Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Kashif Haleem

Humanoide Roboter: NAO und Interaktion

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Kashif Haleem

Humanoide Roboter: NAO und Interaktion

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Olaf Zukunft
Zweitgutachter: Prof. Dr. Klaus-Peter Kossakowski

Eingereicht am: 2. Oktober 2018

Kashif Haleem

Thema der Arbeit
Humanoide Roboter: NAO und Interaktion

Stichworte
Humanoider Roboter, NAO, Kinect, Mensch-Maschinen-Interaktion, Maschine-Umgebung-Interaktion

Kurzzusammenfassung
Im Rahmen dieser Bachelorarbeit wurden zwei Anwendungen für den humanoiden Roboter NAO für eine Mensch-Maschinen-Interaktion und eine Maschinen-Umgebung-Interaktion entwickelt. Die Anwendungen dienen dem Einsatz von NAO im täglichen Leben des Menschen. Hierzu wurden die grundlegenden Techniken verschiedener Hardwarekomponenten und deren Entwicklungsmöglichkeiten diskutiert. Um die Szenarien erfolgreich zu nutzen, wurden einzelne Konzepte und Implementierungen vorgestellt. Im Abschluss wurden verschiedene Tests, um den Erfolg der Implementierung zu testen, durchgeführt.
Kashif Haleem

Title of the paper
Humanoid Robot: NAO and interaction

Keywords
Humanoid robot, NAO, Kinect, Human-machine interaction, Machine-environment interaction

Abstract
As a part of this bachelor thesis, two applications for the humanoid robot NAO for a human-machine interaction and a machine-environment interaction were developed. The applications serve the daily use of NAO in humans life. For this the basic techniques of different hardware components and their development possibilities were discussed. In order to use the scenarios successfully, individual concepts and implementations were presented. In conclusion, various tests were conducted to test the success of the implementions.

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

Ein Roboter ist ein vielseitiges und hoch entwickeltes mechanisches Gerät bzw. Maschinenwesen. In der heutigen Zeit kommen Roboter in verschiedenster Gestalt in unterschiedlichen Anwendungsgebieten zum Einsatz. Das Spektrum reicht von der Erforschung anderer Planeten, z.B. dem Mars, bis hin zur sozialen Kommunikation und kommt als Begleiter eines Menschen im alltäglichen Leben zum Einsatz. Im Jahr 2015 setzte die Bank of Tokyo-Mitsubishi UFJ als Experiment in ausgewählten Niederlassung den humanoiden Roboter ein, der den Kunden bei grundlegenden Anfragen, wie z.B. bei der Bedienung der Bankautomaten, hilft. [1].

Der Einsatz von Roboter über Fernsteuerung findet auch in Bereichen statt, in denen der menschliche Einsatz eine Bedrohung für ihr Leben darstellt, wie z.B. im Militäreinsatz, in Industriegebieten für Forschung von gefährliche Substanzen, als auch in der Automobilindustrie und Medizin als Medizinroboter.

Ein humanoider Roboter ist ein der menschlichen Gestalt nachempfundener Roboter und somit besteht er meist aus einem Kopf, zwei Armen, einem Torso und zwei Beinen, die manipulierbar sind und mit verschiedenen Sensoren und Kameras ausgestattet sind. Eines der grundlegenden Merkmale eines Roboters ist seine Fähigkeit, sich in seine Umwelt zu integrieren. Durch diese Integration ist es ihm möglich die menschlichen Bewegungen, wie z.B. dem Gehen, Armbewegungen und dem Greifen von Objekten nachzumachen. Die eingebauten Kameras und implementierten Algorithmen ermöglichen ihm verschiedene Objekte zu erkennen und zwischen ihnen unterscheiden zu können. [2]

> *„To achieve various tasks in the real world, humanoid robots have to generate whole-body motions in real-time, interacting with the environment and with command input by humans. Whether the control comes from an autonomous controller or from a human operator, establishing an effective whole-body operation method is of great importance.“* [3]

1.2. Ziel der Arbeit

Durch seine Eigenschaften, den Menschen nachzumachen, kann ein humanoider Roboter einen Menschen in verschiedene menschlichen Tätigkeiten ersetzten. Damit die Interaktion zwischen dem humanoiden Roboter und einem Menschen bzw. der Umgebung stattfinden kann, muss der Roboter den Menschen bzw. seine Umgebung erkennen und mit ihnen kommunizieren können. Mit der Kommunikation können wesentliche Informationen zwischen einem Menschen und dem Roboter für eine Mensch-Maschinen-Interaktion (MMI) und einer Maschinen-Umgebung-Interaktion (MUI) ausgetauscht und verarbeitet werden. Ziel dieser Arbeit ist es zwei verschiedene Szenarien für den humanoiden Roboter NAO, der französischen Firma Aldebaran, zu entwickeln. Die Szenarien, Imitation eines Menschen mit Hilfe der Microsoft Kinect Kamera, welche eine Mensch-Maschinen-Interaktion darstellt, und Mülltrennung, welche eine Maschinen-Umgebung-Interaktion darstellt, werden im Laufe der Arbeit näher betrachtet (siehe Analyse- 3 und Konzeptkapitel 4).

2. Grundlagen

Bevor die einzelnen Szenarien näher brachtet werden, werden in diesem Kapitel die Grundlagen vom NAO und der Microsoft Kinect Kamera erläutert.

2.1. Humanoider Roboter NAO

NAO ist ein humanoider Roboter der französischen Firma Aldebaran, der im Jahr 2006 zum ersten Mal vorgestellt wurde. Die Firma Aldebaran wurde in Jahr 2005 von Bruno Maisonnier gegründet, der bereits schon im Jahr 2004 mit der Entwicklung des Roboters NAO unter „Projekt NAO“ begonnen hat [4]. Mittlerweile beinhaltet die Produktfamilie NAO verschiedene Variationen (Stand Ende 2017). Neben den vier Hauptmodellen (NAO-H25, -H21, -T14 & -T2) verfügt die Familie NAO über verschiedene Versionen (V4, V3.3, V3.2 & V3+) welche sich anhand des Hinterkopfdesigns unterscheiden lassen (siehe Abbildung 2.1) [5].

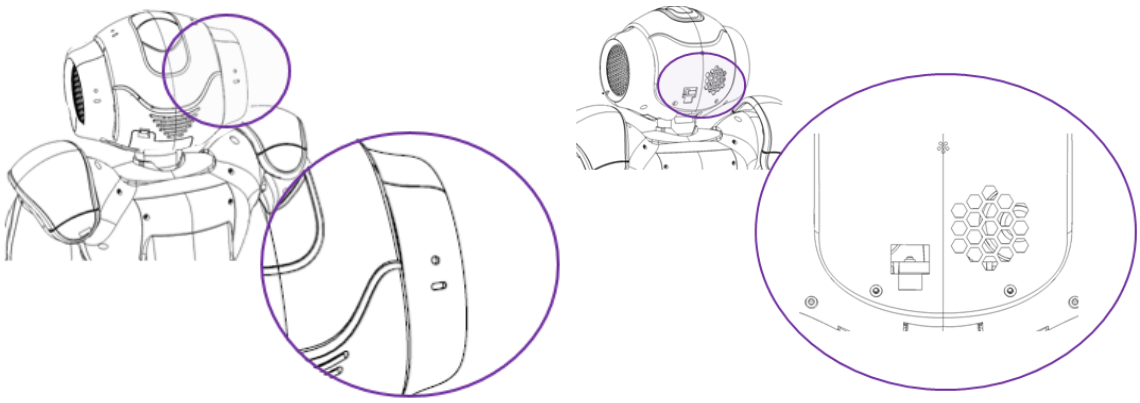


Abbildung 2.1.: Linke Abbildung V5 und Rechte Abbildung V3 [5]

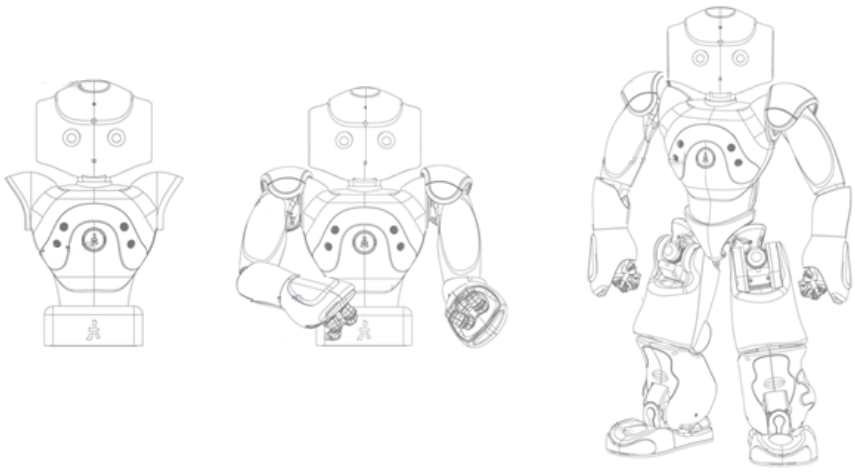


Abbildung 2.2.: Die T- & H-Modelle der Familie NAO: NAO-T2 (links), NAO-T14 (mittig) und NAO-H25 (rechts) [6]

Die T-Modelle der Familie NAO sind keine richtigen humanoiden Roboter im herkömmlichen Sinne. Sie bestehen aus einem Oberkörper und haben entweder einen Kopf und einen Torso (NAO-T2) oder ergänzend dazu noch Arme (NAO-T14). Somit sind die Einsatzgebiete der T-Modelle ortsgebunden. Mit der Entwicklung der H-Modelle der Familie NAO (H21 /H25) ist es Aldebaran gelungen einen nah am Menschen gestalteten kompakten humanoiden Roboter zu entwickeln. Diese besitzen ergänzend zu den T-Modellen auch zwei Beine, die entsprechend über bewegliche Gelenke verfügen und somit deren Einsatzgebiete frei Wählbar sind (siehe Abbildung 2.2).

Masse und Gewicht

Die Bereits erwähnten verschiedenen NAO-Modelle unterscheiden sich außer in der Bauform auch in den Maßen, dem Gewicht und der Anzahl der Sensoren. Der in dieser Bachelorarbeit zu Einsatz kommende NAO-H25 Version 5(siehe Abbildung 2.3 hat in aufrechter Position, vom Scheitel bis zur Sohle, eine Körpergröße von 57,4 cm und inklusive der Batterie ein Gesamtgewicht von 5,4 kg. [7]

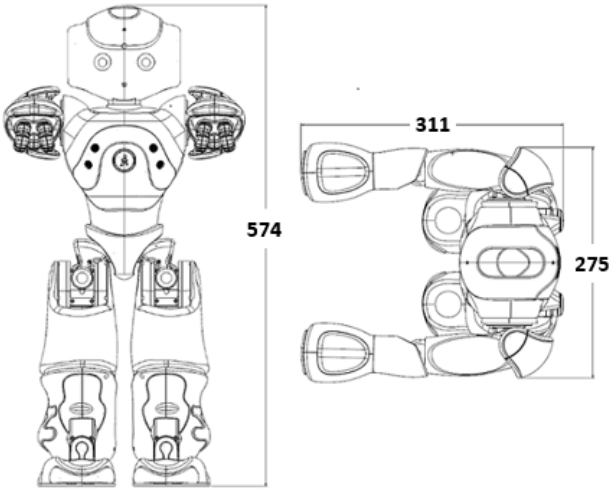


Abbildung 2.3.: Konstruktion des Roboters NAO H25 V5 [7]

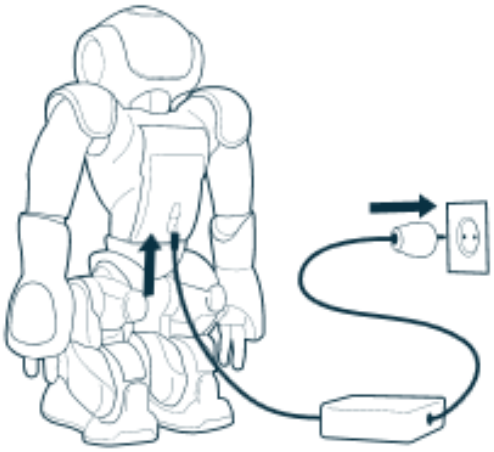


Abbildung 2.4.: Aufladung des NAOs [8]

Stromversorgung

Der NAO kann sowohl per aufgeladener Batterie, als auch direkt über die Stromversorgung per mitgeliefertem Netzteil (Adapter) benutzt werden (siehe Abbildung 2.4). Aber die Batterie wird nur aufgeladen, wenn der NAO ausgeschaltet ist und mit seinem Netzteil (Adapter) verbunden ist. Mit aufgeladener Batterie kann NAO zwischen 60 Minuten im Vollbetrieb bzw. 90 Minuten im Normalbetrieb verwendet werden. [9]

Sensorik

Die eingebaute Vielzahl von verschiedenen Sensoren ermöglichen es NAO-H25 V5 seine eigene Umwelt, sowie seinen eigenen Zustand zu erfassen und dementsprechend zu reagieren. Im Kopf der NAO-H25 sind zwei baugleiche Kameras mit einer maximalen Auflösung von 1280 x 960 Pixels bei 30 Bildern pro Sekunde, wovon sich eine in der Stirn und eine Weitere im Mund befindet [10]. Über diese sind beispielsweise eine Objekt-, Gesichtserkennung oder Fernüberwachung der Umgebung möglich. In seinen Ohren sind zwei Lautsprecher für die Audioausgaben und in seinem Kopf vier Mikrofone, zwei vorne und zwei hinten [11], über die die Sprachinteraktionen, die Lokalisierung der Geräuschquellen und die Verarbeitung von Audiodaten durchgeführt werden können, eingebaut. Durch die eingebauten LEDs kann der Anwender bzw. Entwickler Benachrichtigungen,

wie z.B. Boot-Prozess und Batteriezustand, vom NAO-System oder von einer Anwendung erhalten. [12]

Teil	Position	Anzahl	Beschreibung
A	Kopf Taktiler Sensor	12	16 step blue LEDs
B	Ohren	2x10	16 step blue LEDs
C	Augen	2x8	Full-Color RGB LEDs
D	Brust Taste	1	Full-Color RGB LEDs
E	Füße	2x1	Full-Color RGB LEDs

Tabelle 2.1.: Position und Anzahl der LEDs

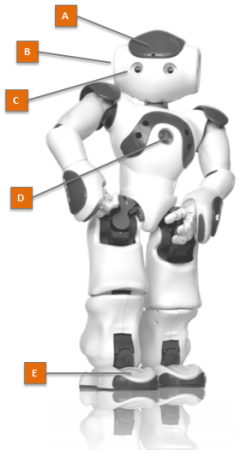


Abbildung 2.5.: NAO-H25 LEDs [13]

Weiterhin verfügt NAO über zwei Ultraschall-Sensoren, einem Beschleunigungssensor, neun taktile Sensoren, sowie acht Berührungssensoren. In seinem Torso ist ein Sonarsubsystem aus Ultraschall-Sensoren, 2 Transmitter und 2 Receiver, eingebaut (siehe Abbildung 2.6), welche sowohl zur Erkennung von Hindernissen als auch für die Entfernungsmessung benutzt werden. Das Sonarsubsystem besteht jeweils aus einem Transmitter und einem Receiver, welche sich auf der linke und auf der rechte Seite des Torsos befinden. Laut dem Datenblatt liegt der Erfassungsbereich der Daten zwischen 0,2 m und 0,8 m. Bei einer Distanz unter 0,2 m gibt es keine Distanzinformationen, sondern lediglich die Information, dass ein Objekt vorhanden ist. Bei einer Distanz von über 0,8 m ist der zurückgegebene Wert eine Schätzung. Anhand eines Trägheitssensors im Torso und vier Drucksensoren jeweils in den Füßen, kann NAO sich balancieren bzw. seinen eigenen Zustand, wie z.B. seine aktuelle Haltung, überwachen. Mehrere Gyrometer, Beschleunigungs- und Drucksensoren ermöglichen es ihm auf unerwartete Bewegungen von außen zu reagieren und somit sein Laufverhalten stabil zu halten bzw. um ein Umfallen zu verhindern [14].

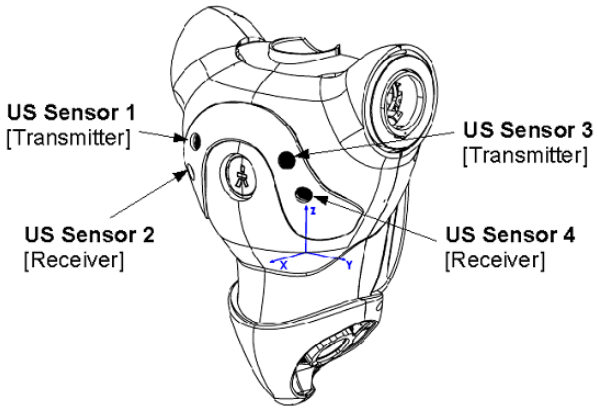


Abbildung 2.6.: NAO Sonarsubsystem [15]

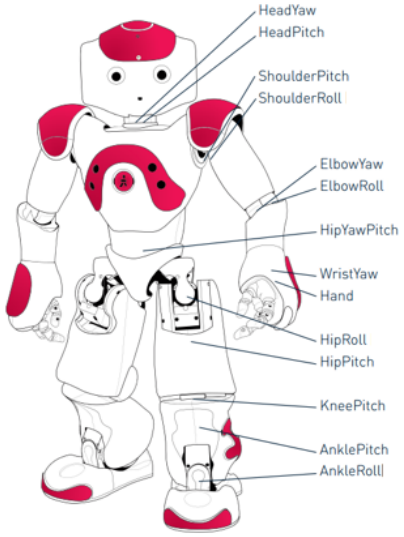


Abbildung 2.7.: NAO H25 Motoren [16]

Freiheitsgrad

Ein wesentlicher Aspekt, den ein humanoider Roboter so besonders macht, ist dessen Fähigkeit die menschlichen Bewegungen nachzumachen. Trotz der menschenähnlichen Gestalt, besitzt ein humanoider Roboter andere Freiheitsgrade als ein Mensch, wodurch es ihm möglich ist manche Bewegungen durchzuführen, die dem Mensch nicht möglich sind, und andersrum. Die Abbildung 2.7 stellt die Freiheitsgrade und die Motorik des Roboters NAO dar. NAO verfügt insgesamt über 25 Freiheitsgrad, dessen Verhalten über 26 Miniaturmotoren gesteuert wird [17]. Aus der Abbildung 2.7 lassen sich folgende Freiheitsgrade 2.2 ablesen.

Gelenk	Anzahl Freiheitsgrad	Beschreibung
Kopf	2	HeadYaw & -Pitch (Neigung und Drehung)
Arme	2 x 5	ShoulderPitch, -Roll, ElbowYaw, -Roll & WristYaw (Bewegung & Drehung)
Becken	1	HipYawPitch (Vor- & Rückbewegung)
Beine	2 x 5	HipRoll, -Pitch, KneePitch, AnklePitch & -Roll (Bewegung & Drehung)
Hände	2 x 1	Hand (Öffnen & Schließen)

Tabelle 2.2.: Freiheitsgrad der NAO

Jedes Gelenk hat einen begrenzten Bewegungswinkel und kann unabhängig voneinander kontrolliert angesteuert werden. Die Abbildung 2.8 stellt die Motorik des NAO H25 V5 detailliert dar. Aus der Tabelle 2.3 können die Drehaschen für jedes Gelenk abgelesen werden [18].

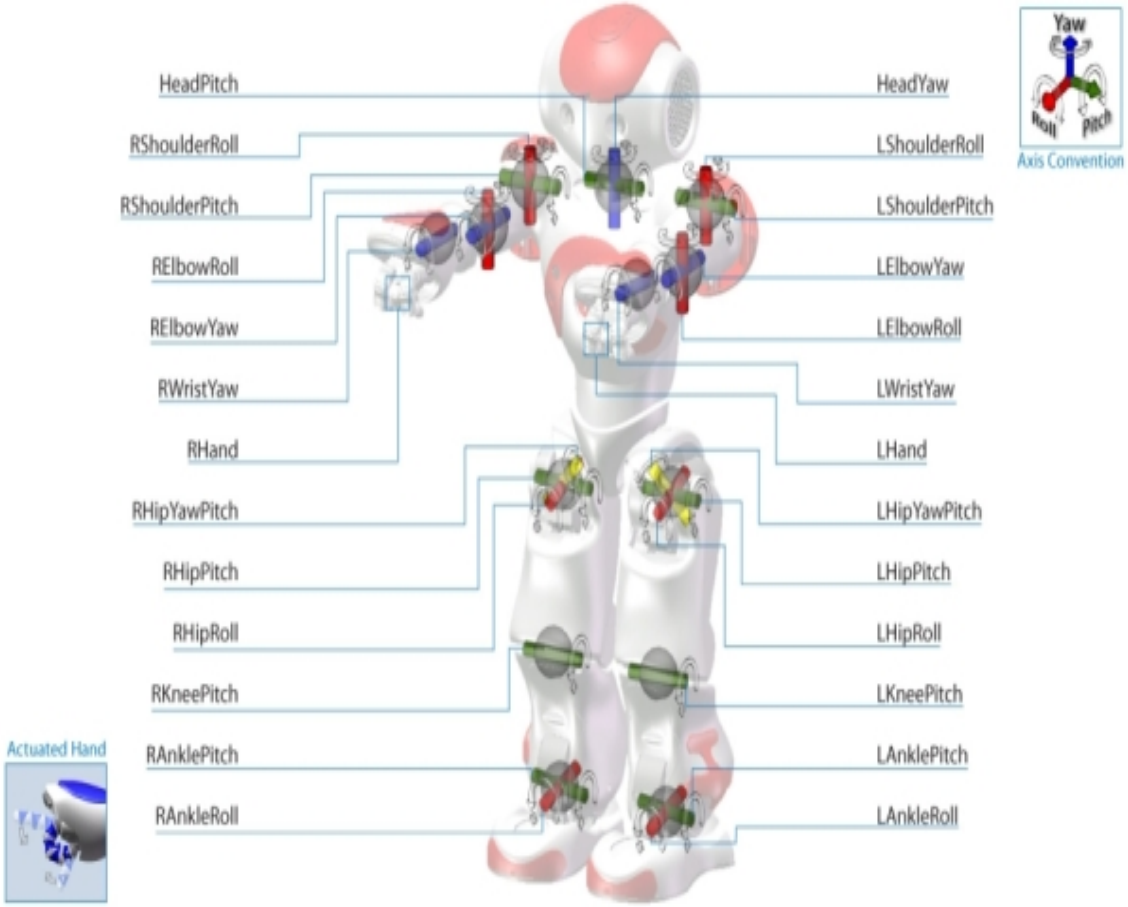


Abbildung 2.8.: Motorik des NAO H25 V5 [19]

Joints	Motion	Range (degrees)	Range (radians)
HeadYaw	Head joint twist (Z)	-119.5 to 119.5	-2.0857 to 2.0857
HeadPitch	Head joint front and back (Y)	-38.5 to 29.5	-0.6720 to 0.5149
LShoulderPitch	Left shoulder joint front and back (Y)	-119.5 to 119.5	-2.0857 to 2.0857
LShoulderRoll	Left shoulder joint right and left (Z)	-18 to 76	-0.3142 to 1.3265
LElbowYaw	Left shoulder joint twist (X)	-119.5 to 119.5	-2.0857 to 2.0857
LElbowRoll	Left elbow joint (Z)	-88.5 to -2	-1.5446 to -0.0349
LWristYaw	Left wrist joint (X)	-104.5 to 104.5	-1.8238 to 1.8238
LHand	Left hand	Open & Close	Open & Close
RShoulderPitch	Right shoulder joint front and back (Y)	-119.5 to 119.5	-2.0857 to 2.0857
RShoulderRoll	Right shoulder joint right and left (Z)	-76 to 18	-1.3265 to 0.3142
RElbowYaw	Right shoulder joint twist (X)	-119.5 to 119.5	-2.0857 to 2.0857
RElbowRoll	Right elbow joint (Z)	2 to 88.5	0.0349 to 1.5446
RWristYaw	Right wrist joint (X)	-104.5 to 104.5	-1.8238 to 1.8238
RHand	Right hand	Open & Close	Open & Close
LHipYawPitch	Left hip joint twist (Y-Z 45°)	-65.62 to 42.44	-1.145303 to 0.740810
RHipYawPitch	Right hip joint twist (Y-Z 45°)	-65.62 to 42.44	-1.145303 to 0.740810
LHipRoll	Left hip joint right and left (X)	-21.74 to 45.29	-0.379472 to 0.790477
LHipPitch	Left hip joint front and back (Y)	-88.00 to 27.73	-1.535889 to 0.484090
LKneePitch	Left knee joint (Y)	-5.29 to 121.04	-0.092346 to 2.112528
LAnklePitch	Left ankle joint front and back (Y)	-68.15 to 52.86	-1.189516 to 0.922747
LAnkleRoll	Left ankle joint right and left (X)	-22.79 to 44.06	-0.397880 to 0.769001
RHipRoll	Right hip joint right and left (X)	-45.29 to 21.74	-0.790477 to 0.379472
RHipPitch	Right hip joint front and back (Y)	-88.00 to 27.73	-1.535889 to 0.484090
RKneePitch	Right knee joint (Y)	-5.90 to 121.47	-0.103083 to 2.120198
RAnklePitch	Right ankle joint front and back (Y)	-67.97 to 53.40	-1.186448 to 0.932056
RAnkleRoll	Right ankle joint right and left (X)	-44.06 to 22.80	-0.768992 to 0.397935

Tabelle 2.3.: Drehaschen des NAO H25 V5 [18]

System

NAO H25 V5 hat einen Intel Atom Z530 Hauptprozessor, welcher mit einer Taktfrequenz von 1,6 GHz getaktet und auf dem Mainboard im Kopf des Roboters integriert ist. Des Weiteren verfügt er über 1 GB Arbeitsspeicher und 2 GB Flash Speicher. Für die NAOqi Software und selbst entwickelter Module steht zusätzlich 8 GB Micro SDHC zur Verfügung [17]. Der Zugriff auf das Computersystem findet entweder über RJ45 Ethernet-Port oder über WLAN statt. Über den USB-Port, welcher im Hinterkopf des Roboters integriert ist, können externe Geräte, wie z.B. die Microsoft Kinect, mit NAO verbunden werden [20]. Eine sekundäre CPU für die Kommunikation und Steuerung der Motoren befindet sich im Torso des Roboters. Die Abbildung 2.9 zeigt die Computerarchitektur NAOs.

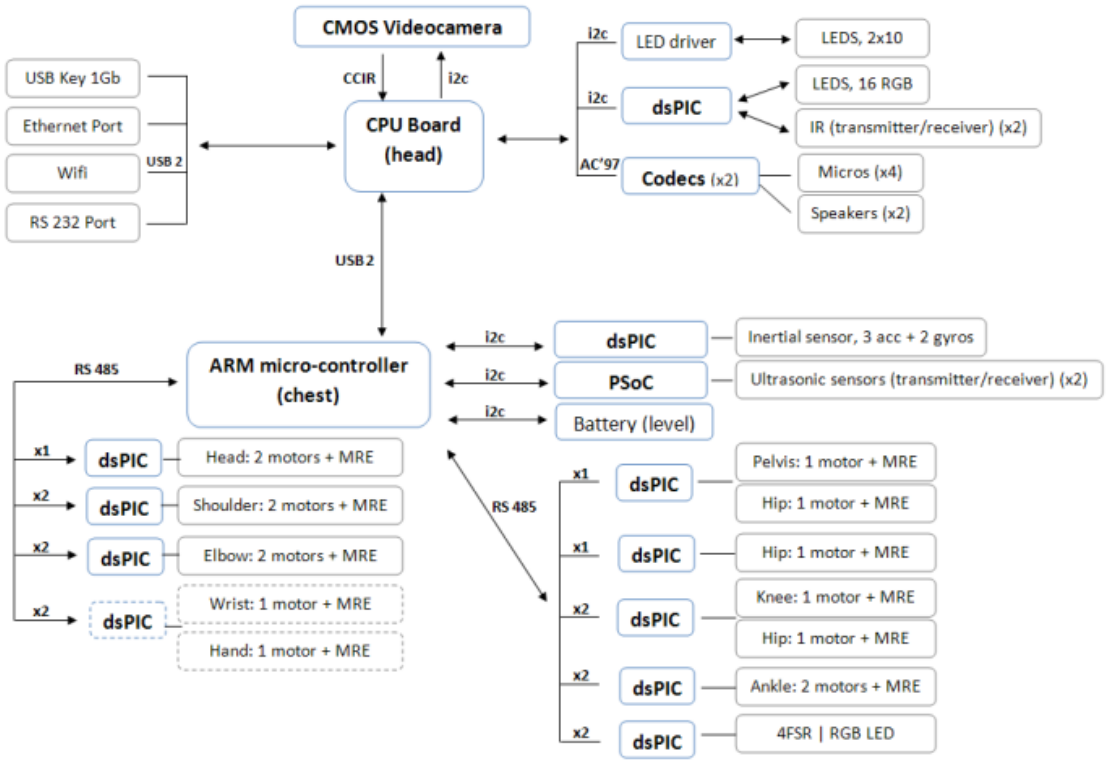


Abbildung 2.9.: Computerarchitektur des Roboters NAO [21]

2.2. Entwiklungsoftware des NAOs

2.2.1. NAOqi & NAOqi Framework

Das embedded Betriebssystem NAOqi, welches von Aldebaren speziell für deren Roboter entwickelt wurde, basiert auf dem Linux Betriebssystem und dient als Middleware, über die eine Interaktion mit der Hardware des Roboters ermöglicht wird. Das NAOqi Framework bietet die Möglichkeit den Roboter mit diversen Programmiersprachen zu programmieren. Es enthält Elemente wie Parallelverarbeitung, Ressourcenverwaltung, Synchronisierung und Verarbeitung von Ereignissen. Auch der Informationsaustausch, die Programmierung über ALMemory, NAOs Speicher, und die Kommunikation zwischen homogenen Modulen, wie Bewegung (Motion), Audio und Video werden ermöglicht. Die umfangreichen Modulbibliotheken erlauben dem Entwickler die Hardware auf verschiedene Art und Weise anzusprechen und gegebenenfalls zu erweitern. Das Framework ist eine Cross-Plattform und kann unabhängig von Betriebssystemen (wie z.B. Windows, Linux und MacOS) und mit verschiedenen Programmiersprachen, wie z.B. Python, C++, Java und JavaScript, benutzt werden [22]. Die Entwickler können die Entwicklungsfortschritte an einem simulierten Roboter testen, welche von dem NAOqi Framework unterstützt wird [23]. Dies bietet den Vorteil die Applikationen in einem virtuellen Robotersimulator zu testen, bevor sie auf dem realen Roboter aufgespielt werden.

NAO wird von einer ausführbaren NAOqi-Datei gesteuert. Der NAOqi-Prozess startet automatisch nach dem das NAOqi-Betriebssystem gestartet wird. NAOqi arbeitet als ein Broker-Objekt und bietet einen Suchtdienst, das für das Auffinden von angeforderten Modulen und deren Methoden zuständig ist. Es lädt die Datei autoload.ini, die festlegt, welche Bibliotheken geladen werden. Jede Bibliothek enthält ein oder mehrere Module, deren Methoden durch den Broker für andere Module bereitgestellt werden. Die Abb. 2.10 zeigt den Zusammenhang zwischen Broker, Bibliotheken und Module.
Als weitere Aufgabe erlaubt der Broker einen Netzwerkzugriff, so dass die Methoden der bekannten Module von außerhalb des Prozesses per Remote aufgerufen werden können. Die Abbildung 2.11 verdeutlicht den NAOqi-Prozess in Zusammenhang zwischen Broker, Module & Methoden.

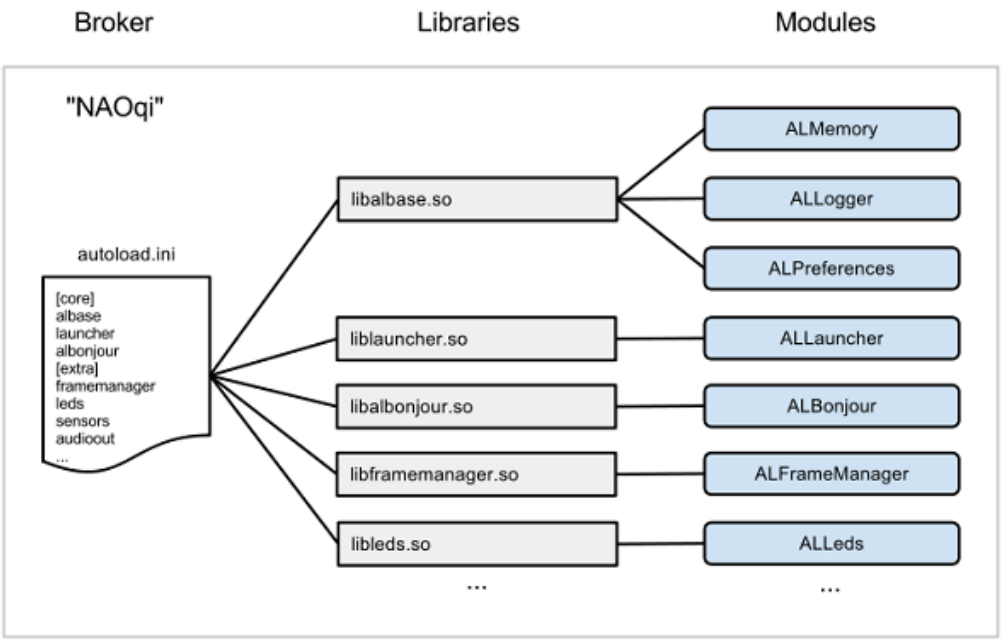


Abbildung 2.10.: Der Zusammenhang zwischen Broker (NAOqi: autoload.ini), Bibliotheken und Module [24]

Jedes Modul verfügt über eine oder mehrere Methoden, um die Funktionen des Roboters aufzurufen und kann remote oder lokal ausgeführt werden. Die Kommunikation mit NAO kann sowohl über Wireless LAN als auch über Ethernetkabel stattfinden. Die lokalen Module laufen im gleichen Prozess und können gegenseitig die Methoden mittels eines Brokers aufrufen. Dies bietet den Vorteil für eine schnelle Kommunikation zwischen Modulen. Die Remote-Module können unabhängig von dem Roboter auf einem PC laufen und brauchen einen zweiten Broker, um mit anderen Modulen zu kommunizieren. Die Modulbibliothek kann mit jeder beliebigen IDE debuggt und genau wie lokale Bibliotheken mit der ausführbaren NAOqi-Datei des Roboters verbunden werden. Da die Kommunikation mit dem Roboter über ein Netzwerk stattfindet, ist eine hohe Geschwindigkeit bei der Kommunikation mit andere Modulen über Remote-Aufrufe nicht gewährleistet [22]. NAOqi besitzt eine Liste von Kern-Modulen (engl. Core Modules) und eine öffentliche API, die immer zur Verfügung stehen. Jedes Modul stellt wie bereits erwähnt mehrere Methoden für die Funktionalität des Roboters zum Abruf breit. Die Funktionalität der Module ist in den folgenden Gruppen unterteilt [26]:

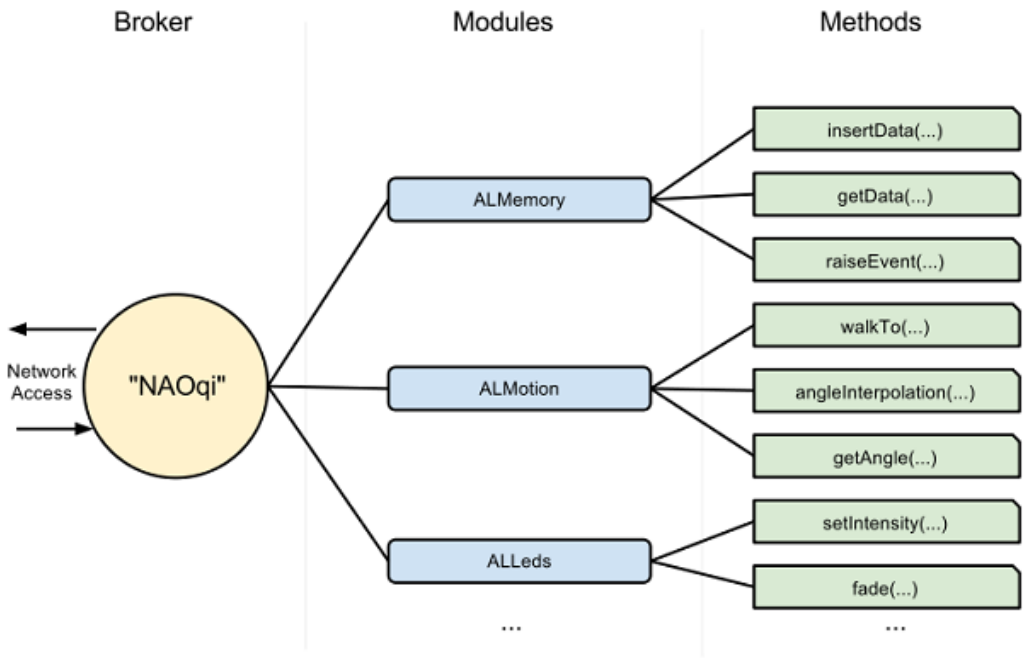


Abbildung 2.11.: Der NAOqi Prozess [25]

- NAOqi Core:

Verfügt über Module, die z.B. für die Speicher-, Ressourcenverwaltung, Kommunikation und Verhalten des Roboters zuständig sind.

- NAOqi Motion:

Module, die die Bewegungsabläufe des Roboters realisieren.

- NAOqi Audio:

Module, die Audio-Softwarekomponenten des Roboters, wie die Sprache des Roboters, Aufnahme und Wiedergabe von Audiodateien und Spracherkennung verwalten.

- NAOqi Vision:

Module, die z.B. für die Video- und Bildverwaltung, sowie der Objekt- und Bilderkennung zuständig sind.

- NAOqi PeoplePerception:

Module um eine Mensch-Roboter-Interaktion zu analysieren.

- **NAOqi Sensors:**
  Module um Sensoren auszulesen.

- **NAOqi Trackers:**
  Modul um die Position eines Zielobjektes, wie z.B. einen Ball oder ein Gesicht, während einer Bewegung zentriert in der Kamera zu halten.

- **ALDiagnosis:**
  führt eine passive Hardwareanalyse durch, ohne dabei das Verhalten des Roboters zu stören.

- **DCM:**
  ist ein Modul, welches ein Softwarebestandteil von NAOqi ist, und bis auf Audioeingaben, -ausgaben und der Kamera für die Kommunikation mit fast jedem elektronischen Gerät zuständig ist.

Die Tabelle 2.4 stellt eine Übersicht der Module dar. Im Zuge der Konzeptrealisierung 4 werden die Module näher betrachtet.

Core
ALAutonomousLife
ALBehaviorManager
ALConnectionManager
ALExtractor
ALMemory
ALModule
ALNotificationManager
ALPreferenceManager
ALResourceManager
ALStore
ALSystem
ALVisionExtractors
ALTabletService
ALUserSession
ALWorldRepresentation
PackageManager

Motion
ALAutonomousMoves
ALMotion
ALNavigation
ALRecharge
ALRobotPosture

Diagnosis
ALDiagnosis

Sensors
ALBattery
ALBodyTemperature
ALChestButton
ALFsr
ALInfrared
ALLaser
ALLeds
ALSensors
ALSonar
ALTouch

Audio
ALAnimatedSpeech
ALAudioDevice
ALAudioPlayer
ALAudioRecorder
ALDialog
ALSoundDetection
ALSoundLocalization
ALSpeechRecognition
ALTextToSpeech
ALVoiceEmotionAnalysis

Trackers
ALTracker

DCM
DCM

Vision
ALBacklightingDetection
ALBarcodeReader
ALCloseObjectDetection
ALColorBlobDetection
ALDarknessDetection
ALLandmarkDetection
ALLocalization
ALMovementDetection
ALPhotoCapture
ALRedBallDetection
ALSegmentation3D
ALVideoDevice
ALVideoRecorder
ALVisionRecognition
ALVisualCompass
ALVisualSpaceHistory

People Perception
ALBasicAwareness
ALEngagementZones
ALFaceCharacteristics
ALFaceDetection
ALGazeAnalysis
ALPeoplePerception
ALSittingPeopleDetection
ALWavingDetection

Tabelle 2.4.: Modulübersicht der NAOqi API [26]

2.2.2. Programmierung

NAO kann mithilfe der von Aldebaran mitgelieferten Software Choregraphe oder SDKs für die verschiedenen Programmiersprachen programmiert werden. Choregraphe ist eine visuelle Entwicklungsumgebung, die eine graphische Programmierung in einer spezifischen Modellierungssprache speziell für NAO erlaubt. Die einzelnen Module des NAOqi-Frameworks werden durch Boxen dargestellt und können über Drag'n'drop und Graphen miteinander verbunden werden. Sie lässt komplexe Verhalten für NAO entwickeln, mit der Möglichkeit die Module mit eigenem Python-Code erweitern zu können. Die Entwicklung kann in einer Simulation bzw. direkt am realen Roboter getestet werden [27]. Die Entwickler können selbst entwickelte Module in das NAOqi-Framework integrieren, in dem sie den Quellcode mit einem Cross-Compiler für NAOqi OS kompilieren. Aktuell ist die NAOqi API in 6 Sprachen verfügbar (siehe Tabelle 2.5), wobei die Module unter unterschiedlichen Entwicklungsumgebungen nur in C++ oder Python entwickeln werden können. Mit dem vollem Zugriff auf die gesamte Funktionalität der API werden nur diese beiden Sprachen unterstützt. Remote vom Computer aus ist dies aber auch mit weiteren Sprachen möglich [28].

Sprache auf dem Roboter	Andere unterstützten Sprachen
C++	.Net
Python	Java
	Matlab
	Urbi

Tabelle 2.5.: Von NAO unterstütze Sprachen

2.2.3. ROS - Robot Operating System Framework

ROS (engl. Robot Operating System) ist ein Open Source Betriebssystem für Roboter, was als Middleware für die Programmierung von Roboter dient. So kann man es als Software-Framework in verschiedenen Roboter-Implementierung einsetzen. Die ursprüngliche Entwicklung des Projektes begann Mitte der 2000er Jahre an der Stanford Universität im Rahmen des Stanford-AI-Robot (STAIR) und Personal Robots (PR) Programm und wurde ab 2007 von der Robotik Institut Willow Garage weiterentwickelt. Seit 2013 wird es von der

Open Source Robotics Foundation (OSRF) gepflegt und ist frei für kommerzielle und Forschungszwecke. Es bietet Hardware-Abstraktion, Gerätetreiber, Bibliotheken, Visualizer, Nachrichtenaustausch (engl. Message-Passing) und Paketverwaltung als Kernkomponenten. Ergänzend dazu bietet es Roboterspezifische Funktionen, wie z.B. Lokalisierung, Mapping, Navigation und Ermittlung der Pose. Mit der Vielzahl von Bibliotheken, die hauptsächlich auf den Programmiersprachen C ++, Python und Lisp basieren. Entwicklungstools wie RIVZ (3D-Visualisierungstool) und RTQ (GUI-Entwicklung Framework für ROS) bieten den Entwicklern die Möglichkeit ein komplexes und robustes Verhalten für Robotern zu entwickeln.
ROS unterstützt die Funktionalität einer großen Anzahl von Robotern. Darunter sind die bekanntesten: NAO, Romeo und Pepper von Aldebaran, TurtleBot von Willow Garage und IRobot Roomba. Die Open-Source-Implementierung von Funktionalitäten und Algorithmen sind in Packages organisiert. Eine vollständige Liste findet man auf der Homepage des ROS [29].

ROS NAOqi Package

Der ROS-Treiber für NAO wurde von Armin Hornung am Freiburger Humanoid Robots Lab im Jahr 2010 entwickelt. Im Wesentlichen ist er ein Wrapper, welcher die NAOqi API umhüllt. Der Treiber beinhaltet zahlreiche Packages, die Basiskonfiguration, Steuerung der Hardwarekomponenten, wie Sensoren und Kamera, und Interkationen, wie Spracherkennung und Teleoperation, von NAO unterstützen. Eines dieser ROS-Packages ist „nao_teleop“, welches bereits über für ROS vorkonfigurierte Joysticks und Gamepads verfügt und die Möglichkeit bietet NAO fernzusteuern. Mit Hilfe solcher Packages ist es möglich den NAO für verschiedene Funktionen bzw. Interaktionen zu programmieren, ohne darüber nachzudenken, welche Logik und welcher Algorithmus dafür verwendet wird [30].

ROS Kinect Package

ROS stellt mehrere Treiber für Kinect zur Verfügung. Einer davon ist der Wrapper „openni_kinect“, welcher die OpenNI Treiber, sowie die Bibliotheken, wie Skeleton und Gestenerkennung, umhüllt. Der Treiber wird von den Betriebssystemen Linux, OS X und Windows unterstützt und bietet vollen Zugriff auf die Kinect-Tiefendaten [31].

2.3. Microsoft Kinect

Seit Anfang November 2010 steht der Kinect-Sensor zum Verkauf auf dem freien Markt. Der Kinect-Sensor wurde von Microsoft zusammen mit der Firma PrimeSense in erster Linie für die Spielkonsole Xbox 360 entwickelt. Damit die Gamer die Spiele durch Körperbewegungen, Gesten und Sprachbefehle, ohne die Benutzung von herkömmlichen Kontoroller wie Gamepads und Joysticks, bedienen können. Schon in wenigen Tagen nach der kommerzielle Marktveröffentlichung wurde ein erster Open-Source-Treiber für das Linux Betriebssystem veröffentlicht [32]. Dies ermöglichte die Daten der Tiefenkamera der Kinect in Echtzeit auszuwerten und z.B. Personen vor der Kamera zu erkennen. Im Juni 2011 stellte Microsoft die erste Kinect SDK in einer Beta-Version unter einer nicht kommerzielle Lizenz für die Öffentlichkeit zur Verfügung. Somit haben die Entwickler Zugriff auf dieselben Tools, die Microsoft für die Entwicklung der Kinect-Anwendung verwendete [33]. Das eingebaute Softwarepaket von Microsoft unterstützt die Erkennung von Menschen, sowie deren Bewegungen und Gesten. Durch diese Eigenschaften kann der Einsatz des Kinect-Sensors eine große Rolle in der Robotik spielen.

2.3.1. Hardware

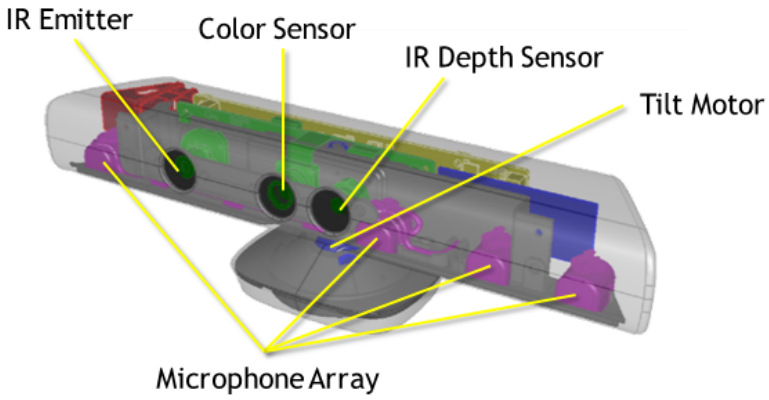


Abbildung 2.12.: Kinect für Windows V1 [34]

Die Abbildung 2.12 zeigt die Basishardware des Kinect-Sensors V1, der in dieser Bachelorarbeit zu Einsatz kommt und aus folgenden Komponenten besteht [34]

- Eine RGB-Farbkamera (engl. Color Sensor):

Die RGB-Kamera mit einer maximalen Auflösung von 1280 x 960 Pixel ermöglicht die Erfassung eines Farbbildes.

- Ein Tiefenmesser (engl. IR Emitter & IR Depth Sensor)

Besteht aus einem Infrarotstrahler (IR Emitter) auf der rechte Seite und einem Tiefensensor (IR Depth Sensor) auf der linke Seite. Der Infrarotstrahler wirft ein Infrarotstrahl in dem Raum bzw. in die Umgebung und der Tiefensensor erfasst den reflektierten Rückstrahl. Die reflektierten Strahlen werden in Tiefendaten umgewandelt, die den Abstand zwischen einem Objekt und dem Sensor messen. Dadurch wird eine tiefe Aufnahme ermöglicht.

- Mehrere Mikrofone (engl. Microphone Array)

Ein Multi-Array-Mikrofon besteht aus vier Mikrofonen, mit der die Erfassung von Audioton (Spracheingabe) ermöglicht wird. Die vier Mikrofone ermöglichen durch die Rausch- und Echounterdrückung den Vorteil eines besseres Audiotones, sowie die Lokalisierung einer Audioquelle.

- Eine motorisierte Basis (engl. Tilt Motor)

Bei Bedarf bewegt bzw. neigt sich der Kinect-Sensor nach oben oder unten, wodurch sich die aktuelle Ausrichtung des Sensors auslesen lässt. [34]

2.3.2. Kinect für Windows SDK

Die verschiedenen Versionen der Kinect-Hardware werden von verschiedenen SDK-Versionen unterstützt. So werden die Kinect für Windows V1 und die Kinect für die Xbox360 von der SDK-Version 1.8 und älteren Versionen unterstützt. Die aktuelle Version von Kinect for Windows SDK 2.0 unterstützt diese zwei Hardware nicht. Die Kinect für Windows V2 und Kinect für Xbox One werden von der SDK Version 2 unterstützt, wobei die Xbox-Versionen einen zusätzlichen Adapter für die Verbindung mit einem PC benötigen.
Das Framework "Kinect for Windows SDK" von Microsoft bietet Entwicklern die Möglichkeit, Anwendungen, die Gesten- und Spracherkennung unterstützen, mit der Kinect als Eingabegerät zu entwickeln. Des Weiteren bietet Microsoft ein Kinect for Windows Developer Toolkit an, welches den Zugriff auf Codebeispiele, Dokumentationen, Tools

und Komponenten erlaubt. Das Framework von Kinect liefert Farbbilder, Tiefenbilder, Audioeingaben und Skelettdaten, mit denen die Entwickler unter anderem Anwendungen mit folgendem Eigenschaften entwickeln können: [35]

- Echtzeit-Videoaufnahme mit dem Farbsensor
- Erkennung von Personen und deren Bewegungsabläufen mit der Hilfe von Skelettdaten
- Bestimmung der Entfernung zwischen einem Objekt und der Kinect-Kamera mit Hilfe von Tiefendaten.
- Sprachgesteuerte Anwendung

2.3.3. Python SDKs

Pykinect

Um ein Kinect-Event in Python tracken zu können, hat Microsoft die PyKinect-Bibliothek entwickelt, mit dem ein Benutzer alle Events, einschließlich visueller und Audiostimulation der Kinect tracken könnte. Das PyKinect-Package unterstützt den Zugriff auf den Kinect-Sensor, und beinhaltet sowohl das NUI-Subpackage als auch das Audio-Subpackage. Das NUI-Package bietet die Möglichkeit der Interaktion mit der Kinect-Kamera, inklusive des Skeleton-Trackings, der Videokamera als auch der Tiefenkamera. Das Audio-Subpackage hingegen unterstützt den Zugriff auf die Kinectmikrofone. [36]

VPython

VPython ist die Programmiersprache Python inklusive eines 3D-Computergrafikmoduls namens Visual. Mit VPython können Objekte wie Kugeln und Kegel in einem 3D-Raum erstellt und in einem Fenster anzeigt werden. Zusammen mit den Berechnungen werden Animationen in Echtzeit erstellt. Dies ermöglicht die Erstellung einer vereinfachten Visualisierung, die es dem Programmierer erlaubt, sich mehr auf die Berechnungsaspekte seines Programms zu konzentrieren. Die Einfachheit von VPython hat es zu einem nützlichen Werkzeug für die Darstellung einfacher Objektstrukturen gemacht. [37]

2.3.4. Skeleton Tracking

Über das sogenannte "Skeleton Tracking" ermöglicht Kinect die Erfassung von bis zu 6 Personen gleichzeitig. In der Kinect for Windows SDK 1.8 können von diesen die Bewegungen bei 2 Personen in Echtzeit erfasst werden. Dazu bildet Kinect eine Person mit einem Skelett von 20 Gelenken ab (siehe Abbildung 2.13).

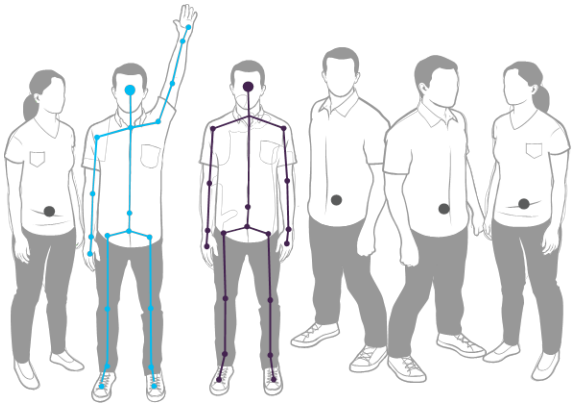


Abbildung 2.13.: Kinect Personen -erkennung [38]

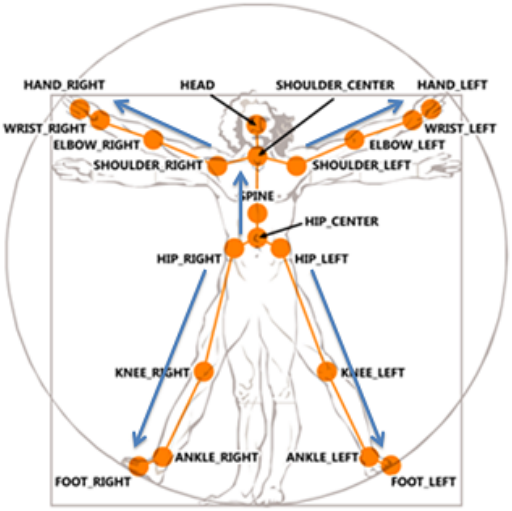


Abbildung 2.14.: Kinect Positions- und Bewegungsermittelung über erkannte Gelenkpunkte [38]

Um dies zu realisieren werden von den Sensoren Tiefendaten, in dem jeder Pixel Informationen des Abstandes zum Objekt liefert, zurückgegeben. Darüber hinaus spielen das maschinelle Lernen und der Farbwahrnehmungsalgorithmus eine wichtige Rolle zur Projizierung des Skeletts. Hiermit können unter anderem die Bewegungen von Schultern, Ellbogen, Handgelenken und Hände einer Person genau erfasst werden (siehe Abbildung 2.14). In der aktuellen Version von Kinect for Windows SDK 2.0 wurde dieses Verfahren erweitert. Somit bildet Kinect einen Skelett von 25 Gelenken je für bis zu 6 Personen ab. Mit diesen Informationen können Anwendungen zur Steuerung und Teleoperation von Robotern entwickelt werden. [38]

3. Analyse

In diesem Kapitel werden die Einsatzgebiete und die Aufgabestellungen der im Einleitungskapitel 1.2 erwähnten Szenarien detaillierter veranschaulicht. Des Weiteren werden verschiedene Entwicklungsmöglichkeiten anhand der Daten aus dem Grundlagenkapitel 2 vorgestellt. Im Anschluss wird die Auswahl der Frameworks für die Entwicklung der Szenarien bestimmt.

3.1. Einsatzgebiete

Der NAO umfasst viele Themen wie Informatik, Kinematik, Mechanik, Physik und vieles mehr. So kann er auch in Bereichen von Forschung, Unterricht in Schulen, interaktiver Messepräsentation bis hin zu gewöhnlichen Haushalten eingesetzt werden. Da der Schwerpunkt dieser Arbeit vor allem auf der Interaktion mit dem NAO liegt, werden die Anwendungsgebiete auch für die in dieser Arbeit entwickelten Szenarien erläutert.
Mit der Fähigkeit die Bewegung des Menschen nachzubilden, kann er als Assistenzroboter im Bereich Gymnastik, Tanzschulen oder auch als Instrukteur eingesetzt werden. Zum Beispiel kann ein Sporttrainer oder Krankengymnast einige Bewegungen vor dem Kinect-Sensor ausführen und NAO macht das nach bzw. NAO übernimmt Aufgaben des täglichen Lebens.

3.2. Aufgabenstellung

Im ersten Szenario wird eine einfache Steuerung des Roboters NAO mit dem Fokus auf den Oberkörper realisiert. Als Basis der Steuerung dienen die menschlichen Bewegungen des Oberkörpers, die mit Hilfe des Microsoft Kinect-Sensors in Echtzeit erfasst werden. Darauf aufbauend, dass der Roboter NAO menschliche Bewegungen imitieren kann, wird im zweiten Szenario eine menschliche Tätigkeit ausgeübt.

Der Hauptkern des zweiten Szenarios ist, dass NAO in der Zusammenarbeit mit einem Menschen zwischen den drei Objekten Plastik, Papier und Aluminium unterscheidet und dies unabhängig davon welche Form diese haben. NAO soll die drei Objekte, z.B. eine Flasche aus Plastik, einen Becher aus Papier oder eine Dose aus Aluminium erkennen und entsprechend in den passenden Papierkorb entsorgen. Für die Entwicklung des zweiten Szenarios sollen die Kriterien Personen-, Objekterkennung, Interaktion über Kamera, Audio und Voice-Kontroller, Lokalisierung und Navigation implementiert werden.

3.3. Auswahlkriterien des Frameworks

Programmiersprache

Nicht notwendig, aber von Vorteil ist es auf Basis der Python SDK zu entwickeln, da die Programmiersprache Python in allen verwendeten Komponenten, NAO und Kinect, genutzt werden kann und dies sogar Plattformunabhängig.

Skeleton Tracking

Für das erste Szenario ist das Skeleten-Traking besonders von Bedeutung, da somit durch Erkennung der Gelenkpositionen und Erfassung der Tiefendaten die Position einer Person im Raum bestimmt wird.

Objekterkennung (Landmark Detection)

Eine bereits im Framework integrierte Möglichkeit Landmarks zu erkennen, kann dafür verwendet werden, um Markierungen, sogenannte "NAOMarks", zu erkennen. Diese können zur Identifizierung genutzt werden und kompensieren damit eine aufwendigere Objekterkennung durch objektinhärente Eigenschaften.

Landmark Tracker (ALTracker)

Zur Auswahl des Frameworks stellt der Landmark-Tracker ein wichtiges Kriterium für das zweite Szenario dar. So ist es unabdingbar, dass sich dieses möglichst fehlerfrei, zuverlässig und genau arbeitet. Durch den Landmark-Tracker ist es möglich, Entfernungen und Positionen auf eine einfache Weise zu erfassen und die gelieferten Daten sinnvoll zu verwenden.

IDE Choregraphe

Wie bereits in dem Kapitel Grundlagen 2 erwähnt, bietet die Entwicklungsumgebung „IDE Choregraphe“ die Möglichkeit NAO anhand graphischer Boxen über Drag’n‘drop zu programmieren. Hinter jeder Box stehen ein oder mehrere Module, womit man NAO ansprechen kann bzw. den Ablauf des Szenarios (Behavior) in einem Flussdiadramm auf einer hohen Abstraktionsebene festlegen kann. Jede einzelne Box hat einen eigenen Python-Script und kann von dem Entwickler weiterbearbeitet werden. Die Boxen können unabhängig voneinander parallel laufen, so kann NAO z.B. während einer Textausgabe auch Bewegungen (Motions) durchführen. Choregraphe arbeitet mit dem Framework NAOqi, welches im Hintergrund der Entwicklungsumgebung läuft und somit die Möglichkeit bietet die Behavoir auf der IDE laufen zu lassen oder mit Einschränkungen, wie z.B. die Sensoren für Berührungen und Voice-Befehle, auf dem virtuellen Roboter zu simulieren bzw. zu testen. Im Anschluss können die Bahavior auf dem realen NAO Roboter gespeichert werden.

Die Abbildung 3.1 zeigt einen Überblick über die Entwicklungsumgebung, die in sechs

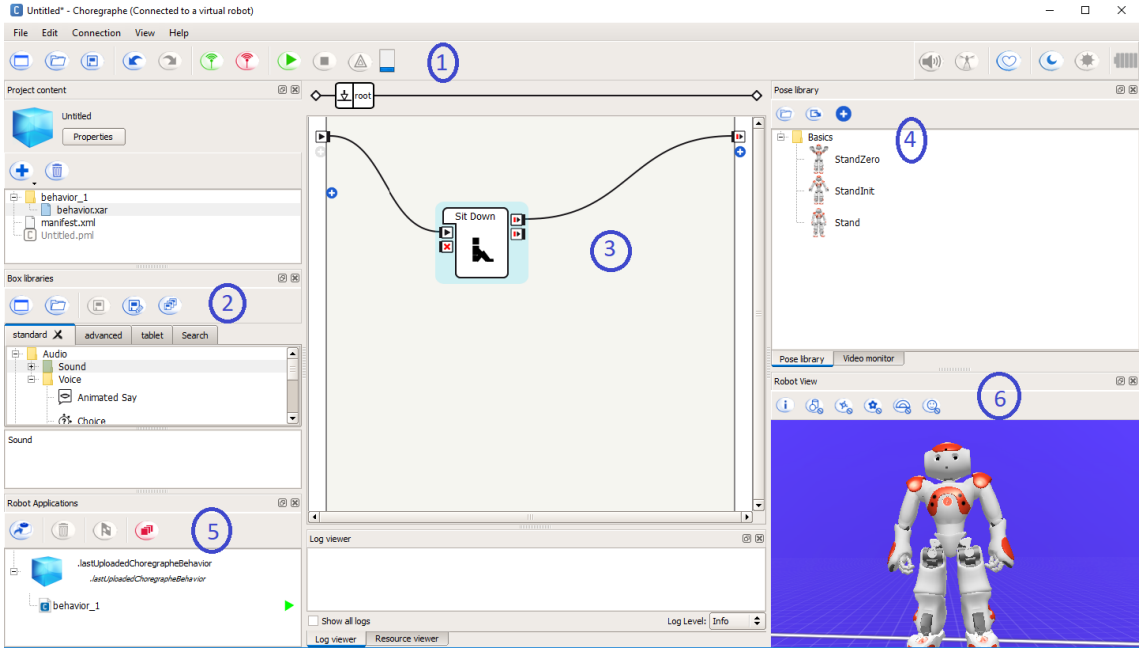


Abbildung 3.1.: Übersicht der Entwicklungsumgebung "Choregraphe"

verschiedene Kategorien unterteilt ist. Ganz oben in der Abbildung befindet sich die Menüleiste mit wichtigen Funktionen, um z.B. ein neues Projekt zu erstellen oder eine

Verbindung mit dem Simulator oder dem echten NAO herzustellen. Die Behavior können über die Schaltfläche mit dem grünen Pfeilsymbol auf den Roboter gespielt werden. Im mittleren linken Bereich, mit der Nummer 2 gekennzeichnet, befindet sich die View "Box libraries", in der alle vordefinierten Funktionen und Behavior aufgeführt sind. In der Mitte unter der Nummer 3 ist der grafische Programmierbereich, in dem ein Entwickler verschiedene Boxen aus den Boxbibliotheken miteinander verbinden kann, um den Ablauf des Behaviors festzulegen. Die Standartposen des Roberts sind rechts oben (Nummer 4) in der View "Pose-Library" enthalten. Unten links unter der Nummer 5 in der View "Robot Applications" können die installierten Behavior auf dem Roboter verwaltet werden. Als letztes befindet sich in der unteren rechten Ecke die mit Nummer 6 markierte View "Robot View", über die ein Entwickler die Bewegungen des simulierten Roberts nachverfolgen kann.

3.4. Teleoperations- & Objekterkennungsarten

Übertragung der Bewegungsabläufe (1:1)

Bewegungen, die direkt 1:1 umgesetzt werden, bieten wesentliche Vorteile bei der Steuerung NAOs. So kann der Benutzer aktiv und spontan auf Ereignisse eingehen und individuell reagieren. Gegenüber dem Aufrufen von Bewegungsmustern (Behavior) ist eine intuitive Steuerung NAOs durch simples vormachen von Bewegungen, ohne diese vorher einprogrammieren zu müssen und daher auch ohne Lernaufwand, möglich. Die Erkennung der Bewegungen des Benutzers stehen im kausalen Zusammenhang mit der Ausführung am NAO. Daher ist eine präzise Erkennung unabdingbar, damit es nicht zu ruckartigen bzw. fehlenden Bewegungen NAOs führt. Nicht so wäre dies mit den vorprogrammierten Bewegungsmustern, den Behaviorn, die nicht durch Bewegungserkennung, sondern lediglich durch das Abrufen ausgeführt werden [2].

Bewegungsabläufe durch Behevior

Behavior sind vorprogrammierte Bewegungsmuster, die ihre Anwendung immer da finden, wo keine Spontanität von Bewegungen erwartet wird, sondern nur vordefinierte Bewegungsabläufe ausreichen. Sie ermöglichen somit unabhängig der Qualität der Bewegungserkennung stets garantierte Bewegungsabläufe und damit auch mehr Sicherheit, wie z.B. der Balanceakt beim Gehen. Der Mensch hat individuelle Proportionen und

entsprechend einen anderen Schwerpunkt, sowohl im Stand als auch in Bewegungen. Dies führt dazu, dass das auf einem Bein stehen nicht 1:1 umgesetzt werden kann, weil aufgrund des unterschiedlichen Schwerpunktes NAO umkippen könnte, was zu einer instabilen Lage und einer unsicheren bis gefährlichen Situation führen könnte. Ebenso verheerend könnten sich Übertragungs- oder Berechnungsfehler auswirken.

Objekterkennung durch NAOMarks

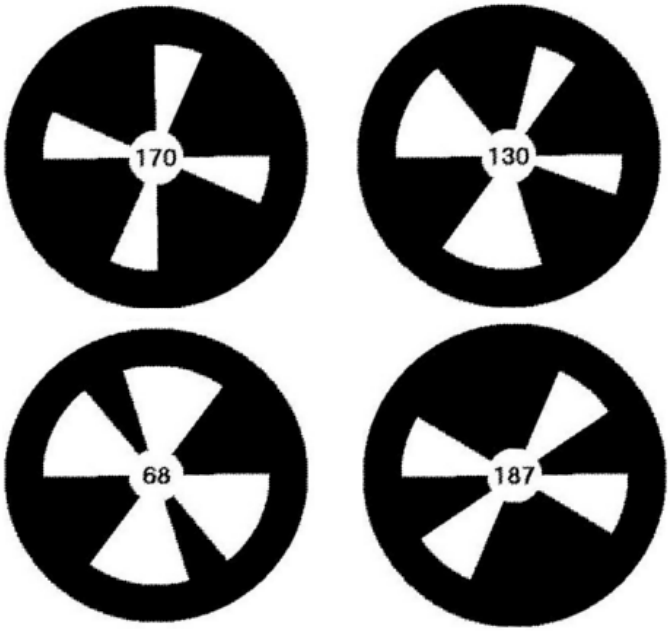


Abbildung 3.2.: Muster NAOMarks [39]

Eine Art der Objekterkennung ist die integrierte Funktion ALLandMarkdetection zu nutzen. Die Landmark wird als NAOMark bezeichnet, welche aus schwarzen Kreisen mit weißen Dreiecksfächern (siehe Abbildung 3.2) bestehen, die in der Mitte des Kreises zentriert sind. Die spezifische Position der verschiedenen Dreiecksfächer wird verwendet, um einen NAOMark von den Anderen zu unterscheiden. Bereits vorgefertigte NAOMarks ermöglichen einem Benutzer das Markieren von Objekten zur Objekterkennung ohne ein Objekt extra fotografisch zu Erzeugen und in einer Datenbank zu hinterlegen. Ein Algorithmus für NAOMarkerkennung und -decodierung wurde bereits von Aldebaran Robotics implementiert und ist in der NAOqi SDK verfügbar, die sich im ALLandMark-Detection Vision-Modul befindet [39].

Objekterkennung durch Bilderkennung

Eine einfache Art NAO Objekte erkennen zu lassen, ist die im Choregraphe eingebaute Funktion "Learn". Der Prozess beginnt mit Choregraphe. Der Benutzer muss den Videomonitor verwenden und zuerst eine neue Datenbank erstellen und dann ein Objekt lernen, das vor der NAO-Kamera angezeigt wird. Sobald das Objekt erfolgreich gelernt wurde, steht dem Benutzer ein Fenster zur Verfügung, in dem er das Objekt über Tags unter den drei Typen "Object", "Book ", "Location" definieren kann [40].

3.5. Ausgewählter Ansatz

Mit den zur Verfügung stehenden Teleoperationsarten ließen sich gut Synergien generieren, da Schwächen der Einen durch Stärken der Anderen kompensiert werden können. In Szenario 1 ist nun der Fokus bewusst auf die 1:1 Bewegungsübertragung gelegt, um ihre Leistungsfähigkeit zu demonstrieren. Die Möglichkeit der Behavior soll im zweiten Szenario zur Ausführung kommen und entsprechendes Potenzial vorführen. Die ausgewählten Ansätze versuchen bestmöglich die beiden Arten der Steuerung jeweils für sich zur Ausführung kommen zu lassen, um die Grenzen der Methoden offenzulegen [2]. Entsprechend der gewählten Bewegungsübertragungsarten fällt die Wahl des Frameworks beim ersten Szenario auf NAOqi SDK für Python auf der Roboterseite, Pykinect um vom Kinect die Bewegungsdaten zu empfangen und VPython zur virtualisierten Darstellung auf dem Computer. Obwohl C++ und Python mit ihren unterschiedlichen Stärken und Schwächen gleichermaßen genutzt werden können, liegt die Wahl für Python in der Einfachheit und Vielseitigkeit begründet, die dennoch alle gewünschten Ansprüche an dieses Projekt umsetzen lassen. Die Auswahl der IDE Choregraphe für das zweite Szenario hat gegenüber anderer Entwicklungsmöglichkeiten den wesentlichen Vorteil einer großen Bibliothek, die aus einer Menge von bereits fertig implementierten Boxen (Module) besteht. Jede Box erfüllt eine gesonderte Aufgabe, wie z.B. die NAOMark und dem LandMark-Tracker, die eine aufwendige manuelle Implementierung von Basisaktivitäten bereits zur Verfügung stellt und dennoch die Freiheit bietet eigene Boxen in Python zu programmieren.

4. Realisiertes Konzept

In diesem Kapitel werden die Konzepte und deren Umsetzung für die Szenarien aus dem Kapitel Analyse 3 dargestellt.

4.1. Bestimmung der Gelenkpunkte

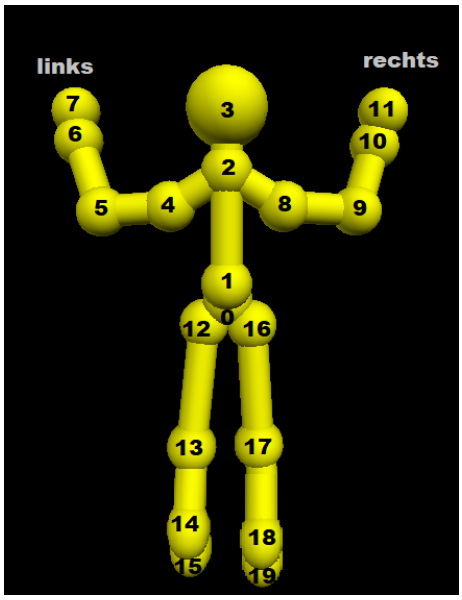


Abbildung 4.1.: Virtuelle Darstellung des Skelettes

Das Pykinect Package für die Programmiersprache Python bietet die Möglichkeit ein Skeleton-Tracking für 20 Gelenkpunkte durchzuführen. Die vom Kinect-Sensor gelieferten Daten werden über Pykinect zur Verfügung gestellt. Diese Daten beinhalten die Positionen der Gelenkpunkte als dreidimensionale Vektoren im kartesischen Koordinatensystem, relativ zur Mitte des Kinect-Sensors. Mit Hilfe des VPython Package können diese Gelenkpunkte benutzt werden, um einen virtuellen Menschen auf dem Computer

darzustellen [41]. Die Abbildung 4.1 zeigt eine virtuelle Darstellung eines Skelettes mit den 20 durchnummerierten Gelenkpunkten und der entsprechenden Achsenorientierung. Anhand von Algorithmen und den bereitgestellten Vektoren ist es nun möglich Gelenkstellungen und somit ein gesamtes Skelett nachzuvollziehen, indem die Winkel zwischen den einzelnen Gelenkpunkten berechnet werden. Damit der NAO den Menschen imitiert, müssen die einzelnen Winkel nun übertragen werden.

4.2. Berechnung der Gelenkwinkel

Durch die Gelenkpunktbestimmung stehen nun Vektoren zur Verfügung, sodass die Winkel zwischen den einzelnen Körperteilen errechnet werden können, indem beispielsweise im Fall eines Ellenbogens (siehe Abbildung 4.2), zwei Strahlen (Vektoren), die von dem Ellenbogen als Scheitelpunkt ausgehen, einen Winkel einschließen [2]. Die Algorithmen für diese Berechnungen basieren auf einer veröffentlichten Abhandlung [43].

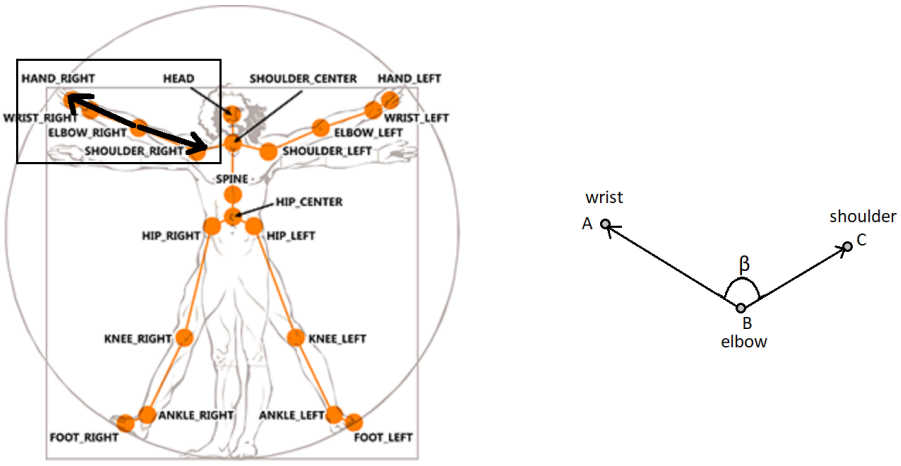


Abbildung 4.2.: Aufgestellte Vektoren am rechten Ellenbogen und dem eingeschlossenen Winkel β zwischen $\vec{BA}$ und $\vec{BC}$

In dem illustrierten Beispiel wird so durch Nutzung der Arc-Cosinus-Funktion

$$\beta = \arccos\left(\frac{\vec{v}_1 \cdot \vec{v}_2}{|\vec{v}_1| \times |\vec{v}_2|}\right) \tag{4.1}$$

der gewünschte Winkel Beta zwischen den zwei Vektoren v1 (WRISTRIGHT - ELBOW-

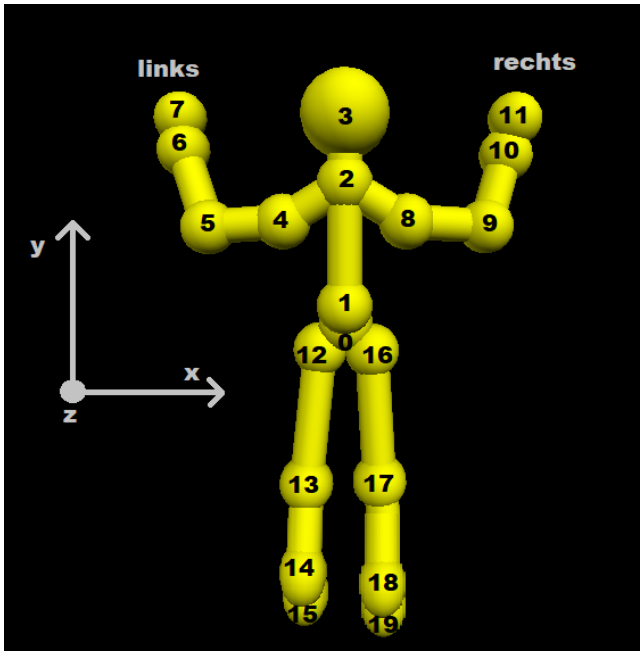


Abbildung 4.3.: Kinect-Skeleton-Model

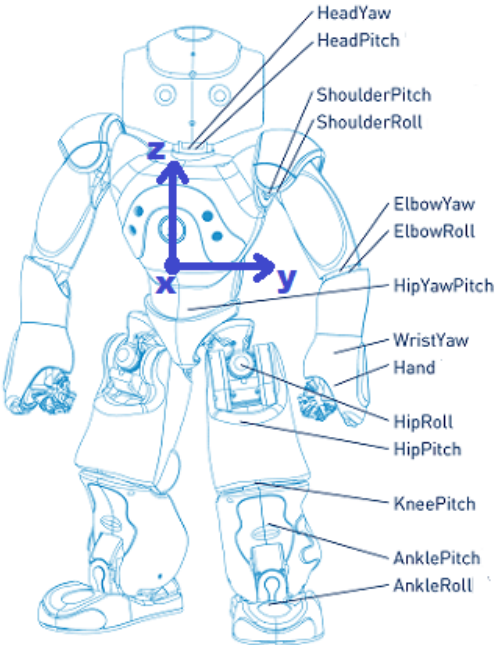


Abbildung 4.4.: NAO-Torso-Model [42]

RIGHT) für den rechten Unterarm und v2 (ELBOWRIGHT - SHOULDERRIGHT) für den rechten Oberarm errechnet. Je weiter sich die beiden Vektoren voneinander entfernen, desto größer wird der Winkel. Diese Winkel unterliegen stets der Prüfung nach realistischen menschlichen Grenzwerten. Die Symmetrie des Arc-Cosinus arccos(-x) = -arccos(x) führt folglich zu dem Wertebereich [0;Pi] [2].
Die Abbildung 4.3 zeigt die virtuelle Darstellung eines Gelenkpunktes eines Menschen, deren Position und Gelenkwinkel auf NAO übertragen werden sollen. Wie in Abbildung 4.4 dargestellt wird dazu ein "Torso-Space", ein abstrahiertes kartesisches Model, auf den NAO-Torso gemappt, wobei die entsprechenden Mittelpunkte kongruent sind. Aus den Abbildungen wird ersichtlich, dass die Positionen und Gelenkwinkel nicht 1:1 vom Kinect-Skeleton-Model zum NAO-Torso-Model übertragen werden können. Für die nachfolgende Berechnung der Gelenkwinkel ist nur die Richtung der Knochen relevant. Die Transformation aus den Kinect-Welt-Koordinaten hin zum Torso-Space ist folgendermaßen:

$$\vec{v}_{\mathbf{a,e}} = \mathbf{A} \cdot (\vec{v}_e - \vec{v}_a) \; with \; a, e \in \{0, 1, 2, ..., 19\} \tag{4.2}$$

Mit dem resultierenden Ergebnisvektor $\vec{v}_{\mathbf{a,e}}$ im Torso-Space, den Endpunkten $\vec{v}_a$ und $\vec{v}_e$ der von der Kinect in Kinect-Welt-Koordinaten vorliegenden Knochen und der Transformationsmatrix **A**. Vektoren im Torso-Space sind in fetten Kleinbuchstaben dargestellt, um sie von den Vektoren in Welt-Koordinaten zu differenzieren. Matrizen sind in fetten Großbuchstaben geschrieben. Für die Transformation der Kinect-Skelettdaten wird ein Dreieck von Endpunkten als Torsoreferenz benötigt. Für die weiteren Berechnungen wird das Dreieck zwischen Schulterzentrum (shoulder centre), dem Gelenkpunkt 2, und den Huftpunkten (Hips), den Gelenkpunkten 12 und 16, benutzt. Die z-Achse des Skeletttorsos ist orthogonal zum gewählten Dreieck und kann aus dem Vektorprodukt

$$\vec{z} = \frac{(\overrightarrow{v}_{16} - \overrightarrow{v}_{12}) \times (\overrightarrow{v}_{16} - \overrightarrow{v}_{2})}{|(\overrightarrow{v}_{16} - \overrightarrow{v}_{12}) \times (\overrightarrow{v}_{16} - \overrightarrow{v}_{2})|} \tag{4.3}$$

berechnet werden. Die x-Achse kann als Vektor zwischen den Hüften angenommen werden,

$$\vec{x} = \frac{\overrightarrow{v}_{16} - \overrightarrow{v}_{12}}{|\overrightarrow{v}_{16} - \overrightarrow{v}_{12}|} \tag{4.4}$$

während die y-Achse aus den Vektorprodukt

$$\vec{y} = \vec{z} \times \vec{x} \tag{4.5}$$

berechnet wird. Und zu guter Letzt sind die Elemente der Transformationsmatrix A folgend bestimmt

$$A = \begin{pmatrix} \frac{\vec{x}\cdot\vec{x}}{|\vec{x}\cdot\vec{x}|} & \frac{\vec{x}\cdot\vec{y}}{|\vec{x}\cdot\vec{y}|} & \frac{\vec{x}\cdot\vec{z}}{|\vec{x}\cdot\vec{z}|} \\ \frac{\vec{y}\cdot\vec{x}}{|\vec{y}\cdot\vec{x}|} & \frac{\vec{y}\cdot\vec{y}}{|\vec{y}\cdot\vec{y}|} & \frac{\vec{y}\cdot\vec{z}}{|\vec{y}\cdot\vec{z}|} \\ \frac{\vec{z}\cdot\vec{x}}{|\vec{z}\cdot\vec{x}|} & \frac{\vec{z}\cdot\vec{y}}{|\vec{z}\cdot\vec{y}|} & \frac{\vec{z}\cdot\vec{z}}{|\vec{z}\cdot\vec{z}|} \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{pmatrix} \tag{4.6}$$

Nach der Transformation der Knochen hin zum Torso-Space, ist die Berechnung jedes einzelnen Gelenkwinkels des Roboters möglich. Die Gelenkwinkel, die in Abbildung

4.4 dargestellt sind, können jetzt einfach auf Basis der Geometrie berechnet werden. Im Folgenden werden die Gelenkwinkelnamen (roll, pitch, yaw) in Anlehnung zur NAO-Choreographe-Softwarenomenklatur verwendet. Die Ellenbogen (elbow rolls) können direkt aus den Winkeln zwischen den korrespondierenden Ober- und Unterarmen berechnet werden. Zum Beispiel folgend für den rechten Ellenbogen

$$\boldsymbol{\alpha}_{elbow\_roll\_right} = \arccos\left(\frac{\overrightarrow{\boldsymbol{v}}_{\mathbf{8,9}} \cdot \overrightarrow{\boldsymbol{v}}_{\mathbf{9,10}}}{|\overrightarrow{\boldsymbol{v}}_{\mathbf{8,9}} \cdot \overrightarrow{\boldsymbol{v}}_{\mathbf{9,10}}|}\right) \tag{4.7}$$

Armbewegungen seitlich zum Körper werden durch den Schulterrollwinkel (shoulder roll) als Winkel zwischen dem Oberarm und der $\vec{\boldsymbol{y}}$-$\vec{\boldsymbol{z}}$-Ebene beschrieben, welche exemplarisch an der rechten Schulter wie folgend beschrieben wird

$$\boldsymbol{\alpha}_{shoulder\_roll\_right} = -\left(\frac{\pi}{2} - \arccos\left(\frac{(\vec{\boldsymbol{y}} \times \vec{\boldsymbol{z}}) \cdot \vec{\boldsymbol{v}}_{\mathbf{8,9}}}{|\vec{\boldsymbol{y}} \times \vec{\boldsymbol{z}}| \cdot |\vec{\boldsymbol{v}}_{\mathbf{8,9}}|}\right)\right) \tag{4.8}$$

Die auf und ab Bewegung des Arms (shoulder pitch) wird durch den Winkel in der Schulter beschreiben, wie z.B. für die rechte Schulter

$$\vec{\boldsymbol{v}}_{upper\_arm\_right} = \vec{\boldsymbol{v}}_{8,9} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \tag{4.9}$$

$$\boldsymbol{\alpha}_{shoulder\_pitch\_right} = -\arctan\left(\frac{v_2}{v_1}\right)$$

Die Rotationsbewegung des Unterarms zum Oberarm (elbow yaw) ist nur bestimmbar wenn der Ellenbogen betroffen ist. Diese Winkelberechnung ist auf Grund der Relativität von Pitch und Yaw zum Oberarm komplexer. Diesbezüglich wird ein Vektor von der

Schulter aus parallel zur z-Achse des zu berücksichtigenden Arms entsprechend der Pitch- und Yaw-Winkel gedreht.

$$\vec{b} = \begin{pmatrix} \cos(\alpha_{shoulder\_roll}) & \sin(\alpha_{shoulder\_roll}) & 0 \\ -\sin(\alpha_{shoulder\_roll}) & \cos(\alpha_{shoulder\_roll}) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(\alpha_{shoulder\_pitch}) & 0 & \sin(\alpha_{shoulder\_pitch}) \\ 0 & 1 & 0 \\ -\sin(\alpha_{shoulder\_pitch}) & 0 & \cos(\alpha_{shoulder\_pitch}) \end{pmatrix} \cdot \vec{z} \tag{4.10}$$

Folglich ist der Ellenbogen-Yaw zwischen dem Vektor $\vec{b}$ und dem Vektorprodukt des Ober- und Unterarms exemplarisch für den rechten Ellenbogen gegeben [43]

$$\begin{aligned} \vec{v}_{arm\_right} &= \frac{\vec{v}_{8,9} \times \vec{v}_{9,10}}{|\vec{v}_{8,9} \times \vec{v}_{9,10}|} \\ \alpha_{elbow\_yaw\_right} &= \left(\frac{\pi}{2} - \arccos\left(\frac{\vec{b} \cdot \vec{v}_{arm\_right}}{|\vec{b}| \cdot |\vec{v}_{arm\_right}|} \right) \right) \end{aligned} \tag{4.11}$$

4.3. Systemarchitektur

Die Abbildungen 4.5 und 4.6 stellen die Systemarchitekturen beider Szenarien dar. Sie bestehen aus fünf bzw. vier Komponenten, welche von dem Akteur bedient werden. Im ersten Szenario stellen der Kinect-Sensor und NAO hierbei die erste und zweite Komponente dar. Die dritte Komponente stellt der Rechner dar, auf welchem die Python Skripte laufen. Als vierte Komponente stellt der Router ein Netzwerk bereit, welches die zweite und dritte Komponente miteinander verbindet. Im zweiten Szenario besteht der einzige Unterschied darin, dass die Komponente Kinect-Sensor nicht gebraucht wird und Choregraphe keine externen Skripte benötigt. Des Weiteren sind die gewählten Systemkompositionen von einem Aktor abhängig, mittels dem die nachzubildenden Bewegungen im ersten Fall aufgenommen werden und NAO letztendlich als Input dienen und im anderen Fall im Austausch von Nachrichten bestehen.

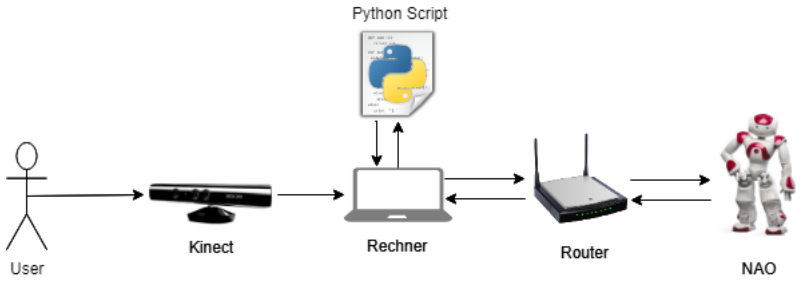


Abbildung 4.5.: Systemarchitektur des ersten Szenarios

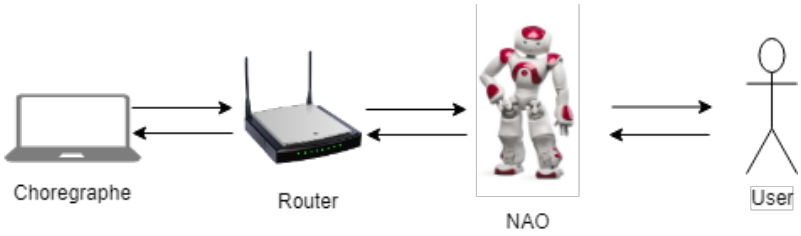


Abbildung 4.6.: Systemarchitektur des zweiten Szenarios

4.4. Technischer Ansatz

Zur Umsetzung der beiden Szenarien sind unterschiedlich Komponenten der Hardware von Nöten, die in der Tabelle 4.1 aufgelistet sind. Des Weiteren ist in ihr nachzuvollziehen, in welchem Szenario diese wie genutzt werden.

Alle NAOqi Module wurden im Grundlagenkapitel 2 tabellarisch 2.4 aufgeführt. Folgende Tabelle 4.2 zeigt farblich hervorgehoben die Zugehörigkeit der Hardwarekomponenten zu den jeweiligen Szenarien.

ALMotion
Das ALMotion-Modul bietet Methoden, die das Bewegen des Roboters erleichtern, wie in Szenario 1 beim setzen der Gelenkwinkel und in Szenario 2 zur Realisierung der Bewegungen zum Greifen nach Objekten.

ALTracker
Mit dem ALTracker-Modul kann der Roboter verschiedene Ziele (roter Ball, Gesicht, Orientierungspunkt, Landmark, usw.) unterschiedlich mitverfolgen (nur Kopf, ganzer Körper, Bewegung usw.).

Hardware ID	Szenario	Hardwarekomponente	Anwendung
Kinect_HW1	1	Kinect Sensor	Skeleton Tracking
Rechner_HW1	1 & 2	Rechner	IDE, Choregraphe, Simulator
Router_HW1	1 & 2	Router	Kommunikation
NAO_HW1	1 & 2	Gelenkmotoren	Körperbewegung
NAO_HW2	2	Kamera	NAOMark- & Gesichtserkennung
NAO_HW3	2	Mikrofon	Voice Befehl um den Szenario zu starten
NAO_HW4	1 & 2	Lautsprecher	Feedback an dem User

Tabelle 4.1.: Benutzte Hardwarekomponenten

Szenario 1		
Motion		
ALMotion		
ALRobotPosture		
Audio		
ALTextToSpeech		

Szenario 2		
Core	**Vision**	**Motion**
ALMemory	ALLandMarkDetection	ALMotion
		ALRobotPosture
Trackers	**Audio**	**Prople Perception**
ALTracker	ALTextToSpeech	ALFaceDetection

Tabelle 4.2.: Benutzte NAOqi-Module

Das Hauptziel dieses Moduls besteht darin, eine Brücke zwischen Zielerkennung und Bewegung zu schlagen, damit der Roboter das Ziel in der Mitte der Kamera im Blick behält. In Szenario 2 wird es benutzt, um die Position einer Landmark (Papierkorb) und der Person zu tracken.

ALMemory

ALMemory ist ein zentraler Speicher, in dem alle wichtigen Informationen zur Hardwarekonfiguration des Roboters gespeichert werden. Es liefert ebenso Informationen über den aktuellen Zustand der Aktoren und Sensoren.

ALLandMarkDetection

ALLandMarkDetection ist ein Vision-Modul, in dem der Roboter spezielle Landmarken

(NAOMarks) mit spezifischen Mustern erkennt. Es findet Einsatz in Szenario 2 um die verschiedenen Objekte zu erkennen.

ALRobotPosture
Das ALRobotPosture-Modul ermöglicht, den Roboter zu verschiedenen vordefinierten Positionen zu bewegen.

ALTextToSpeech
Das ALTextToSpeech-Modul ermöglicht dem Roboter zu sprechen. Er sendet Befehle an eine Text-in-Sprache-Engine und autorisiert auch die Sprachanpassung. Das Ergebnis der Synthese wird an die Lautsprecher des Roboters gesendet.

ALFaceDetection
ALFaceDetection ist ein Vision-Modul, in dem der Roboter versucht, Gesichter vor ihm zu identifizieren und gegebenenfalls wiederzuerkennen.

4.5. Fachliches Datenmodell

4.5.1. Szenario 1

Komponentendiagramm

Das Komponentendiagramm in Abbildung 4.7 gibt eine Übersicht über die geplanten Komponenten und ihre Beziehungsrolle. Zunächst wird eine Komponente "Kinecthandler" zur Breitstellung der Kinect-Sensor-Funktionalität erfordert. Diese greift auf vordefiniert Funktionen des Pykinect Packages zu und gewährleistet folgenden Komponenten den Zugriff auf die von Sensoren getrackten Daten. Diese wurden über die Schnittstelle zum User erfasst und über eine weitere Schnittstelle "Get Skeleton Data" zu Verfügung gestellt. Sie liegen noch nicht in einer verwertbaren Form bereit. Das Aufbereiten dieser Skeletondaten erfordert eine Komponente "Skeleton", die die einzelnen Gelenkpunkte und ihre verbindenden Knochen zu einem dreidimensionalen Vektor verarbeitet und somit über zwei Schnittstellen abgegriffen werden kann. Zum einen über "Visual" zur Visualisierung für den User-Feedback und zum anderen über "Get joints position" an die Komponente "Converter" zur weiteren Transformation, nach Vorgaben des NAO-Roboters, und Gelenk-

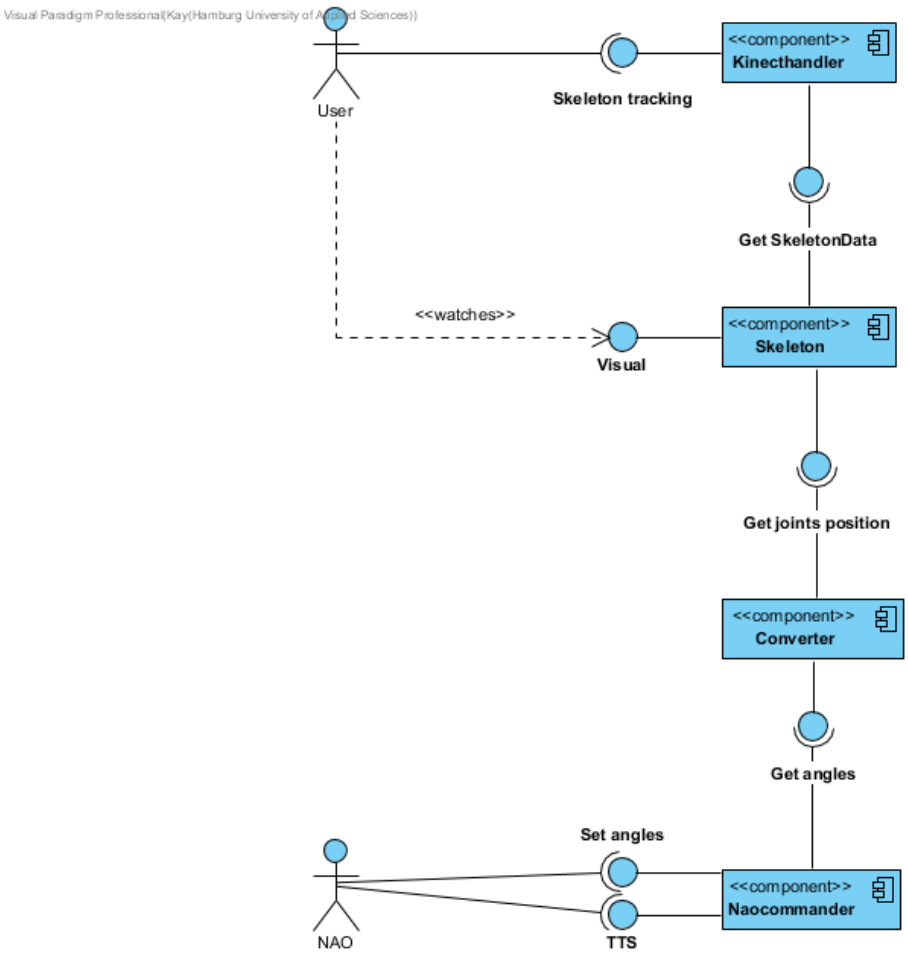


Abbildung 4.7.: Komponentendiagram Szenario 1

winkelberechnung. Eine letzte und für die Nutzung NAOs wesentliche Komponente stellt der "Naocommander" dar. Diese bezieht die Gelenkwickel über die Converter-Schnittstelle "Get angles", baut über das Modul "AlProxy" eine Verbindung zum Roboter NAO her und setzt entsprechende Winkel über seine Schnittstelle "Set angles". Damit der User von NAO Feedback über sein womögliches Fehlverhalten erhält, bekommt NAO über die Naocommander-Schnittstelle "TTS" den Ausgabetext übermittelt.

Klassendiagramm

Das in der Abbildung 4.8 dargestellte Klassendiagramm wurde entsprechend der Aufgabestellung aus dem Kapitel 3 entwickelt. Dieses Diagramm zeigt die Operationen, Eigenschaften und Assoziationen der einzelnen Klassen zueinander.

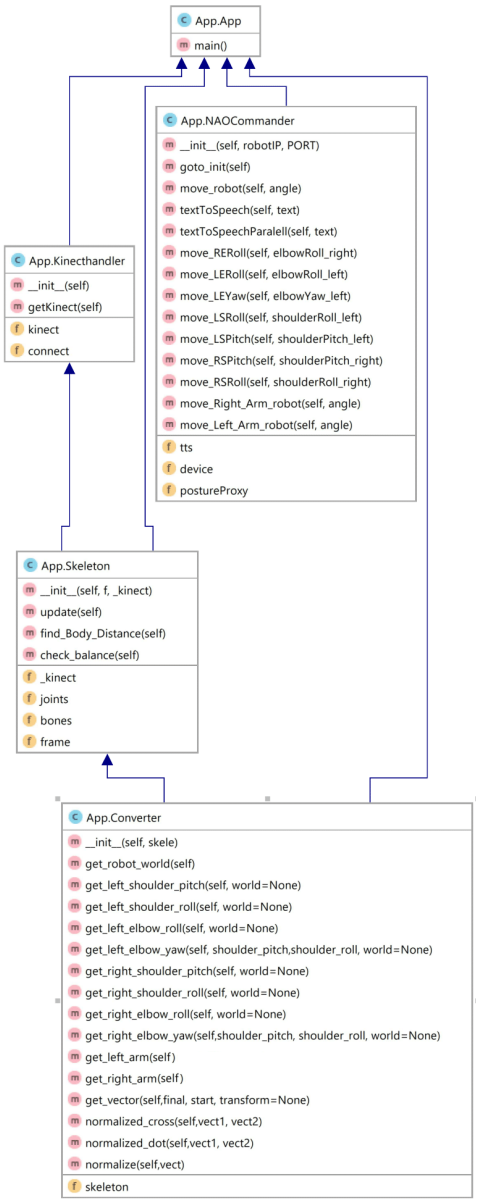


Abbildung 4.8.: Klassendiagramm Szenario 1

App-Klasse

Die App-Klasse ist eine Start-Klasse, in der die IP-Adresse und die Port-Nummer des NAOs fest gelegt werden, womit der Broker die Module registrieren wird. Das Weiteren beinhaltet die Klasse die Logik für den Ablauf des Szenarios.

Kinecthandler-Klasse

In dieser Klasse wird ein Kinect-Objekt initialisiert und bereitgestellt, welches die Verbindung mit dem Kinect-Sensor herstellt. Das Objekt wird über die Methode "getKinect()" abgerufen.

Skeleton-Klasse

Die Klasse Skeleton besitzt ein Kinect- und ein Frame-Objekt, um die Skeleton-Daten aus der Kinect-Tiefenkamera zu empfangen und diese aufbereitet als Skeleton visuell in Frames darzustellen. Des Weiteren werden die einzelnen Gelenkpositionen in das Attribut "joints" gespeichert, welche über die Methode "update()" stetig aktualisiert werden. Die Methode "find_Body_Distance()" überprüft den Abstand zwischen dem User und dem Kinect-Sensor. Die Methode "check_balance()" überprüft, dass der Zustand des User-Körpers nicht zu einer kritischen Imbalance des Roboters NAO führt.

Converter-Klasse

Diese Klasse transformiert die Skeleton-Daten in den NAO-Kontext und beinhaltet alle Methoden zur Gelenkwinkelberechnung, entsprechend der Algorithmen aus dem Kapitel 4.2.

Naocommander-Klasse

Diese Klasse baut mit einer vordefinierter IP-Adresse und einer Port-Nummer über das Proxy-Modul eine Verbindung mit dem NAO Roboter auf. Des Weiteren besitzt die Klasse Methoden zum Setzten der Gelenkwinkel und zum Übermitteln von Ausgabetexten.

Sequenzdiagramm

Mit Hilfe von zwei Sequenzdiagrammen wird zum einen 4.9 der Aufruf der Applikation mit entsprechenden Initialisierungen benötigter Komponenten und zum anderen 4.10 der Ablauf der Bewegungsimitation dargestellt. Das zweite Sequenzdiagramm führt alle wesentlichen Nachrichten zwischen den interagierenden Komponenten auf. Beginnend mit der Erfassung der Bewegungsdaten durch den Kinect-Sensor, über die Gelenkpunktspeicherung, der Virtualisierung, der Transformierung der Daten und der letztendlichen Übermittlung an NAO.

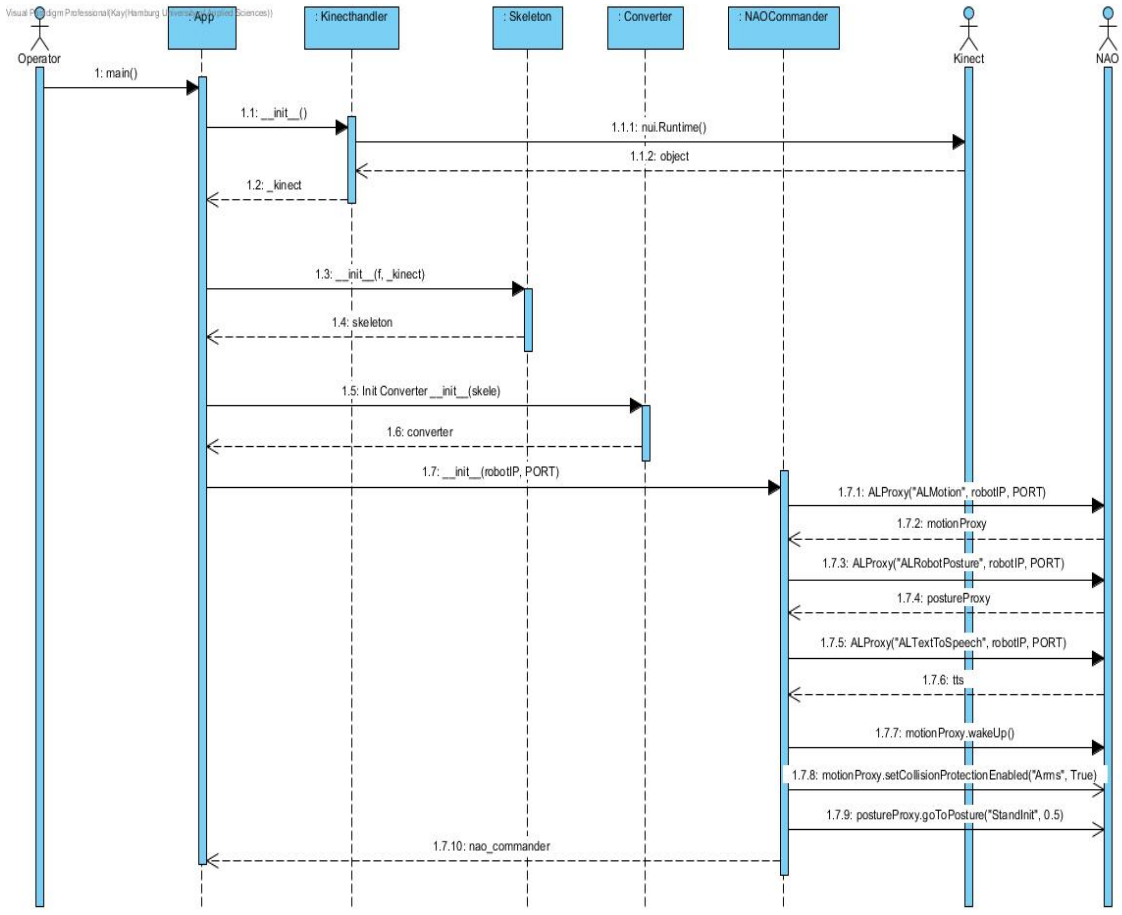


Abbildung 4.9.: Sequenzdiagramm Szenario 1: Initialisierungen der Komponenten

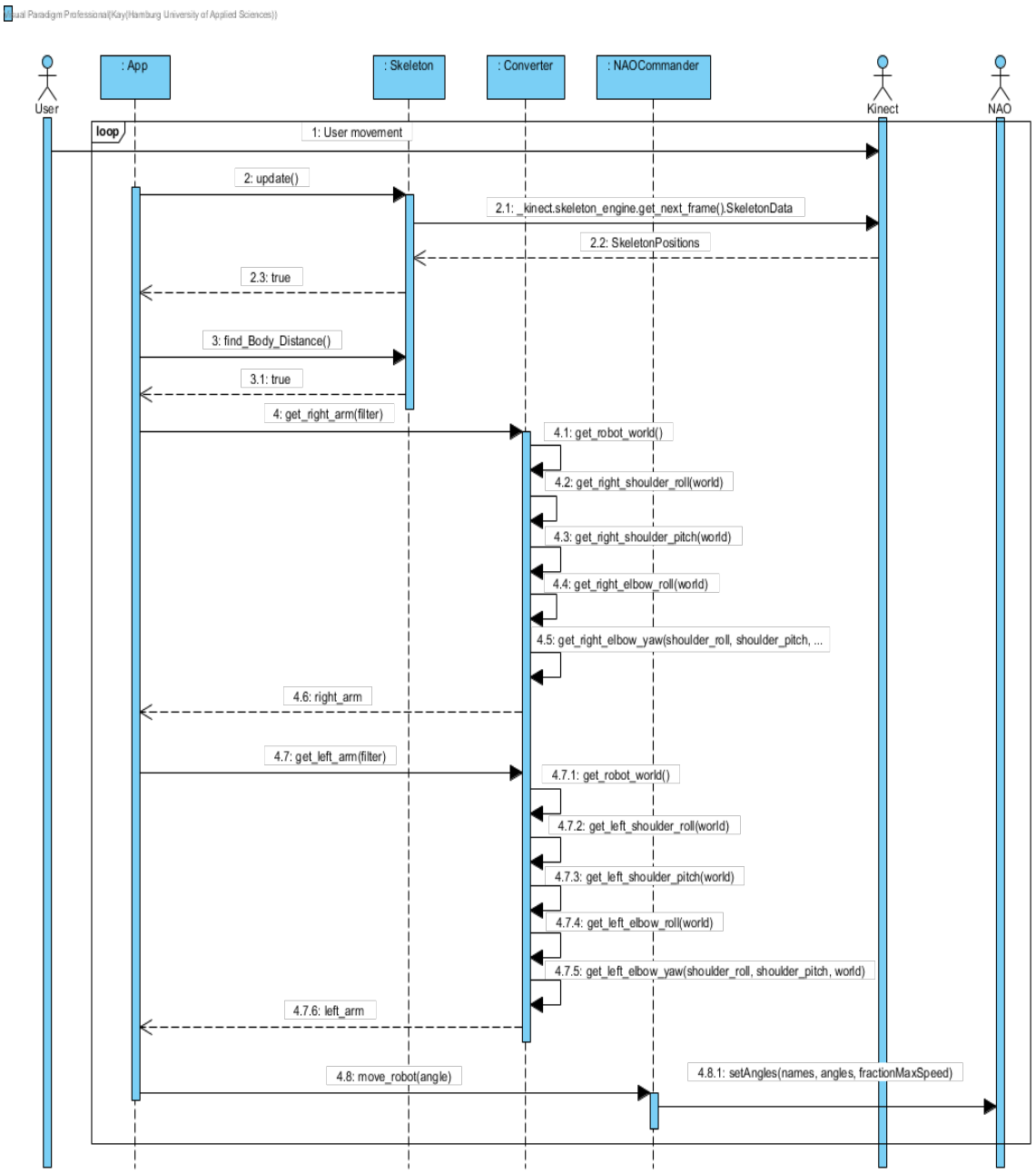


Abbildung 4.10.: Sequenzdiagramm Szenario 1: Ablauf der Bewegungsimitation

4.5.2. Szenario 2

Komponentendiagramm

Das Komponentendiagramm in Abbildung 4.11 beschreibt die anwendungsspezifische Komponente des Szenarios 2, die aus einer Hauptkomponente "NAOqi" besteht. Die Komponente ist im NAO integriert und enthält alle benötigten Module aus dem Abschnitt 4.4 und den Methoden für die Umsetzung des Szenarios 2.

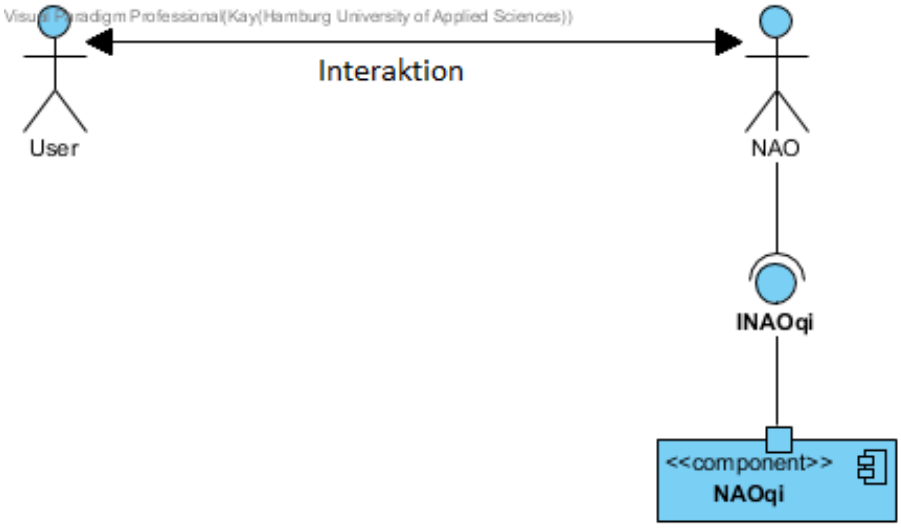


Abbildung 4.11.: Komponentendiagramm Szenario 2

Flussdiagramm

In der Abbildung 4.12 wird entsprechend der Aufgabenstellung 3.2 das Flussdiagramm zum zweiten Szenario in seiner Gesamtheit dargestellt. Eingangs wird NAO durch eine Initialisierung in die richtige Betriebsart versetzt, womit er eine aufrechte Körperhaltung mit definierter Körperspannung hat. Parallel wird die Gesichtserkennung des Users und eine dazugehörige suchenden Kopfbewegung gestartet. Dies wird so lange versucht, bis ein Gesicht erkannt wird. Bei entsprechender erfolgreicher Gesichtserkennung wird das Tracking des Gesichtes durchgeführt, was zur Folge hat, dass sich NAO bis zum erreichen des Users bemüht sich dem zu nähern. Erfolgreich beim User angekommen, beginnt NAO mit der Frage nach einem Auftrag zur Mülltrennung, die durch Bestätigung des Users mit dem Starten der Objekterkennungsroutine beginnt oder mit dem Ende des Szenarios, mit

vorherigem Wechsel NAOs in eine sichere Ruheposition, abschließt. Die Objekterkennung veranlasst nun das dedektieren von NOAMarks in seinem Sichtfeld. Der User sollte nun ein Mülltrennungsobjekt durch das Hinhalten in NOAs Sichtfeld zur Verfügung stellen, damit NAO eine darauf platzierte NAOMark identifizieren kann. NAO wird diesen Vorgang solange wiederholen, bis er eine vorher registrierte NAOMark erkennt, um anschließend die NAOMark einer der drei vordefinierten Müllarten zuzuordnen. Sollte ihm dies nicht gelingen, beginnt NAO erneut mit der Objekterkennung, andernfalls merkt er sich die Objektkategorie und hebt die linke Hand, um ein Müllobjekt in Empfang zu nehmen. Zur Annahme des entsprechenden Müllobjektes fragt NAO, ob er die linke Hand zum Greifen schließen darf. Dies würde er solange wiederholen, bist eine positive Rückmeldung registriert wird, um dann entsprechend die Hand zu schließen und in eine Position zum Gehen zu wechseln. Ab hier geht es um die Mülltonnensuche, für die NAO eine suchende Kopfbewegung und die erneute Dedektion von NAOMarks durchführt. Beides wird solange wiederholt, bis eine der registrierten Mülltonnen erkannt wird und anschließenden dem Trackingverlauf folgend erreicht wird. Bei entsprechendem erreichen der Mülltonne wird die linke Hand über der Mülltonne positioniert und zum Abwerfen des Müllobjektes geöffnet. Abschließend wechselt NAO wieder in die Ausgangspostion zur erneuten Gesichtserkennung, um den gesamten Vorgang erneut durchführen zu können.

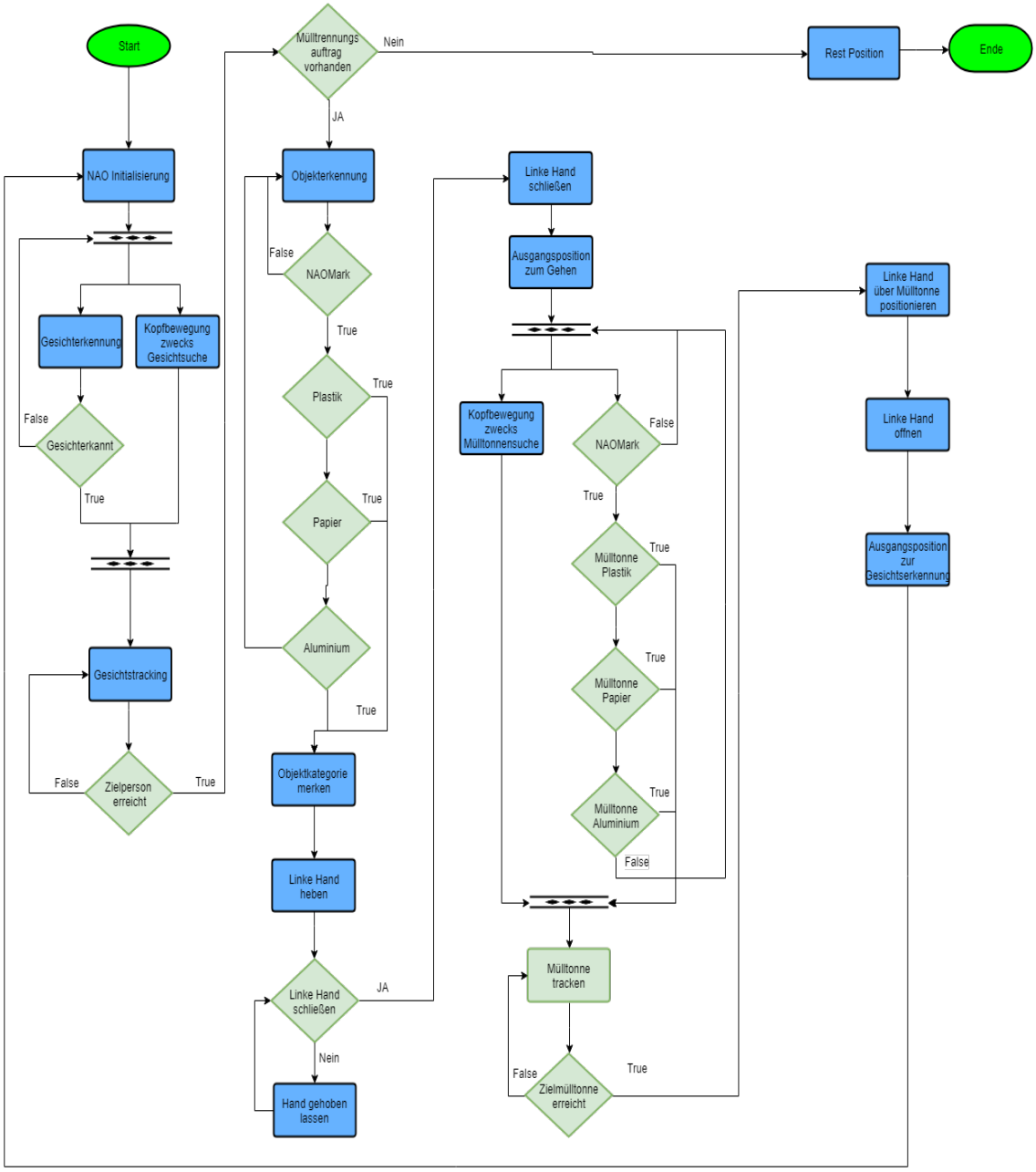


Abbildung 4.12.: Flussdiagramm Szenario 2

Sequenzdiagramm

Ein typischer Ablauf der Objekterkennung wird in der folgenden Abbildung 4.13 als Sequenzdiagramm entsprechend der Aufgabenstellung 3.2 dargestellt. Hierbei werden nicht nur der Nachrichtenaustausch zwischen NAO und dem User für das zweite Szenario bis zur Objekterkennung wiedergegeben, sondern auch die Interaktionen zwischen dem User und NAO berücksichtigt. Damit NAO mit der eigentlichen Objekterkennung beginnen kann, ist dessen Aktivierung über den Front-Kopfsensor vonnöten (1). Nun wird als weiterer Vorschritt eine Gesichtserkennung (3) mit einhergehender suchender Kopfbewegung (4) durchgeführt. Erst wenn dies erfolgreich erfüllt ist, kann NAO zum Gesichtstracking (5) übergehen und erst mit dem Erreichen des Users eine weitere Interaktion mit dem User starten. Diese besteht darin, dass NAO nach einem Auftrag fragt (6) und mit Erhalt einer verbalen Antwort des Users (7) entweder das Szenario in einer Ruheposition beendet (13) oder mit der Objekterkennung beginnt. Zunächst beginnt NAO hierfür mit der NAOMarkerkennung (8), die erst mit der Interaktion des Users, dem Zeigen eines Objektes mit einem bestimmten NAOMark (9), abgeschlossen wird. Bei erfolgreicher Erkennung des Objektes wird NAO dem User die entsprechende Objektkategorie rückmelden (10) oder andernfalls mit der Objekterkennung (8) erneut beginnen. Nach der Objekterkennung merkt NAO sich die Objektkategorie (11) und hebt die linke Hand (12).

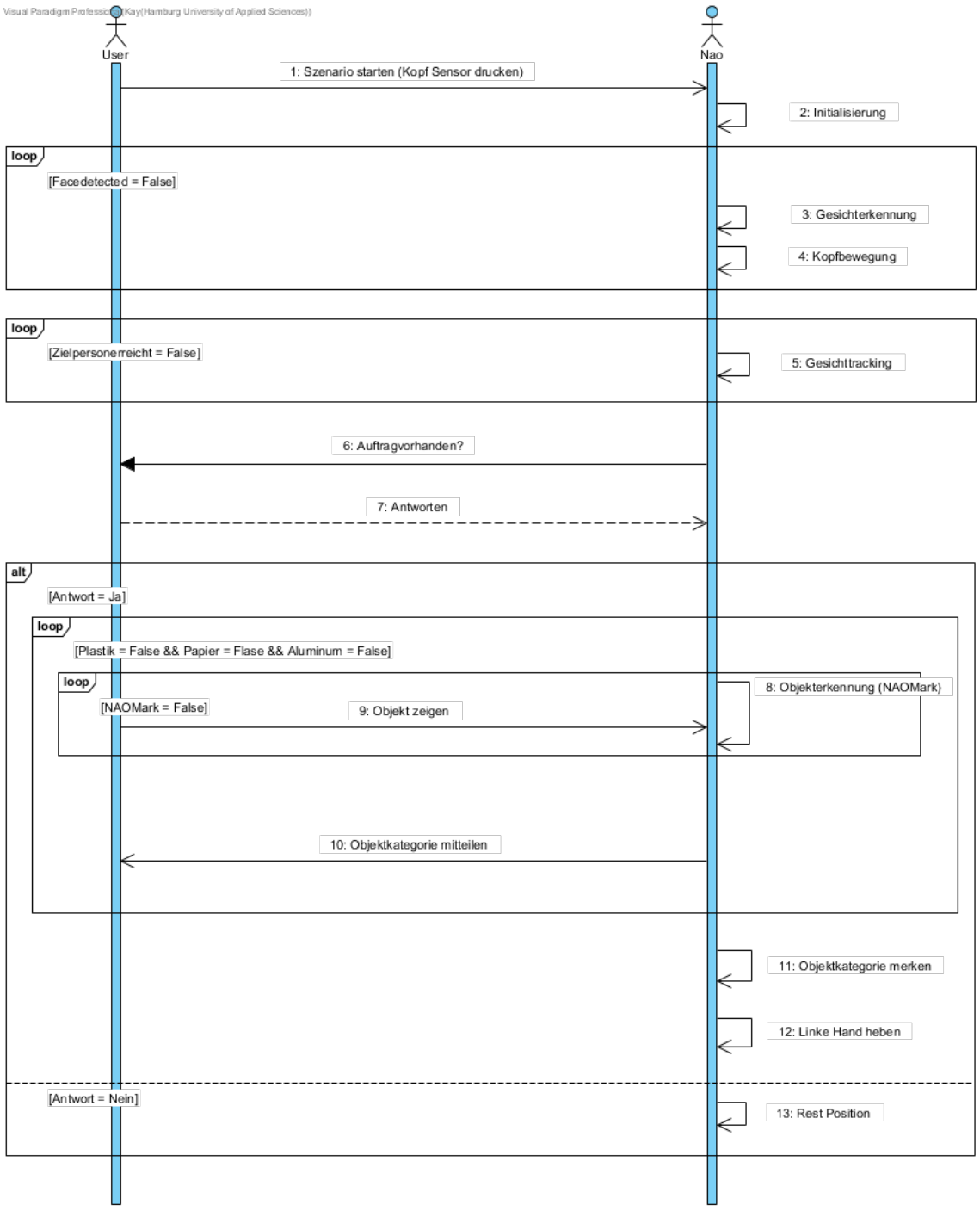


Abbildung 4.13.: Sequenzdiagramm Szenario 2

4.6. Implementierung

4.6.1. Szenario 1

Im folgenden wird die Implementierung des ersten Szenarios veranschaulicht. Die Umsetzung erfolgt nach den fachlichen Datenmodellen aus dem Abschnitt 4.5.1. Alle Klassen wurden in der Programmiersprache Python in der Version 2.7 32-Bit unter dem Betriebssystem Windows 10 geschrieben. Die Kinect-SDK v1.8 für Windows und NAOqi-SDK für Python sind auf dem Entwicklungsrechner installiert und werden durch die Python Bibliotheken (Module) Pykinect und VPython 6 erweitert. Es wird nicht der gesamte implementierte Code aufgeführt, sondern nur wesentliche Abschnitte erläutert, da der Codeumfang zu groß ist.

Erfassung der Bewegungsdaten

Die Erfassung der Bewegungsdaten wird im wesentlich durch die Klasse Skeleton realisiert. Dazu werden die von der Kinect getrackten Skeleton-Daten stetig aktualisiert erfasst.

```
1 for skeleton in self._kinect.skeleton_engine.get_next
2         _frame().SkeletonData:
3 if skeleton.eTrackingState == nui.SkeletonTrackingState.TRACKED:
```

Listing 4.1: Skeleton update-Methode (Skeleton.py)

Virtualisierung

Die für die virtuelle Darstellung der erfassten Bewegungsdaten von dem Kinect-Sensor benötigten Attribute werden in dem Listing 4.2 dargestellt. Das Attribut "_bone_ids" wird benötigt um die einzelnen Gelenkpunkte miteinander zu verbinden. In dem Attribut "joints" werden alle 20 von dem Kinect-Sensor erfassten Gelenkpunkte in Form einer Kugel mit den Radien 0,08 gespeichert. Die einzige Ausnahme stellt dabei der Gelenkpunkt 3 mit einem Radius von 0,125 zur Darstellung des Kopfes dar. Das Attribut "bones" enthält die einzelnen Knochen zwischen den Gelenkpunkten, die in Form von Zylindern mit einem Radius von 0,05 hinterlegt sind. Im Listing 4.3 werden, entsprechend der fachlichen Datenmodelle aus Abschnitt 4.5, die vorher erfassten Daten der Gelenkpositionen ent-

sprechend auf die Gelenkpunkte gemapped und dazwischen liegende Knochen berechnet, um daraus eine visuelle Spiegelung der Userbewegungen darzustellen.

```
_bone_ids = [[0, 1], [1, 2], [2, 3], [7, 6], [6, 5], [5, 4],
        [4, 2], [2, 8], [8, 9], [9, 10], [10, 11], [15, 14],
        [14, 13], [13, 12],[12, 0], [0, 16], [16, 17], [17, 18],
        [18, 19]]

self.frame = f
self.joints = [sphere(frame=f, radius=0.08, color=color.yellow)
        for i in range(20)]
self.joints[3].radius = 0.125
self.bones = [cylinder(frame=f, radius=0.05, color=color.yellow)
        for bone in _bone_ids]
```

Listing 4.2: Skeleton Attribute (Skeleton.py)

```
# Move the joints.
for joint, p in zip(self.joints, skeleton.SkeletonPositions):
  joint.pos = (p.x, p.y, p.z)

# Move the bones.
  for bone, bone_id in zip(self.bones, _bone_ids):
  p1, p2 = [self.joints[id].pos for id in bone_id]
  bone.pos = p1
  bone.axis = p2 - p1
  updated = True
```

Listing 4.3: Visualisierung der Skeletonparameter (Skeleton.py)

Transformierung der Daten

Für die Transformation der Kinect-Skelettdaten wird zunächst ein Dreieck von Endpunkten als Torsoreferenz erstellt, welche für die weiteren Berechnungen dienen. Entsprechend der in Abschnitt 4.1 aufgeführten Algorithmen wird so eine Matrix erstellt, die die letztendliche Transformierung, also der Übertragung der Koordinaten vom Kinect-Skeleton-Model zum NAO-Torso-Model, bewerkstelligt und die Grundlage für die Gelenkwinkelberechnungen schafft.

```
def get_robot_world(self):
  right_hip = self.skeleton.joints[16].pos
  left_hip = self.skeleton.joints[12].pos
  shoulder_spin = self.skeleton.joints[2].pos

  hip_vector = self.get_vector(right_hip,left_hip)
  spine_shoulder_to_hip =self.get_vector(right_hip,shoulder_spin)

  length_hip_vector = np.linalg.norm(hip_vector)

  vector_dot = np.cross(hip_vector,spine_shoulder_to_hip)
  length_vector_dot = np.linalg.norm(vector_dot)

  z = np.divide(vector_dot,length_vector_dot)
  x = np.divide(hip_vector,length_hip_vector)
  y = np.cross(z,x)
  _x = np.array([1, 0, 0])
  _y = np.array([0, 1, 0])
  _z = np.array([0, 0, 1])

  x1 = self.normalized_dot(x,_x)
  x2 = self.normalized_dot(x,_y)
  x3 = self.normalized_dot(x,_z)

  y1 = self.normalized_dot(y,_x)
  y2 = self.normalized_dot(y,_y)
  y3 = self.normalized_dot(y,_z)

  z1 = self.normalized_dot(z,_x)
  z2 = self.normalized_dot(z,_y)
  z3 = self.normalized_dot(z,_z)

  A = np.matrix([[x1, x2, x3], [y1, y2, y3], [z1, z2, z3]])
return [A, np.array([x, y, z])]
```

Listing 4.4: get_robot_world-Methode (Converter.py)

Gelenkwinkelbestimmung für den rechten Ellenbogen

Für die Gelenkwinkelbestimmung sind, wie auch schon in Abbildung 4.2 nachzuvollziehen, drei Gelenkpunkte vonnöten. Diese werden zunächst transformiert, ehe sie weiter Verwendung finden. Ein Gelenkpunkt, in diesem Fall der rechte Ellenbogen, dient als Scheitelpunkt und zwei Weitere werden gebraucht, zu denen jeweils ein Vektor berechnet wird. Der dazwischen liegende Winkel wird dann entsprechend der Formel 4.7 berechnet und stellt nunmehr ein Gelenkwinkel (als Rückgabewert) im NAO-Torso-Model dar. Die Berechnung anderer Gelenkwinkel wird analog umgesetzt. Lediglich die anzuwendende Formel ist entsprechend Roll, Pitch und Yaw angepasst.

```
def get_right_elbow_roll(self, world=None):
  if world is None:
    world = self.get_robot_world()
  shoulder = self.skeleton.joints[8].pos
  elbow = self.skeleton.joints[9].pos
  wrist = self.skeleton.joints[10].pos

  shoulder_elbow = self.get_vector(elbow,shoulder)
  shoulder_elbow = np.dot(world[0], shoulder_elbow)
  shoulder_elbow = np.squeeze(np.asarray(shoulder_elbow))
  len_shoulder_elbow = np.linalg.norm(shoulder_elbow)

  elbow_wrist = self.get_vector(wrist,elbow)
  elbow_wrist = np.dot(world[0], elbow_wrist)
  elbow_wrist = np.squeeze(np.asarray(elbow_wrist))
  len_elbow_wrist = np.linalg.norm(elbow_wrist)

  elbowRoll_dot = np.dot(shoulder_elbow,elbow_wrist)
  len_elbowRoll_dot = np.dot(len_shoulder_elbow,len_elbow_wrist)

  eRoll = (math.acos((elbowRoll_dot/len_elbowRoll_dot)))
  eRoll = max(eRoll, 0.069)
  eRoll = min(eRoll, 1.483)

```

```
25 return eRoll
```

Listing 4.5: get_right_elbow_roll-Methode (Converter.py)

Übermittlung an NAO

Damit die Daten an NAO übermittelt werden, müssen die Module gemäß dem Abschnitt 4.4 über Proxy vom Rechner bei NAO registriert werden. Um dies zu tun, werden im Listing 4.6 als Erstes die drei Module (ALMotion, ALRobotPosture & ALTextToSpeech) bei NAO registriert. Als Nächstes werden über das Proxy-Modul "ALMotion" die Methoden "wakeup()" und "setCollisionProtectionEnabled("Arms", True)" aufgerufen, um die Motoren des NAOs zu starten und den Antikollisionsschutz der Arme des Roboters zu aktivieren. Abschließend wird NAO anhand des Moduls "ALRobotPosture" in eine vordefinierte Haltung gebracht.

```
1  from naoqi import ALProxy
2  import motion
3
4  class NAOCommander:
5
6    def __init__(self, robotIP, PORT):
7      motionProxy = ALProxy("ALMotion", robotIP, PORT)
8      postureProxy = ALProxy("ALRobotPosture", robotIP, PORT)
9      tts = ALProxy("ALTextToSpeech", robotIP, PORT)
10
11     motionProxy.wakeUp()
12     motionProxy.setCollisionProtectionEnabled("Arms", True)
13
14     postureProxy.goToPosture("StandInit", 0.5)
15     pNames = "Body"
16     motionProxy.setStiffnesses(pNames, 0.5)
```

Listing 4.6: Module Verbindung des NAOs und Initialisierung des Startvorgangs (Naocommander.py)

Nun ist NAO bereit seine ersten Gelenkwinkel zu erhalten, wie im Listing 4.7 zu sehen. Die Gelenke, die gesetzt werden sollen, werden mit den von Aldebaran festgelegten Namen in einer Liste eingetragen und eine korrespondierende Liste mit Gelenkwinkel, die von der Klasse Converter im Vorfeld 4.5 berechnet wurden, wird dazu bereitgestellt. Die Methode "setAngles()" setzt die endgültigen Gelenkwinkel am NAO.

```
def move_robot(self, angle):
    jointnames = ["RShoulderRoll","RShoulderPitch","RElbowRoll",
        "RElbowYaw","LShoulderRoll","LShoulderPitch","LElbowRoll"
        ,"LElbowYaw"]
    movement = [angle[0],angle[1], angle[2], angle[3], angle[4],
        angle[5], angle[6], angle[7]]
    self.device.setAngles(jointnames, movement, 0.5)
```

Listing 4.7: move_robot-Mehtode (Naocommander.py)

Abschließend wird über das Proxy-Modul "ALTextToSpeech (tts)" zum einen die Ausgabesprache gesetzt und zum anderen die Übertragung des Ausgabetextes an NAO veranlasst.

```
def textToSpeech(self, text):
 self.tts.setLanguage("English")
 self.tts.say(text)
```

Listing 4.8: textToSpeech-Mehtode (Naocommander.py)

4.6.2. Szenario 2

Die Implementierung des Szenarios 2 erfolgt in der IDE Choregraphe Version 2.1.4.13 unter dem Betriebssystem Windows 10. Die in diesem Szenario eingesetzten NAOMarks sind in der Tabelle 4.3 aufgelistet und stellen ein wichtiges Element bei der Objekterkennung dar. Sie dienen nicht nur zur Erteilung des Mülltrennungsauftrages eine Rolle, sondern auch bei der Mülltonnenfindung und sind entsprechend ein visuelles Element, die klar sichtbar im Blickfeld NAOs sein müssen. Für die Auftragserteilung sind drei NAOMarks entsprechend für die drei Müllkategorien registriert, die NAO in diesem Zusammenhang ausschließlich versucht zu dedektieren. Die NAOMarks sind nicht nur wie im Abschnitt 3.4 beschrieben visuell unterscheidbar, sondern auch durch ihre jeweilige eindeutige ID. Zum Einsatz kommen die NAOMarks 68 für Objekte aus Plastik, 108 für Objekte aus Papier und 112 für Objekte aus Aluminium. Für die Mülltonnen sind 85 für Plastikmüll, 114 für Papiermüll und 119 für Aluminiummüll hinterlegt. Die Auswahl der NAOMarks ist frei von der Semantik der Zahlenwerte der IDs gewählt und entsprechend austauschbar.

NAOMark	Anmerkung	Symbol
68	Plastik Objekte z.B. eine Flasche	
108	Papier Objekte z.B. ein Pappbecher	
112	Aluminium Objekte z.B. eine Dose	
85	Mülltonne für Plastik	
114	Mülltonne für Papier	
119	Mülltonne für Aluminium	

Tabelle 4.3.: Benutzte NAOMarken Szenario 2

Eine Gesamtübersicht des implementierten Szenarios ist in Abbildung 4.16 zu sehen, welches aus der Gesamtheit aller verwendeten Boxen und ihrer Assoziationen zueinander besteht. Zur besseren Verständlichkeit sind logische Gruppen in einzelne Teilbereiche visuell farbig umrandet zusammengefasst dargestellt. Im ersten Bereich findet die Initialisierung der Sprache, der Sensorik und der Motorik statt. Der zweite Bereich ist für die Usersuche zuständig, was sowohl die Gesichtserkennung als auch das Usertracking beinhaltet. Bereich 3 implementiert die Erkennung der einzelnen Müllobjekte, während die entsprechend darauf beruhende Mülltonnenzuordnung im vierten Bereich umgesetzt wird. Die Mülltonnenlokalisierung und Entsorgung des Mülls findet im Bereich 5 statt. Bereich 6 definiert die Abschlussaktivität NAOs, die mit einer Ruhepostion endet.
Exemplarisch sei an dieser Stelle noch eine Box aus der Implementierung herausgegriffen,

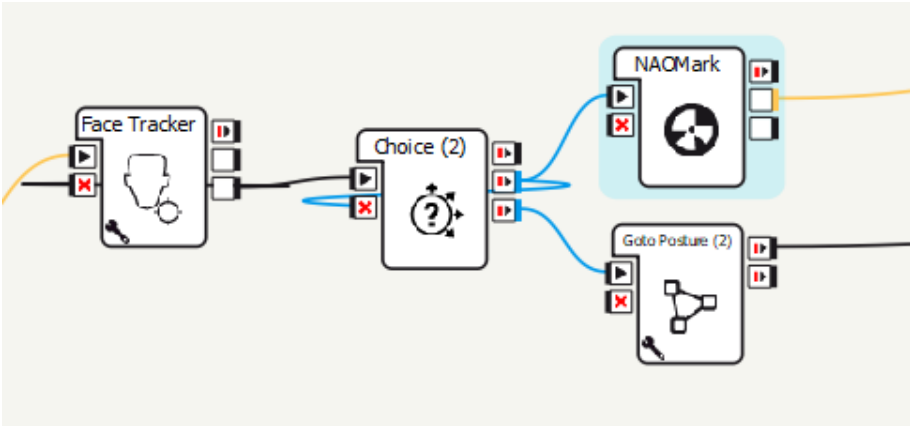


Abbildung 4.14.: Choice Box Szenario 2

um die Funktionalität im Kontext der Implementierung zu veranschaulichen. Die Abbildung 4.14 zeigt die Box der verbalen Interaktion zur Mülltrennungsauftragsannahme, die nach dem Usertracking statt findet. Die Box besitzt entsprechend über Ein- und Ausgänge für die Kommunikation zu weiteren Boxen. Die Eingänge sind onStart Input und onStop Input. Die Ausgänge sind drei onStopped Outputs. Zu erkennen ist, dass ein onStart Signal vom Usertracking eingeht und damit die Box aktiviert und ein onStopped Singnal als Output, das sowohl als Input für die folgende NAOMarkbox, als auch als Input zur Aktivitätsbeendigung der eigenen Box, dient. Der letzte onStopped Output veranlasst unter Umständen die Aktivierung der Folgebox, die NAO in den Ruhezustand versetzt. Die tatsächlich gesendeten Outputsignale unterliegen der Interaktion mit dem User.

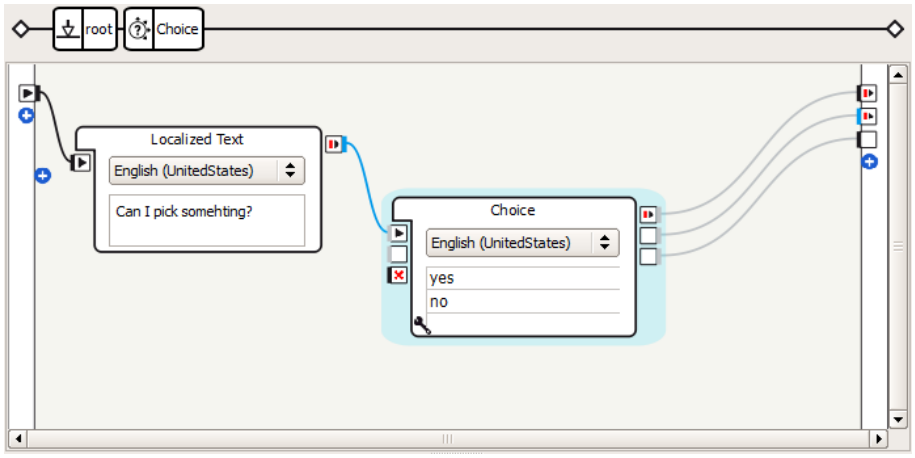


Abbildung 4.15.: Die Logik hinter dem Choice Box Szenario 2

Die hinter der Choice-Box liegende Logik ist in der Abbildung 4.15 dargestellt. Das Bild zeit, dass Boxen durchaus verschachtelt als Container für weitere Boxen dienen und logisch zusammenhängende Einheiten vereinen. Hier sind das die Boxen Localized Text und Choice. Localized Text dient zum Einstellen der Sprache und der auszugebenden Sprachnachricht, die bei der Interaktion mit dem User von NAO ausgegeben wird. Die Choice-Box fungiert als Einheit zur Spracherkennung und Interpretation der bei der Interaktion mit eingegangen Userantwort, was entsprechend unterschiedliche Outputsignale veranlasst. Auf NAOs Frage "Can I pick something?" wird nur eine der voreingetragenen Antwortmöglichkeiten "Yes" oder "No" von der Spracherkennung angenommen und zur Entscheidungsfindung für die Outputsignale interpretiert. Anderweitig würde bei einer Fehlerkennung ein Default-Outputsignal ausgegeben werden.

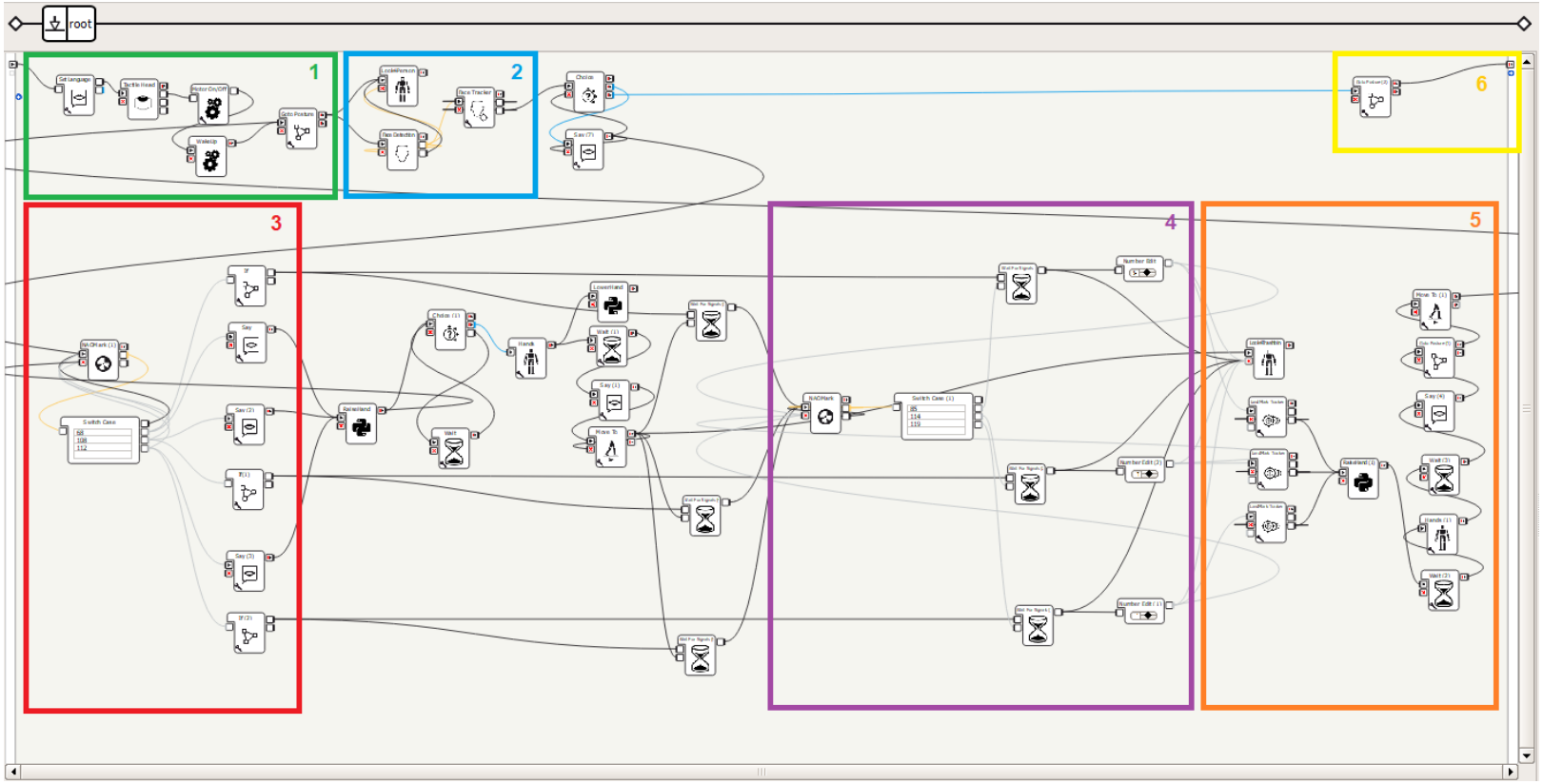


Abbildung 4.16.: Gesamtübersicht der Implementierung Szenario 2

5. Test

Um den Erfolg der implementierten Szenarien zu überprüfen, wurden die Szenarien sowohl auf dem Simulator bzw. Monitor, als auch real auf NAO getestet. Ebenfalls werden die verwendeten Werkzeuge und die Voraussetzungen für die Tests vorgestellt.

5.1. Testvoraussetzungen

Für die Tests stellen einige Hardware- und Softwarekomponenten eine Grundvoraussetzung dar und sind in der Tabelle 4.1 und in dem Kapitel 3 aufgeführt. Des Weiteren bestehen Vorraussetzung an die Umgebung und benutzten Hilfsmittel. Die Raumausleuchtung muss gleichmäßig sein und ist somit von dem Tageslicht unabhängig zu bewerkstelligen. Dies wird durch künstliches Deckenlicht umgesetzt. Damit NAO und der User sich frei bewegen können, muss der Testraum über einen hinreichend großen Freiraum verfügen. Im ersten Szenario müssen NAO und der User ihre Arme weit ausstrecken können und den Mindestabstand von zwei Metern zwischen dem User und der Kinectkamera dennoch gewährleistet sein. Im zweiten Szenario müssen entsprechend für den User mit den aufzugebenden Müllobjekten, den Mülltonnen und dem dazwischen agierenden NAO Platz zur Verfügung stehen. Es dürfen also keine Hindernisse im Wege stehen, NAOMarks müssen frei und möglichst eben angebracht sein und Müllobjekte in Größe und Gewicht handlich für NAO sein. Zu berücksichtigen ist für das zweite Szenario auch, dass die englischsprachigen Befehle "yes" und "no" bekannt sein müssen. Ebenso ist die Höhe der Mülltonnen an NAOs Größe angepasst.

5.2. Szenario 1

In den folgenden Testabläufen wird die Implementierung hinsichtlich des ersten Szenarios getestet.

5.2.1. Verhalten & Darstellung der Armbewegungen

Die Umsetzung einer Folge von Armbewegungen wird überprüft.

Anforderungen

Choregraphe, der Router, die Kinect und NAO sind betriebsbereit bzw. initialisiert und die Applikation wurde erfolgreich gestartet. Der User hat ein Mindestabstand von zwei Metern zur Kinect einzuhalten und eine gleichmäßige Raumausleuchtung wird gewährleistet.

Kriterien

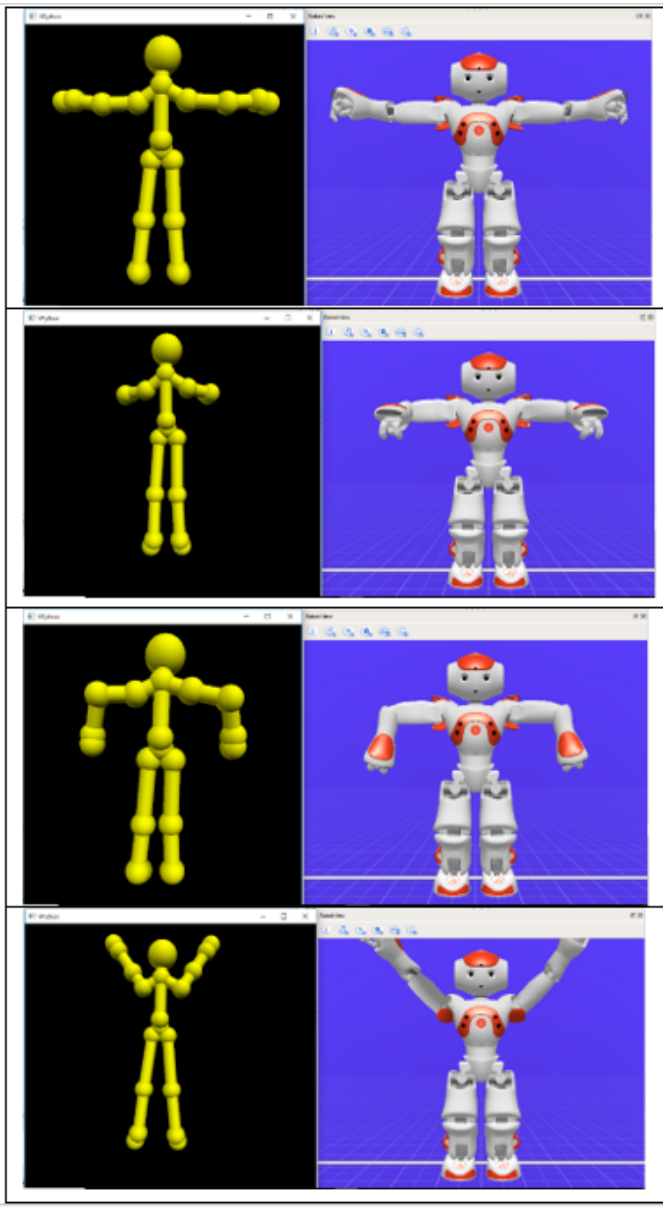
Die Armstellungen sollen sowohl in der Simulation, als auch von NAO widergespiegelt werden und die von NAO umgesetzten Gelenkwinkel sollen somit den berechneten Wickeln entsprechen.

Testablauf

Die Folge von Armbewegungen sieht zunächst vor, dass die Arme seitwärts im Winkel von 90 Grad vom Körper weg gestreckt werden. Anschließend die Ellenbogen um 90 Grad nach vorne ausgerichtet angewinkelt werden. Danach werden die Schultern um 90 Grad verdreht, sodass die Unterarme nach unten ausgerichtet sind. Abschließend werden die Arme gerade nach oben gestreckt.

Auswertung

Die gesamte Folge von Armbewegungen wurde wie in Abbildung 5.1 erfolgreich ausgeführt. Die Abbildung zeigt die jeweiligen Armstellungen in der Simulation, NAO und die dazugehörigen Gelenkwinkel in Rad. Die Auswertung der Gelenkwinkel zeigt nur minimale Diverenzen im Promillebereich zwischen den Soll- und Ist-Werten und kann bezüglich der Zielsetzung aus Abschnitt 3.2 als hinreichend genau interpretiert werden.



Position	Kinect	NAO
RShoulderPitch	0,07201	0,07135
RShoulderRoll	-1,32	-1,32
RElbowYaw	0,64496	0,64027
RElbowRoll	0,28714	0,28580
LShoulderPitch	0,04037	0,04398
LShoulderRoll	1,32	1,32
LElbowYaw	-0,49943	-0,48272
LElbowRoll	-0,28639	-0,24949

Position	Kinect	NAO
RShoulderPitch	0,21257	0,21250
RShoulderRoll	-1,15260	-1,15168
RElbowYaw	0,00481	0,00653
RElbowRoll	1,14254	0,14318
LShoulderPitch	0,17583	0,17908
LShoulderRoll	1,10325	1,10936
LElbowYaw	0,02997	0,01598
LElbowRoll	-0,93496	-0,96791

Position	Kinect	NAO
RShoulderPitch	0,21515	0,20135
RShoulderRoll	-1,20774	-1,20664
RElbowYaw	-1,05466	-1,06279
RElbowRoll	1,22440	1,21443
LShoulderPitch	0,01318	0,01001
LShoulderRoll	1,32	1,32
LElbowYaw	1,18748	1,19248
LElbowRoll	-1,31984	-1,31654

Position	Kinect	NAO
RShoulderPitch	-1,03897	-1,03495
RShoulderRoll	-0,56753	-0,57073
RElbowYaw	-1,09527	-1,08622
RElbowRoll	0,07402	0,07438
LShoulderPitch	-1,03137	-1,03595
LShoulderRoll	0,56029	0,55552
LElbowYaw	0,14310	0,14854
LElbowRoll	-0,09340	-0,93434

Abbildung 5.1.: Simulierte Darstellungen von vier umgesetzten Armstellungen und den dazugehörigen Gelenkwinkel in Rad

5.2.2. Verhalten & Darstellung der Beinbewegungen

Die Umsetzung einer Folge von Beinbewegungen wird auf die Balance NAOs überprüft.

Anforderungen

Choregraphe, der Router, die Kinect und NAO sind betriebsbereit bzw. initialisiert und die Applikation wurde erfolgreich gestartet. Der User hat ein Mindestabstand von zwei Metern zur Kinect einzuhalten, der User steht mit beiden Beinen auf dem Boden und eine gleichmäßige Raumausleuchtung wird gewährleistet.

Kriterien

Die Beinstellungen sollen in der Simulation widergespiegelt werden, von NAO hingegen nicht. Die Beinstellung von NAO soll stets in der initialisierten Ausgangsposition verbleiben. Wenn der User ein Bein anhebt, soll NAO nur mit der akustischen Warnmitteilung "No No No. Please put your foot down." reagieren.

Testablauf

Der User hebt zunächst nur das rechte Bein mindestens 10 cm an, wobei nur die Simulation dies widerspiegelt und NAO dennoch mit beiden Beinen standfest verharrt. Die Warnmitteilung wird akustisch ausgegeben. Dies gilt genauso beim Anheben des linken Beines des Users.

Auswertung

In beiden Fällen zeigte die Simulation das angehobene Bein und NAO behielt wie zu erwarten einen sicheren Stand und gab die Warnmitteilung akustisch aus.

5.3. Szenario 2

In den folgenden Testabläufen wird die Implementierung hinsichtlich des zweiten Szenarios, wie in Abbildung 4.16 dargestellt, blockweise mit verschiedenen Müllobjekten geprüft.

5.3.1. Gesichtserkennung & -tracking

Die Implementierung wird hinsichtlich der Gesichtserkennung und -trackings des Users überprüft.

Anforderungen

Das Behavior muss bereits im NAO gespeichert sein und NAO betriebsbereit und initialisiert sein. Es dürfen keine Hindernisse im Raum sein und eine gleichmäßige Raumausleuchtung wird gewährleistet. Das Gesicht des Users muss frei sichtbar sein.

Kriterien

Nach dem betätigen des Front-Kopfsensors soll NAO mit der Suche nach einem Gesicht beginnen und erkennen. Ein erkanntes Gesicht soll getrackt werden, d.h. auch, dass NAO sich dem getrackten Gesicht nähern soll und nach einem Auftrag fragen soll.

Testablauf

NAOs Front-Kopfsensor wird betätigt, woraufhin dieser nach einem Gesicht Ausschau hält und dieses nach der Erkennung verfolgt. NAO wird sich dem getrackten Gesicht nähern, "Can I pick somehting?" akustisch ausgeben und eine Antwort des Users entgegennehmen.

Auswertung

Nach drücken des Front-Kopfsensors hat NAO nach kurzem Suchen das Gesicht des Users erfolgreich erfasst, getrackt und sich dem User genähert. Die Erfassung des Gesichtes ist in Abbildung 5.2 zu sehen. Ebenso erfolgreich waren die akustische Mitteilung und die Erkennung der Rückantwort des Users.

Abbildung 5.2.: Von NAO erfasstes Gesicht

5.3.2. Müllobjekterkennung & -entgegennahme

Die Implementierung wird hinsichtlich der Müllobjekterkennung & -entgegennahme überprüft.

Anforderungen

Der vorherige Test 5.3.1 bereits mit einer positiven Rückantwort durchlaufen sein. Eines der vordefinierten Müllobjekte muss mit dessen NAOMark frei sichtbar zur Verfügung stehen und dieses NAO bei Bedarf vom User in dessen Hand gelegt werden.

Kriterien

Ein erkanntes Müllobjekt soll akustisch mitgeteilt und gleichzeitig gespeichert werden. Danach soll NAO seine Hand anheben und die akustische Anfrage "Can I close my hand." stellen. Bei positiver Rückmeldung soll NAO seine Hand schließen und senken.

Testablauf

Das Müllobjekt wird NAO gezeigt, woraufhin NAO das erkannte Müllobjekt dem User akustisch mitgeteilt. NAO hebt seine Hand an und stellt dem User die Frage, ob er seine Hand schließen darf. Der User stellt NAO das Müllobjekt greifbar zur Verfügung und bestätigt NAOs Frage, der wiederum seine Hand schließt und senkt.

Auswertung

Die Müllobjekterkennung, -entgegennahme und Kommunikation verliefen reibungslos. Eine erfolgreiche Speicherung des erkannten Müllobjektes ist erst durch einen weiteren Test nachvollziehbar.

5.3.3. Mülltonnenzuordnung

Die Implementierung wird hinsichtlich der Mülltonnenzuordnung überprüft, was nichts anderes ist, als eine Verwertung der vorherigen gespeicherten Müllobjektinformation und einer darauf folgende NAOMarkerkennung.

Anforderungen

Der vorherige Test 5.3.2 muss bereits positiv durchlaufen sein. Die vordefinierten Mülltonnen müssen mit deren NAOMarks frei sichtbar zur Verfügung stehen.

Kriterien

NAO soll entsprechend der zuvor gespeicherten Müllobjektinformation die passende Mülltonne finden.

Testablauf

Zunächst führt NAO eine suchende Kopfbewegung durch. Während dessen wird die NAOMarkerkennung durchführt. Bei erfolgreicher Mülltonnenzuordnung wird ein Signal ausgegeben.

Auswertung

Die Mülltonnenzuordnung wurde erfolgreich durchgeführt, welche mittels Choregraphe nachvollzogen wurde.

5.3.4. Mülltonnenlokalisierung & -entsorgung

Die Implementierung wird hinsichtlich der Mülltonnenlokalisierung und -entsorgung überprüft.

Anforderungen

Es muss bereits eine Mülltonnenzuordnung stattgefunden haben.

Kriterien

NAO soll die zugeordnete Mülltonne lokalisieren und sich ihr nähern. Daraufhin soll er seine Hand anheben und öffnen. Nach kurzem Warten gibt er eine akustische Mitteilung aus und senkt zugleich seine Hand.

Testablauf

NAO nähert sich der zugeordneten Mülltonne und hebt beim erreichen dieser seine geschlossene Hand an, öffnet diese nun zur Müllentsorgung und verharrt zwei Sekunden still, bevor er eine akustische Mitteilung ausgibt und währenddessen seine Hand wieder senkt.

Auswertung

Die Mülltonnenlokalisierung und -entsorgung wurde wie zu erwarten erfolgreich durchgeführt. Jedoch zeigte sich bei einigen Tests, dass NAO nicht stets alle Müllobjekte festhalten konnte, da seine Handspanunnung und Proportionen nicht der eines Menschen entsprechen.

6. Fazit

6.1. Zusammenfassung

Roboter sind in der heutigen Zeit ein fester Bestandteil bei der Fertigung von Produkten aller Art, wie z.B. in der Autoindustrie, in der Medizintechnik oder auch in der Fertigung von Unterhaltungselektronik. Der nächste konsequente Schritt wäre nun humanoide Roboter auch im Alltag zu integrieren. Der besondere Aspekt ist es hier einen Roboter zu erschaffen, der dem Menschen nachempfunden ist, um so leichter eine Interaktion mit dem Menschen zu ermöglichen und eine mögliche Hemmschwelle zu vermeiden. Es sollte nun nicht das Ziel sein den Menschen durch Roboter zu ersetzen, viel mehr sollte sich die Gesellschaft diese zu Nutze machen. Bei dem demographischen Wandel, welcher derzeitig stattfindet, können Roboter Aufgaben von Altenpflegern übernehmen bzw. diese unterstützen.
Die menschliche Interaktion mit Robotern spielt hierbei eine wichtige Rolle und die Relevanz wird anhand der vorliegenden Arbeit erläutert. Zu diesem Zweck wird zunächst in den Grundlagen auf die Eigenschaften des Roboters NAO eingegangen. Hierbei werden wichtige Merkmale des Roboters im Einzelnen erläutert. Darüber hinaus wird die zur Interaktion benötigte Kinect Kamera vorgestellt. Schließlich wird ein Einblick auf die verschiedenen Entwicklungsmöglichkeiten von NAO und der Kinect-Sensor gegeben.
Im Anschluss daran werden bei der Analyse die zwei Szenarien der Aufgabenstellung detailliert veranschaulicht. Des Weiteren wird die Auswahl des gewählten Frameworks zur Umsetzung der Aufgabenstellung begründet und der gewählte Ansatz zur Umsetzung festgelegt.
Im nächsten Schritt wird zunächst der Algorithmus zur Bestimmung der Gelenkwinkel anhand der von Kinect gelieferten Gelenkpunkte für das erste Szenario vorgestellt. Danach werden die Systemarchitekturen und der technische Ansatz zur Realisierung der beiden Szenarien bestimmt. Im Anschluss wird ein Entwurf zur Umsetzung der

Szenarien erstellt, welcher zwei Komponentendiagramme, ein Klassenmodell für das erste Szenario, ein Flussdiagramm für das zweite Szenario und drei Sequenzdiagramme, welche die Abhängigkeiten der Komponenten in den Abläufen verdeutlichen, beinhaltetet. Entsprechend den Anforderungen und dem Konzept wurden dann die Szenarien implementiert. Hierzu werden beim ersten Szenario anhand von Codeabschnitten die wichtigsten Funktionalitäten aufgezeigt. Beim zweiten Szenario werden exemplarisch die für die Implementierung benötigten Choregraphe-Boxen vorgestellt.
Zum Schluss werden geeignet Testfälle kreiert und ausgeführt um die Implementierung auf ihre Richtigkeit zu testen.

6.2. Ausblick

Zu Beginn dieser Arbeit wurde das Ziel festgelegt, zwei verschiedene Szenarien für den humanoiden Roboter NAO auf Basis einer Mensch-Maschinen-Interaktion und einer Maschinen-Umgebung-Interaktion zu entwickeln. Das erste Szenario umfasst die Imitation eines Menschen mit Hilfe der Kinect-Kamera, welcher mit dem Fokus auf den Oberkörper realisiert wurde. Beim zweiten Szenario besteht der Hauptkern in der Zusammenarbeit des NAOs mit einem Menschen. Dieser soll eine Mülltrennung zwischen drei verschiedenen Objekten bewerkstelligen. Hierzu wurde im Rahmen der Arbeit eine Analyse durchgeführt, aus der ein Realisierungskonzept entstand. Die Szenarien wurden dann anhand des Konzept erfolgreich umgesetzt und getestet, wobei einige Möglichkeiten zur Erweiterung der Szenarien bestehen. Beim ersten Szenario besteht die Möglichkeit einer Erweiterung, mit dem Fokus auf den Unterkörper zu realisieren. Die Gelenkwinkel können anhand des bereits vorhandenen Algorithmus der vorliegenden Arbeit berechnet werden, jedoch muss bei der Erarbeitung des Szenarios auf die Balance des NAOs geachtet werden, um so Schäden am NAO zu vermeiden. Eine andere Erweiterungsmöglichkeit besteht darin den Kinect V2 für die Bestimmung der Gelenkpunkte zu benutzen. Der Kinect V2 liefert bis zu 25 Gelenkpunkte, wodurch sich mehrere Möglichkeiten bei der Erweiterung des Szenarios bieten. Beim zweiten Szenario bieten sich viele verschiedene Möglichkeiten der Erweiterung an. Es können beispielsweise Hindernisse beim gehen des NAOs erkannt und umgangen werden. Dies wären unter anderem Themen, die in einer anderen Arbeit erörtern werden könnten.

A. Inhalt der CD-ROM

Dieser Arbeit liegt eine CD-ROM mit folgender Verzeichnisstruktur bei:

- **[Ausarbeitung]** beinhaltet diese Arbeit im PDF-Format.
- **[NAOMark]** beinhaltet in dieser Arbeit verwendete NAOMarks im PDF-Format.
- **[Quellcode]** beinhaltet die Quellcodes der entwickelten Anwendungen inkl. Readme.
  - **[NAOKinect]** Szenario 1
  - **[NAO_Muelltrennung]** Szenario 2

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literatur

[1] Inc Mitsubishi UFJ Financial Group. *Humanoid Robot NAO*. 2015. URL: http://www.mufg.jp/english/ir2015/e_v/it/ (besucht am 09. 10. 2017).

[2] Simon Päusch. *Teleoperation des humanoiden Roboters NAO per Kinect-Kamera. Bachelor thesis*. 30. Nov. 2011. URL: http://www.servicerobotik-ulm.de/drupal/sites/default/files/bachelor-simon-paeusch.pdf (besucht am 20. 10. 2017).

[3] Ee Sian Neo/ Kazuhito Yokoi/ Shuuji Kajita/ Fumio Kanehiro und Kazuo Tanie. "A Switching Command-Based Whole-BodyOperation Method for Humanoid Robots". In: *IEEE/ASME Trans. Mechatronics* 10.5 (5. Okt. 2005). URL: https://www.academia.edu/18542755/A_Switching_Command-Based_Whole-Body_Operation_Method_for_Humanoid_Robots (besucht am 09. 10. 2017).

[4] ALDEBARAN. *Unveiling of NAO Evolution: a stronger robot and a more comprehensive operating system*. 20. Juni 2014. URL: https://www.ald.softbankrobotics.com/sites/aldebaran/files/press-releases/cp_nao_evolution_en_def.pdf (besucht am 09. 10. 2017).

[5] ALDEBARAN. *NAO - Versions and Body Type*. 2017. URL: http://doc.aldebaran.com/2-1/family/body_type.html (besucht am 09. 10. 2017).

[6] ALDEBARAN. *NAO - Technical Guide*. 2017. URL: http://doc.aldebaran.com/2-1/family/index.html (besucht am 14. 10. 2016).

[7] ALDEBARAN. *H25 - Construction*. 2017. URL: http://doc.aldebaran.com/2-1/family/nao_h25/dimensions_h25.html (besucht am 09. 10. 2017).

[8] ALDEBARAN. *Charging the battery*. 2017. URL: http://doc.aldebaran.com/2-1/_images/wiz_step2.png (besucht am 29. 11. 2017).

[9] ALDEBARAN. *NAO - Battery*. 2017. URL: http://doc.aldebaran.com/2-1/family/robots/battery_robot.html#robot-battery (besucht am 10. 10. 2017).

[10] ALDEBARAN. *NAO - Video camera*. 2017. URL: http://doc.aldebaran.com/2-1/family/robots/video_robot.html#robot-video (besucht am 10. 10. 2017).

[11] ALDEBARAN. *NAO - Technical overview*. 2017. URL: http://doc.aldebaran.com/2-1/family/robots/index_robots.html (besucht am 30. 11. 2017).

[12] ALDEBARAN. *Event and state notifications*. 2017. URL: http://doc.aldebaran.com/2-1/nao/voice_notifications.html (besucht am 11. 10. 2017).

[13] ALDEBARAN. *LEDs*. 2018. URL: http://doc.aldebaran.com/2-1/_images/naoh25_leds.png (besucht am 15. 01. 2018).

[14] ALDEBARAN. *Sonars*. 2017. URL: http://doc.aldebaran.com/2-1/family/robots/sonar_robot.html#robot-sonar (besucht am 12. 10. 2017).

[15] ALDEBARAN. *Sonars*. 2017. URL: http://doc.aldebaran.com/2-1/_images/hardware_usposition.png (besucht am 21. 10. 2017).

[16] ALDEBARAN. *Motors*. 2017. URL: http://doc.aldebaran.com/2-1/_images/hardware_motortype_h25V5.png (besucht am 11. 11. 2017).

[17] ALDEBARAN. *Datasheet NAO Next Gen - H21/H25 Model - English version*. 2012. URL: https://www.ald.softbankrobotics.com/sites/aldebaran/files/datasheet_nao_next_gen_en.pdf (besucht am 12. 10. 2017).

[18] ALDEBARAN. *Joints*. 2017. URL: http://doc.aldebaran.com/2-1/family/robots/joints_robot.html (besucht am 09. 10. 2017).

[19] ALDEBARAN. *NAO - Actuator & Sensor list*. 2017. URL: http://doc.aldebaran.com/2-1/_images/hardware_jointname.jpg (besucht am 11. 11. 2017).

[20] ALDEBARAN. *Connectivity*. 2017. URL: http://doc.aldebaran.com/2-1/family/robots/connectivity_nao.html (besucht am 12. 10. 2017).

[21] ALDEBARAN. *Low level architecture*. 2017. URL: http://doc.aldebaran.com/1-14/_images/dcm_electronic_arch.png (besucht am 12. 10. 2017).

[22] ALDEBARAN. *Key concepts*. 2017. URL: http://doc.aldebaran.com/2-1/dev/naoqi/index.html#naoqi-framework-overview (besucht am 11. 10. 2017).

[23] ALDEBARAN. *Simulator SDK package*. 2017. URL: http://doc.aldebaran.com/2-1/ref/simulator_sdk.html (besucht am 11. 10. 2017).

[24] ALDEBARAN. *NAOqi Framework*. 2017. URL: http://doc.aldebaran.com/1-14/_images/broker-libraries-modules.png (besucht am 30. 11. 2017).

[25] ALDEBARAN. *NAOqi Framework*. 2017. URL: http://doc.aldebaran.com/1-14/_images/broker-modules-methods.png (besucht am 30. 11. 2017).

[26] ALDEBARAN. *NAOqi APIs*. 2017. URL: http://doc.aldebaran.com/2-1/naoqi/index.html (besucht am 19. 10. 2017).

[27] ALDEBARAN. *Choregraphe overview*. 2017. URL: http://doc.aldebaran.com/1-14/software/choregraphe/choregraphe_overview.html (besucht am 19. 10. 2017).

[28] ALDEBARAN. *SDKs*. 2017. URL: http://doc.aldebaran.com/1-14/dev/sdk.html (besucht am 19. 10. 2017).

[29] Inc. Open Source Robotics Foundation. *Documentation*. 2017. URL: http://wiki.ros.org/ (besucht am 22. 11. 2017).

[30] I. Open Source Robotics Foundation. *Aldebaran Nao*. 2017. URL: http://wiki.ros.org/nao (besucht am 27. 11. 2017).

[31] I. Open Source Robotics Foundation. *kinect*. 2017. URL: http://wiki.ros.org/kinect (besucht am 27. 11. 2017).

[32] Hartmut Gieselmann. *Open-Source-Treiber für Kinect veröffentlicht*. 11. Nov. 2010. URL: https://www.heise.de/newsticker/meldung/Open-Source-Treiber-fuer-Kinect-veroeffentlicht-1135025.html (besucht am 23. 10. 2017).

[33] James Ashley Jarrett Webb. *Chapter 1: Getting started.* In: *Beginning Kinect Programming with the Microsoft Kinect SDK*. 22. Feb. 2012. ISBN: 978-1-4302-4105-8.

[34] Microsoft. *Kinect for Windows Sensor Components and Specifications*. 23. Nov. 2015. URL: https://msdn.microsoft.com/en-us/library/jj131033.aspx (besucht am 12. 10. 2017).

[35] Microsoft. *Kinect for Windows Programming Guide*. 2017. URL: https://msdn.microsoft.com/en-us/library/dn782037.aspx (besucht am 01. 11. 2017).

[36] The Python Package Index (PyPI). *PyKinect Module for interacting with the Kinect SDK*. 6. März 2012. URL: https://pypi.org/project/pykinect/ (besucht am 20. 02. 2018).

[37] glowscript. *Using VPython to create 3D animations*. 2017. URL: http://www.glowscript.org/docs/VPythonDocs/index.html (besucht am 04. 02. 2018).

[38] Microsoft. *Skeletal Tracking*. 2017. URL: https://msdn.microsoft.com/en-us/library/hh973074.aspx (besucht am 02. 11. 2017).

[39] ALDEBARAN. *ALLandMarkDetection*. 2017. URL: http://doc.aldebaran.com/2-1/naoqi/vision/allandmarkdetection.html (besucht am 04. 05. 2018).

[40] ALDEBARAN. *Recognizing objects. Teaching NAO to recognize objects*. 2017. URL: http://doc.aldebaran.com/1-14/software/choregraphe/tutos/recognize_objects.html (besucht am 04. 05. 2018).

[41] Possibly Wrong. *Kinect skeleton tracking with Visual Python*. 4. Nov. 2012. URL: https://possiblywrong.wordpress.com/2012/11/04/kinect-skeleton-tracking-with-visual-python/ (besucht am 08. 01. 2018).

[42] ALDEBARAN. *H25 - Motors - V 3.3*. 2018. URL: http://doc.aldebaran.com/1-14/_images/hardware_motortype_h25.png (besucht am 25. 05. 2018).

[43] Sven Franz/ Ralph Nolte-Holube/ Frank Wallhoff. “NAFOME: NAO Follows Me – tracking, reproduction and simulation of human motion”. In: (2013). URL: https://www.ige.tu-berlin.de/fileadmin/fg176/IGE_Printreihe/TAR_2013/paper/Session-6-Event-2-Franz.pdf (besucht am 04. 04. 2018).

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 2. Oktober 2018 Kashif Haleem