsets wichtig, da durch die Nähe an den Augen sonst die Pixel der Bildschirme erkannt werden können. Die Bildwiederholrate der Bildschirme liegt bei 90Hz. Die Brille hat ebenfalls einige Sensoren eingebaut. So sind ein SteamVR Tracking Sensor, ein G-Sensor, ein Gyroskop, ein Entfernungssensor und ein IPD, dies ist ein Sensor für den Komfort des Pupillenabstands, in das Vive System integriert [8]. Durch die Möglichkeit den Linsenabstand, die Kopfhörer und das Kopfband zu verstellen, ist das System für eine große Menge an Anwendern geeignet.

Des Weiteren bietet das System ebenfalls die Möglichkeit kabellos verwendet zu werden. Ein solcher Adapter ist vorhanden, jedoch wurde dieser während der Umsetzung des Projektes nicht verwendet. Zudem werden für diese Arbeit noch Vive Tracker verwendet. Diese sind Objekte, welche an Objekte in der realen Welt befestigt werden können, sodass diese in der virtuellen Realität ausgewertet und verwendet werden können. Die folgende Abbildung zeigt ein Beispiel, wie die Tracker eingesetzt werden.



Abbildung 3: Beispiel für die Verwendung der HTC Vive Tracker, [8]

In der Abbildung ist die Möglichkeit zu erkennen, den Tracker an einen Tennisschläger zu montieren, sodass dieser für ein VR Tennisspiel verwendet werden kann. Dies gibt dem Anwender noch mehr das Gefühl gerade Tennis zu spielen und somit tiefer in das Spielerlebnis eintauchen zu können. Ein Tracker kann für 120€ käuflich erworben werden (Stand 2021; [8]).

3 Anforderungsanalyse

Aus Punkt *Motivation und Zielsetzung* geht hervor, dass die Funktionen des Roboters erweitert werden müssen. Bisher ist das Laufen, Drehen und in die Knie gehen des Roboters nur möglich, wenn der Status des Roboters in einen Zustand gewechselt wird, in dem das Bewegen des Kopfes und der Arme nicht mehr möglich ist. Dies beeinträchtigt die Immersion des Anwenders mit der Anwendung, sodass dies geändert werden muss.

Es muss also ein Weg gefunden werden, der sowohl die alten Funktionalitäten als auch das vorige benannte Bewegen, sei es Laufen, Drehen oder Ducken, ohne einen vorigen Zustandswechsel ermöglicht. Hierbei sollten die HTC VIVE Tracker als Hilfe verwendet werden, sodass es nicht mehr möglich sein darf, den Roboter durch Betätigen von Knöpfen der Controller zu bewegen.

4 Probleme des aktuellen Stands

Bevor Konzepte zu den neuen Anforderungen angegangen werden, muss zunächst sichergestellt sein, dass der aktuelle Stand ordnungsgemäß funktioniert. Bei näherer Betrachtung der Ausarbeitung und der Qualität liegt dies jedoch nicht vor. Im Nachfolgenden werden Fehler und Probleme analysiert, damit diese bei Erweiterungen auf neue Funktionalitäten nicht im Wege stehen.

4.1 Kamerahalterung & Kameras

In der vorherigen Studienarbeit wurde eine Kamerahalterung modelliert und mit einem 3D Drucker ausgedruckt. In Dieser Kamerahalterung sollen zwei Kameras festsitzen und auf dem Kopf des Nao befestigt werden. Die Bilder dieser Kameras werden dann jeweils auf ein Auge in der VR Brille angezeigt.

Das Problem daran ist, dass durch Stürze und Experimente im vorherigen Projekt, diese Halterung gebrochen ist (Siehe *Abbildung 4: Kaputte Kamerahalterung*). Dadurch kann diese Vorrichtung nicht mehr benutzt werden. Des Weiteren wurde die Box der Halterung, in der nur die Kameras sitzen, beschädigt, sodass eine Kamera nicht mehr richtig in dieser Box befestigt ist und wackelt. Das Bild, welches angezeigt wurde, war demnach unbrauchbar.



Abbildung 4: Kaputte Kamerahalterung

Die Kameras selbst sind jedoch unbeschadet geblieben. Es muss jedoch eine Lösung für das Problem mit der Halterung gefunden werden, sodass die Vorrichtung wieder am Roboter befestigt werden kann. Außerdem muss die Box, in der sich die Kameras befinden repariert werden, damit die Kameras fest verstaut sind und es zu keinen Bildfehlern durch Wackeln der Kameras kommen kann.

4.2 Funktionalität Greifen

Eines der Features, welches in der vorigen Studienarbeit programmiert wurde, war die Möglichkeit Dinge mit dem Roboter zu greifen. Jedoch konnte diese Funktionalität niemals überprüft werden. Immer, wenn versucht wurde, mit dem Roboter zu greifen, ging das nicht.

Hierfür wurde von den Bearbeitern der vorigen Studienarbeit eine Klasse verwendet, welche sich mit dem Greifen auseinandersetzt, die Klasse „GrabManager“. Hier wurde versucht, mit Hilfe einer Message-Klasse die Information, welche Hand gerade geöffnet oder geschlossen werden soll, an den ROS-Client zu senden. Dadurch, dass diese Funktionalität niemals nachgewiesen werden konnte, wurde die Klasse GrabManager gelöscht, sodass eine neue Idee umgesetzt werden konnte, damit das Greifen des Roboters immer und vor allem zuverlässig funktioniert.

Des Weiteren wurde bei der Sichtung der alten Studienarbeit in diesem Bereich bemerkt, dass durch die Kameras das Modell des Roboters angezeigt wird, ähnlich wie in einer AR-Applikation. Durch diesen Umstand war es nicht möglich, dass der Anwender der Applikation sehen konnte, wenn er versucht hatte, etwas zu greifen, da das Modell des Roboters die Hände des realen Roboters verdeckt hatte.

4.3 Posen des Nao-Roboters

Der Nao-Roboter hat die Möglichkeit sich in vordefinierte Posen zu bewegen. Eine Sammlung dieser Posen ist in *Abbildung 5: Einnehmbare Posen des Nao-Roboters* zu erkennen. Diese Posen können beispielsweise über die Anwendung Choreographie ganz simpel eingenommen werden.

Im vorherigen Projekt wurde ein „PoseController“ verwendet, um die Posen während der Unity Anwendung auszuführen. Nao sollte sich beim Starten der Anwendung in die Initialpose bewegen, welches die „StandZero“ Pose darstellt.

Des Weiteren soll sich der Roboter mithilfe der HTC VIVE Controller durch einen Knopfdruck ebenfalls in die Posen begeben können.

Dazu sollte der obere Knopf benutzt werden, um den Roboter aufstehen zu lassen sowie den unteren Knopf, um ihn in die Hocke gehen zu lassen, welches die Pose „Crouch“ in der *Abbildung 5: Einnehmbare Posen des Nao-Roboters* darstellt.

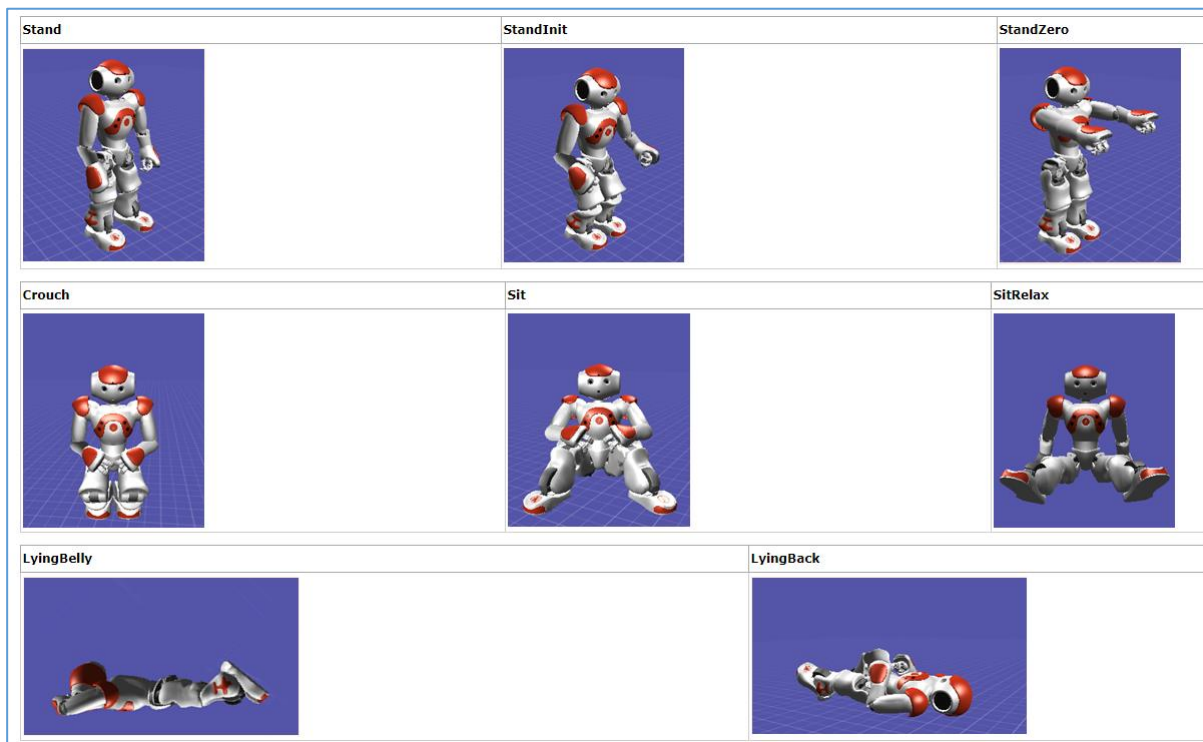


Abbildung 5: Einnehmbare Posen des Nao-Roboters [9]

Allerdings passierte nicht einmal etwas, als die Knöpfe betätigt wurden. Dies lässt sich auf die Tatsachen zurückführen, dass entweder der „PoseController“ nicht funktionsfähig ist oder dass dieser mittlerweile nicht mehr unterstützt wird. Eine weitere Erkenntnis ist, dass sich der Roboter auch nicht beim Starten des Programms in die Initialpose begibt, sodass es nicht an der Knopfbelegung liegen kann, dass die Posen durch den Roboter nicht eingenommen werden.

Das ist ein Problem, welches zwingend behoben werden muss, um weitere Funktionalitäten zu implementieren.

4.4 Community Aldebaran

Der Nao-Roboter, welcher von der Firma Aldebaran Robotics hergestellt wird, benötigt Software, um jede Funktion benutzen zu können. Hierzu gibt es vom gleichen Unternehmen eine Webseite, in der benötigte Software gefunden werden kann. Diese Webseite war unter Community Aldebaran zu finden bzw. unter folgenden Link: <https://community.ald.softbankrobotics.com/>

Aufgrund des hohen Alters des Nao V4 Roboters und entsprechenden weiteren neuen Modellen wie Nao V5 und Nao V6 wird keine direkte Unterstützung mehr für den Nao V4 Roboter bereitgestellt. Dies wird deutlich, wenn versucht wird sich auf die Community Aldebaran Webseite zu begeben. Bei einem Aufruf der Webseite erscheint nur noch die Meldung, welche in *Abbildung 6: Community Aldebaran Webseite* zu sehen ist.

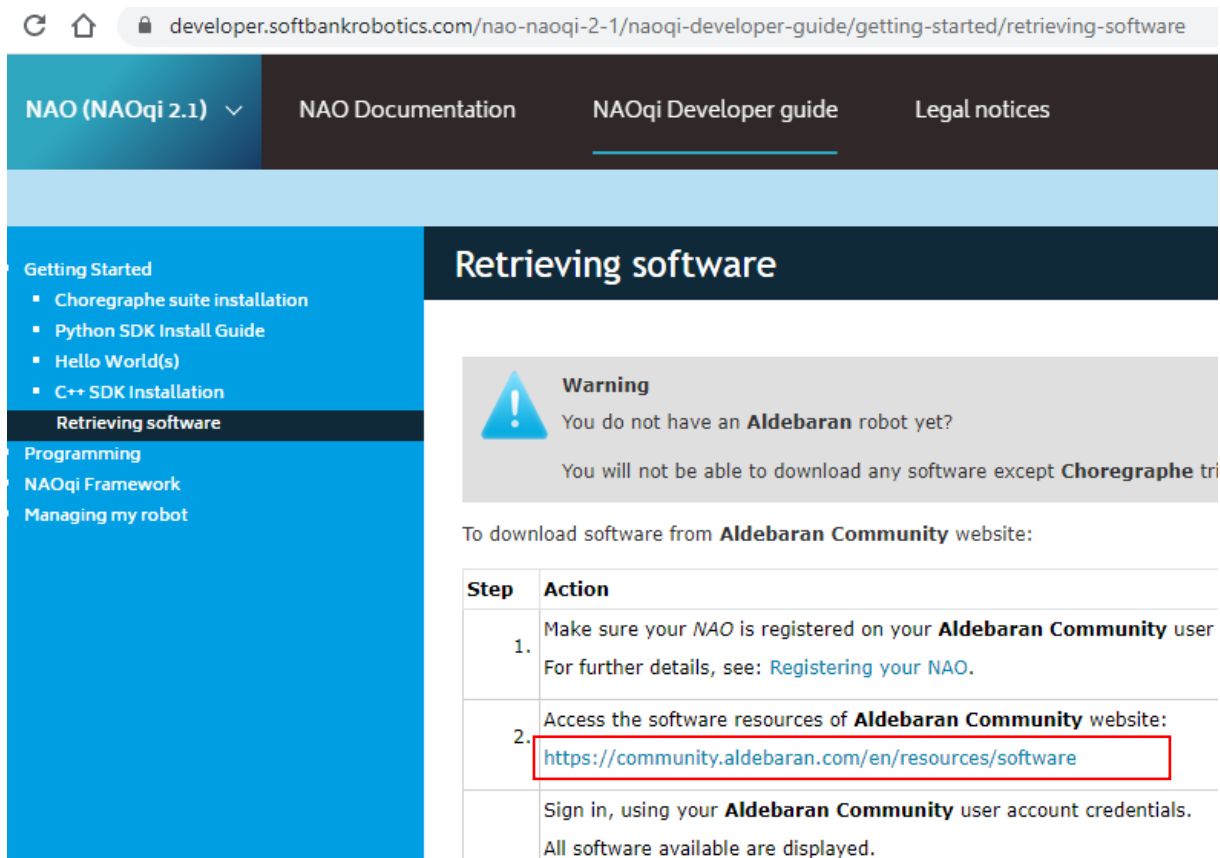


Abbildung 6: Community Aldebaran Webseite [10]

Hier wird darauf hingewiesen, dass die Webseite nicht länger verfügbar ist und die Software nur noch im Developer Center zur Verfügung steht. Dazu muss lediglich der Link im roten Kasten benutzt werden.

Jedoch wird direkt der Nao V6 Roboter sowie der Pepper Roboter auf der Startseite des Developer Centers angezeigt und nicht der gewünschte V4 Roboter (Stand 2021). Auf der Weiterleitung zum V6 Roboter gibt es die Möglichkeit, eine Seite über die Naoqi 2.1, also die richtige Version für den V4 Roboter, aufzurufen. Jedoch gibt es hier nicht die Möglichkeit notwendige Software für die Arbeit an dem Roboter herunterzuladen. Der Aufrufer der Webseite hat lediglich die Auswahl zwischen der technischen Dokumentation des Roboters, welche für diese Arbeit nicht relevant ist, sowie den Developer Guide [11].

Der Developer Guide wäre genau die richtige Stelle, um notwendige Software zu oder notwendige Informationen zu erhalten. Hierzu gibt es auch einige Anlaufstellen, jedoch ist diese auch noch zu einem älteren Zeitpunkt entstanden. Hier wird nämlich erneut auf die Community Aldebaran Seite verwiesen (siehe *Abbildung 7: Developer Center Webseite*), welches keinen aktuellen Stand darstellt, da es die Seite, wie vorhin erwähnt, nicht mehr gibt.



developer.softbankrobotics.com/nao-naoqi-2-1/naoqi-developer-guide/getting-started/retrieving-software

NAO (NAOqi 2.1) ▾ NAO Documentation NAOqi Developer guide Legal notices

Retrieving software

Warning
You do not have an **Aldebaran** robot yet?
You will not be able to download any software except **Choregraphe** tri

To download software from **Aldebaran Community** website:

Step	Action
1.	Make sure your <i>NAO</i> is registered on your Aldebaran Community user For further details, see: Registering your NAO .
2.	Access the software resources of Aldebaran Community website: https://community.aldebaran.com/en/resources/software
	Sign in, using your Aldebaran Community user account credentials. All software available are displayed.

Abbildung 7: Developer Center Webseite

4.5 Batterien für Kameras defekt

Die angesprochenen Kameras in der Kamerahalterung benötigen eine Stromzufuhr. Dazu werden im aktuellen Stand beide Kameras mit einer Nanobatterie versorgt. In der *Abbildung 8: Turnigy nano Tech 300mAh 2s und Compact Charger* ist die Nano Batterie „Turnigy nano-Tech 300mAh 2S“ zu sehen. Davon sind vier vor Ort in der DHBW in Karlsruhe, welche von den Vorgängern benutzt worden sind. Beim Aufsetzen des alten Standes haben jedoch zwei der Batterien gar nicht funktioniert. Beim Versuch diese mit einem „Turnigy E3 Compact Charger“ aufzuladen, welcher ebenfalls in der *Abbildung 8* zu erkennen ist, haben alle LEDs geblinkt. Dieser Hinweis deutet darauf hin, dass ein Fehler vorliegt. Der genaue Fehler lautet „There is a 300mv difference of voltage between battery pack“. Bei den beiden funktionsfähigen Batterien tritt der Fehler ebenfalls auf, nachdem sie entleert wurden. Nach einer durchschnittlichen Ruhezeit von 2 Stunden, haben diese manchmal wieder funktioniert und konnten erneut aufgeladen werden.

Beim Testen mit beiden Kameras ist eine Batterie allerdings nach ungefähr 20 Minuten aufgebraucht und da nur eine als Puffer vorhanden war, eignet sich dieser Umstand als unzuverlässige Methode.

Dieses Problem wurde direkt gelöst, indem 2 weitere voll funktionsfähige Batterien bestellt wurden.



Abbildung 8: Turnigy nano Tech 300mAh 2s und Compact Charger

4.6 Roboter gebraucht

Der Nao V4 Roboter, welcher in dieser Studienarbeit verwendet wird (Siehe 2.1 *Nao-Roboter*), ist an der DHBW in Karlsruhe schon sehr lange im Einsatz. Dieser wurde schon in mehreren Studienarbeiten involviert und hat dementsprechend einige Gebrauchsspuren. Äußerlich sind einige Schrammen am orangenen oder weißen Plastik zu erkennen, die darauf zurückzuführen sind, dass der Roboter in früheren Projekten öfter gestürzt ist. Diese sind jedoch nur von optischer Natur und schränken die Funktionalitäten des Nao-Roboters nicht ein.

Ein größeres Problem stellt die Akkulaufzeit des Roboters dar. Bei einem neuen Modell der V4 Generation beträgt die Laufzeit mindesten 60 Minuten. Bei einem neuen Roboter der V6 Generation bereits 90 Minuten. Durch den langen Verwendungszeitraum des Roboters ist der Akku nicht mehr so funktionsfähig wie zu Beginn. Die Folge ist, dass der Nao so oft wie möglich mit Strom versorgt werden muss. Wenn Tests mit dem Roboter durchgeführt werden, muss allerdings beachtet werden, dass beim Bewegen des ganzen Körpers (z.B. Laufen, Rotieren) das Kabel nicht am Roboter befestigt werden kann, da es die Bewegungen des Nao-Roboters einschränkt. Daraus resultieren erneute Wartezeiten beim Experimentieren, um den Akku wieder aufzuladen.

Die Leistung des Prozessors sollte auch nicht außenvorgelassen werden. Diese ist bei weitem nicht mehr so hoch wie in der Anfangsphase. Wenn der Roboter an einem Labortag das erste Mal eingeschalten wird, benötigt das nach aktuellem Stand vier bis fünf Minuten.

Nachdem der Roboter hochgefahren ist, gibt der Nao Auskunft über seinen aktuellen Status. Dabei gibt er Auskunft über seine Motoren. Aufgrund des hohen Alters hat der Nao selbst nach einer Woche, ohne hochgefahren worden zu sein, die Meldung gegeben, dass Arme- und Bein-Motoren zu heiß sind. Das wird dann auch bereits deutlich, wenn über „Choreographie“ Posen eingenommen werden. Bei der Pose „StandZero“ beispielsweise, sollten beide Arme parallel nach vorne gestreckt sein. Wenn allerdings der linke Arm-Motor bereits zu heiß ist, kann es vorkommen, dass der Arm nicht so hoch gehalten wird wie der rechte Arm.

4.7 Laboraufenthalte & Corona

Die aktuelle Gegebenheit mit dem Corona Virus betrifft auch den Umgang mit dieser Studienarbeit. Geräte wie die HTC VIVE mit Zubehör, der Roboter selbst sowie die Arbeitsflächen mit Tastatur und Maus müssen jedes Mal desinfiziert werden. Nur dann ist sichergestellt, dass sich an das Hygienekonzept der Dualen Hochschule gehalten wird und niemand einem Risiko ausgesetzt ist. Des Weiteren werden für die HTC VIVE von der DHBW Karlsruhe Haarnetze sowie austauschbare Gesichtspolster bereitgestellt.

Ebenso ist es zu den bedauernden Zeiten nur durch Termin möglich sich gezielt zu zweit zu treffen und an der Studienarbeit weiterzuarbeiten. Es besteht nicht die Möglichkeit, wie zu Zeiten ohne den Virus, nach einer Vorlesung den Raum zu wechseln und an der Studienarbeit weiterzuarbeiten. Bei dieser Art von Studienarbeit ist es nämlich zwingend notwendig vor Ort zu sein, da sonst keine Experimente durchführbar sind.

Wie auch bereits in der vorigen Studienarbeit beschrieben, wäre es sinnvoll gewesen, die Anwendung durch eine repräsentative Menge von Benutzern testen zu lassen. Aufgrund der aktuellen Lage (Stand Mai 2021) sind noch starke Beschränkungen vorhanden. Somit ist es nicht möglich, die Anwendung durch mehrere Personen testen zu lassen. Zwischenzeitlich waren wegen zu hohen Inzidenzwerten Laborbetriebe an der Hochschule nicht gestattet. An der DHBW Karlsruhe war es also nur noch möglich notwendige Prüfungen stattfinden zu lassen, sodass es nicht realisierbar war nacheinander mehrere Benutzer den Roboter bedienen zu lassen.

Mithilfe von Rückmeldungen mehrerer Benutzer, könnte die Qualität der Anwendung besser bewertet werden sowie Verbesserungen im Sinne der Interaktionen mit dem Roboter verwirklicht werden.

5 Umsetzung

Dieses umfangreiche Kapitel ergibt sich aus den Anforderungen sowie den erläuterten Problemen des aktuellen Systems. Dabei wird näher auf die Vorgehensweise von Lösungen dieser Probleme eingegangen. Außerdem werden hier die neu implementierten Funktionalitäten näher erläutert und die Veränderungen mit dem momentanen Stand gegenübergestellt.

5.1 Inbetriebnahme VM und Verbindung zu ROS

Für die Verwendung der Anwendung, die mit dieser Studienarbeit in Verbindung steht, muss zunächst die Ubuntu VM erstellt und eingerichtet werden. Analog zu den Vorgängern wurde diese selbst erstellt. Lediglich eine andere Virtualisierungssoftware nämlich VM Workstation Pro wurde verwendet anstatt Oracle Virtual Box.

Da weitere Pakete für die Interaktion mit dem Roboter benötigt werden, müssen diese auch erst installiert werden. Zuerst wird eine ROS Distribution installiert. Dazu gibt es verschiedene Varianten wie ROS Noetic Ninjemys, ROS Melodic Morenia, ROS Indigo Igloo und viele weitere. Weil sich der Vorgänger für die Indigo Igloo Variante entschieden hat, haben wir diese ebenfalls benutzt. Damit konnten wir direkt an die Ergebnisse der alten Arbeit anknüpfen und nicht von neuem eine unterschiedliche Distribution auf Funktionalitäten überprüfen.

Die Grundlage der Installation sind die Anleitungen aus dem Wiki der ROS [12] Dokumentation. Hier werden detaillierte Schritte zum erfolgreichen Installieren der notwendigen Komponenten gezeigt. Dazu gehören beispielsweise das Einrichten

einer Sources Liste, damit Software von ROS akzeptiert werden sowie „rosdep“, welches ebenfalls benötigt wird, um ROS zu benutzen. Zur Verbindung mit dem Roboter werden zusätzliche Pakete benötigt. Darunter auch die Python SDK [13]. Zum erfolgreichen Installieren steht hierfür auch eine ROS Anleitung zur Verfügung, an der sich orientiert wurde. Das Wichtigste zur Installation ist die Schnittstelle zum Nao herzustellen. Dazu benötigt man spezifische Pakete. Diese sind `naoqi_driver`, `naoqi_bridge` und `nao_description`. Während der Arbeit mit der VM ist es sehr hilfreich mit Snapshots zu arbeiten. Es handelt sich dabei um eine Sicherung bzw. ein Backup der VM. Damit lassen sich eine falsche Installation oder noch fehlende Pakete besser identifizieren, denn es werden einige Befehle benötigt, um alle Komponente von ROS zu installieren.

Nachdem alles korrekt eingerichtet ist, ist es wieder wie beim alten Stand möglich sich mit dem Nao zu verbinden. Dazu ist es nötig sich im selben Netzwerk wie der Nao zu befinden. Falls der Nao eingeschalten ist, muss der Knopf auf seiner Brust betätigt werden, um die IP-Adresse des Nao zu erfahren. Ist dies nicht der Fall und der Nao gibt keine Auskunft über seine IP-Adresse, muss zunächst überprüft werden, ob der Router eingeschalten ist. Falls dies der Fall ist, hilft meistens ein Neustart des Roboters. Hilft dies jedoch auch nicht, muss ein Ethernet Kabel benutzt werden, um den Nao manuell mit dem Router zu verbinden. Anschließend kann das Kabel wieder entfernt werden und der Roboter behält seine Verbindung zum Router.

Wenn Erfahrung über die IP-Adresse des Nao gebracht wurde, gilt es selbst eine zu erhalten. Dazu muss sich wie bereits erwähnt in dasselbe Netzwerk verbunden werden. Die Einstellungen, die dazu in der VM nötig sind, wurden im Kapitel 1.3 *Entwicklungsumgebung* nähergebracht. Sobald das System, auf dem die Virtualisierungssoftware betrieben wird, mit demselben Router verbunden ist, erhält dieser eine IP-Adresse im gleichen Netzwerk. Das Entscheidende ist, dass mit den Netzwerkeinstellungen in der VM, die VM selbst eine eigene IP-Adresse erhält. Diese ist mit dem Befehl „ifconfig“ aufrufbar und wird in der weiteren Verwendung zwingend benötigt. Wie ein solcher Befehl aussieht, ist in der nächsten Abbildung zu erkennen. Die Adresse, welche entscheidend ist, wurde hervorgehoben.

```
dennis@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet  Hardware Adresse 00:0c:29:b4:b9:11
          inet Adresse:192.168.100.11  Bcast:192.168.100.255  Maske:255.255.255.
          0
          inet6-Adresse: fe80::20c:29ff:feb4:b911/64  Gültigkeitsbereich:Verbindu
ng
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metrik:1
          RX-Pakete:237 Fehler:0 Verloren:0 Überläufe:0 Fenster:0
          TX-Pakete:189 Fehler:0 Verloren:0 Überläufe:0 Träger:0
          Kollisionen:0 Sendewarteschlangenlänge:1000
          RX-Bytes:53215 (53.2 KB)  TX-Bytes:19731 (19.7 KB)

lo        Link encap:Lokale Schleife
          inet Adresse:127.0.0.1  Maske:255.0.0.0
          inet6-Adresse: ::1/128  Gültigkeitsbereich:Maschine
          UP LOOPBACK RUNNING  MTU:65536  Metrik:1
          RX-Pakete:218 Fehler:0 Verloren:0 Überläufe:0 Fenster:0
          TX-Pakete:218 Fehler:0 Verloren:0 Überläufe:0 Träger:0
          Kollisionen:0 Sendewarteschlangenlänge:1
          RX-Bytes:15924 (15.9 KB)  TX-Bytes:15924 (15.9 KB)
```

Abbildung 9: Erhalten der IP-Adresse der VM

Als nächstes wird der ROS Core gestartet. Dieser Befehl verwendet die installierte Indigo Igloo Distribution. Es startet die Kernprogramme und ist zwingend notwendig, um weitere ROS Komponente zu verwenden. Für die nächsten Schritte ist also die Reihenfolge der Befehle zu beachten. Wenn alle Komponenten ordnungsgemäß installiert sind, sollte es nach dem Befehl „roscore“ wie in folgender Abbildung aussehen.

```
dennis@ubuntu:~$ roscore
... logging to /home/dennis/.ros/log/53af3376-8e19-11eb-a751-000c29b4b911/roslaunch-ubuntu-2834.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:38603/
ros_comm version 1.11.21

SUMMARY
=====

PARAMETERS
* /rostdistro: indigo
* /rosversion: 1.11.21

NODES

auto-starting new master
process[master]: started with pid [2846]
ROS_MASTER_URI=http://ubuntu:11311/

setting /run_id to 53af3376-8e19-11eb-a751-000c29b4b911
process[rosout-1]: started with pid [2859]
started core service [/rosout]
```

Abbildung 10: Starten der Kernprogramme mit "roscore"

Anschließend gilt es die Verbindung zum Nao mit Unity zu ermöglichen. Das wird mit dem Paket naoqi_bridge ermöglicht. Es muss also ein ROS_Bridge Server gestartet werden. Der Befehl ist „roslaunch rosbridge_server rosbridge_websocket.launch“.

Wenn der Befehl wieder erneut korrekt ausgeführt wird, sollte das folgendermaßen aussehen.

```
dennis@ubuntu:~$ roslaunch rosbridge_server rosbridge_websocket.launch
... logging to /home/dennis/.ros/log/53af3376-8e19-11eb-a751-000c29b4b911/rosla
unch-ubuntu-2906.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:33758/

SUMMARY
=====

PARAMETERS
* /rosbridge_websocket/address:
* /rosbridge_websocket/authenticate: False
* /rosbridge_websocket/delay_between_messages: 0
* /rosbridge_websocket/fragment_timeout: 600
* /rosbridge_websocket/max_message_size: None
* /rosbridge_websocket/port: 9090
* /rosbridge_websocket/retry_startup_delay: 5
* /rostdistro: indigo
* /rosversion: 1.11.21

NODES
/
  rosapi (rosapi/rosapi_node)
  rosbridge_websocket (rosbridge_server/rosbridge_websocket)

ROS_MASTER_URI=http://localhost:11311

core service [/rosout] found
process[rosbridge_websocket-1]: started with pid [2924]
process[rosapi-2]: started with pid [2925]
registered capabilities (classes):
- rosbridge_library.capabilities.call_service.CallService
- rosbridge_library.capabilities.advertise.Advertise
- rosbridge_library.capabilities.publish.Publish
- rosbridge_library.capabilities.subscribe.Subscribe
- <class 'rosbridge_library.capabilities.defragmentation.Defragment'>
- rosbridge_library.capabilities.advertise_service.AdvertiseService
- rosbridge_library.capabilities.service_response.ServiceResponse
- rosbridge_library.capabilities.unadvertise_service.UnadvertiseService
[INFO] [WallTime: 1616752553.726895] Rosbridge WebSocket server started on port
9090
```

Abbildung 11: Ausschnitt zur Erstellung eines ROSBridge_Servers

Der letzte Befehl ist „roslaunch nao_bringup nao_full.launch nao_ip:=<IP_Adresse-_des_Nao> roscore_ip:=<IP_Adresse_der_Ubuntu_VM>“. Aufgrund der Einstellung der VM, unterscheidet sich dieser Befehl leicht von denen des Vorgängers. Hier wird nämlich kein weiterer Parameter benötigt. In der vorherigen Studienarbeit wurde zusätzlich noch „network_interface:=enp0s81“ benötigt, da in der Virtual Box andere Einstellungen vorgenommen wurden. Der Aktuelle Stand benötigt aber nur noch den Befehl wie in *Abbildung 12: Startet die Verbindung mit dem Nao mit der Python SDK* gezeigt.

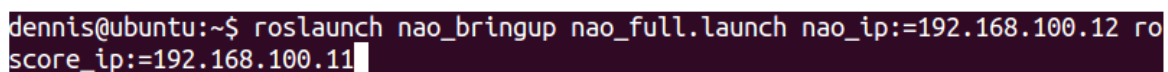
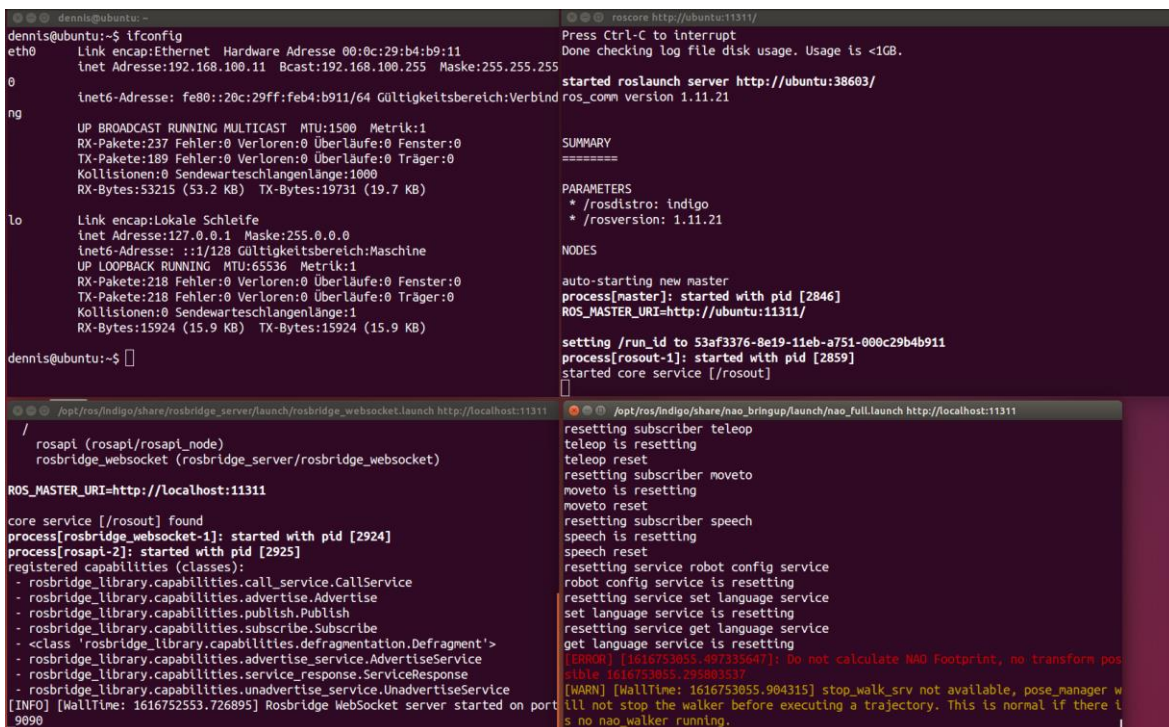


Abbildung 12: Startet die Verbindung mit dem Nao mit der Python SDK

Das waren alle Befehle, die jedes Mal ausgeführt werden müssen, um ROS zu starten und die Verbindung zum Nao herzustellen. Wichtig ist dabei, jeden Befehl in einem separaten Terminal auszuführen. Es sollte auch gewartet werden, bis ein Befehl vollständig durchgeführt wurde, bevor der nächste ausgeführt wird. Eine gute Übersicht wie es beim Ausführen aussehen kann, wird in *Abbildung 13: Alle Terminals zum Starten von ROS & Verknüpfung mit Nao* deutlich.

Im letzten Befehl ist es der Fall, dass ständig ein Footprint Error geworfen wird. Dieser kann aber ignoriert werden. Des Weiteren treten in manchen Fällen gelbe Warnungen auf, die die weitere Funktionalität mit dem Roboter nicht einschränken. Sobald nur noch Footprint Error geworfen werden, ist es das Zeichen, dass die Verbindung erfolgreich war.

Es können aber tatsächliche Fehler auftreten. Üblich sind beispielsweise, wenn der „roslaunch“ aufgrund des „naoqi_driver“ nicht ausführbar ist. Das hängt entweder mit dem Nao-Roboter oder mit den Hintergrundaktivitäten der VM zusammen. Wenn es aufgrund des Nao-Roboters passiert, liegt es daran, dass er überhitzt ist. Die Lösung dafür ist demnach den Nao auszuschalten, um ihn so abzukühlen. Liegt es daran, dass in der VM Hintergrundaktivitäten den naoqi_driver blockieren, muss die VM lediglich erneut hochgefahren werden, um das Problem zu beheben. Falls keines dieser Lösungen hilft, ist die Installation nicht ordnungsgemäß durchgeführt worden.



```
dennis@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet  Hardware Adresse 00:0c:29:b4:b9:11
          inet Adresse:192.168.100.11  Bcast:192.168.100.255  Maske:255.255.255
          inet6-Adresse: fe80::20c:29ff:feb4:b911/64  Gültigkeitsbereich:Verbind
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metrik:1
          RX-Pakete:237 Fehler:0 Verloren:0 Überläufe:0 Fenster:0
          TX-Pakete:189 Fehler:0 Verloren:0 Überläufe:0 Träger:0
          Kollisionen:0 Sendewarteschlangenlänge:1000
          RX-Bytes:53215 (53.2 KB)  TX-Bytes:19731 (19.7 KB)

lo        Link encap:Lokale Schleife
          inet Adresse:127.0.0.1  Maske:255.0.0.0
          inet6-Adresse: ::1/128  Gültigkeitsbereich:Maschine
          UP LOOPBACK RUNNING  MTU:65536  Metrik:1
          RX-Pakete:218 Fehler:0 Verloren:0 Überläufe:0 Fenster:0
          TX-Pakete:218 Fehler:0 Verloren:0 Überläufe:0 Träger:0
          Kollisionen:0 Sendewarteschlangenlänge:1
          RX-Bytes:15924 (15.9 KB)  TX-Bytes:15924 (15.9 KB)

dennis@ubuntu:~$
```

```
roscore http://ubuntu:11311/
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:38603/
ros_comm version 1.11.21

SUMMARY
=====

PARAMETERS
* /roscpp: indigo
* /rosversion: 1.11.21

NODES
auto-starting new master
process[master]: started with pid [2846]
ROS_MASTER_URI=http://ubuntu:11311/

setting /run_id to 53af3376-8e19-11eb-a751-000c29b4b911
process[roscout-1]: started with pid [2859]
started core service [/roscout]
```

```
/opt/ros/indigo/share/rosbridge_server/launch/rosbridge_websocket.launch http://localhost:11311
/
  rosapi (rosapi/rosapi_node)
  rosbridge_websocket (rosbridge_server/rosbridge_websocket)

ROS_MASTER_URI=http://localhost:11311

core service [/roscout] found
process[rosbridge_websocket-1]: started with pid [2924]
process[rosapi-2]: started with pid [2925]
registered capabilities (classes):
- rosbridge_library.capabilities.call_service.CallService
- rosbridge_library.capabilities.advertise.Advertise
- rosbridge_library.capabilities.publish.Publish
- rosbridge_library.capabilities.subscribe.Subscribe
- <class 'rosbridge_library.capabilities.defragmentation.Defragment'>
- rosbridge_library.capabilities.advertise_service.AdvertiseService
- rosbridge_library.capabilities.service_response.ServiceResponse
- rosbridge_library.capabilities.unadvertise_service.UnadvertiseService
[INFO] [WallTime: 1616752553.726895] Rosbridge WebSocket server started on port
9090
```

```
/opt/ros/indigo/share/nao_bringup/launch/nao_full.launch http://localhost:11311
resetting subscriber teleop
teleop is resetting
teleop reset
resetting subscriber moveto
moveto is resetting
moveto reset
resetting subscriber speech
speech is resetting
speech reset
resetting service robot config service
robot config service is resetting
resetting service set language service
set language service is resetting
resetting service get language service
get language service is resetting
[ERROR] [1616753055.897329647]: Do not calculate NAO Footprint, no transform pos
sible 1616753055.295089577
[WARN] [WallTime: 1616753055.904315] stop_walk_srv not available, pose_manager w
ill not stop the walker before executing a trajectory. This is normal if there i
s no nao_walker running.
```

Abbildung 13: Alle Terminals zum Starten von ROS & Verknüpfung mit Nao

5.2 Kamerahalterung

Aus dem Kapitel *Kamerahalterung & Kameras* geht hervor, dass eine Lösung für das Problem mit den Kameras und der Halterung benötigt wird. Dazu wurde zunächst das kaputte Teil identifiziert. Anhand der Stelle, an dem das Teil gebrochen ist, musste eine Anpassung an dem Modell vorgenommen werden. Die Bruchstelle ist nämlich genau entlang der Löcher, in die die Schrauben zum Verbinden der Komponenten hineingehören.

Zur Modellierung einer verbesserten Variante wurde Blender [14] verwendet (siehe *Abbildung 14: Ausschnitt aus Blender, 3D-Modell für Kamerahalterung*). Die Grundform bleibt dieselbe, da es sich sehr gut auf den Kopf des Nao platzieren lässt. Damit die Stabilität jedoch weiterhin gegeben ist, wurden minimale Anpassungen an der Dicke der Form vorgenommen. Das Ziel war es, das Gewicht der Halterung nicht unnötig zu erhöhen, da dies mehr Belastung für den Roboter bedeuten würde.

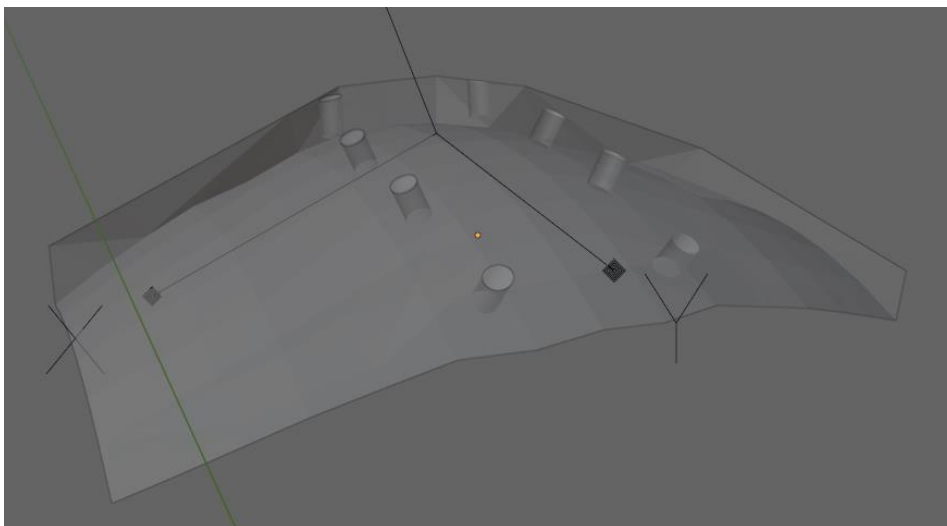


Abbildung 14: Ausschnitt aus Blender, 3D-Modell für Kamerahalterung

Der wichtigste Punkt sind jedoch die Löcher. Diese wurden diesmal nur angedeutet und nicht vollständig tief beigelassen. Die Vermutung ist nämlich, dass die Kamerahalterung aufgrund der Tiefe der Löcher, nicht dem Gewicht der Kameras und der Kamerabox standhalten konnte.

Das 3D-Modell wurde mit einem 3D Drucker vor Ort in der DHBW Karlsruhe gedruckt. Dabei wurde besonders auf die Füllichte geachtet. Bei einer zu geringen Füllichte ist das gedruckte Teil viel zu instabil. Sie sollte aber auch nicht zu hoch sein, da der 3D Druck dann zu schwer wird. Es wurde sich hierbei für ein Mittelmaß sowie eine Gitterstruktur (Einstellung: Cura Infill) entschieden, welches den oberen Schichten mehr Stabilität gibt.

Damit jedoch weiterhin die Löcher für die Schrauben benutzt werden können, wurde mit einem Akkubohrer vorsichtig die Tiefe der Löcher vorgenommen. Durch die manuelle Erstellung der Löcher ist nämlich die 3D-Struktur weiterhin gegeben.

Um dem Problem des Wackelns der Kamera entgegenzukommen, muss die komplette Vorrichtung auseinander gebaut werden. Sobald die einzelnen Komponenten betrachtet werden, ist das Problem eindeutig. Die Schrauben für die Kameras waren nicht mehr zusammen mit der Halterung verbunden. Es war allerdings keine Beschädigung vorhanden. Die Schraube muss durch einen Sturz des Nao-Roboters aus der Kamerahalterung entfernt worden sein.

Durch sorgfältiges Zusammenführen der einzelnen Komponenten war das Wackeln der Kamera nicht mehr vorhanden und sie sitzt wieder fest in der Kamerabox. Nach der Zusammenführung ist die Brille für den Nao-Roboter (siehe *Abbildung 15: Vollständig reparierte Kamerahalterung*) wieder vollständig einsatzbereit und es wird ein Bild angezeigt.

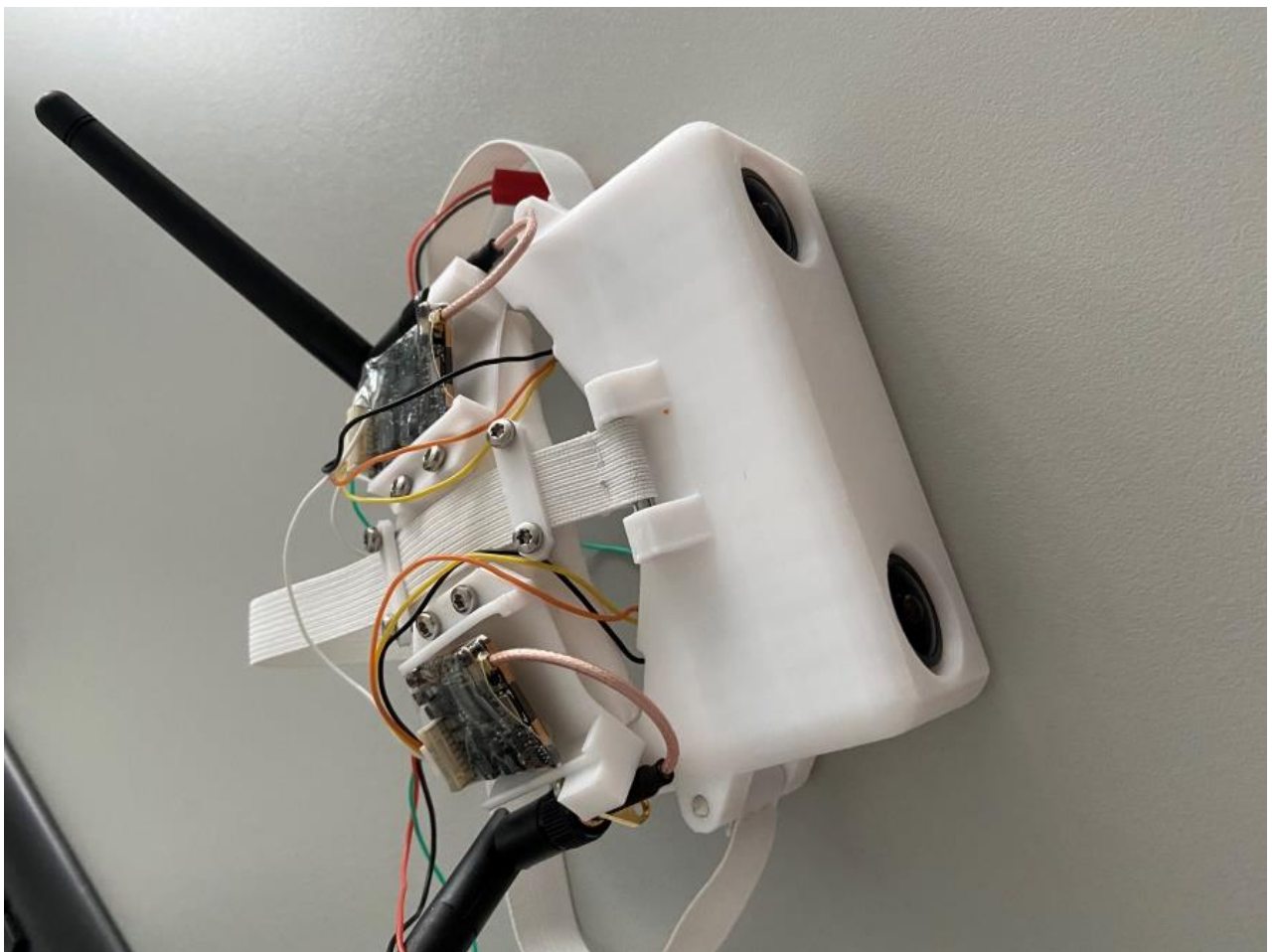


Abbildung 15: Vollständig reparierte Kamerahalterung

5.3 Anpassung der Kameras

Nachdem die Kamerahalterung neu gedruckt wurde und die Kameras in der Kamerabox nicht mehr gewackelt haben, sind neue Erkenntnisse über das übertragene Bild entstanden. Das übertragene Bild bzw. die übertragenen Bilder von den Kameras stimmen nicht aufeinander ab. Das bedeutet, dass der Benutzer nicht das Gefühl bekommt, dass er ein und dieselbe Perspektive besitzt.

Das Phänomen, welches dabei entsteht, nennt sich Motion Sickness bzw. Cyber Sickness. Dieses Thema wurde bereits von den Vorgängern näher erläutert, jedoch sind einige Punkte näher zu betrachten. Dadurch, dass die beiden Bilder für das linke und rechte Auge nicht korrekt aufeinander abgestimmt sind, könnte bei Benutzung der Anwendung Unwohlsein oder gar Schwindelgefühle auftreten. Das Problem ist nämlich, dass der Träger der VR Brille für beide Augen jeweils unterschiedliche virtuelle Realitäten wahrnimmt.

Zuerst war die Annahme, dass beim Zusammenbauen der einzelnen Komponenten die Kameras eventuell verkehrt herum festgeschraubt wurden. Zur Überprüfung dieser These, wurde die Halterung erneut auseinander gebaut und alle Möglichkeiten durchprobiert. Keine Position der Kameras hat ein besseres Ergebnis geliefert. Lediglich noch weiter entferntere Winkel der beiden Kameras waren aufzufinden. Es besteht die Möglichkeit, dass die beiden Kameras doch kleine Schäden beim Stürzen in der vorherigen Studienarbeit erlitten haben, sodass das Bild an sich noch funktioniert, allerdings die Entfernung der Kameras in der Kamerahalterung nicht mehr korrekt ist.

Um nicht mehr Zeit für das Problem zu investieren und den Fokus auf wichtigere Implementierungen setzen zu können, gab es eine andere Idee. Diese bestand darin nur noch eine Kamera für beide Augen in der VR Brille anzeigen zu lassen. Das entfernt zwar die Funktionalität der Tiefenschärfe und des 3D Effekt durch die beiden Kameras, jedoch gibt es kein Motion Sickness Phänomen mehr. Für den Träger der VR Brille ist es nun deutlich angenehmer die Anwendung zu bedienen, da bei dem aktuellen Stand, der Träger ein Auge schließen musste, um gezielte Objekte zu betrachten ohne Doppelbilder bzw. verzerrte Bilder zu erhalten.

Wie das Bild in der VR Brille in der finalen Version aussieht, ist in der nachfolgenden Abbildung zu sehen.



Abbildung 16: Point of View des Nao-Roboters

5.4 Funktionalität Greifen

Als erster Schritt zur Korrektur dieser Funktionalität wurde, nachdem der GrabManager gelöscht wurde, das Modell des Roboters ausgeblendet, sobald der Anwender die Software initialisiert hat. Dadurch, dass Kameras zum Bestandteil der Softwareerfahrung zählen, kann der Anwender so die tatsächlichen Arme des Roboters sehen und somit auch sehen, wenn er Gegenstände greifen möchte.

Eine erste Idee der Umsetzung dieser Funktionalität ist die, dass für jede Hand an jedem Controller des HTC VIVE-Systems ein Button ausgewählt wird, welcher durch Betätigen des Buttons, die Hand des NAO-Roboters schließt. Ein Gedrückthalten des Buttons bedeutet in dem Fall, dass die Hand des Roboters geschlossen wird. Wenn der Anwender den Button wieder loslässt, sollte sich dann auch die Hand des Roboters wieder öffnen. Dies hat zu Beginn auch funktioniert, gegen Ende der Programmiersession dann jedoch nicht mehr, was erstmals auf die Laufzeit des Roboters geschoben wurde. Wie in Punkt *Roboter* beschrieben, ist der Roboter durch die vielen Jahre an der DHBW veraltet und hat viele Probleme mit den Motoren oder dem Akku. Als der erste Prototyp des Greifens umgesetzt worden ist, lief der Roboter schon mehrere Stunden. Deswegen ging das Greifen nach einiger Zeit nicht mehr zu 100%. Das Überprüfen, ob es an der Implementierung oder dem Roboter lag, wurde deshalb auf das nächste Treffen verschoben, da der Roboter so eine Ruhezeit hatte und somit seine Motoren wieder abkühlen konnte.

Aber auch im nächsten Meeting war die Funktionalität nicht zu 100% gegeben. Dies hat den Grund, dass die Manager in Unity so aufgebaut sind, dass diese einmal pro Frame die Update-Methode aufrufen [15]. Wird also ein Mal pro Frame überprüft, ob ein Button gerade gedrückt wird, oder nicht, wird dem Roboter ein Mal pro Frame die Anforderung geschickt, seine Hand zu schließen. Wenn der Roboter oft eine Anforderung erhält, kommt er damit nicht zu recht. Es kann nicht sicher gesagt werden, ob das am Alter des Roboters liegt oder an zu wenig Speicher des Roboters, mit dem er die Anforderungen bearbeitet, da kein neuer Roboter zur Verfügung steht, um dieses Problem nachprüfen zu können.

Mit der zweiten Version dieser Feature-Umsetzung werden zwei neue Bool-Variablen im Code verwendet, durch die die Anforderung des Greifens oder Loslassens des Roboters nur noch einmal kommt und nicht mehr ein Mal pro Frame. Dies bietet aber auch nicht eine 100%ig zufriedenstellende Version, weswegen diese noch ein letztes Mal bearbeitet worden ist. Dadurch, dass für das Greifen einer Hand des Roboters der Button am Controller gedrückt gehalten werden musste, kam nach dem die Message erfolgreich bearbeitet worden ist, wieder eine Message, was nicht der gewünschten Funktionalität entspricht.

Die beste Lösung, welche es nun auch in die finale Version der Umsetzung geschafft hat, ist die, dass es zwar immer noch pro Hand des Roboters einen Button an den Controllern des Anwenders gibt, jedoch müssen diese nun nur noch einmal betätigt werden, damit der Roboter das Greifen beginnt, oder seine Hand öffnet. Drückt der Anwender den Button, schließt der Roboter die Hand, welche zum Controller des Anwenders passt und lässt diese so lange geschlossen, bis der Anwender den Button noch einmal drückt. Damit es nicht wieder zu vielen

Anforderungen, die an den Roboter geschickt werden, kommen kann, wird für das Greifen oder das Öffnen der Hände des Roboters nun eine Sekunde eingeplant, welche die Software abwartet, bevor sie eine weitere Anforderung in Bezug auf die Hände des Roboters entgegennimmt.

Durch diese Umsetzung des Features ist es in allen bisherigen Tests noch zu keinem Fehlverhalten der Anwendung gekommen und der Roboter hat bisher mit 100-prozentiger Zuverlässigkeit seine Hände geöffnet oder geschlossen.

5.5 Implementation der Posen des Nao-Roboters

Aus den Problemen, speziell aus dem Kapitel *4.3 Posen des Nao-Roboters*, geht hervor, dass es keine Möglichkeit gibt, die vorinstallierten Posen im aktuellen Stand zu verwenden.

Eine Möglichkeit besteht darin, innerhalb der Unity Anwendung und somit im C# Code Daten wie bei der Bewegung der Arme über die Bridge an den Nao-Roboter zu senden. Die Problematik dabei ist, dass Unity bzw. das System nicht über die Bibliotheken verfügt, um vorinstallierte Posen zu verwenden.

Die Idee zur Lösung des Problems ist, die Python SDK auf dem System, auf dem Unity läuft, zu installieren. Damit müssen keine weiteren Daten über den Server an den Nao gesendet werden, sondern können direkt aus dem C# Code an den Nao gesendet werden, da auch dieses System im gleichen Netzwerk ist wie der Roboter selbst.

Mithilfe der benutzten SDK und der IP-Adresse des Nao können auf Bibliotheken zugegriffen werden, die benötigt werden. Dazu gehört „ALProxy“, welche die

Posen des Roboters besitzt. Damit können nun Pythonskripte wie in der folgenden Abbildung zu sehen ist, geschrieben werden. Die Pose, die im unten zu sehenden Code verwendet wird, ist „StandZero“.

```
def main(robotIP):  
  
    try:  
        postureProxy = ALProxy("ALRobotPosture", robotIP, 9559)  
    except Exception, e:  
        print "Could not create proxy to ALRobotPosture"  
        print "Error was: ", e  
  
    postureProxy.goToPosture("StandZero", 1.0)  
  
    print postureProxy.getPostureFamily()
```

Abbildung 17: Ausschnitt aus Pythonskript

Die einzige Hürde, die noch besteht, ist es, diese Pythonskripte über die Unity Anwendung zu gegebenen Zeitpunkten auszuführen. Dazu wird im C# Code eine Methode benötigt, die diese Skripte über die Shell bzw. Kommandozeile ausführen kann. Die Methode wird in der *Abbildung 18: C# Methode zum Ausführen von Pythonskripten* näher gezeigt. Als einziger Parameter erhält die Methode die auszuführende Datei. Je nach Pose wird also ein anderer Parameter übergeben.

Das Wichtige dabei ist, dass durch den Aufruf kein anderer State benötigt wird. Näheres dazu im Kapitel *Anpassung der States*. Das bedeutet, dass während ein Skript ausgeführt wird, die Bewegung der Arme und des Kopfes automatisch blockiert wird und erst nach Abschluss dieser Pose, diese wieder freigegeben werden. Die Bewegung in eine solche Pose ist auch garantiert, da durch das

Pythonskript sichergestellt ist, dass auf Basis der aktuellen Position in die gewählte Pose gewechselt werden kann. In Choreographie wird der Unterschied anhand der Pose „Stand“ deutlich. Wenn der Nao sich in der Hocke befindet, muss er sich nur noch aufrecht hinstellen. Liegt der Roboter auf dem Boden erkennt er dies und versucht mit den Armen seinen Rücken zu stützen, sodass er aufstehen kann. Es wird also je nach aktueller Position eine unterschiedliche Bewegung ausgeführt.

```
public void Pose(string fileName)
{
    ProcessStartInfo start = new ProcessStartInfo();
    start.FileName = @"C:\Python27\python.exe";
    start.Arguments = fileName;
    start.UseShellExecute = false;
    start.RedirectStandardOutput = true;
    using (Process process = Process.Start(start))
    {
        using (StreamReader reader = process.StandardOutput)
        {
            string result = reader.ReadToEnd();
            Console.WriteLine(result);
        }
    }
}
```

Abbildung 18: C# Methode zum Ausführen von Pythonskripten

5.6 Nao's Design in VR

Ein weiterer kleiner Punkt in dieser Umsetzung ist die Änderung der Farbe des Robotermodells in der virtuellen Realität. Dieses ist bisher blau und dadurch, dass der Roboter in echt orange ist, nimmt dieses kleine Detail auch etwas die Immersion für den Anwender tatsächlich in einem Cockpit zu sitzen und einen Roboter zu steuern. Hierfür wird das Bild im Programmverzeichnis gesucht, welches die Farbe auf das Robotermodell spannt. Dieses wird in einer Bildveränderungssoftware geöffnet und die blauen Stellen werden auf den Farbwert des Orange des Roboters geändert. Nach einer erneuten Kompilation des Projektes, ist das Robotermodell nach dem nächsten Programmstart nun auch orange. Die folgende Abbildung zeigt das Bild, welches dafür abgeändert worden ist.

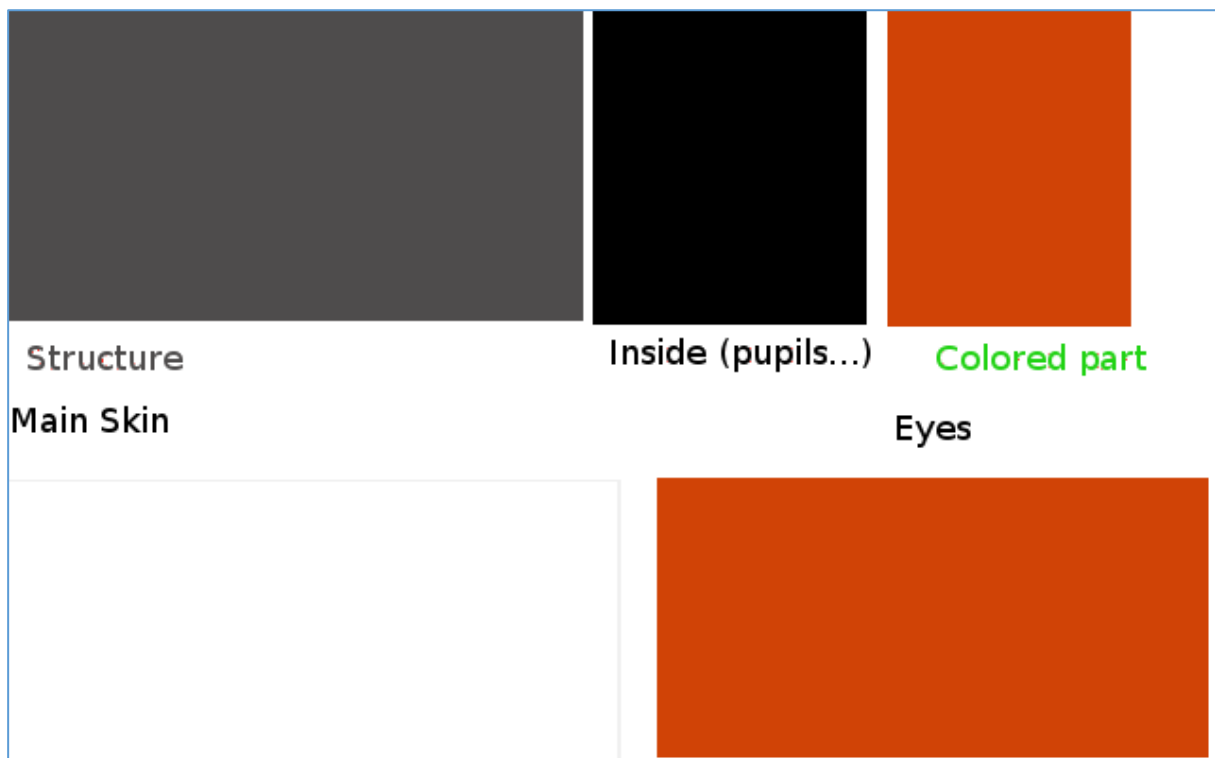


Abbildung 19: Bild anhand, welches das Robotermodell eingefärbt wird

In dieser Abbildung sind einige Farben zu sehen, welche verwendet werden, um das Robotermodell in der virtuellen Realität einzufärben. Hierbei stehen die einzelnen Farben für das, was darüber beziehungsweise darunter steht.

So sind in diesem Fall die Pupillen des Roboters schwarz. Dies wird jedoch nie gesehen, da der Kopf des Roboters zu keinem Zeitpunkt eingeblendet ist. Der hier wichtige Farbwert ist der für „Colored Part“ – oben in grün geschrieben. Diese Farbe wird für die bunten Elemente des Roboters verwendet. Nach dem Ändern auf die oben zu sehende Farbe, ist das Modell des Roboters in der VR orange und somit bietet die Anwendung für den Benutzer dieser eine realistischere Erfahrung, den Roboter zu steuern.

5.7 Anpassung der States

Die bisherige Version der Software, welche mit dem Nao-Roboter verwendet wird, war darauf ausgelegt, dass man den Oberkörper des Roboters nur dann steuern kann, wenn in der Software ein bestimmter Status gesetzt ist. Die verschiedenen Status, welche in der Software verwendet werden, werden durch eine Enumeration dargestellt. Diese Enumeration beinhaltet die folgenden Werte und somit auch die Status, die in der Software erreicht werden können: „init“, „positioned“, „calibrated“, „disarmed“ und „armed“. Hierbei werden die ersten drei Status dafür verwendet, um den Anwender korrekt in der virtuellen Realität zu positionieren. Dies ist wichtig, sodass das Robotermodell an den Anwender angepasst werden kann.

Nachdem der Anwender seine Position kalibriert hat, wird in den ersten für die Anwendung der Software wichtigen Status gewechselt: „armed“. In diesem Status

findet die Bewegung des Oberkörpers des Roboters statt. Der Anwender kann seine Arme und seinen Kopf bewegen und diese Bewegungen werden dann auf den Roboter übertragen.

In der vorigen Umsetzung der Software war es so, dass man für die Lauf-, Bück- oder Drehbewegungen des Roboters den Status von „armed“ auf „disarmed“ wechseln musste. Hierfür wurde ein Button eines HTC VIVE Controllers verwendet. Wird dieser Button betätigt, wird der Status gewechselt und der Roboter übernimmt nicht mehr die Bewegungen des Menschen. So kann der Anwender in Ruhe den Roboter zum Laufen oder Drehen bringen.

Durch die Änderung an der Software, dass die oben genannten Bewegungen nun durch die Auswertung von HTC VIVE Trackern durchgeführt werden, ist der disarmed-Status nun obsolet.

Am Status „armed“ erfolgt ebenfalls eine Änderung. Dadurch, dass das Greifen mit dem Roboter nun immer funktioniert, muss sichergestellt werden, dass der Anwender auch immer sieht, was er greifen möchte. Durch das Anzeigen des Robotermodells in der Sicht des Anwenders, kann dieser die realen Hände des Roboters nicht sehen. Deswegen wird das Robotermodell im Status armed nun ausgeblendet, indem die localScale-Property des transform-Objektes des Roboters auf den Nullvektor gesetzt wird [15]. So sieht der Anwender zwar nicht den für die Anwendung modellierten Roboter, jedoch sieht er den realen Roboter in der VR-Brille, was die Anwendung ebenfalls etwas realistischer gestaltet.

5.8 Tracker in Steam VR

Der größte Punkt, der für die Umsetzung der Studienarbeit relevant ist, ist das Bewegen des Roboters mit Hilfe von Trackern, die am Benutzer festgemacht werden. Damit das funktioniert müssen die HTC VIVE Tracker mit SteamVR gekoppelt werden. Dafür werden die benötigten Treiber installiert, sodass die Tracker als neue Objekte in SteamVR hinzugefügt werden können. Des Weiteren müssen die Empfänger der Tracker ebenfalls mit dem PC verbunden sein, sodass diese ohne Probleme erkannt werden können. Nach Einschalten der Tracker werden diese als zusätzliche Objekte in der VR Umgebung von Steam erkannt. Diese können nun für SteamVR oder, im hier benötigten Fall, für Unity verwendet werden.

5.9 Tracker in Unity

Damit die Tracker nun auch in Unity ohne Probleme erkannt werden, muss für jeden Tracker ein neues Objekt in Unity erzeugt werden. Diese werden im linken Bereich unter der Kamera erzeugt.

Für jedes Objekt kann nun angegeben werden, welches SteamVR-Objekt es in der VR abbildet. Nachdem hierfür die korrekten Tracker aus SteamVR ausgewählt worden sind, kann im Programmcode auf die Tracker reagiert werden [15]. Darauf wird jedoch genauer unter Punkt *Auswertung Tracker* und Punkt *Synchronisierung der Bewegungen* eingegangen.

Beim Starten der Anwendung unter Unity, erkennt der Anwender die Tracker jedoch noch nicht. Das liegt daran, dass es für die Tracker keine auf dem Rechner

installierten 3D-Modelle für die VR-Umgebung gibt. Es wurde einfach Abhilfe geschaffen, in dem für die Controller andere Formen zur Ansicht gewählt worden sind. So können die Tracker auch dann erkannt werden, wenn der Benutzer die VR Brille aufhat und das Programm noch nicht richtig läuft.

5.10 Tracker am Benutzer

Das größte Ziel dieser Studienarbeit ist das Steuern des Roboters mit Hilfe der Tracker. Dafür müssen die Tracker am Anwender angebracht werden. Durch die Vorarbeit an diesem Projekt waren an der DHBW Karlsruhe bereits zwei zusätzliche Tracker für das HTC VIVE System vorhanden. Für das Anbringen der Tracker am Körper des Anwenders wurden ebenfalls Halterungen gedruckt, die mit Hilfe eines Bandes am Anwender angebracht werden können.

Die Halterungen sind für die tatsächliche Umsetzung der Bewegungen robust genug, sodass keine neuen Halterungen entwickelt und gedruckt werden müssen.

Ein Problem waren die momentan verfügbaren Bänder für die Halterungen. Es handelt sich um einfache Klettbänder, welche für einen durchschnittlichen Knöchel viel zu eng sind. Durch das Besorgen neuer, größerer Klettbänder kann diese Problematik aus der Welt geschafft werden und die Tracker können nun ohne Probleme und ohne Schmerzen nach langer Benutzung am Anwender angebracht werden.

Da sich die Tracker beim Anschalten erst in den virtuellen Raum begeben müssen, kann es vorkommen, dass sich der Tracker unverhältnismäßig in der virtuellen Umgebung umherbewegt. Das lässt sich in Steam VR sehr gut beobachten. Dies kann ebenfalls passieren, wenn die Unity Anwendung gestartet wird. Es muss also bei der Auswertung beachtet werden, dass sich der Tracker nicht nur in der realen Umgebung in einer Ruheposition am Körper des Benutzers befindet, sondern dies auch in der virtuellen Umgebung so wahrgenommen wird. Andernfalls kann es zu Fehlern kommen, sobald die Tracker ausgewertet werden und Funktionalitäten erhalten. Später wird dazu weiter im Kapitel *Auswertung Tracker* darauf eingegangen.

5.10.1 Prototyp der Trackerverwendung

Für die tatsächliche Umsetzung der Bewegung des Roboters durch das Bewegen als Anwender, wurde zuerst der erste offensichtliche Ansatz gewählt: Das Anbringen der Tracker an jeweils einem Fuß des Anwenders. So wäre es möglich, dass der Anwender mit jeweils einem Bein das passende Bein des Roboters bewegen kann. Die folgende Abbildung zeigt die Anbringung der Tracker im ersten Stadium der Umsetzung.



Abbildung 20: Prototyp der Anbringung der Tracker

Nach ersten Tests der Software wurde diese Idee aber nicht weiter realisiert. Dadurch, dass sich der Anwender in der realen Welt so bewegen kann, wie er will, wäre die Sicherheit des Roboters nicht mehr gewährleistet. Auf die unvorhersehbaren Bewegungen des Anwenders kann die Software nicht reagieren, sodass eine normale und gute Nutzung der Software nicht mehr gegeben wäre.

Zudem gäbe es Probleme bei der Umsetzung der Funktionalität, dass der Roboter in die Knie gehen kann. Wenn die Tracker jeweils an den Knöcheln des Anwenders angebracht werden, kann bei einem in die Knie gehen des Anwenders

nicht ermittelt werden, ob der Roboter ebenfalls in die Knie gehen muss, da die Knöchel des Anwenders beim in die Knie gehen nicht in eine für die Software gut bemerkbare Position wechseln, sodass der Roboter auf diesen Positionswechsel reagieren könnte.

Somit muss das Konzept der Roboterbewegung neu erarbeitet werden und es muss so erarbeitet werden, dass es nicht zu Unfällen des Roboters kommt, sodass sichergestellt ist, dass der Roboter nicht kaputt gehen kann und dass der Anwender die Software ohne Probleme verwenden kann.

5.10.2 Finale Version der Trackerverwendung

Nach gründlicher Überlegung ist die Verwendung der Tracker nun in einem zufriedenstellenden Zustand. Da die Idee mit dem identischen Bewegen des Roboters verworfen wurde, sind beide Tracker an den Knöcheln nicht mehr notwendig.

Es besteht nun die Umsetzungsüberlegung, dass nur ein Tracker am rechten Knöchel des Anwenders angebracht wird und der zweite Tracker am Bauch des Anwenders. So kann ein Tracker für das Laufen des Roboters verwendet werden und der andere für das in die Knie gehen und das Drehen des Roboters, wenn der Anwender diese Bewegungen durchführt.



Abbildung 21: Finale Anbringung der Tracker am Anwender

In der oben zu sehenden Abbildung ist die finale Positionierung der Tracker zu erkennen. Der Tracker mit dem Sticker wird am rechten Fuß angebracht und der andere Tracker am Bauch. Für den Tracker am Bauch werden mehrere Klettbinden verwendet, da hier ein längeres Band benötigt wird. Auf die genaueren Funktionalitäten und den Zweck der Anbringung der Tracker wird in *Punkt Auswertung Tracker* eingegangen.

5.11 Auswertung Tracker

Durch die gerade beschriebenen Positionierungen der Tracker am Körper des Anwenders der Software, sollen die Tracker folgendermaßen in der VR ausgewertet werden:

- Für das Bewegen soll die Position des Trackers am Bein in Y-Bewegung im VR-Raum ausgewertet werden
- Für das in die Knie gehen soll die Position des Trackers am Bauch in Y-Bewegung im VR-Raum ausgewertet werden
- Für das Drehen des Roboters soll der Winkel des Trackers am Bauch ausgewertet werden

Jeder dieser Punkte wird im zugehörigen Punkt des Umsetzungskapitels genauer beschrieben.

5.12 Synchronisierung der Bewegungen

Das folgende Kapitel befasst sich mit der Synchronisierung des Roboters anhand der ausgewerteten Bewegungen der Tracker, welche am Anwender der Software angebracht sind.

Wie bereits beschrieben wird sich der Roboter nicht zu 100% identisch zum Anwender bewegen. Dies hat den Grund, dass der Roboter ein unterschiedliches Gleichgewichtsgefühl zum Menschen hat und viel leichter umfällt als ein Mensch. Aufgrund der vorigen Tests und vor allem der vorigen Studienarbeit konnte dies festgestellt werden [1].

Des Weiteren kann eine komplett identische Bewegung des Roboters nicht mit nur zwei Trackern umgesetzt werden. Hierfür müsste an jedem Gelenk des Menschen ein Tracker angebracht werden, da die Bewegung der Beine komplexer ist als eine einfache Armbewegung. Zudem kann sonst nicht sichergestellt werden, dass der Roboter zu Schaden kommen würde, wenn eine unerwartete Bewegung der Beine gemacht werden würde.

In einem Gespräch mit den Vorarbeitern dieses Projektes, wurde von ihnen erwähnt, dass der Roboter oft gestürzt ist, obwohl in der vorigen Arbeit das Hauptaugenmerk auf die Bewegung des Oberkörpers gelegt worden ist.

Für die Umsetzung der einzelnen Punkte für die Tracker wurden die Tracker als „SteamVR_TrackedObject“ im Programmcode initialisiert. Bei einem „SteamVR_TrackedObject“ handelt es sich um eine Klasse aus dem Valve.VR-Namespace [16], welche verwendet wird, um Objekte aus der realen Welt in der VR zu analysieren und auszuwerten. Hierfür muss dann am Objekt eine ID gesetzt werden, sodass die Software weiß, welches Objekt aus der realen Welt ausgewertet werden muss [15].

5.12.1 Laufen

Die Grundfunktionalitäten für das Laufen des Roboters war wegen der Vorarbeit an diesem Projekt bereits durch zwei Methoden für das Starten und Beenden des Laufvorgangs gegeben [1]. Bisher war es so, dass der Anwender den aktuellen Status der Software wechseln musste, damit der Roboter auf das Drücken eines Knopfes am Controller des HTC VIVE Systems reagieren kann und somit den Laufvorgang startet oder beendet. Durch Testen der Anwendung, das Aufrufen der

Methoden zum Starten und Beenden des Laufvorgangs wurde auf einen Tastendruck auf der Tastatur gelegt, konnte die Funktionalität der Methoden verifiziert werden. Es muss nur noch die Umsetzung mit Hilfe des Trackers erfolgen.

Hierfür wird in einem ersten Versuch eins der beiden SteamVR_TrackedObject-Objekte verwendet. Der WalkerController ist eine Klasse im Code des Projektes, welches sich um das Laufen des Roboters kümmert. In dieser Klasse sind die oben erwähnten Methoden zu finden. Jeder Controller in SteamVR ist so aufgebaut, dass ein Mal pro Frame eine Update-Methode aufgerufen wird [15]. In dieser Methode können Dinge überprüft werden und vor allem andere Dinge gestartet oder beendet werden. In diesem ersten Versuch der Umsetzung wird die aktuelle Position des Trackers am Fuß in einer weiteren Variablen gespeichert und am Ende der Methode mit der aktuellen Position des Trackers verglichen. Hierfür werden einfach die Positionen in Y-Richtung miteinander verglichen. Besteht ein Unterschied, läuft der Roboter eine Sekunde in die aktuell anvisierte Richtung. Dies hat am Anfang auch funktioniert, jedoch zu gut. Der Anwender hat eine Laufbewegung gemacht und es kam zu einer Differenz in Y-Richtung, der Roboter bewegt sich also. Da die Update-Methode ein Mal pro Frame aufgerufen wird, wird diese Überprüfung auch ein Mal pro Frame gemacht und beim kleinsten Unterschied bewegt sich der Roboter. Dies hat zur Folge, dass der Roboter immer zu Laufen begann, auch wenn der Anwender keine tatsächliche Laufbewegung gemacht hat. Es muss also ein anderes Konzept erarbeitet werden.

Die Positionierung des Trackers im realen Raum bleibt der wichtige Punkt für die Bewegung des Roboters, jedoch soll die Bewegung jetzt nicht mehr anhand der

Differenz in Y-Richtung initiiert werden. Stattdessen wird in der Update-Methode nun überprüft, ob die Position des Trackers einen bestimmten Wert überschreitet. Beim Starten der Anwendung wird der Tracker mit einem Nullvektor initialisiert. Hebt der Anwender nun das Bein, an dem sich der Tracker befindet und der Wert in Y-Richtung überschreitet den benötigten Schwellwert, führt der Roboter eine Laufbewegung durch. Auch hier kam es durch den Aufbau in der Programmierung in Unity und SteamVR zu Problemen mit dem Roboter, da die Aufforderung zum Laufen auch dann kommen konnte, wenn der Roboter seinen Laufvorgang noch gar nicht beendet hatte. Um weiteren Problemen entgegenwirken zu können, wurde eine Bool-Variable eingeführt, welche den Laufprozess des Roboters abschirmt, sodass keine weiteren Befehle an den Roboter geschickt werden können. Zudem wurde die Laufanimation von einer Sekunde auf 1.25 Sekunden erweitert. Nach Tests mit der Anwendung ist diese Zeit näher an der Laufbewegung des Menschen, sodass dies noch etwas realer wirkt. Es folgt der Code, der für diese Funktionalität relevant ist.

```
var walkPosition = _walkTracker.transform.position.y;
if (walkPosition > 0.75)
{
    _walking = true;
    walkAhead();
    System.Threading.Thread.Sleep(1250);
    stopMoving();
    _walking = false;
}
```

Abbildung 22: Code für das Laufen des Roboters

In der Abbildung ist die korrekte Ausführung der Idee zu erkennen. Die Position des Trackers wird ermittelt und anhand eines Wertes wird überprüft, ob der Roboter laufen soll oder nicht.

5.12.2 Hocke & Aufstehen

Für die Möglichkeit des Roboters in die Hocke zu gehen und wieder aufzustehen, wird die Position des Trackers am Bauch des Anwenders ausgewertet. Hierfür wird ebenfalls im WalkerController in der Update-Methode die aktuelle Position des Trackers ermittelt. Da der Fehler, dass zu viele Arbeitsanweisungen in der Update-Methode an den Roboter geschickt werden, bereits bei der Laufumsetzung gemacht worden ist, wurde hier ebenfalls direkt auf einen bestimmten Wert im Code geprüft, sodass der Roboter in die Hocke gehen kann. Dadurch, dass der Roboter auch nicht nur halb in die Hocke gehen kann, war eine andere Umsetzung dieser Funktionalität gar nicht möglich. So wird nun überprüft, ob die Trackerposition unter einem bestimmten Wert ist. Ist dies der Fall, startet der Roboter die Hockanimation. Hierfür wird ein Pythonskript verwendet, welches ausgeführt wird, um den Roboter in die CROUCH-Pose zu versetzen.

Steht der Anwender nun auf, erreicht der Tracker in Y-Richtung einen bestimmten Wert, sodass der Roboter nun aufsteht. Hierfür wird ebenfalls ein Pythonskript verwendet. Warum Pythonskripte anstelle der Realisierungen der vorigen Programmierer dieses Projektes verwendet werden, kann unter Punkt *Implementation der Posen des Nao-Roboters* nachgelesen werden. Es folgt der für diese Funktion relevante Code.

```
var crouchPosition = _turnTracker.transform.position.y;
if (crouchPosition < 0.7 && !_crouched)
{
    _crouched = true;
    stiffnessController.speech.Pose(stiffnessController.speech.CROUCH);
}
else if (crouchPosition > 2.5 && _crouched)
{
    _crouched = false;
    stiffnessController.speech.Pose(stiffnessController.speech.STAND_INIT);
}
```

Abbildung 23: Code für das in die Hocke gehen des Roboters

In dieser Abbildung ist der benötigte Code zu sehen. Eine weitere Besonderheit in diesem Code ist die Bool-Variable „_crouched“. Diese wird verwendet, damit der Roboter nicht öfter den Auftrag, in die Hocke zu gehen oder aufzustehen erhält. Da die Arbeit mit dem Roboter einige Probleme mit sich zog (siehe Punkt *Roboter gebraucht*), musste für den Code eine Idee entwickelt werden, sodass der Roboter nicht zu schnell von seinen zu heißen Motoren klagt. In diesem Codebeispiel ist ebenfalls der Methodenaufruf zu erkennen, welche verwendet wird, um die Pythonskripte aufzurufen und auszuführen. Darauf wird jedoch in *Implementation der Posen des Nao-Roboters* näher eingegangen.

5.12.3 Drehen des Roboters

Für das Drehen des Roboters wird ein ähnlicher Ansatz wie für das Laufen und in die Hocke gehen gewählt. Es wird sich vom Tracker, welcher sich am Bauch des Anwenders befindet, die Startposition zum Zeitpunkt des Aufrufs der Update-Methode gemerkt und wenn der Anwender sich dreht, wird der so entstehende Winkel des Trackers, also die Bewegung vom Startpunkt zum aktuellen Punkt, ausgewertet. Es wurde erst ein Ansatz gewählt, durch den es für den Anwender möglich gewesen wäre, den Roboter winkengenau bewegen zu können. So wären

Drehungen im vollen Spektrum eines Kreises möglich gewesen. Jedoch gab es auch hier Probleme mit dem Alter des Roboters und der Codestruktur von Projekten in Unity. Auch hier wird die Update-Methode verwendet, welche ein Mal pro Frame aufgerufen wird. Dadurch, dass die HTC VIVE Brille eine Bildwiederholrate von 90Hz hat [8], wären das 90 Aufrufe der Update-Methode pro Sekunde. Die Ermittlung des Winkels des Roboters hat gleich auf Anhieb funktioniert, die Drehung des Roboters in die anvisierte Position wäre also ohne Probleme möglich gewesen, jedoch kam dieser mit 90 Befehlen pro Sekunde für die Drehung nicht zurecht und die Anwendung zeigte keine Funktionalität mehr. Deswegen wurde ein neues Konzept erarbeitet, durch das es dem Anwender nur noch möglich ist, den Roboter in 90° Schritten zu bewegen. Hierfür wird überprüft, ob sich der Winkel, welcher der Tracker bei der Bewegung des Anwenders zurückgelegt hat, in der Nähe von 90° befindet, um so dem Anwender noch einen kleinen Spielraum bieten zu können. Ist dies der Fall, wird die Drehbewegung des Roboters initiiert und dieser dreht sich dann in die vom Anwender aufgeforderte Richtung. Eine kleine Schwierigkeit in dem Fall war die Ermittlung, in welche Richtung sich der Anwender gedreht hat, um den Roboter in die korrekte Richtung drehen zu können. Der hierfür benötigte Code wird im Folgenden gezeigt und näher erläutert.


```
private Tuple<bool, float, float> GetRotateDirection(float from, float to)
{
    float clockWise = 0f;
    float counterClockWise = 0f;

    if (from <= to)
    {
        clockWise = to - from;
        counterClockWise = from + (360 - to);
    }
    else
    {
        clockWise = (360 - from) + to;
        counterClockWise = from - to;
    }
    var direction = (clockWise <= counterClockWise);
    return new Tuple<bool, float, float>(direction, clockWise, counterClockWise);
}
```

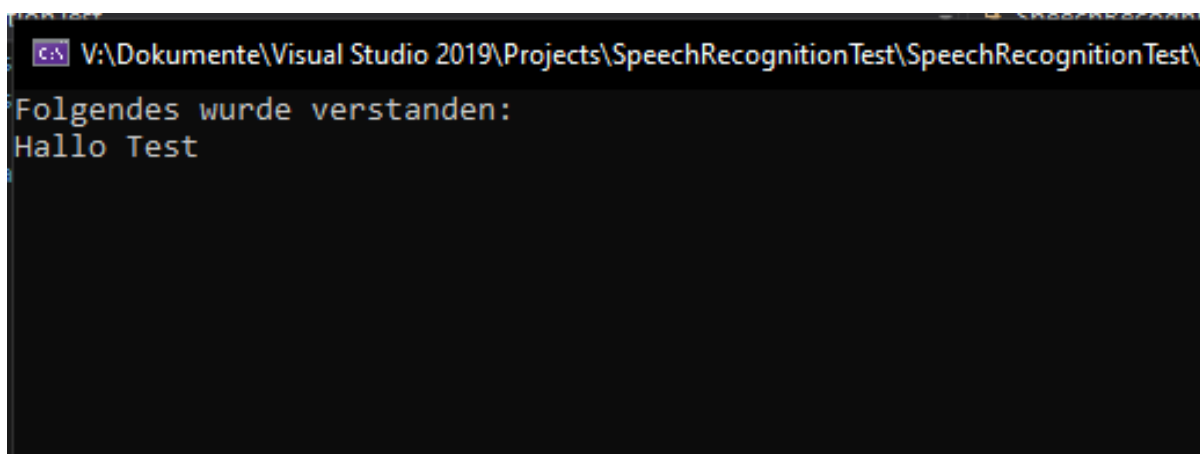
Abbildung 24: Code für die Ermittlung der korrekten Rotationsrichtung

In der Abbildung oben ist die Methode „GetRotateDirection“ zu erkennen. Diese erwartet zwei Parameter: Den Startwinkel des Trackers und den Winkel des Trackers, an welchem sich der Tracker zum Zeitpunkt des Methodenaufrufs befindet. Anhand dieser Informationen werden die Variablen „clockWise“ und „counterClockWise“ berechnet, welche im späteren Verlauf des Programms für die Drehung des Roboters zuständig sind. Je nach dem, in welchem Fall der if-Anweisung das Programm abläuft, werden beide Variablen berechnet und am Ende in das Tupel gesteckt, welches als Rückgabe-Objekt der Methode verwendet wird. Der erste Wert des Tupels ist eine bool-Variable, welche darstellt, in welche Richtung und somit auch, welche der beiden anderen Werte des Tupels, für die Drehung des Roboters zur Überprüfung des Winkels im Bereich von 90° verwendet werden soll.

5.13 Sprachfeature des Roboters

Eine Idee für ein weiteres Feature des Roboters in Bezug auf diese Studienarbeit, war die Möglichkeit, den Roboter sprechen zu lassen. Dadurch, dass der Nao-Roboter mit Lautsprechern und einer Sprachsoftware für verschiedene Sprachen ausgestattet ist, würde sich ein solches Feature anbieten. Da der Roboter den Anwender nach Start der Applikation ebenfalls sprachlich begrüßt, ist diese Erweiterung der Anwendung sinnvoll. Eine Umsetzung einer eigenen Engine für das Zuhören und Nachsprechen ist nicht geplant, da Microsoft hierfür schon eine eigene Api anbietet [17].

Um die Funktionalität der Api nachzuweisen, wurde eine Konsolenanwendung geschrieben, welche die Api verwendet und den gehörten Text in der Konsole ausgibt. Die folgende Abbildung zeigt die Funktionalität der Testanwendung.



```
C:\V:\Dokumente\Visual Studio 2019\Projects\SpeechRecognitionTest\SpeechRecognitionTest>
Folgendes wurde verstanden:
Hallo Test
```

Abbildung 25: Ergebnis der Testanwendung SpeechRecognitionTest

In der Abbildung ist zu erkennen, dass der Text, welcher vom Anwender gesprochen wurde, in der Konsole ausgegeben wird. Somit steht die Testanwendung für einen Prototyp der Umsetzung in der Software der Studienarbeit. Die in der Testanwendung verwendete Api kann jedoch nicht verwendet werden, da es für Unity eine eigene Api gibt, die aber dieselben Funktionalitäten bietet.

Diese wird nun eingebunden und beim Start der Anwendung wird der DictationRecognizer der Api initialisiert, sodass dieser bei Bedarf verwendet werden kann. Hierfür werden auch zwei Events der Klasse abonniert, welche in einem Fehlerfall oder nach erfolgreichem Zuhören des Recognizers benötigt werden.

Damit der Roboter dem Anwender zuhören kann, ist diese Funktionalität wieder auf eine Schaltfläche am Controller gelegt, sodass der Anwender den Sprachmodus selbstständig starten und beenden kann. Betätigt der Anwender nun eine Schaltfläche, wird je nach vorigem Zustand das Zuhören des Roboters aktiviert oder deaktiviert. Im aktiven Zustand des Zuhörens kann der Anwender nun Dinge sagen, die so lange in einer Variablen gespeichert werden, bis der Anwender erneut die Schaltfläche betätigt. Ist dies der Fall, wird die Variable durch die Sprachfähigkeit des Roboters ausgegeben und geleert, damit der Roboter die gleichen Dinge nicht mehrmals sagt – es sei denn, der Anwender sagt genau das Gleiche nochmal.

Die Umsetzung ist aber auch nicht die perfekte Variante. Bei der Implementierung gab es die Probleme, dass ab und zu kein Sprachfeedback des Roboters zustande kam. Dies lag daran, dass bei der Ausführung der Anwendung diese im Debug-Modus gestartet worden ist und deshalb der Fokus nicht auf der Anwendung, sondern auf Visual Studio lag. Damit das Sprachfeature des Roboters also korrekt funktioniert, darf der Fokus der Anwendung nicht verlassen werden.

5.14 Veränderte Bedienung der Software

Zusammenfassend wird hier die Bedienung der Anwendung zum vorherigen Stand gegenübergestellt. Der Beginn der Ausführung der Software bleibt bestehen. Es muss wie vorher auch zunächst die VM gestartet werden sowie die Befehle (siehe *5.1 Inbetriebnahme VM und Verbindung zu ROS*) ausgeführt werden, damit die Verbindung von Unity und dem Nao-Roboter hergestellt werden kann. Der Benutzer startet anschließend die Unity Applikation und setzt die Brille auf. Der Unterschied ist jetzt dabei, dass neben den Controllern in den Händen noch vorher die Tracker am Bein und am Bauch angelegt werden müssen. Als nächstes muss sich der Anwender mit Hilfe des rechten Grip Buttons kalibrieren und daraufhin mit dem linken Grip Button mit dem Nao synchronisieren.

Aufgrund der State Änderung besteht nicht mehr die Möglichkeit zwischen dem „armed“ und dem „disarmed“ State zu wechseln. Näheres dazu befindet sich im Kapitel *5.7 Anpassung der States*.

Des Weiteren gibt es nicht mehr die Möglichkeit mit Hilfe des linken HTC VIVE Controllers die Fortbewegung des Naos auszuführen. Somit fallen „Turn Left“, „Turn Right“, „Walk Forward“ sowie „Crouch“ und „Stand up“ weg. Anstelle von „Walk forward“ wird dieser Knopf für das Sprachfeature verwendet.

Die Funktionalitäten dieser Knöpfe fallen jedoch nicht weg. Sie sind lediglich mit den Trackern ermöglicht worden, damit die Interaktion mit dem Nao realitätsnäher geworden ist.

Die beiden hinteren Tasten bei den Controllern werden weiterhin für das Greifen benutzt. Der Unterschied besteht darin, dass es jetzt konstant ausführbar vonstattengeht und jeweils das Drücken der Taste das Öffnen und Schließen der Hand durchführt, je nachdem, ob die Hand geöffnet ist oder nicht.

Eine Übersicht über die Veränderungen der Bedienung der Controller zur vorherigen Studienarbeit ist in der folgenden Darstellung nochmals aufgezeigt.

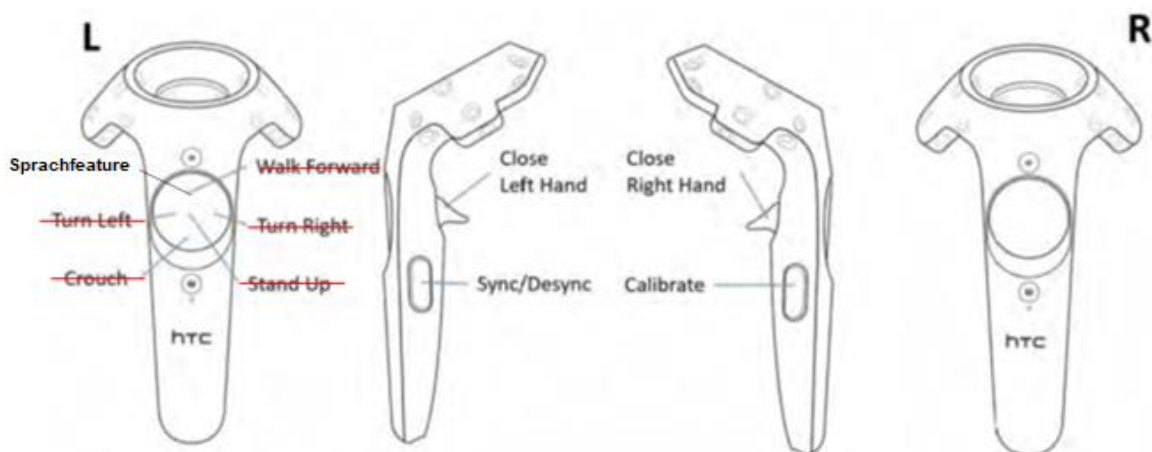


Abbildung 26: Veränderte Bedienung der Controller der HTC Vive

6 Evaluation

Dieses Kapitel befasst sich mit der Evaluation des Projektes und der möglichen Verbesserungen beziehungsweise Probleme bei der Softwareerfahrung.

6.1 User Experience

Was die User Experience bei Nutzung der Software angeht, kann gesagt werden, dass sich diese klar verbessert hat. Dadurch, dass der Anwender nun immer die Möglichkeit hat, den Roboter frei im Raum zu bewegen, erhöht das die Immersion, sodass der Anwender nun mehr als vorher denkt, tatsächlich in einem Cockpit eines Roboters zu sitzen. Die Tracker sind durch die neuen Klettbander viel angenehmer zu tragen und der Anwender kann so auch die Software nach einigen Minuten des Nutzens ohne kleinere Schmerzen wieder beenden.

Da jetzt für die Sicht des Anwenders nur noch eine Kamera verwendet wird, welche das gleiche Bild auf beiden Augen projiziert, ist das anfängliche Problem der Motion Sickness nicht mehr gegeben. Das Bild ist durch diese Verwendung und den Neudruck der Brille für den Roboter besser als bei der vorigen Arbeit, sodass der Anwender mit keinen Problemen bei der Softwareerfahrung zu rechnen hat.

Des Weiteren ist die Tatsache, dass das Modell des Roboters nach der korrekten Initialisierung ausgeblendet wird, ebenfalls besser für die User Experience. So kann der Anwender nun den tatsächlichen Roboter sehen und ebenfalls auch das sehen, was er gerade greifen möchte, oder wenn sich bereits etwas in einer der Hände des Roboters befindet.

Durch die korrekte Programmierung der neuen und alten Funktionalitäten ist nun auch sichergestellt, dass jeder der Knöpfe, welche für eine Funktionalität in der Software verwendet wird, auch immer das macht, was von ihm erwartet wird. Die Sprachfunktion funktioniert ohne Probleme und auch das Greifen der Roboter funktioniert zuverlässig.

Für die User Experience gibt es aber auch einige Wermutstropfen. So ist es dem Alter des Roboters zu schulden, dass nach einiger Benutzung der Software, die Motoren des Roboters zu heiß werden. So ist es nicht mehr gegeben, dass der Roboter zu 100% das ausführt, was der Anwender möchte. Zudem ist die Akkulaufzeit des Nao auch nicht mehr zufriedenstellend. Bei fast jeder Nutzung der Software muss der Roboter am Ladekabel hängen, da der Akku sonst nach einigen Minuten leer geht und so eine Benutzung der Software nicht mehr möglich ist. Die Tatsache, dass der Roboter am Ladekabel hängt, grenzt den Anwender auch bei der Verwendung der Lauf- oder Drehbewegungen ein. Allzu weit kann der Roboter nämlich nicht laufen, wenn er am Kabel hängt und wenn sich der Anwender zu oft in die gleiche Richtung drehen würde, würde sich der Roboter im Ladekabel verfangen. Sobald dies der Fall ist, kann der Nao über das Kabel stolpern und stürzt auf den Boden.

6.2 Latenz in der Ausführung

Ein großes Problem bei der Verwendung der Software war die immer größer werdende Latenz bei längerer Ausführung der Software. Dies hat zum einen den Grund, dass die Motoren des Nao zu heiß wurden. Dies resultierte in einer Arbeitsverweigerung des Roboters. Er ahmte keine Bewegungen mehr nach und wenn doch, dann zeitverzögert. Um den zu heißen Motoren entgegenzuwirken, wurde der Nao runtergefahren und nach einer Pause, durch die es auch nicht mehr möglich war, am Projekt weiterzuarbeiten, wurde der Roboter wieder gestartet. Nicht selten war es aber der Fall, dass die Bearbeitung am Projekt für den Tag beendet werden musste, da der Roboter, obwohl eine Pause durchgeführt wurde, nicht mehr weitermachen wollte.

Manchmal war die Verbindung zur VM auch der Grund für zu lange Latenzzeiten. Dies konnte durch einen Neustart der VM korrigiert werden. Die genaue Grundursache, warum es zu diesem Problem kommen konnte, ist nicht bekannt. Meist war es ausreichend die Befehle, welche den roscore, den Bridge-Server und die Verbindung zum Nao initialisiert, neu auszuführen. Das führte zum Resultat, dass die Latenz wieder geringer wurde und die Benutzung angenehmer gestaltet ist.

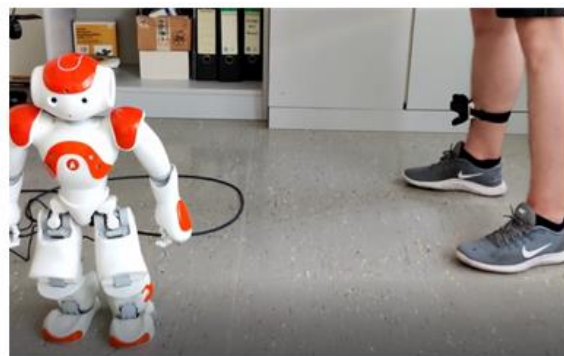
Ein weiteres Problem, welche die Bearbeitung des Projektes ins Stocken brachte, war der Akku des Roboters. Dieser war, wenn der Nao nicht am Ladekabel hing, nach kurzer Zeit bereits leer und der Roboter hat sich selbstständig runtergefahren. Aber den Nao die ganze Zeit am Kabel lassen war auch keine Möglichkeit, da so der Laufradius eingegrenzt war und durch die ständige

Verbindung zum Strom wurden die Motoren des Nao schneller heiß, welches dann im ersten genannten Problem dieses Kapitels resultierte.

Die Bewegung der Arme und dessen Latenz wurde bereits in der vorherigen Studienarbeit dargestellt. Zum Vergleich wird nun die Latenz der Bewegung des Nao mit den Trackern in Betracht gezogen. Dazu wird die Drehung aus einem Video mit 30 FPS (siehe *Abbildung 27: Ausschnitte aus einem Video*) gemessen. Nachdem der Benutzer sich gedreht hat, vergehen 10 Bilder. Das entspricht einer Latenz von 330ms. Das ist eine etwas geringere Latenz als die 360ms, die bei der Bewegung der Arme in dem vorherigen Stand ausgewertet wurde [1]. Das könnte auf die schnellere Datenübertragung mithilfe der Pythonskripte beruhen. Jedoch könnte das auch daran liegen, wie lange der Nao bereits im Einsatz war.



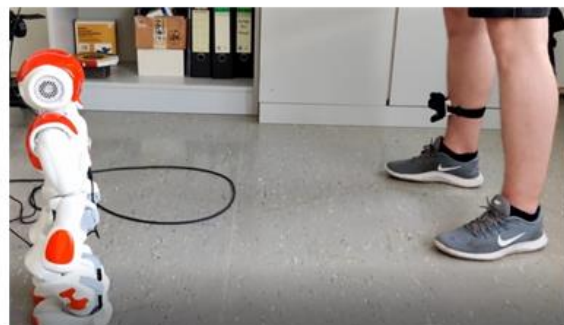
1. Ausgangssituation



2. Ende der Bewegung beim Benutzer



3. Nao beginnt Bewegung



4. Ende der Bewegung beim Nao

Abbildung 27: Ausschnitte aus einem Video

7 Fazit

In diesem Kapitel wird ein Fazit für das Projekt gezogen. Es fasst die Arbeit noch einmal zusammen und bietet einen Ausblick auf vielleicht weiterführende Arbeiten an diesem Projekt.

7.1 Zusammenfassung

Zusammenfassend lässt sich sagen, dass diese Arbeit erfolgreich durchgeführt werden konnte. Dadurch, dass die Aufgaben klar definiert waren, konnten diese auch umgesetzt werden. Jedoch konnte nicht sofort an den Neuprogrammierungen gearbeitet werden, da der alte Stand der Arbeit noch einige Fehler aufwies. So mussten diese Funktionen erst korrigiert und vor allem so programmiert werden, dass sie durchgängig und zuverlässig funktionieren. Nachdem diese Funktionen korrekt umgesetzt werden konnten, war es nun möglich die neuen Programmierungen anzugehen. Hierbei konnten die Anforderungen, das Laufen, Drehen und in die Hocke gehen durch Bewegungen des Anwenders, mit Hilfe von den HTC VIVE Trackern umgesetzt werden. Ein komplett, zum Anwender synchronen, Bewegen des Roboters wurde wegen möglichen Unfällen und Stürzen des Roboters jedoch nicht umgesetzt.

Das Verwenden der Software unterliegt jedoch einigen Grenzen. So ist der Roboter nach einiger Zeit nicht mehr voll funktionstüchtig, da seine Motoren zu heiß werden. Dadurch ist es nicht mehr gegeben, dass der Roboter die Bewegungen nachahmt oder das nachspricht, was der Anwender sagt. Das ist das

größte Problem bei dieser Anwendung und diesem Projekt, da der Roboter auch nach einer kleinen Pause meistens immer noch zu heiße Motoren hat.

Dennoch lässt sich sagen, dass die Software, wenn der Roboter normal temperierte Motoren hat und wenn es zu keinen Latenzzeiten wegen der benötigten VM kommt, sehr viel Spaß macht.

7.2 Ausblick

Dieses Kapitel bietet einen kleinen Ausblick auf weiter mögliche Features oder Umsetzungsmöglichkeiten. Das Projekt bietet auch weiterhin die Möglichkeit, durch Folgeprojekte weiterbearbeitet zu werden.

7.2.1 Weitere Sprachfunktionalitäten

Die Sprache ist eine wichtige Möglichkeit, mit anderen Personen zu interagieren. Dadurch, dass ein kleines Sprachfeature, dem Nachsagen des vom Anwender Gesprochenen, in dieser Arbeit implementiert wurde, könnte man versuchen, das Mikrofon des Roboters auszuwerten und somit das, was der Roboter durch das Mikrofon hört, dem Anwender an die Kopfhörer der VIVE Brille zu übertragen. So könnte der Roboter vielleicht in einem anderen Raum sein und dort mit anderen Personen „sprechen“, in dem der Anwender des Programms auf das Gehörte reagiert und so den Roboter sprechen lässt.

Des Weiteren könnte man auch an eine Umsetzung von Sprachbefehlen denken. So könnte der Anwender einige Befehle nennen, die durch die Software dann ausgewertet und durch weitere Befehle an den Roboter geschickt werden. So

wäre durch Befehle wie „Lauf“ oder „Dreh dich nach links“ ebenfalls eine Bewegung des Roboters durch Personen, die im Rollstuhl sitzen, möglich.

7.2.2 Verwenden anderer Kameras

Eine weitere Möglichkeit, das Projekt fortzuführen wäre das Einbinden neuer Kameras. Bisher ist es durch die Verwendung von Kameras, derer Empfänger mit dem PC verbunden sind, so, dass die Lokation, an der der Roboter verwendet werden kann, relativ klein ist. Eine Möglichkeit der Erweiterung der Räumlichkeiten würden WLAN-Kameras bieten. So könnte der Anwender des Programms den Roboter durch ganze Gebäude steuern, indem er sich in einem Raum befindet. Durch Implementierung des vorher genannten Features, könnte er durch den Roboter mit anderen Personen in diesem Gebäude interagieren. Die Kameras sollten aber eine akzeptable Akkulaufzeit und Auflösung haben, sodass die Softwareerfahrung dadurch nicht schlechter wird.

7.2.3 Beschränkungen des Nao V4

Sollte dieses Projekt fortgeführt werden, sollte aber ein anderer, neuerer Nao-Roboter verwendet werden. Die hier verwendete V4 Generation hat nach der langen Zeit zu viele Probleme, durch die eine Ausführung der oben genannten Features ohne weiteres nicht möglich wäre. Da die Akkulaufzeit des Roboters zu schlecht ist, wäre eine Umsetzung der Funktion, dass sich der Roboter durch Gebäude bewegen kann, nicht gut. Des Weiteren werden die Motoren des Roboters zu heiß und das Arbeiten mit ihm machte dadurch keinen Spaß, da der Nao sich immer über irgendetwas beschwert hat und so seine Aufgaben nicht erfolgreich durchführte. Der Nao V6 bietet eine längere Akkulaufzeit und durch die besseren Prozessoren würde es vermutlich zu weniger Latenzzeiten kommen [3]. Durch die viel besseren Kameras könnten diese vielleicht als die primären Kameras des Projektes verwendet werden, sodass keine weiteren Kameras mehr benötigt werden würden.

Wird darüber nachgedacht einen aktuelleren Roboter zu verwenden, könnten die internen Kameras des Roboters verwendet werden, um so beispielsweise Gegenstände oder Personen zu erkennen und so personalisiert mit den Personen zu interagieren oder Gegenstände durch den Roboter klar zu benennen. Dies würde den Raum für Spielereien mit dem Roboter bieten, sodass weitere Folgeprojekte umgesetzt werden könnten.

Mit Hilfe der nachfolgenden Abbildung wird deutlich, inwiefern sich die Ausstattung der verschiedensten Nao Generationen über die Jahre geändert hat.

Robot Version	Nao V3+ (2008)	Nao V3.2 (2009)	Nao V3.3 (2010)	Nao Next Gen (V4) (2011) ^[30]	Nao Evolution (V5) (2014) ^[31]	Nao Power 6 (V6) (2018) ^[32]
Height	573.2 millimetres (22.57 in)		573 millimetres (22.6 in)		574 millimetres (22.6 in)	
Depth	290 millimetres (11 in)		311 millimetres (12.2 in)			
Width	273.3 millimetres (10.76 in)		275 millimetres (10.8 in)			
Weight	4.836 kilograms (10.66 lb)		4.996 kilograms (11.01 lb)	5.1825 kilograms (11.425 lb)	5.305 kilograms (11.70 lb)	5.48 kilograms (12.1 lb)
Power supply	lithium battery providing 27.6 Wh at 21.6V				lithium battery providing 48.6 Wh at 21.6V	lithium battery providing 62.5 Wh at 21.6V
Autonomy	60 minutes (active use)				90 minutes (active use)	
Degrees of freedom	25 ^[33]					
CPU	x86 AMD GEODE 500MHz			Intel Atom Z530 @ 1.6 GHz		Intel Atom E3845 Quad Core @ 1.91 GHz
RAM	256 MB			1 GB		4 GB DDR3
Storage	2 GB Flash memory			2 GB Flash memory + 8 GB Micro SDHC		32 GB SSD
Built-in OS	OpenNAO 1.6 (OpenEmbedded-based)	OpenNAO 1.8 (OpenEmbedded-based)	OpenNAO 1.10 (OpenEmbedded-based)	OpenNAO 1.12 (gentoo-based)	NAOqi 2.1 (gentoo-based)	NAOqi 2.8 (openembedded-based)
Compatible OS	Windows, Mac OS, Linux					
Programming languages	C++, Python, Java, MATLAB, Urbi, C, .Net					
Simulation environment	Webots					
Cameras	Two OV7670 58°DFOV cameras			Two MT9M114 72.6°DFOV cameras		Two HD OV5640 67.4°DFOV cameras

Abbildung 28: Ausschnitt der Ausstattung der verschiedenen Nao Modelle

Die V5 Generation hat nicht nur Gemeinsamkeiten mit dem V4 Robotern, sondern auch mit dem neuesten Modell. Allerdings ist es nicht empfehlenswert eine Version aus dem Jahre 2014 zu wählen. Zum neuesten Modell, die V6 Generation, gibt es die meiste Kompatibilität und das neuste Softwarepaket. Außerdem sprechen die bessere CPU und RAM für bessere Leistungen dieses Nao-Roboters und der Anwendung aus dieser Studienarbeit sowie zukünftige Projekte.

Literaturverzeichnis

- [1] J. Schneider und D. Jahnke, NaoVR - Nao Access Over Virtual Reality, Karlsruhe, 2020.
- [2] 2021 VMware, Inc., [Online]. Available: <https://www.vmware.com/de.html>. [Zugriff am 4 November 2020].
- [3] „Wikipedia,“ [Online]. Available: [https://en.wikipedia.org/wiki/Nao_\(robot\)](https://en.wikipedia.org/wiki/Nao_(robot)). [Zugriff am 25 April 2021].
- [4] „Choreographie User Documentation,“ Aldebaran, [Online]. Available: <http://doc.aldebaran.com/2-1/software/choregraphie/index.html>.
- [5] „eduporium,“ [Online]. Available: <https://www.eduporium.com/nao-v6-robot.html>.
- [6] „Oanda Währungsrechner,“ Oanda, [Online]. Available: <https://www1.oanda.com/lang/de/currency/converter/>.
- [7] „HTC VIVE Wiki,“ [Online]. Available: https://en.wikipedia.org/wiki/HTC_Vive.
- [8] „HTC VIVE Webseite,“ HTC, [Online]. Available: <https://www.vive.com>.
- [9] SoftBank Robotics Europe, [Online]. Available: http://doc.aldebaran.com/2-1/family/robots/postures_robot.html. [Zugriff am 20 April 2021].
- [10] SoftBank Robotics, „Community Aldebaran,“ [Online]. Available: <https://community.ald.softbankrobotics.com/>. [Zugriff am 20 Oktober 2020].

- [11] SoftBank Robotics, „Developer Guide,“ [Online]. Available:
<https://developer.softbankrobotics.com/>. [Zugriff am 20 Oktober 2020].
- [12] Open Robotics, [Online]. Available: <http://wiki.ros.org/>. [Zugriff am 20 Oktober 2020].
- [13] SoftBank Robotics, „Software,“ [Online]. Available:
<https://www.softbankrobotics.com/emea/en/support/nao-6/downloads-softwares>.
[Zugriff am 20 Oktober 2020].
- [14] Blender Foundation, [Online]. Available: <https://www.blender.org/>. [Zugriff am 7 November 2020].
- [15] „Unity Scripting Documentation,“ Unity, [Online]. Available:
<https://docs.unity3d.com/ScriptReference/>.
- [16] „SteamVR Api Documentation,“ Valve, [Online]. Available:
https://valvesoftware.github.io/steamvr_unity_plugin/.
- [17] „.NET API,“ Microsoft, [Online]. Available: <https://docs.microsoft.com/de-de/dotnet/api/>.