

# **Evaluation der Nao Plattform für den Einsatz maschineller Lernverfahren**

## **Bachelor-Thesis**

Studiengang Medieninformatik  
der  
Hochschule der Medien Stuttgart

**David Bertram**

Erstprüfer: Prof. Dr. Johannes Maucher  
Zweitprüfer: Prof. Dr. Jens-Uwe Hahn

Bearbeitungszeitraum: 15. Juni 2011 bis 15. September 2011

Stuttgart, September 2011



## Kurzfassung

Im Rahmen dieser Bachelor Thesis wurde die Nao Roboterplattform für den Einsatz in Verbindung mit maschinellen Lernverfahren evaluiert. Anhand verschiedener Aufgabenstellungen war geplant unterschiedliche Verfahren zu testen und deren Tauglichkeit, mit Hilfe eines Simulators und eines realen Nao Roboters zu bewerten. Die Algorithmen wurden in der Programmiersprache Python entwickelt, und um genügend Rechenleistung zur Verfügung zu haben, auf einem Desktop PC durchgeführt.

Um ein Gefühl für die Möglichkeiten und Einschränkungen der Roboterplattform zu bekommen, wurden zu Beginn der Bearbeitung der Roboter, dessen Steuerung, und die entsprechenden Schnittstellen, sowie die zur Plattform gehörenden Tools relativ frei getestet. Die Erfahrungen und Ergebnisse wurden gesammelt und im Anschluss der Bearbeitung wird ein Wiki für weitere Projekte erstellt werden.

Die Aufgabenstellungen ergaben sich aus der Idee für ein Spiel, bei dem zwei identische Roboter gegeneinander antreten. Jeder Roboter hat einen bestimmten Bereich zu verteidigen und muss bestimmte Objekte finden, aufheben und in den Bereich des Gegners werfen.

Eine vollständige Umsetzung dieser Idee war, wie erwartet, nicht im Rahmen dieser Arbeit möglich, da einzelne Teile bereits erhebliche Komplexität und entsprechenden Aufwand erfordern, und zu den hier bearbeiteten Problemstellungen noch weitere, zum Beispiel optimale Laufwege und Spielstrategie, hinzukommen. Die Auswahl eines derart hoch gegriffenen Ziels war jedoch gewünscht um die Grenzen der Plattform zu erreichen und Schwierigkeiten aufzeigen zu können.

### Schlagwörter:

Nao, Roboter, Machine Learning, Roboter Simulation, Python

# Inhaltsverzeichnis

<b>Kurzfassung.....</b>	<b>3</b>
<b>Inhaltsverzeichnis.....</b>	<b>4</b>
<b>Motivation.....</b>	<b>6</b>
<b>1     Überblick .....</b>	<b>8</b>
<b>2     Ziele .....</b>	<b>9</b>
2.1    Erfahrungen und Informationen sammeln .....	9
2.2    Aufgabenstellungen .....	10
2.3    Maschinelles Lernen .....	11
<b>3     Nao Roboterplattform .....</b>	<b>12</b>
3.1    Software .....	12
3.1.1   Choregraphie.....	12
3.1.2   NAOqi SDK.....	14
3.1.3   Telepathe und NAO OS .....	15
3.2    Hardware.....	16
3.2.1   Allgemeines .....	17
3.2.2   Skelett .....	18
3.2.3   Gelenke und Motoren .....	19
3.2.4   Sensoren und Kameras.....	21
3.2.5   Schnittstellen.....	22
3.2.6   Akku.....	22
3.2.7   Interner Prozessor .....	23
3.2.8   LEDs .....	24
3.2.9   Fähigkeiten.....	25
<b>4     Werkzeuge .....</b>	<b>26</b>
4.1    Sonstige Umstände.....	26
4.2    Software .....	27
4.2.1   Netbeans IDE.....	27
4.2.2   Webots .....	27
4.2.2.1 Unterschiede zwischen realem und simuliertem Nao .....	30
4.3    Python und verwendete Libraries .....	31
4.3.1   OpenCV .....	31
4.3.2   Pygame.....	32
4.3.3   PyBrain .....	32

<b>5</b>	<b>Theoretische Grundlagen.....</b>	<b>34</b>
5.1	Lokalisierung einer Kugel.....	34
5.1.1	Bildverarbeitung .....	34
5.1.2	Kreiserkennung.....	37
5.1.3	Probleme bei der optischen Lokalisierung.....	37
5.2	Maschinelles Lernen .....	38
5.2.1	Maschinelles Lernen und Künstliche Intelligenz.....	38
5.2.2	Lernverfahren.....	39
5.2.3	Künstliche neuronale Netze .....	40
5.2.4	Reinforcement Learning .....	43
<b>6</b>	<b>Durchführung.....</b>	<b>46</b>
6.1	Planung .....	46
6.2	Prototypen & Versuche.....	47
6.2.1	Steuerung des Nao .....	47
6.2.2	Optische Lokalisierung einer Kugel .....	48
6.2.3	Lernversuche mit PyBrain .....	49
<b>7</b>	<b>Ergebnisse.....</b>	<b>52</b>
7.1	Auswertung .....	52
7.1.1	Evaluierung der Nao Plattform .....	52
7.1.2	Steuerungsmodule.....	53
7.1.3	Robotersimulation.....	53
7.1.4	Machine Learning .....	54
7.1.5	Auswertung der geplanten Teilaufgaben .....	55
7.2	Fazit und Ausblick .....	56
	<b>Anhang A: Inhaltsbezogene Anhänge .....</b>	<b>57</b>
A.1	RoboCup - Objective .....	57
A.2	Grafiken der Gelenke .....	58
A.3	Grafiken der Sensoren.....	61
	<b>Anhang B: Überblick der entwickelten Softwaremodule .....</b>	<b>63</b>
	<b>Literaturverzeichnis.....</b>	<b>66</b>
	<b>Erklärung.....</b>	<b>69</b>

## Motivation

Das Wort „Roboter“ ist ein relativ junger Begriff mit dem *stationäre oder mobile Maschinen, die nach einem bestimmten Programm festgelegte Aufgaben erfüllen*<sup>1</sup> bezeichnet werden. Abgeleitet von „robota“, dem tschechischen Wort für Arbeit/Fronarbeit/Zwangsarbeit wurde der Begriff zu Beginn des 20. Jahrhunderts durch die Schriftsteller Josef und Karel Čapek zunächst in der Science-Fiction Literatur für gezüchtete menschenähnliche künstliche Arbeiter verwendet. Darüber hinaus gibt es mittlerweile einige Definitionen um Geräte zu spezifizieren, welche als Roboter bezeichnet werden können. Diese Definitionen sind nicht zu 100% identisch und werden von den verschiedenen Konsortien unterschiedlich ausgelegt. Die meisten Definitionen basieren jedoch auf ähnlichen Festlegungen und unterscheiden sich in Details und den Anforderungen, die die Geräte erfüllen müssen um als Roboter bezeichnet zu werden. Die gemeinsame Forderung der meisten Definitionen besteht darin, dass die Geräte unterschiedlichste Aufgaben durch frei programmierbare Bewegungsabläufe erfüllen können.

In den letzten Jahrzehnten wurde die für die Robotik benötigte Technik ständig weiterentwickelt und verbessert. Damit erweiterten sich die Fähigkeiten und Aufgabenstellungen welche von Robotern übernommen werden können. Mittlerweile werden Roboter für verschiedenste Aufgaben eingesetzt. In der Chirurgie führen manuell gesteuerte Roboter schwierige Operationen durch und die Forschung verfolgt das Ziel Kleinstroboter der Größe von Blutzellen zu entwickeln, die dann selbstständig im Inneren des menschlichen Körpers bestimmte Aufgaben, wie zum Beispiel die Bekämpfung von Krebszellen, erfüllen sollen. Die technische Industrie verwendet mittlerweile sehr viele Roboter, unter anderem für Montageaufgaben, da Roboter hierbei eine sehr hohe Geschwindigkeit und Präzision erreichen können, die der durchschnittlichen menschlichen Leistung bei weitem überlegen sind. Daneben gibt es viele weitere Einsatzfelder für Roboter, so können auch die Maschinen die die Menschheit bisher zum Beispiel auf dem Mond oder auf dem Mars einsetzt, zum Teil als Roboter bezeichnet werden.

Die Forderung, dass Roboter ähnlich wie in Science-Fiction Filmen nahezu menschlich agieren, kommunizieren und letztendlich sogar über ein Bewusstsein und Gefühle oder vergleichbare Eigenschaften verfügen ist mit dem heutigen Stand der Technik nicht zu verwirklichen und es bleibt zweifelhaft ob diese Eigenschaften überhaupt jemals entwickelt werden können. Es gibt Versuche diese Eigenschaften zu simulieren, so dass ein Mensch, wenn auch nicht 100%, so doch zu einem gewissen Grad, einen Roboter nicht auf Anhieb als solchen erkennt. Der japanische Professor Hiroshi Ishiguro zum Beispiel versucht Menschen möglichst perfekt nachzubilden und lässt diese Roboter die Bewegungen (Gestik, Mimik) der Menschen, denen die Roboter nachempfunden sind, ausführen. Seiner Meinung nach ist es grundsätzlich möglich einen Roboter zu entwickeln, der von einem Menschen, zumindest bei kurzen Begegnungen, nicht als solcher erkannt wird. Die Ergebnisse seiner Arbeit erfahren regelmäßig eine gewisse Aufmerksamkeit in den Broadcast Medien und im Internet lassen sich ohne großen Suchaufwand Präsentationsvideos seiner Roboter finden.

---

<sup>1</sup> [Wikipedia], Begriff „Roboter“

Mit Professor Ishiguro's Projekten ist ein großes Feld der Robotikforschung angesprochen. Neben den bereits heute in zahlreichen praktischen Anwendungen zu findenden Robotern beschäftigen sich unterschiedliche Forschungsgebiete (Psychologie, Künstliche Intelligenz, ...) mit Robotern und der Mensch-Maschine-Interaktion, die mit dem Einsatz von humanoiden<sup>2</sup> Robotern, einen ihrer Höhepunkte anstrebt. Es hat sich gezeigt, dass auch ohne perfekt simuliertes menschliches Aussehen ein gut simuliertes menschliches Verhalten immerhin den Eindruck eines lebendigen, selbständig agierenden Wesens erzeugen kann. Je weniger statisch die Aktionen des Roboters sind, umso eher wird dieser als selbstständig agierend empfunden.

Bis vor einigen Jahren gab es kaum in Serie produzierte (humanoide) Roboter zu erschwinglichen Preisen und die Entwicklung von Prototypen erfordert ein erhebliches Budget. In den letzten Jahren hat sich hier einiges getan und mittlerweile existieren Produkte die auf dem freien Markt erhältlich sind. So hat Sony im Jahr 1999 einen Roboterhund, Aibo genannt, auf den Markt gebracht der bis 2008 die Plattform für eine der verschiedenen Ligen des RoboCup<sup>3</sup> verkörperte. Mit dem Auslaufen der Produktion dieses Vierbeiners wurde als Plattform für diese Liga der Roboter „Nao“<sup>4</sup> der Firma Aldebaran ausgewählt. Da sich das Projekt „RoboCup Soccer“ das Ziel gesetzt hat eines Tages (angepeilt ist das Jahr 2050) mit einer Robotermannschaft gegen den amtierenden menschlichen Weltmeister gewinnen zu können<sup>5</sup>, ist der Wechsel zu einem auf zwei Beinen laufenden Roboter ein wichtiger Schritt für dieses Projekt.

Um im eigenen Haus Projekte mit Robotern zu ermöglichen hat die Hochschule der Medien Stuttgart zwei Exemplare dieses Roboters gekauft. Im Rahmen dieser Bachelorthesis sollen nun erste Erfahrungen gesammelt und erste Ansätze für maschinelles Lernen getestet werden um künftigen Projekten den Einstieg zu erleichtern und mögliche Aufgabenstellungen entsprechend den Fähigkeiten und Einschränkungen dieser Roboterplattform formulieren zu können.

---

<sup>2</sup> Roboter mit menschlichem Aussehen (je 2 Arme, Beine, Füße, Hände, ..., je 1 Kopf, Rumpf, ...)

<sup>3</sup> Wettbewerb, bei dem verschiedene Forschungsteams mit Fußball spielenden Robotern gegeneinander antreten

<sup>4</sup> Im Folgenden nur noch Nao genannt

<sup>5</sup> s. Anhang A.1 RoboCup – Objective

# 1 Überblick

In diesem Kapitel wird ein Überblick über das vorliegende Dokument gegeben und die Inhalte der folgenden Kapitel kurz umrissen.

In Kapitel 2 werden die zu Beginn definierten Ziele dieser Arbeit und die geplanten Lösungsansätze beschrieben.

Kapitel 3 geht auf die Roboterplattform und deren Bedienung ein. Ausserdem enthält dieses Kapitel Informationen über die verwendete Hard- und Software.

Die für die Bearbeitung dieser Arbeit eingesetzten Werkzeuge (Software, Bibliotheken, Entwicklungsumgebung, ...) werden in Kapitel 4 behandelt.

Kapitel 5 stellt die theoretischen Hintergründe zu den bearbeiteten Problemstellungen bereit und beschreibt die Ideen der, in der Fachliteratur erwähnten, Lösungsansätze. Zum einen werden Grundlagen zur Objekterkennung angesprochen, zum anderen wird ein Einblick in Methoden der künstlichen Intelligenz und des maschinellen Lernens gegeben.

Die eigentliche Bearbeitung der Teilaufgaben wird in Kapitel 6 beschrieben.

In Kapitel 7 werden die erarbeiteten Ergebnisse ausgewertet und in Bezug zu den Zielen der Arbeit gesetzt. Dieses Kapitel schließt die Arbeit mit dem Fazit und einem Ausblick ab.



## 2 Ziele

Dieses Kapitel beschreibt die Ziele, die zu Beginn der Bearbeitung definiert wurden und die am Ende für eine reflektierende Bewertung des Vorgehens dienen sollen.

### 2.1 Erfahrungen und Informationen sammeln

Da diese Arbeit die Grundlagen für zukünftige Projekte mit den beiden Naos erarbeiten soll, war es eines der Hauptziele zunächst die Roboterplattform kennenzulernen und einen Einstieg in deren Bedienung und Anwendung zu finden. Zur dieser Plattform gehört, neben dem Nao selbst, ein umfangreiches Softwarepaket das im Wesentlichen aus folgenden Komponenten besteht:

- **Choregraphie** erlaubt die visuelle Programmierung von sogenannten „Behaviours“<sup>6</sup>
- **NAOqi SDK** das Development Kit zur Einbindung der Schnittstelle zum Nao in eigene Programme
- **Telepathe** Tool mit dessen Hilfe einige Funktionen des Naos abgefragt und getestet werden können (Marker Detection<sup>7</sup>, Arbeitsspeicher<sup>8</sup>, ...)
- **NAO OS** Betriebssystem, das auf dem Roboter läuft und dessen Schnittstelle zur Außenwelt bereitstellt

Neben der mitgelieferten Software galt es auch die Hardware des Roboters selbst zu testen um mögliche Schwachstellen zu erkennen. Menschen sind normalerweise in der Lage komplexe Bewegungen auch in dynamischen Situationen gut an eine sich verändernde Umwelt anpassen zu können. Dafür setzt der Mensch unzählige Muskeln, Gelenke und Sinneswahrnehmungen wie zum Beispiel das Sehen und Tasten ein. Die Naos verfügen zwar über relativ große Bewegungsfreiheiten, die dennoch mit den Fähigkeiten eines gesunden Menschen kaum vergleichbar sind.

Einen weiteren wichtigen Teil der Plattform bildet die Schnittstelle, über die der Nao kontrolliert werden kann. Diese Schnittstelle wird über das Development Kit (NAOqi SDK) bereitgestellt und hat großen Einfluss darauf, wie viel Hintergrundwissen zur Funktionsweise des Naos nötig und wie kompliziert die Ansteuerung der Funktionen des Roboters für unerfahrene Benutzer ist.

Die Ergebnisse dieses Teils der vorliegenden Arbeit bestehen also aus einer Einschätzung der Hard- und Software sowie der Steuerungsschnittstelle der Plattform. Die gesammelten Informationen und Einschätzungen sollen später in Form eines Wikis innerhalb der Hochschule für zukünftige Projekte verfügbar gemacht werden.

---

<sup>6</sup> Definieren Verhaltensweisen die auf den Roboter übertragen werden können

<sup>7</sup> Für den Nao gibt es von der Firma Aldebaran eigene Marker und Methoden zu deren Erkennung entwickelt. Diese Marker können auf Objekte geklebt werden und dienen dazu diese Objekte mit Hilfe eines Kamerabildes zu identifizieren.

<sup>8</sup> Im Arbeitsspeicher des Naos können zur Laufzeit Daten und Informationen abgelegt werden. Eine Schnittstelle ermöglicht den Zugriff.

## 2.2 Aufgabenstellungen

Um die Plattform realistisch einschätzen zu können wurden Teilaufgaben definiert, die die Fähigkeiten des Naos fordern und den Roboter an dessen Grenzen bringen. Hierbei gab es kaum Vorgaben und am Ende tatsächlich eine funktionierende Lösung zu finden war nicht unbedingt nötig, im Gegenteil waren Aufgabenstellungen die der Nao nicht, oder nur mit großen Schwierigkeiten lösen kann wünschenswert.

Es wurde ein Szenario entwickelt, das die Teilaufgaben enthält und sich mit entsprechenden Lösungen zu einem Spiel, bei dem 2 Naos gegeneinander antreten, ausbauen ließe. Das Spielfeld besteht aus einer Fläche mit festgelegten Bereichen. Die Grundidee ist, dass jeder Mitspieler einen der Bereiche „verteidigen“ muss. Das bedeutet dass im eigenen Bereich keine Bälle liegen sollen. Gleichzeitig muss jeder Roboter versuchen den Gegner „anzugreifen“ indem versucht wird, Bälle in dessen Bereich zu befördern.

Es wurden 3 Teilaufgaben auf folgende Weise definiert:

- **Lokalisieren:**

Um ein Objekt greifen zu können ist es notwendig Informationen über dessen Position zu haben. Zur Lösung dieses Problems kommen hauptsächlich die beiden Kameras des Naos in Frage. Das bedeutet zur Lösung dieses Problems ist es mindestens notwendig, ein Kamerabild vom Nao abrufen zu können und dieses zu untersuchen. Mit Hilfe der Ergebnisse dieser Teilaufgabe soll der Nao in der Lage sein, sich selbst in eine zum Greifen geeignete Position zu bringen.

- **Greifen:**

Damit ein Objekt aus dem eigenen Bereich entfernt werden kann, muss der Roboter die Möglichkeit haben mit diesem zu interagieren. Denkbar sind hier Methoden wie Stoßen, Kicken, Schieben und eben das Objekt zunächst zu greifen und unter Kontrolle zu bringen. Um eine anspruchsvolle Aufgabe zu haben und dabei die Fähigkeiten des Roboters auszuwerten und an dessen Grenzen bringen zu können, wurde die Option des Greifens gewählt.

- **Werfen:**

Im Falle, dass ein Spielobjekt erfolgreich gegriffen wurde, benötigt der Roboter anschließend eine Methode um das Objekt in den Bereich des Gegners bringen zu können. Da die Methode des Greifens gewählt wurde, erfordert diese Aufgabe entweder die Fähigkeit, das Objekt in den Bereich des Gegners zu tragen und dort fallenzulassen, oder, falls das Betreten des Bereichs des Gegners nicht erlaubt ist, die Fähigkeit das Objekt zielgerichtet zu werfen.

Die bearbeiteten Lösungsansätze zu den jeweiligen Teilaufgaben und deren Auswertung finden sich in Kapitel 6 und 7.

## 2.3 Maschinelles Lernen

Eine der wenigen Bedingungen, die an die Bearbeitung der Teilaufgaben gestellt waren war, dass die Lösungsansätze möglichst wenige statische Teile enthalten und stattdessen versuchen sollen, durch den Einsatz von Methoden aus dem Bereich des maschinellen Lernens und der Künstlichen Intelligenz möglichst flexible und universelle Lösungen anzupeilen.

Da es sehr aufwändig sein kann solche Algorithmen von Grund auf selbst zu programmieren, wurde die Entscheidung getroffen die Methoden des existierenden Frameworks „PyBrain“ zu verwenden.

Damit ist also die Verwendung von PyBrain in Kombination mit dem Nao ein weiteres Ziel dieser Arbeit gewesen. Da PyBrain umfangreich ist und die Anwendung des Frameworks in aller Regel einiges an Wissen über die jeweilige Methode erfordert, ist dieses Ziel mit der Einarbeitung in dessen Funktionsweise verbunden. Im Vordergrund stand dabei das Ausarbeiten einer Möglichkeit die Methoden von PyBrain mit der Schnittstelle des Roboters zusammenzubringen und deren Zusammenspiel zumindest in Teilen zu analysieren ohne dabei unbedingt eine optimal funktionierende Lösung für Teilaufgaben finden zu müssen.

## 3 Nao Roboterplattform

Dieses Kapitel enthält Beschreibungen der Hardware des Roboters und der in der Plattform enthaltenen Software, sowie die entsprechenden Erfahrungen, die bei den praktischen Versuchen dieser Arbeit gesammelt werden konnten.

### 3.1 Software

An dieser Stelle wird die zur Plattform gehörende Software beschrieben und deren Einsatzmöglichkeiten aufgezeigt. Die Software kann von der Community-Seite von Aldebaran bezogen werden, dafür ist allerdings ein Zugang zu diesem Portal [Aldebaran] erforderlich. Diesen bekommt man in aller Regel automatisch mit dem Erwerb eines Naos. Auf Internetseite stehen auch die (regelmäßig erweiterte) Dokumentation der Plattform inklusive Tutorials und ein Forum für Diskussionen und Fragen bereit.

#### 3.1.1 Choregraphie

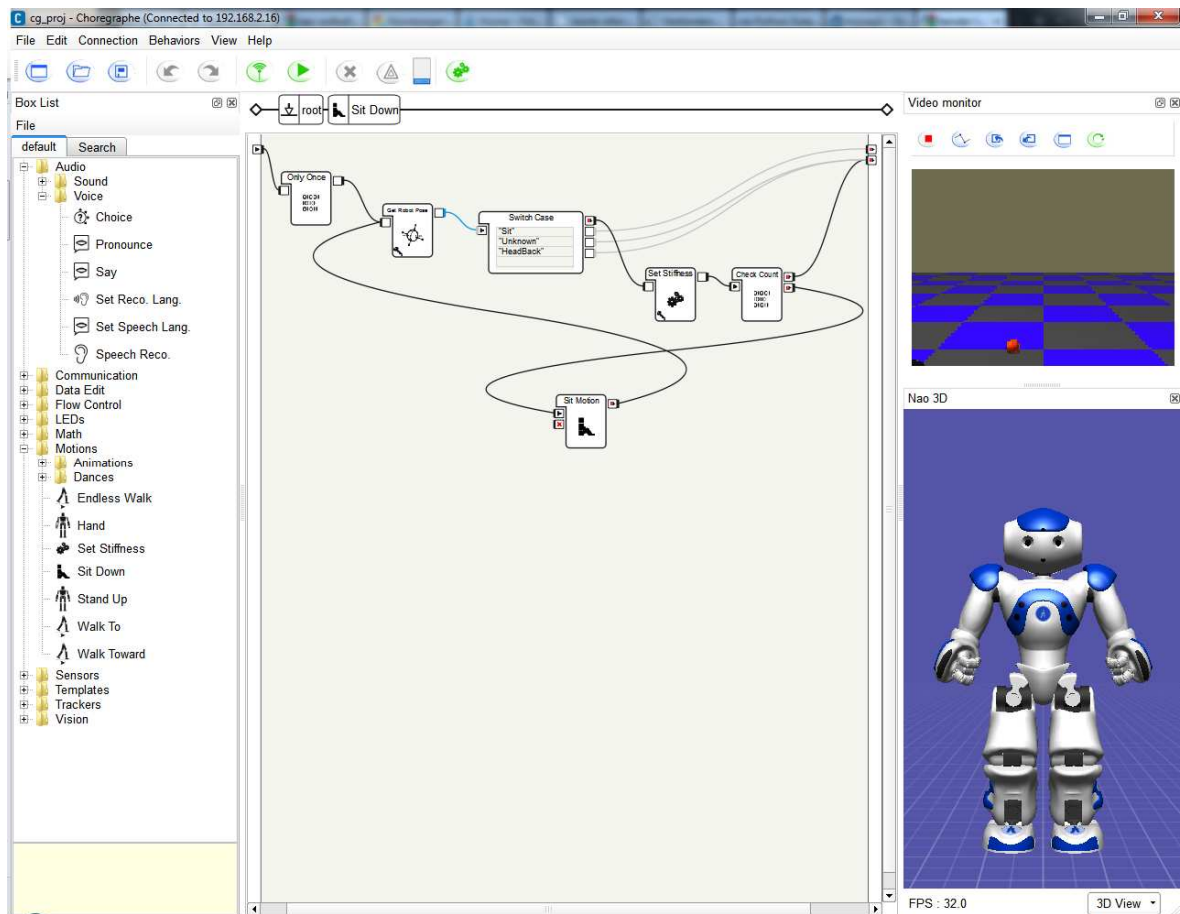
Dieses Programm wurde entwickelt um auch Benutzern ohne tiefergehende Programmierkenntnisse eine intuitive und einfache Möglichkeit bereitzustellen, den Roboter mit individuellen Verhaltensweisen programmieren zu können. Dafür wird eine grafische Oberfläche verwendet, auf der einzelne Bausteine, welche wiederum einzelne oder aus anderen zusammengesetzte Aktionen darstellen, miteinander zu einem Programm verknüpft werden können. Dies gestaltet sich in der Praxis sehr einfach und sollte es jedem interessierten Benutzer erlauben innerhalb kurzer Zeit eigene Verhaltensweisen für den Nao zu erstellen.

Trotz der einfachen Bedienung ist dieses Programm sehr mächtig und erlaubt tiefe Eingriffe in das Framework und die dem Roboter zur Verfügung stehenden Methoden. So ist es zum Beispiel möglich komplette Bewegungsabläufe, die die Komplexität eines Tanzes<sup>9</sup> haben, aufzuzeichnen und direkt als Aktion zu verwenden. Ausserdem ist der Zugriff auf sämtliche Sensoren des Roboters möglich, was die Erstellung dynamischer Verhaltensweisen erlaubt. Je nach Komplexität erreicht man dabei allerdings früher oder später einen Punkt an dem die Übersichtlichkeit leidet und Benutzer mit gewisser Programmiererfahrung die Möglichkeit zur Nutzung von eigenem Code suchen um sich in den gewohnten Kontrollstrukturen der („klassischen“) Programmierung bewegen zu können. Das bedeutet nicht, dass Choregraphie keine Möglichkeiten zur Verwendung von Schleifen und Bedingungen bereitstellt. Diese Bausteine müssen allerdings entweder aufwändig hierarchisch zu zusammengesetzten Aktionen verschachtelt werden oder führen schnell zu eben erwähnten Übersichtsproblemen. Sogar eigenen Code kann man in Choregraphie einbinden und daraus Aktionen und Module für den Nao generieren. Dennoch bleibt für die Programmierung des Roboters das im Folgenden beschriebene NAOqi SDK die erste Wahl.

---

<sup>9</sup> Mittlerweile sind mehrere fertige Tänze für den Nao verfügbar, ein Tai-Chi Tanz ist in der aktuellen Version von Choregraphie direkt und weitere auf Anfrage verfügbar

Wie auf diesem Bild erkennbar, bietet Choregraphe die Möglichkeit zur Laufzeit das aktuelle Kamerabild und die aktuellen Positionen aller Komponenten des Roboters anhand eines 3D-Modells darzustellen sowie die visuelle Programmierung.



### 3.1.2 NAOqi SDK

Hinter dem Namen NAOqi SDK verbirgt sich das Development Kit für die Nao Plattform. Neben Schnittstellen für weitere Programmiersprachen wie C++ beinhaltet das SDK die für die Einbindung eines Naos in eigene Python Programme nötigen Libraries.

Da im Rahmen dieser Arbeit nahezu ausschließlich mit Python gearbeitet wurde, beschränken sich die durchgeführten Versuche und deren Ergebnisse auf die Python betreffenden Teile des SDK. Die Schnittstellen in anderen Programmiersprachen sollten jedoch auf die gleiche Weise funktionieren und damit im Großen und Ganzen die gleichen Fähigkeiten und Einschränkungen wie die Schnittstelle zu Python zu besitzen.

Das SDK besteht zunächst aus einem Zip-Archiv, das im Unterverzeichnis „lib“ die Python Libraries enthält. Hat man dieses Verzeichnis für die Python Umgebung verfügbar gemacht (zum Beispiel durch Import des Verzeichnisses in ein Netbeans<sup>10</sup> Projekt) erhält man mittels:

```
from naoqi import ALProxy
```

Zugriff auf die Schnittstelle des Roboters.

Diese Schnittstelle ist so konstruiert, dass man spezielle Proxy-Objekte (ALProxy) verwendet die jeweils eine bestimmte Art von Funktionalität über eine entsprechende Menge an Methoden bereitstellen.

Einer der wichtigsten Proxy-Typen ist der sogenannte „ALMotion Proxy“. Über diesen Proxy können die einzelnen Gelenke des Roboters gesteuert, die entsprechenden Sensoren ausgelesen oder sogar komplexere Bewegungen wie zum Beispiel „Laufen“ angesteuert werden.

Im Rahmen dieser Arbeit war ausserdem der Proxy „ALVideoDevice“ ein wichtiger Baustein um Bilder der Kamera abzufragen und diese anschließend weiter zu verarbeiten.

Ein weiterer Proxy ist zum Beispiel der „ALBehaviourManager“ mit dessen Hilfe, mittels Choreographie vordefinierte, „Behaviours“ gestartet werden können. Dies ist zwar nicht unbedingt nötig, stellt aber eine bequeme Möglichkeit zur Verfügung um Verhaltensweisen in eigenen Code einzubinden, die bereits verfügbar sind und zum Beispiel aus früheren Projekten oder von dritten Entwicklern stammen.

Es existiert eine ganze Reihe weiterer Proxies, deren Verwendung in der Regel mit Hilfe der verfügbaren Dokumentation<sup>11</sup> relativ einfach nachvollzogen werden kann. Eine Auflistung der verfügbaren Proxies und deren Methoden, kann für jeden Nao durch Zugriff auf dessen Webinterface<sup>12</sup> angezeigt werden.

Ausser dem Bereitstellen dieser Schnittstellen wird das SDK auch benötigt um den Nao Controller für den später beschriebenen Roboter Simulator „Webots“ zu erstellen. Genaueres hierzu findet sich in Kapitel 4.2.2.

---

<sup>10</sup> Programmierumgebung, vergleichbar mit „Eclipse“ oder „MS Visual Studio“

<sup>11</sup> [http://users.aldebaran-robotics.com/docs/site\\_en/reddoc/framework/framework.html](http://users.aldebaran-robotics.com/docs/site_en/reddoc/framework/framework.html)

<sup>12</sup> In einem beliebigen Browser zu Port 9559 unter der IP-Adresse des betreffenden Nao navigieren

### 3.1.3 Telepathe und NAO OS

Ein weiteres Tool der Plattform ist „Telepathe“, mit dem im Rahmen dieser Arbeit allerdings nicht gearbeitet wurde, und letztendlich das Betriebssystem, welches auf dem Roboter selbst läuft und dessen Schnittstellen zur Laufzeit bereitstellt.

„Telepathe“ bietet die Möglichkeit sich sämtliche interne Werte des Roboters zur Laufzeit anzeigen zu lassen und diese in gewissem Maße zu analysieren, sowie Zugriff auf Funktionen wie zum Beispiel eine mitgelieferte Markererkennung.

„NAO OS“ bezeichnet das Betriebssystem des Roboters. Dieses kann bei Bedarf aktualisiert werden, Aldebaran stellt regelmäßige Updates bereit und versucht damit den Funktionsumfang des Roboters zu erweitern. Man kann dieses Betriebssystem theoretisch selbst modifizieren und eigene Module sowie Funktionen implementieren. Dies erfordert allerdings tieferes Einarbeiten in das System und wurde ebenfalls im Rahmen dieser Arbeit nicht eingehender untersucht.

Das Betriebssystem des Roboters stellt, sobald eine Netzwerkverbindung<sup>13</sup> besteht, ein Webinterface zur Verfügung, mit dessen Hilfe der Nao konfiguriert<sup>14</sup> werden kann. Eine erste Konfiguration nimmt man in der Regel mit Hilfe einer kabelgebundenen Verbindung vor, und konfiguriert dann den Zugang zu einem WLAN.

---

<sup>13</sup> Siehe Kapitel 3.2.5

<sup>14</sup> Unter anderem Lautstärke, Bezeichnung, Netzwerkeinstellungen, ...

## 3.2 Hardware

Die folgenden Unterkapitel beschreiben den Roboter selbst und dessen Fähigkeiten mit seiner Umwelt zu interagieren. Die wichtigste Rolle spielen hierbei bilden die Gelenke und Motoren sowie die Sensoren des Roboters. Von diesen hängt die Anzahl der Freiheitsgrade<sup>15</sup>, die die Bewegungsmöglichkeiten bestimmen, und die Möglichkeit Informationen über den eigenen Zustand und die unmittelbare Umgebung zu erhalten, ab.

Die im Folgenden verwendeten Abbildungen und Daten stammen von der Nao Community-Seite, dort unter dem Punkt: Software → Documentation → Hardware<sup>16</sup> zu finden.

---

<sup>15</sup> „Richtungen“ in die ein Gelenk bewegt werden kann, eine einfache Schwingtüre zum Beispiel hätte demnach einen Freiheitsgrad von 1

<sup>16</sup> [http://users.aldebaran-robotics.com/docs/site\\_en/reddoc/hardware/hardware.html](http://users.aldebaran-robotics.com/docs/site_en/reddoc/hardware/hardware.html)

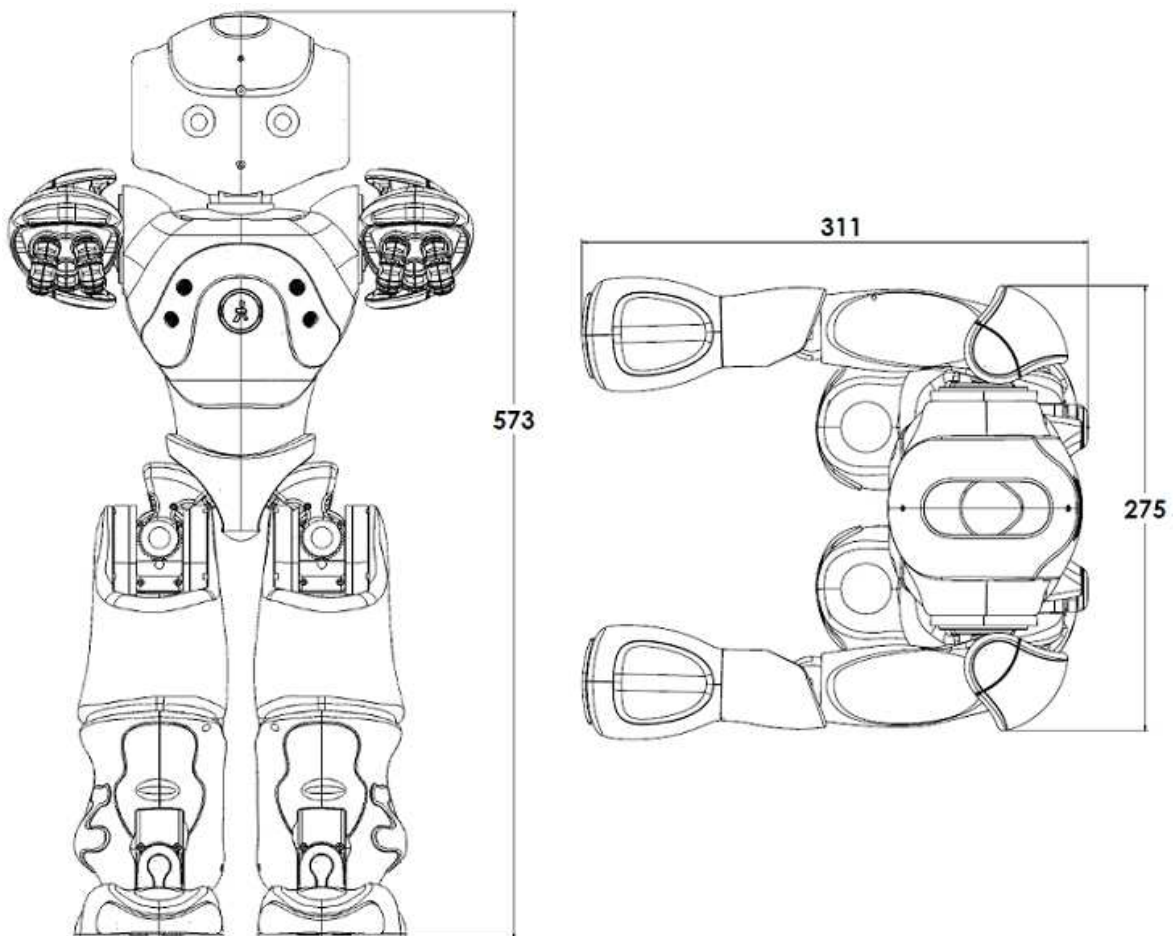


### 3.2.1 Allgemeines

Der Nao Roboter ist knapp 60 cm groß und hat ein Gewicht von knapp viereinhalb Kilogramm. Damit kann der Roboter ohne große Mühe von einer Person alleine zum Beispiel zu Präsentationen mitgeführt werden.

Um Demonstrationen mit dem Roboter durchführen zu können reicht es im allerbesten Fall aus, den Roboter mit geladenem Akku mitzunehmen. Das erfordert allerdings, dass sämtliches Verhalten vorprogrammiert und auf dem Roboter verfügbar gemacht wurde. Um volle Kontrolle über den Roboter zu haben und dessen Programmierung jederzeit anpassen oder austauschen zu können empfiehlt es sich zusätzlich ein Notebook, und zur Verbindung zwischen Roboter und Notebook, der Einfachheit halber einen WLAN-Accesspoint oder -Router mitzuführen. Diese Geräte stehen an der HdM gemeinsam mit dem Nao zur Verfügung.

Insgesamt ist der Nao relativ stabil gebaut und so konstruiert, dass er beim Stolpern und Hinfallen keine Schäden nimmt. Dennoch bleibt der Roboter ein empfindliches Gerät und erfordert entsprechenden Umgang. So sollte der Nao nur eingeschaltet werden, wenn er sich auf dem Boden befindet oder so gesichert ist, dass er nicht fallen kann.



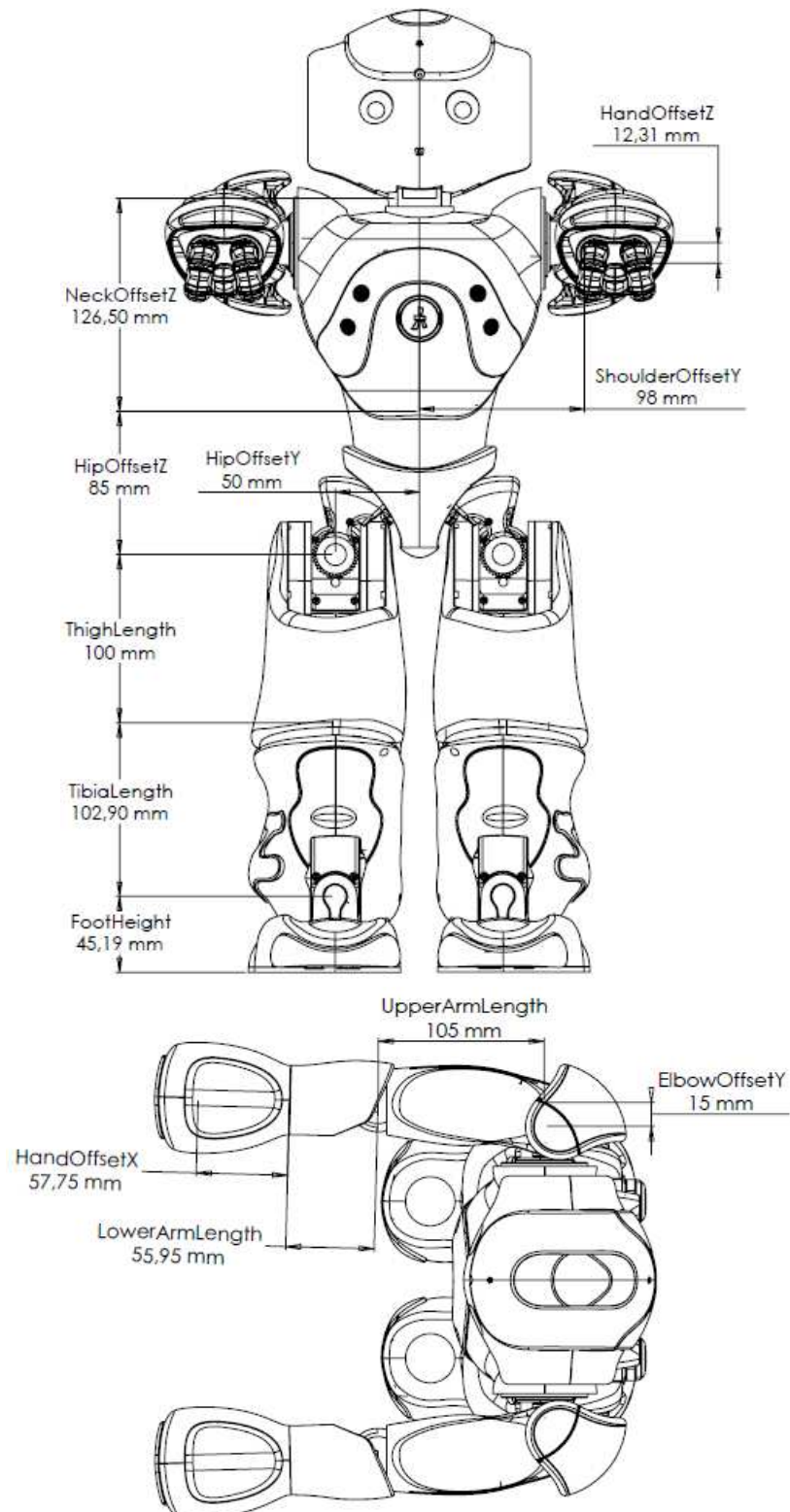
Höhe: ~ 60 cm

Gewicht: ~ 4,5 kg

<sup>17</sup> Angaben in Millimetern

### 3.2.2 Skelett

Der Roboter besteht technisch gesehen aus Verbindungsstücken, die mit Gelenken verbunden sind und deren Auslenkung mit Hilfe von Motoren gesteuert werden kann. An dieser Stelle werden die Verbindungsstücke dargestellt, im folgenden Kapitel findet sich die Beschreibung der Gelenke, deren Daten zunächst weitaus interessanter sind, da diese bei der Programmierung von Bewegungen direkt benötigt werden.



### 3.2.3 Gelenke und Motoren

Der Nao verfügt über insgesamt 25 Gelenke, von denen jedes einzeln angesteuert werden kann. Die einzelnen Gelenke haben unterschiedliche maximale Auslenkungen, so kann sich der Kopf zum Beispiel deutlich weiter zur Seite, als nach oben und unten, bewegen. Dabei werden Schrittmotoren eingesetzt, deren genaue Daten in der Online-Dokumentation<sup>18</sup> zu finden sind. Interessanter als diese Daten sind allerdings die subjektiven Eindrücke, die die Arbeit mit dem Roboter hinterlässt, da erst diese eine realistische Einschätzung der praktischen Fähigkeiten erlauben.

Um Beschädigungen der Motoren zu vermeiden, überprüft der Roboter deren Temperaturen selbstständig und regelt die maximale Kraft, mit der die Motoren versuchen die gewünschte Stellung einzunehmen. Bei kurzen Demonstrationen bleiben die Temperaturen in der Regel unterhalb der kritischen Werte und man kann bei Bedarf die volle Kraft der Motoren nutzen. Hierfür gibt es zu jedem Gelenk den Parameter „Stiffness“ der angibt, wie viel Prozent der maximal möglichen Kraft verwendet wird. Bei längeren Versuchen oder Versuchsreihen kann es durchaus vorkommen dass einzelnen Motoren Überhitzung droht, der Roboter meldet dies dann zunächst akustisch und regelt die Kraft des jeweiligen Motors automatisch herab falls die Temperatur weiter ansteigt.

Besonders die Motoren in den Schulter- und die Kniegelenken neigen dazu recht schnell heiß zu werden. Der Grund dafür liegt im hohen Gewicht, das besonders an den Motoren dieser beiden Gelenke „zieht“. Bei seitlich ausgestrecktem Arm zieht das gesamte Gewicht (inklusive Hebelwirkung) am Motor im Schultergelenk und bereits nach einigen Minuten fängt ein zu Beginn noch kalter (Zimmertemperatur) Motor an heiß zu werden. Die Motoren im Kniegelenk werden häufig stark belastet, da der Roboter beim Laufen ein gutes Stück in die Knie geht und diese Motoren dabei einen Großteil des Gewichts des gesamten Roboters tragen müssen. Ähnlich wie beim Menschen erfordert dies also an den Beinen besonders starke Muskeln bzw. Motoren um dieses Gewicht auch bei nicht durchgestrecktem Kniegelenk tragen zu können. Um diese Belastung zu minimieren sollte der Roboter also nachdem er gelaufen ist, eine Position einnehmen in der er mit durchgestreckten Beinen steht. Bei überhitzenden Motoren im Kniegelenk ist Vorsicht geboten, denn wird deren Kraft herab geregelt knicken dem Roboter im wahrsten Sinne des Wortes die Beine weg und er fällt. Ausserdem steht der Roboter mit durchgestrecktem Kniegelenk besonders stabil, da er in dieser Position aktiv versucht ein Umkippen zu verhindern. Drückt man zum Beispiel (leicht) von vorne gegen den Kopf des Roboters, so gleicht dieser diese Kraft mit Hilfe der Motoren in seinen Füßen aus und kann sich bis zu einem gewissen Grad stabilisieren.

Im Folgenden sind Abbildungen der Gelenke und eine Tabelle mit deren maximalen Auslenkungen aufgeführt. Die Ansteuerung der Motoren geschieht über einen Wert zwischen dem Minimum und dem Maximum der mit „Range (radian)“ bezeichneten Spalte.

---

<sup>18</sup> [http://users.aldebaran-robotics.com/docs/site\\_en/reddoc/hardware/motor\\_type.html](http://users.aldebaran-robotics.com/docs/site_en/reddoc/hardware/motor_type.html)

Zur Veranschaulichung welcher Gelenkname welches Gelenk bezeichnet finden sich im Anhang A.2 die entsprechenden Grafiken, welche ebenfalls auf der Community-Seite gefunden werden können.

Joint name	Motion	Range (degrees)	Range (radian)
<b>HeadYaw</b>	Head joint twist (Z)	-119.5 to 119.5	-2.0857 to 2.0857
<b>HeadPitch</b>	Head joint front and back (Y)	-38.5 to 29.5	-0.6720 to 0.5149
<b>RShoulderPitch</b>	Right shoulder joint front and back (Y)	-119.5 to 119.5	-2.0857 to 2.0857
<b>RShoulderRoll</b>	Right shoulder joint right and left (Z)	-76 to 18	-1.3265 to 0.3142
<b>RElbowYaw</b>	Right shoulder joint twist (X)	-119.5 to 119.5	-2.0857 to 2.0857
<b>RElbowRoll</b>	Right elbow joint (Z)	2 to 88.5	0.0349 to 1.5446
<b>RWristYaw</b>	Right wrist joint (X)	-104.5 to 104.5	-1.8238 to 1.8238
<b>RHand</b>	Right hand	Open and Close	Open and Close
<b>LShoulderPitch</b>	Left shoulder joint front and back (Y)	-119.5 to 119.5	-2.0857 to 2.0857
<b>LShoulderRoll</b>	Left shoulder joint right and left (Z)	-18 to 76	-0.3142 to 1.3265
<b>LElbowYaw</b>	Left shoulder joint twist (X)	-119.5 to 119.5	-2.0857 to 2.0857
<b>LElbowRoll</b>	Left elbow joint (Z)	-88.5 to -2	1.5446 to 0.0349
<b>LWristYaw</b>	Left wrist joint (X)	-104.5 to 104.5	-1.8238 to 1.8238
<b>LHand</b>	Left hand	Open and Close	Open and Close
<b>LHipYawPitch*</b>	Left hip joint twist (Y-Z 45°)	-65.62 to 42.44	-1.145303 to 0.740810
<b>RHipYawPitch*</b>	Right hip joint twist (Y-Z 45°)	-65.62 to 42.44	-1.145303 to 0.740810
<b>LLHipRoll</b>	Left hip joint right and left (X)	-21.74 to 45.29	-0.379472 to 0.790477
<b>LLHipPitch</b>	Left hip joint front and back (Y)	-101.63 to 27.73	-1.773912 to 0.484090
<b>LKneePitch</b>	Left knee joint (Y)	-5.29 to 121.04	-0.092346 to 2.112528
<b>LAnklePitch</b>	Left ankle joint front and back (Y)	-68.15 to 52.86	-1.189516 to 0.922747
<b>LAnkleRoll</b>	Left ankle joint right and left (X)	-44.06 to 22.79	-0.769001 to 0.397880
<b>RLHipRoll</b>	Right hip joint right and left (X)	-42.30 to 23.76	-0.738321 to 0.414754
<b>RLHipPitch</b>	Right hip joint front and back (Y)	-101.54 to 27.82	-1.772308 to 0.485624
<b>RKneePitch</b>	Right knee joint (Y)	-5.90 to 121.47	-0.103083 to 2.120198
<b>RAnklePitch</b>	Right ankle joint front and back (Y)	-67.97 to 53.40	-1.186448 to 0.932056
<b>RAnkleRoll</b>	Right ankle joint right and left (X)	-22.27 to 45.03	-0.388676 to 0.785875

\*LHipYawPitch und RHipYawPitch sind zwei Bezeichnungen für ein und denselben Motor und können nicht unabhängig voneinander gesteuert werden.

### 3.2.4 Sensoren und Kameras<sup>19</sup>

Der Nao verfügt neben Kameras unter anderem über berührungsempfindliche Touch-Sensoren an den Handaußenseiten und auf dem Kopf. Diese Sensoren erlauben es noch zur Laufzeit Kommandos an den Nao zu geben, so kann man beispielsweise ein Verhalten definieren, bei dem der Roboter zunächst sitzt und aufsteht, sobald einer der 3 Touch-Sensoren auf dem Kopf ausgelöst wird.

Ausserdem können 2 Ultraschallsensoren, jeweils leicht nach vorne zur Seite gerichtet, verwendet werden um Hindernisse in einer Entfernung von 0 bis 70 cm zu erkennen. Dabei können allerdings nur Entfernungen größer als 15 cm genau angegeben werden, befindet sich das Hindernis näher, kann nur noch dessen Anwesenheit erkannt werden.

Mit insgesamt 4 Mikrofonen ist der Nao in der Lage Geräuschquellen mit Hilfe einer eingebauten Funktion zu lokalisieren. Die Mikrophone sind rund um den Kopf angebracht und nach vorne und hinten, sowie links und rechts gerichtet.

Da der Nao einen Beschleunigungssensor besitzt, kann festgestellt werden, ob und in welche Richtung der Roboter beschleunigt. Dadurch kann im Falle eines Sturzes versucht werden, die Extremitäten so auszurichten, dass bei Bodenkontakt möglichst wenig Kraft auf empfindliche Teile wie die Schrittmotoren trifft. Genauere Daten zum Beschleunigungssensor finden sich in der Online-Dokumentation und diesem Link<sup>20</sup>

Um in der Lage zu sein, die aktuellen Ausrichtungen der Gelenke auszulesen, besitzt jedes Gelenk einen Sensor. Dieser liefert entsprechend der aktuellen Auslenkung einen Wert, der dem entspricht den man als Kommando senden müsste um das Gelenk in die aktuelle Position zu bringen.

In den Fußsohlen des Roboters befinden sich druckempfindliche Sensoren, mit deren Hilfe Informationen über die Beschaffenheit des Bodens auf dem der Nao steht gewonnen werden können. Es wäre denkbar mit Hilfe dieser Sensoren an einer Kante entlang zu laufen und zu erkennen wann der Fuß nicht mehr komplett auf festem Boden steht.

Darüber hinaus besitzt der Nao auch zwei fest am Kopf montierte Kameras die 30 Bilder pro Sekunde mit einer Auflösung von 640x480 Pixeln liefern können. Die Qualität der Bilder ist ausreichend, das Bildrauschen fiel im subjektiven Vergleich zu Bildern von Digitalkameras der unteren bis mittleren Preisklasse nicht über- oder unterdurchschnittlich auf. Die Kameras sind (leider) übereinander angeordnet und sind nicht parallel ausgerichtet. Das erlaubt zwar den fußballspielenden Robotern des RoboCup Projekts auch Bälle, die direkt vor deren Füßen liegen ohne große Kopfbewegungen zu erkennen. Es verhindert allerdings gleichzeitig den Einsatz von stereoskopischen Methoden<sup>21</sup> um Tiefeninformationen zu ermitteln.

---

<sup>19</sup> Grafiken finden sich in Anhang A.3

<sup>20</sup> [http://users.aldebaran-robotics.com/docs/site\\_en/reddoc/hardware/inertial\\_unit.html](http://users.aldebaran-robotics.com/docs/site_en/reddoc/hardware/inertial_unit.html)

<sup>21</sup> Damit ist der Einsatz von Stereokameras, vergleichbar zur menschlichen Wahrnehmung mittels 2 Augen, gemeint. Übereinander angeordnete Kameras ließen sich durch Anpassung der Algorithmen realisieren, die aufeinander abgestimmte Ausrichtung der Kameras ist allerdings zwingend nötig und beim Nao nicht möglich. Detaillierte Informationen dazu finden sich im Buch „Learning OpenCV“ von Gary Bradski und Adrian Kaehler (2008). Mögliche Ergebnisse zeigt zum Beispiel das Video „OpenCV Stereo Vision Software“ unter: <http://www.youtube.com/watch?v=WHBMXzBPZS4>

### 3.2.5 Schnittstellen

Verbindungen zum Nao kann man auf unterschiedliche Arten herstellen. Interessant und praktikabel sind vor allem die Funkverbindungen WLAN und Bluetooth.

- WLAN

Mit Hilfe einer herkömmlichen WLAN Netzwerkkarte kann sich der Nao zu WLANs verbinden und bei bestehender Verbindung komplett kabellos gesteuert werden.

- Bluetooth

Um eine Möglichkeit bereitzustellen, mit deren Hilfe mehrere Naos auch ohne vorhandenes WLAN miteinander Daten austauschen können, kann der Nao ausserdem mittels Bluetooth-Verbindung kommunizieren.

- LAN

Schließlich gibt es noch einen gewöhnlichen Netzwerkanschluss, mit dem der Nao Zugriff zu kabelgebundenen Netzwerken bekommt. Wie bereits anfangs erwähnt ist dies allerdings nur selten nötig und in aller Regel konfiguriert man den Roboter nur einmal über diesen Anschluss und verwendet später die bequemere kabellose Variante.

### 3.2.6 Akku

Als Akku kommt eine Lithium-Ionen Batterie mit 2 Ah (2000 mAh) Kapazität zum Einsatz und liefert nominal 21.6 Volt. Die Batterie wiegt ca 350g und benötigt ca. 2-3 Stunden um aufgeladen zu werden. Mit einer vollen Batterie, kann der Roboter je nach Art und Häufigkeit der Aktivitäten bis zu eine Stunde autonom agieren. Diese Zeit kann in der Praxis allerdings deutlich geringer oder auch höher ausfallen, da vor allem Bewegungen viel Energie kosten und damit erheblichen Einfluss auf die Laufzeit haben.

Sobald der Ladestand des Akkus eine Grenze unterschreitet, meldet sich der Nao akustisch mit dem Hinweis „battery low“ und schaltet sich schließlich sogar ganz ab, bevor der Akku komplett entladen wird, da dieser ansonsten Schaden nehmen könnte. Auch hier ist ähnliche Vorsicht geboten wie wenn der Roboter heiße Motoren in den Kniegelenken meldet, da die Gelenke beim Ausschalten jeglichen Widerstand verlieren und nicht mehr in der Lage sind, den Roboter in einer nicht zu 100% ausbalancierten Stellung zu halten. In der Folge fällt der Roboter zusammen und könnte dabei Schaden nehmen.

### 3.2.7 Interner Prozessor

Der Nao besitzt einen Prozessor vom Typ „x86 AMD Geode 500 MHz“. Diese CPU verbraucht zwar wenig Strom und produziert im Vergleich zu herkömmlichen Desktop CPUs weniger Abwärme, bietet dafür allerdings auch weitaus weniger Rechenleistung als diese. Dennoch kann man auch aufwändigere Berechnungen in selbst erstellten Modulen für das NAO Betriebssystem kompilieren und diese bei Bedarf auf dem internen Prozessor ausführen.

In dieser Arbeit wurden sämtliche Berechnungen auf einem externen Desktop PC ausgeführt und die Funktionen des Roboters über dessen Netzwerkschnittstelle angesteuert. Damit steht deutlich mehr Rechenleistung zur Verfügung und man reduziert ausser dem Stromverbrauch auch die aus dem Robotergehäuse abzuführende Wärme. Falls der interne Prozessor heiß werden sollte, meldet sich der Nao automatisch mit der akustischen Meldung „head processor hot“. Um die Hardware nicht über Gebühr zu strapazieren sollte in diesem Fall dafür gesorgt werden, dass die Temperatur wieder sinkt. Am Hinterkopf des Nao ist ein Lüfter angebracht, der die Abwärme des internen Prozessors aus dem Gehäuse bläst, ein blockierter Luftstrom kann die Ursache für einen heiß gewordenen Prozessor sein. Grundsätzlich sollte es ausreichen den Nao für 15 bis 30 Minuten auszuschalten um die Temperatur wieder zu senken.

Den Ursachen für einen überhitzenden Prozessor sollte auf jeden Fall nachgegangen werden, unter Umständen kann übermäßiges Beanspruchen des Prozessors bei relativ hoher Raumtemperatur die Temperatur in die Höhe schnellen lassen. Zum Beispiel wird, bei jedem Abrufen eines Videobildes, dieses mit Hilfe des Prozessors konvertiert und anschließend ausgeliefert. In hoher Frequenz durchgeführt erzeugt dies eine vergleichsweise hohe Rechenlast und verursacht damit ein Steigen der Temperatur des Prozessors.

### 3.2.8 LEDs

Um sich auf visuellem Weg mitteilen zu können, sei es einem anderen Nao oder einem Menschen, verfügt der Nao über insgesamt 19 RGB-LEDs<sup>22</sup>, davon 8 in jedem Auge sowie in Brust und bei den Füßen jeweils eine. Besonders die LEDs in den Augen können zu interessanten Farben/-spielen geschaltet werden und zum Beispiel verwendet werden um Emotionen darzustellen. Sanfte Helle Augenfarben suggerieren zum Beispiel Ruhe, leuchtendes Rot dagegen möglicherweise Unruhe oder gar Wut.

Darüber hinaus sind in jedem Ohr, also an den Kopfseiten, 10 blaue LEDs angebracht, die in jeweils 16 Stufen geregelt werden können. Anhand dieser LEDs wird der Bootprozess des Nao nach folgendem Schema<sup>23</sup> visualisiert:

Leuchstärke	Fortschritt des Bootprozesses
10%	entering in the initrd and before finding partition to boot
20%	end of initrd, the partition is found
30%	start of init script
40%	the wireless process is launched
50%	after ifplugd daemon and web server
60%	after firmware update
70%	after NAOqi start
100%	NAO is ready to use

Die Beschreibung dieses Schemas wurde nicht übersetzt, da zum Verständnis ohnehin die Kenntnis der Fachbegriffe und des Bootvorgangs eines Computersystems erforderlich ist. Wichtig ist an dieser Stelle hauptsächlich, dass mögliche Probleme beim Bootprozess erkannt werden können.

<sup>22</sup> RGB-LEDs können die 3 Grundfarben Rot, Grün und Blau in unterschiedlichen Anteilen mischen und damit eine breite Farbpalette darstellen

<sup>23</sup> Zu finden unter: [http://users.aldebaran-robotics.com/docs/site\\_en/reddoc/hardware/leds.html](http://users.aldebaran-robotics.com/docs/site_en/reddoc/hardware/leds.html)



### 3.2.9 Fähigkeiten

Um die Verwendung eines humanoiden Roboters der aufrecht auf zwei Beinen läuft möglichst einfach zu gestalten liefert Aldebaran den Nao mit einigen vorgefertigten Fähigkeiten aus. Diese werden immer wieder erweitert und betreffen unterschiedlichste Funktionen.

Eine der wichtigsten Fähigkeiten die der Nao bereits bei Auslieferung beherrscht ist gerade das Laufen auf zwei Beinen sowie das stabile Stehen. Die Methode zum Laufen funktioniert ganz gut, erlaubt allerdings kein allzu schnelles Vorwärtskommen. Daher implementieren einige RoboCup Teams eigene Laufalgorithmen und erzielen damit eine zum Teil deutlich schnellere Fortbewegung<sup>24</sup> allerdings mitunter auf Kosten der möglichen Betriebszeit, da bei schnelleren Bewegungen der Stromverbrauch sowie die Hitzeentwicklung in den Gelenkmotoren deutlich steigen kann.

Ausser Laufen kann der Nao von Beginn an bereits vom Stand in eine stabile Sitzposition wechseln, in der er auch in ausgeschaltetem Zustand stabil ist, sowie aus beliebigen Positionen aufstehen. Dafür wird mittels des Beschleunigungssensors und der Gelenksensoren ermittelt in welcher Position er sich gerade befindet und anschließend eine geeignete Aktion ausgewählt um aufzustehen.

Bei Demonstrationen beeindrucken, können Tänze wie der bereits erwähnte Tai-Chi Tanz. Diese sind allerdings an sich fast reine Choreographien und sehen zwar toll aus, enthalten jedoch relativ wenig „Programmiertechnik“. Diese Tänze zu erstellen ist trotzdem absolut nicht einfach und insbesondere Bewegungen der Füße erfordern ein differenziertes Ausbalancieren zu jedem Zeitpunkt der Choreographie. Einen Tanz „dynamisch“ zu programmieren, also nicht einfach als festen Ablauf, ist grundsätzlich durchaus denkbar, die Umsetzung allerdings sicherlich aufwändig.

Über diese motorischen Fähigkeiten hinaus ist der Nao in der Lage einfache Kommandos akustisch zu erkennen und darauf zu reagieren sowie mit Hilfe eines Moduls zur Sprachsynthese Texte vorzulesen. Die Qualität der Sprachsynthese ist brauchbar, allerdings bisher nicht für die deutsche Aussprache verfügbar. Mit einem Trick können deutsche Wörter dennoch auf verständliche Weise ausgesprochen werden. Schreibt man die deutschen Wörter so, wie deren Phoneme am besten in englischer Schreibweise angenähert werden können, lassen sich passable Ergebnisse erzielen.<sup>25</sup>

Hören sich die Worte: *Guten Morgen*

bei englischer Aussprache zum Beispiel eher wie: *Gjiut Moagn an*,

so erhält man mit folgender Schreibweise eine deutlich bessere Aussprache: *Gootan Morrgan*

Für tiefergehende Versuche stellt der Nao selbst eine Methode zur Verfügung, die versucht im aktuellen Kamerabild einen roten Ball und dessen Position relativ zum Roboter zu berechnen. Auf diese Methode kann über einen Proxy (ALRedBallDetection), ähnlich wie auf andere Sensordaten zugegriffen werden. Die nötigen Berechnungen werden dabei auf dem Prozessor des Naos ausgeführt und das Ergebnis über einen weiteren Proxy (ALMemory) verfügbar gemacht.

---

<sup>24</sup> RoboCup 2010: Nao walking at 444 mm/s [ [http://www.youtube.com/watch?v=3aOuQ1\\_e—k](http://www.youtube.com/watch?v=3aOuQ1_e—k) ]

<sup>25</sup> Diese Idee wurde von Professor Dr. Hahn während einer Demonstration des Naos eingebracht.

## 4 Werkzeuge

Dieses Kapitel beschreibt die Arbeitsumgebung und die verwendeten Werkzeuge. Darunter sind die Computer, die verwendete Software und zusätzliche Hardware zu verstehen.

### 4.1 Sonstige Umstände

Der Nao wird an der HdM im MobileLab<sup>26</sup> aufbewahrt und sollte in der Regel auch dort verbleiben. Daher wurde ein beträchtlicher Teil der Arbeiten dort durchgeführt. Es stehen ausreichend viele PCs mit leistungsstarken Komponenten zur Verfügung und die Arbeitsatmosphäre ist gut, der Raum ist gut besucht, aber keinesfalls überfüllt und man findet auch in kniffligen Situationen die nötige Ruhe.

Neben den dort verfügbaren PCs wurde eigens für die Naos ein Notebook angeschafft, das in erster Linie dafür dienen soll, die für Demonstrationen nötigen Daten und Programme auch mobil zur Verfügung zu haben. Dieses Notebook kann auch zu Entwicklungszwecken verwendet werden, an einen der verfügbaren großen TFT-Monitore angeschlossen kann gut damit gearbeitet werden. Möchte man allerdings mit einem Simulator<sup>27</sup> arbeiten, empfiehlt es sich diesen auf einem der freien PCs zu installieren, da der Simulator recht hohe Ansprüche an die Rechenleistung stellt und das Notebook diesen nicht ganz gerecht wird. Für den Betrieb des Notebooks ist es ratsam sich im Falle einer Neuinstallation des Systems, zunächst alle benötigten Treiber direkt von der Website des Herstellers zu besorgen und diese direkt nach der Systeminstallation zu installieren.

Wie bereits erwähnt wurde ausserdem ein WLAN-Router angeschafft der ein vorkonfiguriertes Netzwerk für den Nao bereitstellt und der bei den in dieser Arbeit beschriebenen Versuchen per LAN-Kabel mit dem Notebook verbunden wurde. In dieser Konstellation reicht es aus, die Kabel anzuschließen, den Nao, das Notebook und den WLAN-Router einzuschalten um eine Verbindung zwischen Notebook und Nao herzustellen.

Nach den ersten Kontakten mit dem Nao entstand die Idee diesen zunächst manuell zu steuern und so ein Gefühl für dessen Fähigkeiten zu bekommen. Als Eingabegerät wurde hierfür das Gamepad<sup>28</sup> der xBox 360 vom Hersteller Microsoft gewählt, da dieses unter den Windows-Betriebssystemen ohne Probleme erkannt wird und mit 4 analogen Achsen, zahlreichen Tasten sowie 2 analogen Schaltern ausreichende Möglichkeiten bietet, um zum Beispiel einen Arm des Roboters komplett zu steuern. Auch von diesen Gamepads wurde für jeden Nao eines besorgt und eines davon befindet sich zusammen mit dem Nao, dem Notebook und dem WLAN-Router im MobileLab.

Programmierungsansätze für die Verwendung des Gamepads zur Steuerung finden sich in Kapitel 6.2 sowie im Quellcode auf der beiliegenden CD.

---

<sup>26</sup> Labor für Projekte und Versuche der Studiengänge Medieninformatik und Mobile Medien

<sup>27</sup> Siehe Kapitel 4.2

<sup>28</sup> Eingabegerät für PCs oder Spielkonsolen, in der Regel mit mehreren Tasten und analogen Eingabeachsen die zum Beispiel durch kleine Steuerknüppel bedient werden

## 4.2 Software

Wie bereits angesprochen gehört zur Nao Plattform das Programm „Choregraphe“. Dieses wurde im Verlauf der Arbeit immer wieder benutzt und leistet, auch wenn es nicht zur Erstellung von „Behaviours“ bzw. zur Programmierung des Roboters benutzt wird, gute Dienste. Es erlaubt es zum Beispiel bestehende „Behaviours“ auf den Roboter zu laden oder einzelne Gelenke direkt zu steuern oder deren „Stiffness“ einzustellen oder sich einfach das Kamerabild „live“ anzuschauen, um zu wissen was sich gerade tatsächlich im Blickfeld des Naos befindet.

Zur eigentlichen Programmierung und für die Versuche bei denen ein Simulator eingesetzt wurde, wurden Netbeans IDE, Virtual Box und Webots benutzt.

### 4.2.1 Netbeans IDE

Netbeans IDE ist eine Programmierumgebung, die ähnlich wie Eclipse mit Hilfe von Plugins erweitert werden kann. Mit deren Hilfe können der Programmierumgebung neue Programmiersprachen hinzugefügt werden und Hilfsfunktionen wie Code-Highlighting<sup>29</sup> können verwendet werden. Die Wahl der Programmierumgebung ist zunächst von deren Möglichkeiten abhängig, darüber hinaus bleibt dem Programmierer mehr oder weniger die freie Wahl und persönliche Erfahrungen und Vorlieben können die Entscheidung beeinflussen solange keine diesbezüglichen Vorgaben bestehen.

### 4.2.2 Webots

Ein wichtiges Werkzeug für Versuche mit Robotern sind Simulatoren. Diese erlauben es Versuche automatisiert und unter 100% kontrollierten Bedingungen durchzuführen. Um Versuche aus dem Bereich des maschinellen Lernens durchzuführen braucht man in aller Regel mehrere Durchläufe, häufig hunderte oder mehr. Theoretisch kann man diese Versuche auch real nachstellen und durchführen, allein jedoch die Ausgangsbedingungen zu kontrollieren ist praktisch eine aufwändige und manchmal nicht vollständig lösbare Aufgabe. Im Simulator hingegen ist dies relativ einfach möglich und man kann automatisiert sehr viele Versuche ohne Überwachung durchführen.

Ein weiteres Argument für die Verwendung von Simulatoren sind Verschleißerscheinungen am realen Roboter. Bei neuen Versuchen kommt es gelegentlich vor, dass mehr oder weniger unerwartete Probleme auftreten. Sollte der Roboter dabei ungeschickte Bewegungen machen passiert es schnell, dass Motoren oder Gelenke unnötig stark belastet werden. In einem Simulator riskiert man im Zweifelsfall keine Beschädigung eines teuren Roboters und kann Versuche entsprechend vorbereiten und viele mögliche Probleme bereits im Vorfeld beseitigen. Trotzdem sind simulierte Versuche in der Regel nicht direkt auf die reale Welt übertragbar und erfordern meist Anpassungen verschiedener Details.

Aktuell existieren zwei Versionen von Webots, eine EDU Version (320 CHF / 290 €) mit, im Vergleich zur PRO Version (2.300 CHF / ~2.000 €) deutlich reduziertem Preis.

---

<sup>29</sup> Hervorheben von Schlüsselworten und Kontrollstrukturen in Quellcodes

Unterschiede im Funktionsumfang kann man folgender Tabelle<sup>30</sup> entnehmen:

Webots	PRO	EDU
Supervisor capability	√	
Physics plugin programming	√	
Fast simulation mode	√	
Multiple platform: Windows, Mac & Linux	√	√
Floating License Option	√	√
Transfer to real robots	√	√
Packaged or electronic Version	√	√
1 year Premier service included	√	√

Die 5 gemeinsamen Features erlauben Versuche mit einem (oder mehreren) simulierten Nao, sowohl in der EDU als auch in der PRO Version.

Für ausgedehnte Versuchsreihen empfiehlt es sich, den „Supervisor“ der PRO Version zu verwenden, mit dessen Hilfe die Parameter einzelner Versuche kontrolliert werden können. Da dieses Feature jedoch nicht getestet wurde, können an dieser Stelle keine Erfahrungen dazu festgehalten werden. In Kapitel 6.2.3 findet sich ein Ansatz mit dem auch in der EDU Version einfache Versuchsreihen durchgeführt werden können.

Falls man die Physik der Simulation verändern möchte um bestimmte Umgebungsbedingungen nachzustellen, benötigt man ebenfalls die PRO Version. Für die meisten Versuche mit dem Nao dürfte das vermutlich allerdings nicht benötigt werden.

Auf einem leistungsfähigen PC ist Webots in der Lage Simulationen nahezu in Echtzeit durchzuführen. Falls die Rechenleistung aufgrund von schwacher Hardware oder sehr komplexen Versuchen jedoch nicht ausreicht, gibt es in der PRO Version mit „Fast Simulation“ die Möglichkeit, das lokale Rendern der „Welt“ zu deaktivieren, der Nao ist dabei weiterhin in der Lage ein Kamerabild seiner simulierten Umgebung zu liefern.

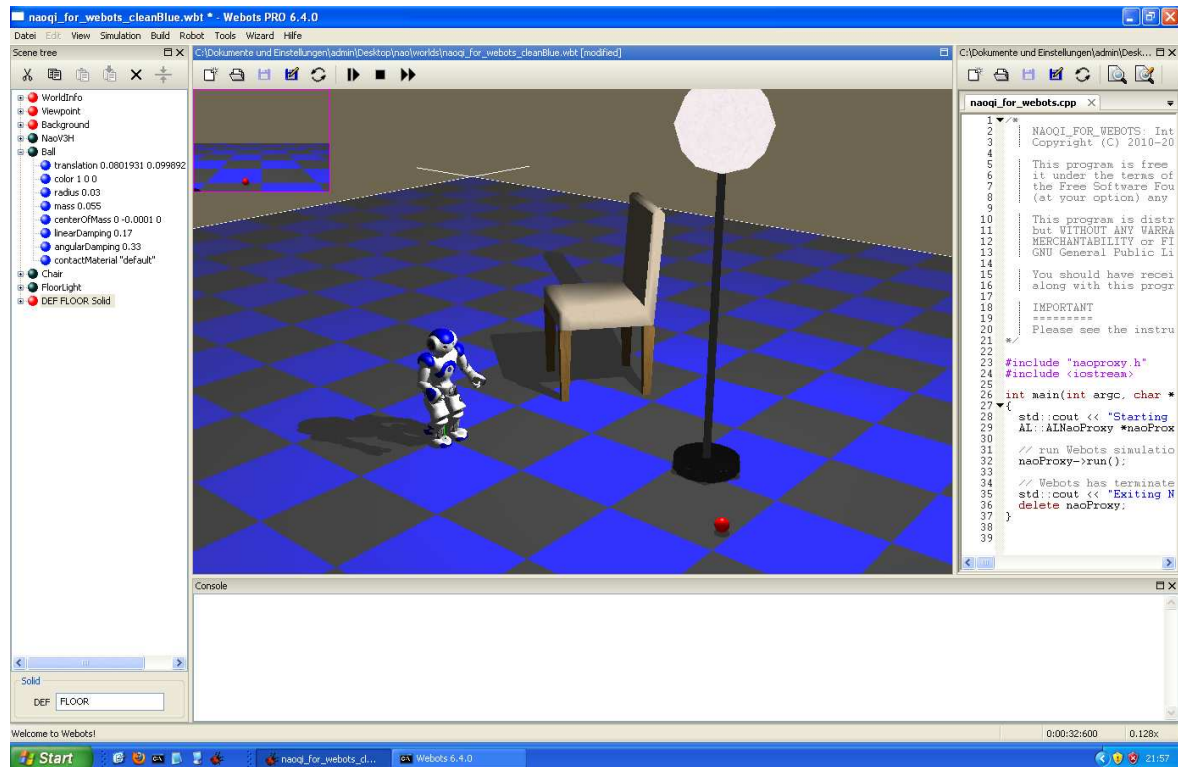
Die 3 „Extra-Features“ der PRO Version sind sicherlich interessant und für viele (fortgeschrittene) Versuche eine große Hilfe. Für grundlegende Versuche sind sie nicht nötig und die EDU Version sollte ausreichenden Funktionsumfang bieten. Mit der angesprochenen Alternative zur Durchführung von Versuchsreihen bietet Webots auch in der EDU Version alles Benötigte für einen Einstieg in die Robotersimulation.

Webots ist als 30 Tage Trial-Version ohne Einschränkungen (sowohl EDU als auch PRO) verfügbar und kann mit wenig Aufwand zur Simulation eines Naos verwendet werden. Die Schnittstelle eines mit Webots simulierten Nao wird mit Hilfe eines sogenannten Controllers bereitgestellt. Zur Erstellung dieses Controllers benötigt man Quellen aus dem NAOqi SDK sowie den Quellcode des Controllers. Glücklicherweise ist der gesamte Prozess sehr einfach und besteht letztendlich nur

<sup>30</sup> Entstammt der Broschüre zu Webots: <http://www.cyberbotics.com/webots/webots6.pdf>

daraus die Quellen bereit zu stellen, ein fertiges Visual Studio 2008 Projekt zu laden und dieses zu kompilieren.<sup>31</sup>

Hat man den Controller erstellt, kann man eine bereits fertige „Welt“ laden die einen Nao und ein paar Gegenstände enthält.



Startet man diese „Welt“, aktiviert der simulierte Nao seine Netzwerkschnittstelle und ist zunächst genau wie ein physikalisch vorhandener Nao erreichbar und steuerbar. Bis auf einige Einschränkungen ist der simulierte Nao in der Lage die gleichen Befehle auszuführen wie ein realer Roboter und erlaubt so das Testen von Versuchen.

<sup>31</sup> Anleitung verfügbar unter: <http://www.cyberbotics.com/nao/>

#### 4.2.2.1 Unterschiede zwischen realem und simuliertem Nao

Es gibt natürlich auch Unterschiede zwischen einem simulierten und einem echten Nao. Der erste Unterschied der sich im Rahmen dieser Arbeit zeigte hat mit der Schwerkraft zu tun. In der simulierten Umgebung bewegt der Roboter seinen Arm exakt bis auf die gewünschte Winkelstellung, in der realen Welt machen sich unter anderem kleine Ungenauigkeiten der Mechanik bemerkbar und der Arm sackt minimal, aber dennoch bemerkbar nach unten sobald die Zielposition erreicht ist und nun diese Position gehalten werden muss. Die Folge waren beim realen Nao ruckelnde, beim simulierten flüssige Armbewegungen. Genauer dazu ist in Kapitel 6.2 beschrieben.

Ein weiterer Unterschied betrifft die in Kapitel 3.2.9 angesprochenen mitgelieferten Fähigkeiten. So kann der simulierte Nao zwar laufen und auch über Choregraphie mit „Behaviours“ geladen werden, die Methode zur Erkennung eines roten Balles<sup>32</sup> fehlt allerdings und somit ist es leider mit der aktuellen Webots Version nicht möglich Versuche, die auf dieser Methode basieren, im Simulator durchzuführen.

Die Modellierung des Naos im Simulator sieht auf den ersten Blick ziemlich gut aus, betrachtet man aber die Funktionalitäten genauer, so zeigt sich, dass einige Details sehr gut gelöst werden konnten, andere jedoch in der aktuellen Version 6.4.1 leider noch nicht ausreichend gut implementiert sind. So sind zum Beispiel das 3D-Modell des Naos und dessen Netzwerkschnittstelle sehr gut gelungen. Die bisher fehlende Methode zur Ball Erkennung lässt sich noch durch eigene Methoden ersetzen<sup>33</sup>, die Modellierung der simulierten Roboterhand hingegen hat zur Folge, dass Greifbewegungen im Simulator bisher noch nicht zufriedenstellend möglich sind. Vereinfacht gesagt liegt es daran, dass die Finger der Roboterhand sozusagen durch das zu greifende Objekt „hindurch greifen“ und keinen Halt finden<sup>34</sup>.

Webots ist somit ein vielversprechender Simulator für den Einsatz mit der Nao Plattform, hat allerdings in der aktuellen Version noch die ein oder andere Schwäche und ist nur für Versuche mit gewissen Einschränkungen geeignet. Grundsätzlich ist der Support des Herstellers sehr bemüht und reagierte auf Anfragen sehr schnell mit mehreren Lösungsvorschlägen was die Hoffnung erlaubt, dass eine zukünftige Version möglicherweise eine verbesserte und vollständigere Simulation bieten kann.

Die Bedienung von Webots ist insgesamt sehr intuitiv gestaltet und Bedarf relativ wenig Erklärung. Man kann „Welten“ speichern, dabei wird die gesamte Szene exakt festgehalten und kann diese bei Bedarf erneut als Ausgangssituation laden. Veränderungen an der „Welt“ nimmt man entweder über Konfigurationsdateien oder direkt über die grafische Oberfläche vor, man kann Objekte zum Beispiel einfach mit der Maus anklicken und verschieben oder drehen. Die Eigenschaften aller Objekte der „Welt“ werden links auf dem Monitor dargestellt und die meisten dieser Parameter, die Größe beispielsweise, können zur Laufzeit angepasst werden und wirken sich sofort auf die simulierte „Welt“ aus.

---

<sup>32</sup> ALRedBallDetection

<sup>33</sup> Siehe Kapitel 5.1: Lokalisierung

<sup>34</sup> Die Gründe liegen in der aktuellen Modellierung der Roboterhand in Webots durch mehrere einzelne Motoren die gleichzeitig bewegt werden und fehlender Reaktion auf Kollisionen mit Objekten.

## 4.3 Python und verwendete Libraries

Zur Programmierung kam die Sprache Python<sup>35</sup> zum Einsatz. Diese findet mittlerweile immer mehr Verbreitung, und wird zum Beispiel häufig bei Projekten zur Entwicklung von Prototypen eingesetzt. Python ist (unter anderem) eine objektorientierte Programmiersprache und fällt besonders durch zwingend notwendige Einrückungen für Codeblöcke anstelle von Klammerungen auf.

Python gibt es in vielen verschiedenen Versionen, verbreitet sind aktuell Versionen von 2.5.x bis hin zu 3.2.2 und davon jeweils 32 Bit und 64 Bit Versionen. Das mag anfangs verwirrend erscheinen, da tatsächlich verschiedene Versionsnummern immer wieder aktualisiert werden und die höchste Nummer nicht zwingend die zuletzt veröffentlichte ist. Das erklärt sich allerdings dadurch, dass es einige Bibliotheken<sup>36</sup> gibt, die auf älteren Python Versionen aufbauen und nicht zu 100% mit neueren kompatibel sind. Um die Unterstützung für diese allerdings nicht vollständig aufgeben zu müssen werden auch ältere Versionen von Python noch mit Updates und in erster Linie sicherheitsrelevanten Patches gepflegt.

Für diese Arbeit wurde die 32 Bit Version von Python 2.6.6 eingesetzt, da die verwendeten Libraries alle mit dieser Version funktionieren.

### 4.3.1 OpenCV

Bei der Suche nach einer Alternative für die im Simulator fehlende Funktion zur Erkennung eines roten Balls wurde eine Methode des Frameworks OpenCV getestet. Es handelt sich dabei um eine Methode zur Erkennung von Kreisen in Bildern<sup>37</sup>.

OpenCV wurde ursprünglich im Jahr 1999 von der Firma Intel entwickelt und hat das Ziel Funktionen zur Bildverarbeitung auf effiziente und schnelle Weise in der Programmiersprache C bereitzustellen. Mittlerweile wird die Entwicklung von OpenCV hauptsächlich von der Robotik Firma *Willow Garage*<sup>38</sup> vorangetrieben und die Library bietet eine Vielzahl unterschiedlichster Methoden aus der aktuellen Forschung aus Bereichen der Bildverarbeitung, Objekterkennung und weiteren.

Neben der Schnittstelle zu C++ werden mehrere Programmiersprachen unterstützt, darunter Python und Java. Das Einbinden von OpenCV in eine Python 2.6 Umgebung erfordert keine Installation im eigentlichen Sinne, es reicht aus, sich das aktuelle Archiv, für Windows zum Beispiel unter diesem Link<sup>39</sup>, herunterzuladen und aus dessen Unterverzeichnis: `...\OpenCV2.3\build\Python\2.6\Lib\site-packages` die beiden Dateien `cv.pyd` und `cv2.pyd` in folgendes Unterverzeichnis der entsprechenden Pythoninstallation zu kopieren: `...\Lib\site-packages`. Danach sind die OpenCV Methoden unter Python verfügbar.

---

<sup>35</sup> weitere Informationen zu dieser Programmiersprache unter: <http://www.python.org/>

<sup>36</sup> Libraries, Frameworks, ...

<sup>37</sup> Genaueres in Kapitel 5.1: Lokalisierung

<sup>38</sup> Weitere Informationen zu OpenCV unter: <http://opencv.willowgarage.com/wiki/>

<sup>39</sup> OpenCV Download: <https://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.3.1/>

Um einen tieferen Einblick in OpenCV zu bekommen bietet sich [BK] an. Es beschreibt einen Großteil der verfügbaren Funktionen und hält einfach gehaltene Erklärungen zu den jeweiligen theoretischen Hintergründen bereit.

### 4.3.2 Pygame

Pygame stellt unter anderem Möglichkeiten zur Verfügung um mit Pythonprogrammen auf Eingabegeräte zugreifen zu können. Wie der Name „Pygame“ schon suggeriert, handelt es sich um eine Library die der Spieleentwicklung dienen soll und entsprechend werden nicht nur Maus und Tastatur unterstützt, sondern auch Joysticks und Gamepads.

Daneben bietet Pygame die Möglichkeit Grafikausgaben und Sound zu generieren. Hauptsächlich eingesetzt wurde es bei dieser Arbeit jedoch um Eingabewerte von einem Gamepad zu lesen und den Roboter damit manuell zu steuern.

Bei Verwendung von Pygame ist zu beachten, dass die seit 2009 aktuelle Version (stable 1.9.1) leider mit aktivierten Debugausgaben kompiliert wurde, was zur Folge hat, dass jedes Abfragen des Eingabegerätes eine Zeile über die Konsole ausgibt. Da dieses Abfragen des Eingabegerätes mindestens 20-25 mal pro Sekunde durchgeführt werden muss um eine brauchbare Steuerung zu ermöglichen bedeutet dies, dass die Konsole kaum noch für weiteres Debugging<sup>40</sup> benutzbar ist. Abhilfe schafft hier die inoffizielle neuere Version „pre 1.9.2“<sup>41</sup>

### 4.3.3 PyBrain

Für den Einsatz von Methoden der Künstlichen Intelligenz und maschineller Lernverfahren empfiehlt es sich zunächst Ansätze mit bereits bestehenden Verfahren zu testen und diese bei Bedarf anzupassen. In der Fachliteratur finden sich zahlreiche Beschreibungen von Algorithmen und Verfahren, die man natürlich auch selbst implementieren kann. Dazu sollten diese allerdings vollständig verstanden sein und idealerweise bereits einige Erfahrung mit deren Anwendung vorhanden sein. Ist dies nicht gegeben, kann es sehr aufwändig und langwierig werden bis die erste Idee zu einer funktionierenden Lösung gewachsen ist.

Für verschiedene Programmiersprachen, darunter C++, Java, Python und weitere, existieren verschiedene Frameworks und Libraries die solche Algorithmen bereitstellen. Bei dieser Arbeit sprachen mehrere Gründe für PyBrain. Wie der Name vermuten lässt, handelt es dabei um ein Framework für Python, mit dem Ziel eine Umgebung für verschiedenste Versuche im Bereich der Künstlichen Intelligenz und des Maschinellen Lernens zu bieten. PyBrain verfügt über verschiedenste Algorithmen<sup>42</sup>, von *Back-Propagation* zum Training künstlicher neuronaler Netzwer-

---

<sup>40</sup> In vielen Fällen relativ langwierige und komplizierte Fehlersuche bei der Programmentwicklung

<sup>41</sup> Neben vielen weiteren Python Libraries zu finden unter: <http://www.lfd.uci.edu/~gohlke/pythonlibs/>. Vorsicht! Aktuell (7.September 2011) wird beim Aufruf dieser Seite von Avira AntiVir Alarm ausgelöst und ein Trojaner gemeldet. Der Onlinedienst „AVG Online Web Page Scanner“ findet jedoch keine Bedrohung. Diese Internetseite sollte trotzdem bis auf weiteres nur mit entsprechenden Vorsichtsmaßnahmen geladen werden!

<sup>42</sup> Eine Übersicht findet sich in der Dokumentation zu PyBrain und unter: <http://pybrain.org/pages/features>



ke, über *Principle Component Analysis*, bis hin zu verschiedenen Verfahren des *Bestärkenden Lernens* (englisch: *Reinforcement Learning*).<sup>43</sup>

Die für die vorliegende Arbeit getesteten und verwendeten Verfahren, sowie die entsprechenden theoretischen Hintergründe werden in Kapitel 5.2 in Grundzügen erklärt. Die Online Dokumentation zu PyBrain<sup>44</sup> enthält neben Erklärungen und einer Übersicht des Frameworks auch einige Beispiele und Tutorials.

Es empfiehlt sich einen Blick auf den aktuellen Quellcode des Projekts zu werfen da dort einige weitere interessante Beispiele enthalten sind. Den aktuellen Quellcode erhält man über das Git-Repository<sup>45</sup> des Projektes. *Git* ist ein Versionsverwaltungssystem, unter Windows kann zum Beispiel *msysgit*<sup>46</sup> für den Zugriff benutzt werden.

---

<sup>43</sup> Siehe Kapitel 5: Theoretische

<sup>44</sup> PyBrain Dokumentation: <http://pybrain.org/docs/tutorial/intro.html>

<sup>45</sup> Downloadanleitung unter: <http://pybrain.org/pages/download>

<sup>46</sup> msysgit: <http://code.google.com/p/msysgit/>

## 5 Theoretische Grundlagen

Um die durchgeführten Versuche und Experimente nachvollziehen zu können bietet dieses Kapitel einen Einblick in die theoretischen Grundlagen der angewendeten Verfahren. Dabei werden hauptsächlich zwei Themen vorgestellt und die jeweiligen Lösungsansätze in Grundzügen beschrieben. Die Perspektive bleibt relativ abstrakt und Erfahrungen der praktischen Versuche mit den beschriebenen Methoden finden sich in den Kapiteln 6 und 7. Quellen zur Vertiefung der hier genannten Verfahren werden an den entsprechenden Stellen und im Literaturverzeichnis genannt.

### 5.1 Lokalisierung einer Kugel

Für die Bearbeitung der ersten Teilaufgabe dieser Arbeit wurde eine Methode benötigt, um in Kamerabildern eine rote Kugel mit einem Durchmesser von ca 3-30 cm in einer Entfernung von 50 cm bis ca. 4 m zu erkennen. Bei der Durchführung im Simulator wurden zu Testzwecken verschiedene Ballgrößen verwendet, aufgrund der Größe der Hand des Roboters kann dieser nur Gegenstände mit einer maximalen Größe von 3-4 cm greifen. Da sich das Problem zunächst darauf reduzieren lässt in Bildern Kreis(-flächen) zu finden, spielt die Kugelgröße eine untergeordnete Rolle. Zur Lösung des Problems standen konkret 3 verschiedene Ansätze zur Auswahl. Die Naoplattform selbst enthält bereits eine relativ gut funktionierende Methode und kann die Position einer roten Kugel relativ zum Körper des Roboters berechnen. Leider ist diese Methode im Simulator Webots (noch) nicht verfügbar und es wurde eine Alternative benötigt, die wenigstens vergleichbare Ergebnisse liefert. Die zweite Option, die auch diese Bedingung erfüllen kann, verwendet die unter Kapitel 5.1.2 beschriebene Methode zur Kreiserkennung und wird durch OpenCV bereitgestellt. Die dritte Möglichkeit bestand darin selbst einen einfachen Ansatz zu suchen und zu implementieren. Da die Ergebnisse der zweiten Methode nicht die gewünschte Genauigkeit erreichten, wurde ein einfacher eigener Ansatz getestet und für einige Versuche im Simulator verwendet.<sup>47</sup>

Weitere Ansätze um aus Bildern Informationen über die Umwelt zu bekommen sind zum Beispiel in [RN] beschrieben.

#### 5.1.1 Bildverarbeitung

Um mit einem Programm oder Algorithmus ein Bild verarbeiten zu können, muss das Format dessen Repräsentation bekannt sein. Es gibt viele verschiedene Formate die jedoch auch Gemeinsamkeiten haben.

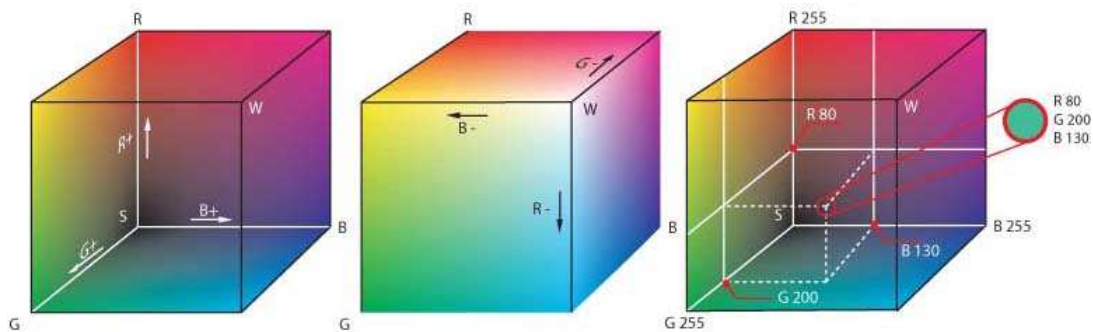
Farben auf einem Computerbildschirm setzen sich aus Anteilen verschiedener Grundfarben zusammen. Einzelne Pixel herkömmlicher Computermonitore setzen sich fast immer aus roten, grünen und blauen Sub-Pixeln zusammen. Dies führt zu einem einfachen Farbmodell und wird RGB<sup>48</sup> genannt. Für jede Grundfarbe wird einfach die Intensität jeder Grundfarbe in einem Array gespeichert. Diese Pixel können dann entsprechend ihrer Zeile und Spalte in einem weiteren 2-dimensionalen Array abgelegt werden. Es gibt Varianten dieser Methode, zum Beispiel verwendet OpenCV intern eine andere Reihenfolge der Grundfarben (BGR) oder es können einzelne Grund-

---

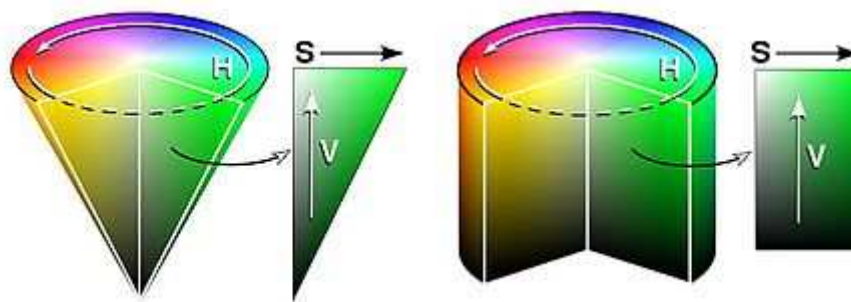
<sup>47</sup> Siehe Kapitel 6.2.2

<sup>48</sup> <http://de.wikipedia.org/wiki/RGB-Farbraum>

farben durch Verwendung von mehr Bits höher aufgelöst werden um der menschlichen Farbwahrnehmung näher zu kommen. Dieses Format lässt sich entsprechend folgender Grafik<sup>49</sup> durch einen Würfel darstellen.



Ein anderes Format wird mit HSV bezeichnet, was für Hue-Saturation-Value steht. Hue bezeichnet den Farbton, Saturation die Farbsättigung und Value die Helligkeit. Der Farbton wird als Winkel angegeben, Farbsättigung und Helligkeit in Prozent. Damit lässt sich dieses Modell durch einen Kegel oder einen Zylinder darstellen<sup>50</sup> und bietet die Möglichkeit Farben mit bestimmtem Farbton auf einfachere Weise als im Würfel zu identifizieren.<sup>51</sup>



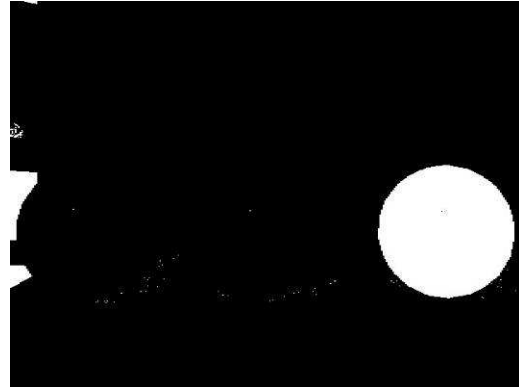
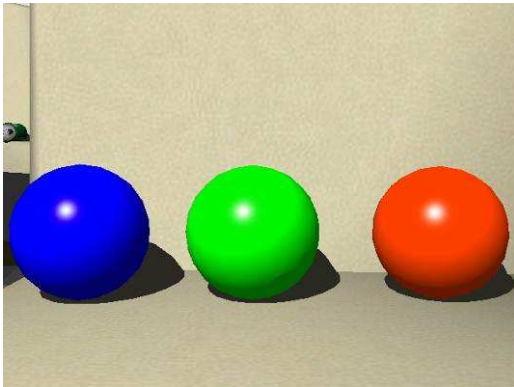
Um in diesem Modell einen Farbfilter auf ein Bild anzuwenden, prüft man alle Pixel eines Bildes und behält nur jene Pixel, die einen Farbton innerhalb eines bestimmten Winkelbereichs haben. Dabei muss man berücksichtigen, dass Pixel mit niedrigem Sättigungs- und gleichzeitig hohem Helligkeitswert einen Farbwert in Richtung weiß und mit niedrigem Helligkeitswert in Richtung schwarz haben. Um also nur Pixel mit einem „richtigen“ Farbwert übrig zu behalten, müssen die zu hellen und die zu dunklen entfernt werden. Man kann entweder alle Pixel bis auf die übrig gebliebenen „schwarz schalten“ indem man (im HSV-Farbmodell) ihre Helligkeit auf 0% setzt oder man verwendet eine sogenannte binäre Maske, die einem schwarz-weiß Bild entspricht und setzt dort für jeden passenden Pixel eine 1 und ansonsten eine 0. Nun kann man entweder versuchen direkt in dieser Maske Kreisflächen zu finden, oder man verwendet sie um aus dem originalen Bild die ursprünglichen Farbwerte der gesuchten Pixel auszulesen und arbeitet mit diesen weiter.

<sup>49</sup> Bildquelle: <http://de.wikipedia.org/wiki/RGB-Farbraum>

<sup>50</sup> Bildquelle: <http://de.wikipedia.org/wiki/HSV-Farbraum>

<sup>51</sup> Die beiden Modelle lassen sich in einander umrechnen, bieten also insgesamt die gleiche Information.

Das Ergebnis eines Rot-Filters kann zum Beispiel so aussehen. Bei diesem Beispiel<sup>52</sup> erkennt man, dass am linken Bildrand noch zu dunkle Bereiche übrigbleiben, also ist hier der mindestens nötige Helligkeitswert noch nicht hoch genug gewählt.



---

<sup>52</sup> Ergebnis eigener Versuche

### 5.1.2 Kreiserkennung

Digitale Kameras weisen in der Regel ein gewisses Rauschen<sup>53</sup> auf und in Kamerabildern demnach ein bestimmter Anteil von Pixeln eines Bildes verfälschte Farbwerte hat, enthalten Kamerabilder in der Regel auch nach der Farbfilterung noch vereinzelte Pixel die nicht zu den gesuchten Bildbereichen gehören. Durch Weichzeichnen und Schärfen oder Erodieren des Bildes kann man vereinzelte störende Pixel entfernen [BK]. Nachdem nun idealerweise nur noch die Bereiche des Bildes, die die rote Kugel anzeigen, übrig sind, muss in diesen Bereichen die Kugel identifiziert werden. Die Silhouette einer Kugel ist immer eine Kreisfläche, deren Größe entsprechend der Entfernung variiert.

Um den (oder die) besten Kreise zu berechnen ist in [BK] die „Hough Circle Transformation“ implementiert. Diese Methode basiert darauf, zunächst durch Berechnung der Gradienten der Farb- bzw. Helligkeitsverläufe (auch binär) Kanten und damit mögliche Linien in einem Bild zu finden. Daraus werden mögliche Kreiszentren und -radien berechnet. Eine genaue Beschreibung des Algorithmus findet sich [BK] auf den Seiten 153 bis 161, sowie den dort genannten Originalquellen. Zu erwähnen ist jedoch, dass diese Methode schwarz-weiße oder „graue“ Bilder als Eingabe benötigt, die im letzten Kapitel beschriebenen „Masken“ eignen sich als Eingabe.

Das Problem der Kreiserkennung selbst kann sehr detailliert bearbeitet werden, dies lag jedoch nicht im Fokus der vorliegenden Arbeit, sondern trat eher als Teilproblem auf.

### 5.1.3 Probleme bei der optischen Lokalisierung

Bei optischen Erkennungsverfahren gibt es einige grundsätzliche Störungsquellen die berücksichtigt werden sollten um brauchbare Ergebnisse zu erhalten. Besonders die Beleuchtungssituation hat sehr großen Einfluss auf die Genauigkeit und die Qualität der Erkennung. Bei wechselnder Helligkeit passen viele Kameras automatisch die Belichtungszeiten an, oder verstärken (bzw. verringern) die Empfindlichkeit der optischen Sensoren. Dabei können Farbinformationen verfälscht oder verloren gehen. Vermeiden lässt sich dies in erster Linie dadurch, dass man bei Versuchen und an potentiellen späteren Einsatzorten eine möglichst konstante Beleuchtung sicherstellt.

Ausserdem haben alle mit Linsen arbeitenden Kamerasysteme das Problem, dass kleinste Ungenauigkeiten der Linsenkrümmung Auswirkungen auf die Geometrie des projizierten Bildes haben. So können gerade Linien gekrümmt erscheinen und Teile des Bildes leicht verzerrt sein. In manchen Anwendungen mag das vernachlässigbar sein, bei der Kreiserkennung ist es hingegen denkbar schlecht, wenn ein Kreis zu einer Ellipse verzerrt erscheint, da dann möglicherweise nur noch der Kreismittelpunkt korrekt erkannt werden kann. Je nachdem wie stark eine Linse verzerrt, ist es besser aufgenommene Bilder zunächst zu entzerren. OpenCV bietet auch dazu eine Möglichkeit, inklusive dem Erzeugen der dazu notwendigen Parameter durch Kalibrieren des Systems mit Hilfe eines Schachbrettmusters. Das entsprechende Vorgehen kann in [BK] ab Seite 430 nachgelesen werden.

---

<sup>53</sup> Mit Rauschen bezeichnet man im Allgemeinen (geringe) zufällige Abweichungen von Werten in Datensätzen.

## 5.2 Maschinelles Lernen

Das zweite Thema, das im Rahmen dieser Arbeit bearbeitet wurde ist das „Maschinelle Lernen“. Unter diesem Begriff kann man verschiedenste Aufgabenstellungen und deren Lösungsansätze zusammenfassen, daher ist es sinnvoll zunächst eine Definition des Begriffs zu geben und die hier interessanten Teilbereiche einzugrenzen. Anschließend werden die diese Arbeit betreffenden Teilgebiete des maschinellen Lernens in Grundzügen beschrieben.

### 5.2.1 Maschinelles Lernen und Künstliche Intelligenz

Maschinelles Lernen „...befasst sich mit künstlichen intelligenten Systemen, die unter Ausnutzung von Erfahrung ihre Leistungsfähigkeit automatisch verbessern. Umgesetzt wird das maschinelle Lernen indem einem Algorithmus eine Menge von Daten übergeben wird. Aufgabe des Algorithmus ist das Auffinden von Mustern und Eigenschaften in diesen Daten. ...“ [Maucher].

Bereits hier wird deutlich, dass der Begriff sehr weit gefasst ist und viele Teilgebiete umfassen kann. Und tatsächlich kommen Verfahren, die dem maschinellen Lernen zugeordnet werden können, in unterschiedlichsten Bereichen zum Einsatz. In vielen modernen Computerspielen<sup>54</sup> wird immer öfter von Computergegnern mit künstlicher Intelligenz gesprochen. Dabei ist keinesfalls gemeint, dass diese über eine der menschlichen vergleichbare Intelligenz verfügen, sondern vielmehr meist sehr spezialisierte Verhaltensweisen anwenden und diese auf die eine oder andere Art ändern und an bestimmte Situationen anpassen können. Die Bezeichnung *künstlich* spielt also eine große Rolle und grenzt den Begriff vom sonst üblichen allgemeinen Verständnis des Begriffs Intelligenz ab. Künstliche Intelligenz ist ein mächtiges Teilgebiet der Informatik und maschinelles Lernen ist als Teilbereich davon zu betrachten. Im Rahmen dieser Arbeit wurden unter anderem künstliche neuronale Netze verwendet, welche den grundlegenden Methoden der Künstlichen Intelligenz zugeordnet sind.

Analog dazu sollte man sich unter maschinellem Lernen nicht exakt das vorstellen was wir unter menschlichem Lernen verstehen. Die Unterschiede sind bei beiden Begriffen zwar mehr im Detail zu sehen, für das Verständnis ist es dennoch wichtig, sich dieser Unterschiede zu den allgemeinen Begriffen bewusst zu sein. Wird es wie oben auf wenige knappe Sätze reduziert entspricht das grundsätzliche Prinzip zwar, nach heutigem Verständnis, dem des menschlichen Lernens, die Möglichkeiten eines Computersystems kommen denen eines Menschen allerdings bisher nicht im Entferntesten nahe. Das gilt jedoch nur für die Betrachtung der potentiellen Erkenntnisse und deren Übertragung auf neue, gänzlich unbekannte Sachverhalte. Bei der Lösung von einzelnen speziellen Problemen mit angepassten Algorithmen oder Verfahren sind Maschinen bzw. Computer den Menschen bei den meisten Anwendungen überlegen, da sie keine Flüchtigkeitsfehler, Vergesslichkeit oder Müdigkeit kennen.

Ein universelles System, das sich wie ein menschliches Gehirn selbst an beliebige Aufgaben anpasst, konnte bisher noch nicht in großem Maßstab entwickelt werden. Zum einen ist die Funktionsweise des Gehirns noch nicht umfassend genug verstanden und die Komplexität eines solchen

---

<sup>54</sup> Computerspiele wurden hier nur als Beispiel gewählt. KI-Methoden werden ausserdem zur Musteranalyse in beliebigen Daten, Vorhersage- und Empfehlungssystemen jeglicher Art, und vielen weiteren Bereichen verwendet.

Systems übersteigt die Möglichkeiten heutiger Computersysteme aufgrund der Anzahl der verarbeitenden Elemente und deren Verschachtelung.

### 5.2.2 Lernverfahren

Die Aufgabe eines Lernverfahrens besteht darin, anhand von Trainingsdaten eine oder mehrere Regeln finden, die es erlauben Eingabedaten auf Ausgabedaten abzubilden. Dabei gibt es unterschiedliche Varianten, je nachdem wie viele bzw. ob überhaupt Daten für das Training bereitstehen, wie deren Verteilung beschaffen ist und ob das System später weiterhin lernen soll. Ebenso hat die Art der Ausgabedaten Einfluss darauf, welche Verfahren zum konkreten Versuch passen und welche nicht. Für Aufgabenstellungen, bei denen zum Beispiel eine Entscheidung zwischen *Ja* und *Nein* zu treffen ist, bieten sich andere Möglichkeiten an, als wenn mehrere kontinuierliche Werte ausgegeben werden sollen.

Es existieren verschiedene Grundtypen von maschinellem Lernen. Das Überwachte und das Unüberwachte und das Bestärkende Lernen. Eingehende Untersuchungen dazu finden sich beispielsweise in [Alpaydin].

- **Überwachtes Lernen:** Stehen sowohl die Eingabe- als auch die entsprechenden Ausgabedaten bereit, kann ein Algorithmus selbst überwachen ob die Eingabe auf die korrekte Ausgabe abgebildet wird. Mit Hilfe der Abweichungen kann das Ergebnis nach und nach verbessert werden.
- **Unüberwachtes Lernen:** Sind dagegen nur mögliche Eingabe- aber keine Ausgabedaten verfügbar kann man versuchen Strukturen oder Gesetzmäßigkeiten in den Daten zu finden und daraus Entscheidungen abzuleiten.

Diese beiden Verfahren sind für Aufgaben geeignet, bei denen die Lösung direkt aus den Eingabedaten ermittelt werden kann. Dies ist nicht jedoch in allen Fällen möglich. Beschreiben die Eingabedaten zum Beispiel den Zustand eines Systems, das durch wiederholte kleine Veränderungen, im Folgenden Aktionen genannt, in einen bestimmten Zielzustand gebracht werden soll, benötigt man einen anderen Ansatz um die jeweils am besten geeignete Aktion zu bestimmen.

- **Bestärkendes Lernen:** Tragen einzelne Aktionen nur einen Teil zur Lösung des Problems bei, so muss man Aktionsketten suchen um ein System in den gewünschten Zielzustand zu bringen. Dadurch verliert eine einzelne Aktion an Bedeutung und erst die Kombination vieler einzelner Aktionen führt zur gesuchten Lösung. Hierfür setzt man eine sogenannte Taktik, auch Strategie genannt, ein. Diese wird im Verlauf des Lernprozesses durch den Algorithmus angepasst und erlaubt danach für beliebige Zustände eine Aktion auszuwählen, die erwarten lässt, dass sie zur Lösung beiträgt. Dabei müssen einzelne Aktionen nicht optimal sein, das Ziel ist es eine Aktionskette zu finden, die das System mit möglichst wenig Kosten<sup>55</sup> in den gewünschten Zustand bringt.

---

<sup>55</sup> Kosten können abhängig von der Aufgabenstellung zum Beispiel die Anzahl der Aktionen, deren Energieverbrauch oder ähnliches sein.

### 5.2.3 Künstliche neuronale Netze

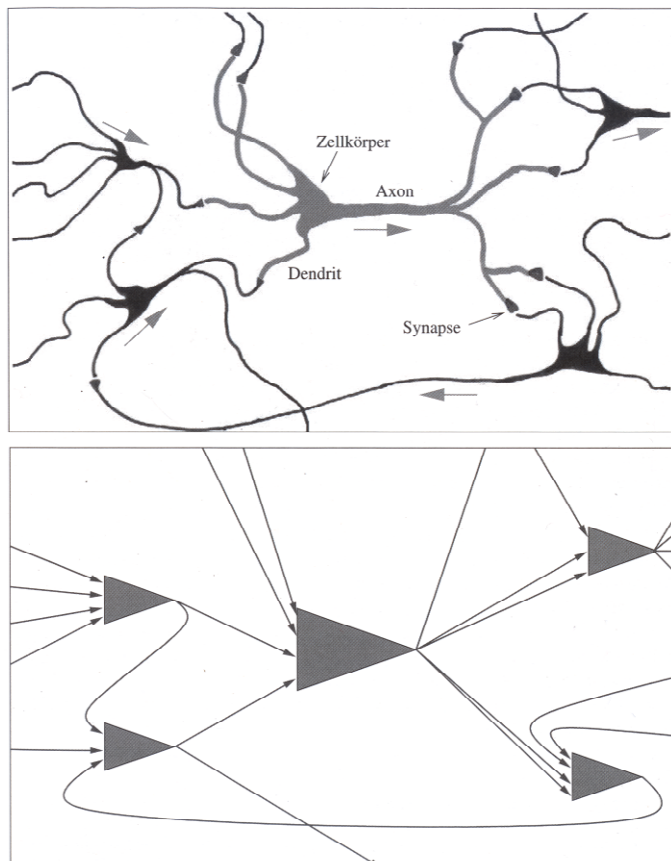
Im Bereich des maschinellen Lernens werden häufig künstliche neuronale Netze verwendet. Sie basieren auf der Funktion echter Nervenzellen und sind seit mehreren Jahrzehnten Objekt intensiver Forschungen. Mittlerweile sind die Grundtypen gut erforscht und künstliche neuronale Netze werden erfolgreich für verschiedenste Aufgaben eingesetzt.

Das Gehirn eines Menschen verfügt über ca 20 Milliarden Nervenzellen. Die Anzahl der entsprechenden künstlichen Neuronen gebräuchlicher künstlicher neuronaler Netze beträgt in der Regel dagegen nur einen Bruchteil davon, oft in der Größenordnung von einigen wenigen bis zu tausend Neuronen. Die Verarbeitungszeit von echten Nervenzellen ist sehr viel langsamer als die von künstlichen Neuronen. Da im Gehirn allerdings sämtliche Nervenzellen gleichzeitig arbeiten und bei der künstlichen Simulation des neuronalen Netzes alle Berechnungen mehr oder weniger sequentiell durchgeführt werden müssen, ist das Gehirn im Endeffekt wesentlich leistungsfähiger. Moderne Hardware, sowohl CPUs und insbesondere auch moderne Grafikprozessoren, verwenden ebenfalls immer stärkere Parallelisierung von einzelnen Berechnungen. Damit vervielfacht sich die Geschwindigkeit der Verarbeitung von neuronalen Netzwerken zwar, nach heutigem Stand der Technik ist man jedoch noch sehr weit davon entfernt auch nur annähernd die Leistungsfähigkeit eines Gehirns zu erreichen. Bedeutende Verbesserungen erhofft man sich erst von neuen Prozessorarchitekturen, die näher am Aufbau eines Gehirns orientiert sind. Aktuelle Forschungen in diese Richtungen lassen jedoch keine

Zur Veranschaulichung des Aufbaus einer echten Nervenzelle und der entsprechenden künstlichen Nachbildung soll folgende, aus [Ertel] entnommene, Abbildung dienen.

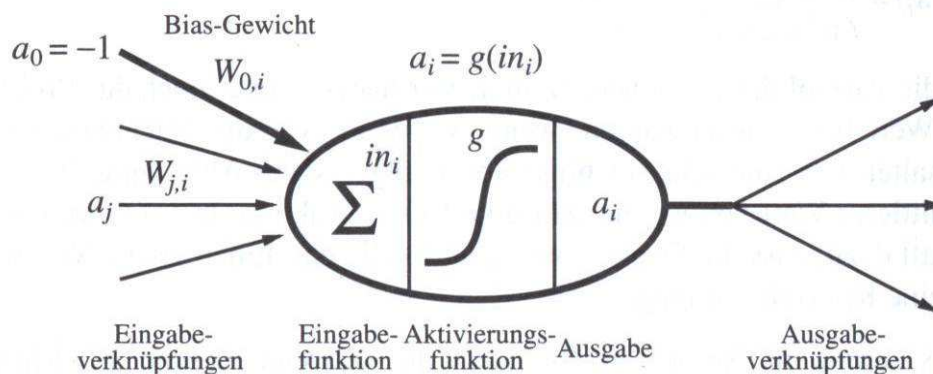
Die obere Grafik zeigt schematisch den Aufbau der Zellstrukturen eines echten Gehirns. Im unteren Bild wird die entsprechende künstliche Nachbildung dargestellt.

Eine echte Nervenzelle besteht vereinfacht gesagt aus Dendriten, einem Zellkörper, einem Axon und Synapsen. Die Dendriten entsprechen praktisch Eingängen über die elektrische Signale anderer Nervenzellen aufgenommen werden können. Der Zellkörper verarbeitet diese Signale und erzeugt daraus einen Ergebniswert, der an das Axon weitergegeben und von diesem über die Synapsen zu weiteren Nervenzellen geleitet wird.



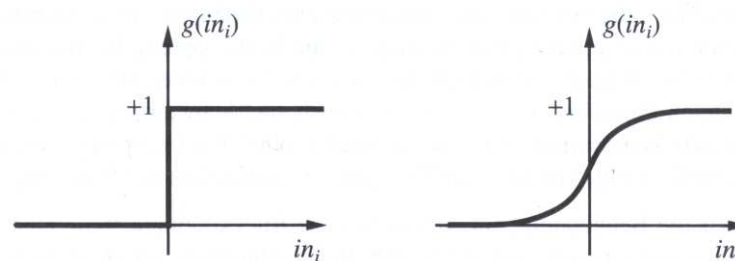


Den Aufbau eines künstlichen Neurons verdeutlicht diese Abbildung, die in [RN] auf Seite 896 zu finden ist.



Entsprechend den Dendriten einer echten Nervenzelle besitzt ein solches künstliches Neuron hier mit  $a_j$  bezeichnete Eingänge zu denen jeweils eine Gewichtung  $W_{j,i}$  gehört. Häufig wird zusätzlich ein *Bias* genannter Eingang mit vorgelegtem Wert hinzugefügt.

Die Eingabefunktion summiert die Werte der Eingänge entsprechend ihrer Gewichtungen und gibt das Ergebnis an eine sogenannte Aktivierungsfunktion. Diese Funktion entscheidet ob der Ausgang des Neurons ein Signal auslöst oder nicht. Prinzipiell können dabei beliebige Funktionen verwendet werden, üblicherweise werden Schwellwerte (Abb. links) oder eine Sigmoid-Funktion (Abb. rechts) verwendet. Die Achse  $in_i$  entspricht der Summe der Eingabewerte, auf der Achse  $g(in_i)$  ist der jeweilige Ausgabewert eingetragen. Die Abbildungen stammen ebenfalls aus [RN].



Bezeichnet man die Aktivierungsfunktion mit  $g$ , so ergibt sich folgende Formel zur Berechnung des Ausgabewertes eines einzelnen Neurons:


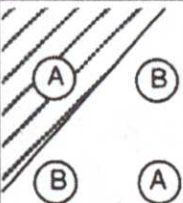
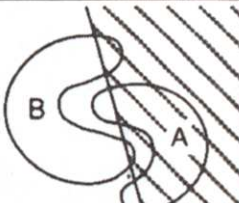
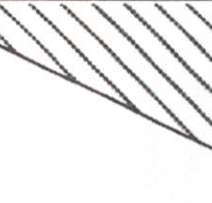

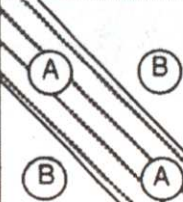
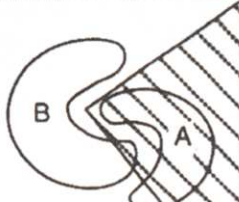


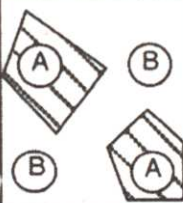


$$a_i = g\left(\sum_{j=0}^n W_{j,i} a_j\right)$$

Die Gewichte werden mit Hilfe eines Lernalgorithmus und der Trainingsdaten während der Lernphase angepasst. Häufig eingesetzt wird hierfür oft der, zum Beispiel in [RN] auf Seite 907 beschriebene *Back-Propagation* Algorithmus, der die zu Beginn meist zufällig initialisierten Gewichtungen  $W_{j,i}$  dadurch anpasst, dass auftretende Fehler über das gesamte Netz „rückwärts“ verfolgt und in jedem Neuron möglichst minimiert werden.

Beim Training künstlicher neuronaler Netze muss berücksichtigt werden, dass eine zu genaue Anpassung eines Netzes an die Trainingsdaten meist nicht gewünscht ist (Überanpassung). Vielmehr wird eine gewisse Ungenauigkeit benötigt, um auch für Datensätze, welche nicht zu den Trainingsdaten gehören, „korrekte“ Ergebnisse zu erzielen (Generalisierung). Einfluss darauf haben zum

einen die Anzahl und Auswahl der Trainingsdaten, die Anzahl der Trainingsdurchläufe, die Anzahl der künstlichen Neuronen sowie Parameter des verwendeten Lernalgorithmus.

Künstliche neuronale Netze können, abhängig von ihrer Architektur, für unterschiedliche Aufgaben eingesetzt werden, darunter Klassifikation und Funktionsannäherung. Einen Überblick dazu gibt [Patterson] auf Seite 204 mit folgender Tabelle. Skizziert sind die jeweils bestmöglichen Ergebnisse für 2 Problemstellungen und eine allgemeine Skizzierung der möglichen Klassifizierung in einem zweidimensionalen Eingaberaum.

Struktur	Entscheidungsbereichstypen	Exklusiv-ODER-Problem	Klassen mit ineinander verflochtenen Bereichen	Allgemeine Bereichsumrisse
Eine Schicht 	Halbebene, begrenzt durch Hyperebene			
Zwei Schichten 	Konvex geöffnete oder geschlossene Bereiche			
Drei Schichten 	Zufällig (Komplexität nur begrenzt durch die Anzahl der Knoten)			

Ohne weitere Vertiefung sei an dieser Stelle bemerkt, dass entsprechend dieser Tabelle bereits mit einem zweischichtigen Feed-Forward Netz beliebige stetige Funktionen approximiert werden können. Mit drei Schichten können sogar unstetige Funktionen abgebildet werden. Ein Feed-Forward Netz erlaubt nur das „Weiterreichen“ von Signalen an „höhere“ Schichten, also nicht an Neuronen der eigenen oder gar einer „früheren“ Schicht. Es gibt weitere Typen von künstlichen neuronalen Netzen mit unterschiedlichster Vernetzung, die unterschiedliche Fähigkeiten und Eigenschaften haben. Oft benötigt man für diese jedoch speziell angepasste Lernalgorithmen und könnte die Aufgabenstellung ebenso mit einem Feed-Forward Netz lösen.

Die Analyse von künstlichen neuronalen Netzen und deren Leistungsfähigkeit ist ein sehr komplexes Thema, Beweise dazu wurden und werden immer noch entwickelt und untersucht. In [Patterson] sind auf Seite 198 ff. verschiedene Beweise zur Abbildungsfähigkeit verschiedener Netze aufgeführt.

### 5.2.4 Reinforcement Learning

Das bereits angesprochene *Reinforcement Learning* bzw. *bestärkende Lernen* benötigt im Unterschied zu den erwähnten überwachten und unüberwachten Lernverfahren prinzipiell keinerlei Anfangsdaten oder Wissen über mögliche Lösungen. Stattdessen ist Wissen über die eigentliche Aufgabenstellung notwendig. Zumindest muss entscheidbar sein, ob ein Problem mit den durchgeführten Aktionen gelöst werden konnte. Diese, vom Zustand des Systems abhängende, Entscheidung wird durch eine, für die jeweilige Aufgabe zu definierende, Bewertungsfunktion getroffen und dient dem Algorithmus dazu, die durchgeführten Aktionen zu bewerten.

Vereinfacht ausgedrückt basiert *Reinforcement Learning* darauf, den Zustand eines Systems (so gut wie möglich) zu kennen und dementsprechende Aktionen auszuwählen und durchzuführen. In der Regel werden mehrere Aktionen in Folge ausgeführt und der Erfolg der gesamten Aktionskette erst am Ende ausgewertet. Dieser Gesamterfolg muss anschließend möglichst genau auf die Einzelaktionen verteilt werden um deren Nutzen zu lernen. Der Lösungsansatz erfordert die Lösung von *Markov-Entscheidungsproblemen*. Die Grundidee basiert auf einer bedingten Wahrscheinlichkeitsverteilung, die es in Kombination mit der *Markov-Annahme*, dass der Zustand eines Systems nur vom unmittelbar vorherigen abhängt, erlaubt die Übergangswahrscheinlichkeiten von einem in andere Zustände anzugeben. Dabei muss der mit „vorherig“ bezeichnete Zustand nicht zwingend zeitlich vorher stattgefunden haben, grundsätzlich sind beispielsweise auch räumliche Sequenzen realisierbar.

Das Auswählen von Aktionen in einem bestimmten Zustand ist einer der Grundbausteine dieses Verfahrens und das dafür nötige Wissen wird durch mehr oder weniger geschickt durchgeführte Aktionen während der Lernphase gesammelt. Gerade in der Art und Weise wie dies geschieht unterscheiden sich die verschiedenen Verfahren des Reinforcement Learning.

Im einfachsten Fall hat ein Experiment wenige diskrete Zustände und in jedem Zustand steht nur eine geringe Anzahl von ebenfalls diskreten Aktionen zur Auswahl. In diesem Fall kann man eine Tabelle aller Kombinationen der möglichen Zustände und Aktionen aufstellen und durch iteratives durchprobieren die beste Lösung finden. Häufig wird ein solches Szenario anhand eines Labyrinths veranschaulicht. Dabei soll ein „Agent“ den schnellsten Weg von einem Start- zu einem Zielpunkt finden. Zerlegt man das Labyrinth in einzelne quadratische Felder können diese jeweils als ein Zustand betrachtet werden. Und definiert man die möglichen Aktionen einfach als Bewegung nach oben, unten, links und rechts, so benötigt man nur noch eine Funktion, die den aktuellen Zustand als Eingabe bekommt und entscheidet ob der Agent das Ziel erreicht hat oder nicht und dem entsprechend eine Bewertung zurückgibt. Diese Funktion kann beim Reinforcement Learning in aller Regel nicht nach jeder einzelnen Aktion ausgewertet werden, sondern zum Beispiel nur bei Erreichen des Ziels, oder nach einer festgelegten Anzahl von Aktionen was zum Problem der Zuordnung von Bewertung zu Einzel-Aktionen führt.

Zusammengefasst lassen sich die Komponenten des Reinforcement Learning also folgendermaßen gliedern:

- Zustände
- Aktionen
- Bewertungsfunktion
- Strategie(-repräsentation)
- Lernalgorithmus

Zustände und Aktionen müssen nicht diskret sein, erfordern jedoch eine geeignete Strategierepräsentation. Im Labyrinth-Beispiel stellt die erwähnte Zustand-Aktions-Tabelle die Strategierepräsentation dar. Diese wird nach und nach durch den Lernalgorithmus so angepasst, dass nach Beendigung des Lernens, durch einfaches „Eingeben“ des Zustands die optimale Aktion abgelesen werden kann. Für kontinuierliche Zustands- oder Aktionsparameter können zum Beispiel die im vorherigen Kapitel beschriebenen künstlichen neuronale Netze eingesetzt werden.

Prinzipiell würde es für einen Lernalgorithmus ausreichen zufällige Aktionen auszuwählen, durchzuführen und sobald eine Belohnung erhalten wird, diese auf die zuletzt durchgeführte Aktionskette jeweils anteilmäßig zu verteilen. Damit erhält man einzelne bewertete Aktionen mit denen man die verwendete Strategie erweitern kann. Es kann allerdings sehr lange dauern bis diese Methode (zufällig) die beste oder eine ausreichend gute Aktionskette findet. Daher wurden und werden geschicktere Verfahren entwickelt, die anhand der bereits gesammelten Erfahrungen versuchen im Voraus die Erwartungswerte neuer Aktionen abzuschätzen und den Aktionsraum gezielt zu explorieren.

Viele gebräuchliche Lernalgorithmen basieren auf dem, unter dem Namen *Q-Learning* bekannten, Verfahren, welches in [Watkins] erstmals im Jahr 1989 vorgestellt wurde. Das Verfahren beruht auf folgender Formel<sup>56</sup>.

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha_t(s_t, a_t)}_{\text{learning rate}} \times \left[ \underbrace{R(s_{t+1})}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})}_{\text{max future value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right]$$

Die Funktion  $Q$  entspricht dabei der Strategierepräsentation. Mit ihrer Hilfe wird für jeden besuchten Zustand und jede dort durchgeführte Aktion die „Qualität“ gespeichert. Diese Funktion wird durch den Algorithmus entsprechend der aufgeführten Formel angepasst.

Mit  $s_t$  wird der Zustand des Systems zum Zeitpunkt  $t$  bezeichnet, mit  $a_t$  die dann durchgeführte Aktion. Der Parameter  $\alpha$  enthält die sogenannte *Learning Rate* und kann, wie in der Formel erkennbar, für verschiedene Paare aus einem Zustand und einer Aktion unterschiedlich definiert werden. Häufig wird jedoch einfach ein konstanter Wert für alle möglichen Paare gewählt. Die Funktion  $R$  stellt die Bewertungsfunktion dar und bildet einen Zustand auf eine skalare Bewertung ab.

<sup>56</sup> Quelle der Formel: <http://en.wikipedia.org/wiki/Q-learning>

Die, der aktuellen Funktion  $Q$  entsprechend, maximal erwartete Bewertung der im nächsten Zustand durchgeführten Aktion, wird mit dem *Discount Factor*  $\gamma$  multipliziert.

Zusammengefasst aktualisiert der Algorithmus also die erwartete Bewertung von Zustand-Aktions-Paaren indem zum alten Wert dessen Differenz zur Bewertung des Zustandes, in den die durchgeführte Aktion das System gebracht hat, plus die zu erwartende Bewertung der Besten der möglichen nächsten Aktionen addiert wird.

Die beiden Parameter haben folgende Auswirkungen:

- *Learning Rate*

Die Learning Rate beschreibt wie stark neue Informationen gegenüber älteren gewichtet werden. Bei einer Learning Rate von 0 wird keinerlei neue Information gelernt, bei einem Wert von 1 würde dagegen nur die aktuelle Information betrachtet und alles bisherige vergessen.

- *Discount Factor*

Hiermit wird beschrieben wie stark zukünftige Belohnungen berücksichtigt werden. Ein Wert von 0 bedeutet hier, dass nur die aktuelle Aktion betrachtet wird. Strebt der Wert gegen 1, so werden spätere Belohnungen immer stärker gewertet und der Algorithmus „sucht“ praktisch nach solchen. Erreicht oder überschreitet dieser Parameter den Wert 1 sind Konvergenzprobleme zu erwarten.

Eine weitere Beschreibung dieses Algorithmus findet sich zum Beispiel auf Seite 408 ff. von [Alpaydin].

Für erweiterte Verfahren gibt es natürlich viele weitere Parameter. Deren Funktionsweise kann in den entsprechenden Veröffentlichungen und Beschreibungen nachgelesen werden. Grundsätzlich sind diese Parameter für den Einsatz der Verfahren von großer Bedeutung, versteht man diese nicht korrekt oder falsch, so hat das Auswirkungen auf Versuchsergebnisse und die entsprechende Auswertung.

Fortgeschrittene Ansätze versuchen über die Gradienten der sogenannten Strategie diese zu verbessern. Ein solches Verfahren ist beispielsweise in [PGPE] und in [MPGPE] nochmals erweitert.

Zur Vertiefung kann [SB] herangezogen werden. Für die Bearbeitung dieser Arbeit lag leider kein Exemplar vor, falls noch nicht vorhanden wäre es eine Überlegung wert dieses Werk über die Bibliothek der Hochschule verfügbar zu machen. Auch wenn es hier nicht konkret verwendet wurde ist es dennoch im Literaturverzeichnis aufgeführt, da viele wissenschaftliche Artikel auf die dort beschriebenen Grundlagen zu *Reinforcement Learning* verweisen.

## 6 Durchführung

Dieses Kapitel beschreibt den Ablauf der Bearbeitung dieser Bachelor-Thesis. Zunächst wird die Planung der bereits angesprochenen Teilaufgaben beschrieben und anschließend auf die konkrete Durchführung eingegangen.

### 6.1 Planung

Entsprechend der Zielsetzung wurden die erforderlichen Arbeiten grob entsprechend folgender Gliederung geplant:

- Recherchen
- Einarbeitung in die Nao Plattform und die Steuerung des Roboters
- Einarbeitung in das Framework PyBrain
- Entwicklung einer Codebasis um den Nao auf bequeme Art in Kombination mit PyBrain verwenden zu können
- Durchführung von Versuchen
- Auswertung und Erstellen des vorliegenden Dokumentes

Bei den vorbereitenden Recherchen wurde hauptsächlich nach Beschreibungen von Projekten oder Verfahren gesucht, die sich mit Robotern und maschinellem Lernen beschäftigen. Die Papers und Artikel sind auf Seite 67 ff. gelistet. Aufgrund ihrer Anzahl wird auf eine detaillierte Beschreibung verzichtet. Besonders interessant waren darunter folgende Artikel:

- [RMAX]: Verfahren mit dessen Hilfe ein Nao lernt Elfmeter zu schießen
- [RC]: Beschreibung eines Robotersystems das Bälle aus der Luft fangen kann
- [Thorne]: Einführung in die Bildverarbeitung mit Python.
- [Sanchez]: Eine Master-Thesis in Bezug auf Artificial Vision mit der Nao Plattform

Die Einarbeitung in die Nao Plattform wurde anfangs relativ frei gestaltet. Das bedeutet, dass ohne feste Zielsetzungen die Möglichkeiten des Roboters ausprobiert wurden. Dabei wurde anfangs vor allem das Programm Choregraphe verwendet und, nachdem die grundsätzlichen Möglichkeiten bekannt waren, zum Development Kit NAOqi SDK gewechselt.

Um gezielt bestimmte Bewegungen des Roboters ausführen und einschätzen zu können kam irgendwann die Idee auf, den Roboter, ähnlich wie ein fernsteuerbares Auto, mit Hilfe eines Gamepads zu steuern. Diese Idee erwies sich als interessant und wurde erfolgreich umgesetzt. Eine knapp gehaltene Beschreibung der entwickelten Möglichkeiten findet sich in Anhang B.

Der nächste Schritt war, die Steuerungsmethoden für den Roboter mit PyBrain zusammen zu bringen und entsprechend der Teilaufgaben Versuche zu entwickeln und durchzuführen.

## 6.2 Prototypen & Versuche

Ein Großteil der verwendeten Arbeitszeit wurde zur Entwicklung der Steuerungsmodule des Roboters sowie zur Vorbereitung und Durchführung von Versuchen mit verschiedenen Algorithmen von PyBrain aufgewendet.

### 6.2.1 Steuerung des Nao

Um einen Roboter mit Hilfe eines Gamepads zu steuern muss dieses zunächst in ein eigenes Programm eingebunden werden. Hierfür gibt es für die meisten Programmiersprachen mittlerweile fertige Libraries, für diese Arbeit wurde dabei Pygame benutzt.

Damit bei manueller Steuerung flüssige<sup>57</sup> Bewegungen durchgeführt werden können, müssen pro Sekunde viele einzelne Bewegungen ausgeführt werden. Ein zeitlicher Abstand von einer Sekunde wäre viel zu langsam. Bei manueller Steuerung müssen viele Bewegungen nachkorrigiert werden und nach jedem Bewegungskommando müsste man vergleichsweise lange warten, bis das nächste Kommando abgesetzt werden kann. Mit 20-30 Abfragen pro Sekunde erreicht man bereits relativ flüssige Bewegungen, entsprechend müssen natürlich auch die jeweiligen Kommandos so angepasst werden, dass die benötigte Zeit zur Durchführung nicht zu lang ist.<sup>58</sup> Die Möglichkeit 50 Kommandos pro Sekunde zu geben verliert ihren Sinn, wenn das erste Kommando zu lange braucht bis es durchgeführt ist und die folgenden entsprechend verworfen oder (noch schlimmer) verzögert durchgeführt werden.

Erste Ansätze in einem Simulator funktionierten ohne weitere Maßnahmen zufriedenstellend. Bei den ersten Versuchen am realen Nao stellte sich jedoch heraus, dass die Bedingungen im Simulator zu perfekt und in der Realität zusätzliche Maßnahmen erforderlich sind. Im Simulator spielt das Gewicht einzelner Teile des Roboters so gut wie keine Rolle, beim realen Nao kommen allerdings zum Gewicht noch kleine Ungenauigkeiten in der Mechanik der Gelenke dazu. Zusammen bedeutet dies, zum Beispiel das Kommando für eine Armbewegung nach oben, wird zwar korrekt durchgeführt, allerdings sackt der Arm minimal ab sobald die Bewegung abgeschlossen ist und auf das nächste Kommando gewartet wird bzw. der Arm in dieser Position gehalten werden soll. Verwendet man einfach bei jedem Abfragen (also 20-30 mal pro Sekunde) immer die aktuellen Sensorwerte um das nötige Kommando für eine Bewegung zu berechnen und setzt den neuen Wert als Kommando ab, so wird der Arm langsam aber sicher immer weiter sinken.

Der Grund hierfür liegt darin, dass das neue Kommando dann praktisch immer der, vom eigentlich gewünschten Wert abweichende Sensorwert ist. Der Effekt wiederholt sich bis der Einfluss der Schwerkraft nicht mehr groß genug ist.

Das Problem kann zunächst auf einfache Art gelöst werden. Prüft man jeweils, ob überhaupt eine Veränderung der aktuellen Position gewünscht ist, also ein Eingabewert ungleich 0 gelesen wurde, und berechnet nur dann einen neuen Wert und setzt diesen nur dann als Kommando ab, wird das Sinken des Arms bereits zuverlässig verhindert.

Dennoch ist diese Lösung nicht grundsätzlich für alle Szenarien ideal. Das eigentliche Problem ist

---

<sup>57</sup> Zumindest von Menschen als flüssig wahrgenommen

<sup>58</sup> Also je kürzer die verfügbare Zeit, umso kleinere Bewegungen.

auf diese Weise nicht wirklich gelöst, sondern nur soweit reduziert, dass es praktisch nicht in dem Maße auffällig ist, wie ohne jegliche Maßnahmen. Eine vollständigere Lösung besteht darin, dass man sich von den Sensorwerten trennt und diese gar nicht, oder nur beim Einschalten des Systems verwendet und ansonsten intern den aktuell gewünschten Wert für alle Gelenke speichert. Auch diese Methode ist nicht unbedingt in allen Fällen ideal, je nachdem was benötigt wird kann man eine oder beide der Maßnahmen verwenden. Weitere Möglichkeiten sind selbstverständlich nicht ausgeschlossen.

Die Funktionen des Nao, welche von den genannten Proxies bereitgestellt werden, wurden mit Hilfe einer Schnittstelle gekapselt. Diese Schnittstelle baut praktisch eine Verbindung zum Nao auf und bietet einheitliche Methoden an, die wiederum die Funktionen der Nao Proxies aufrufen.

Dieses Konzept erlaubt eine einfache Zuordnung von Funktionen des Naos zu den Eingabemöglichkeiten des Gamepads bzw. deren Einbindung in Versuche mit PyBrain.

### 6.2.2 Optische Lokalisierung einer Kugel

Da die Teilaufgaben darauf basieren, Informationen über die Position einer roten Kugel zu ermitteln (Lokalisierung) oder diese zur Verarbeitung benötigen, war eine brauchbare Erkennung der Kugel eine der Grundvoraussetzungen. Der Nao bringt bereits eine eigene Methode mit, diese ist im Simulator allerdings aktuell nicht verfügbar und eine Beschreibung des Algorithmus konnte in der Dokumentation der Plattform nicht gefunden werden.<sup>59</sup> Daher wurde eine entsprechende Alternative benötigt.

Die erste Idee war, bereits fertige Methoden zu verwenden, hier bot sich das Framework OpenCV an. Allerdings führten Versuche damit zunächst nicht zu Ergebnissen der gewünschten und erforderlichen Stabilität (kleine Änderungen der Situation führten zu großen Änderungen des Ergebnisses). Die Verwendung von OpenCV für diesen Zweck sollte jedoch nicht ausgeschlossen werden. Im Rahmen dieser Bachelor-Thesis wurde die Entscheidung getroffen die Verwendung von OpenCV nicht zum Schwerpunkt zu machen und nach einigen wenig erfolgreichen Versuchen nach einer weiteren Alternative gesucht.

Die Möglichkeit zur Erkennung von Kugeln in einer echten Umgebung ist von mehreren Parametern abhängig, nicht zuletzt von der Beleuchtungssituation. Tageslicht und künstliches Licht haben Einfluss darauf, wie Farben von der Kamera des Roboters erkannt werden und in einer Situation sehr gut funktionierende Farbfilter können in einer anderen nahezu völlig versagen. Daher wurde zunächst für den Simulator eine sehr einfache (und rechenintensive) Methode implementiert. Diese liefert stabile Ergebnisse solange das Szenario ausser der roten Kugel keine weiteren roten Objekte enthält. Die Methode führt zunächst eine Farbfilterung durch, und ermittelt einen möglichen Mittelpunkt mit Hilfe des Mittelwertes aller gefundenen roten Pixel. Damit wird klar, dass die Methode versagt, sobald rote Pixel auftauchen, die nicht zur roten Kugel gehören. Im Simulator lässt sich dies jedoch mehr oder weniger einfach erreichen, indem man sicherstellt, dass kein anderes Objekt<sup>60</sup> eine rote Farbe hat.

---

<sup>59</sup> Der Algorithmus kann möglicherweise im NAOqi SDK direkt eingesehen werden.

<sup>60</sup> Besonders der rot-schwarz gekachelte Fußboden von Webots sollte zum Beispiel in blau-schwarz geändert werden.



Anschließend wird für jeden Pixel der Abstand zum ermittelten Mittelpunkt berechnet und die Anzahl der Pixel für jeden berechneten Abstand gezählt. Der Abstand mit den meisten Treffern wird als Radius des Kreises übernommen. Idealerweise sollte der Kreisumfang berücksichtigt werden und eher das Verhältnis von Treffern auf dem möglichen Kreis zur Anzahl der Pixel auf dem Kreis, die nicht „passen“, verwendet werden um den besten Radius zu ermitteln. Auch ohne diese Verbesserung lieferte dieser Ansatz jedoch zuverlässig den Mittelpunkt und den Radius einer im Bild vorhandenen roten Kugel.

### 6.2.3 Lernversuche mit PyBrain

Mit PyBrain wurden zahlreiche Versuche unterschiedlichster Art durchgeführt, auch um das Framework zunächst kennen zu lernen und mit einzelnen Methoden vertraut zu werden.

Einer der ersten Versuche bestand daraus, einem künstlichen neuronalen Netz eine mathematische Funktion beizubringen. Hierfür wurde kurzerhand die Formel

$$a^2 + b^2 = c^2$$

verwendet. Nach einigen Versuchen gelang es endlich auch Eingaben die nicht in der Trainingsmenge enthalten waren mehr oder weniger gut auf die korrekten Ergebnisse abzubilden. Insgesamt waren dazu allerdings sehr viele Trainingsdatensätze nötig<sup>61</sup> und daher wurde die ursprüngliche Idee verworfen, den Kopf des Nao mit Hilfe eines künstlichen neuronalen Netzes so auszurichten, dass die rote Kugel in der Bildmitte erscheint. Stattdessen wurde dafür eine statische Lösung, nach dem Prinzip „zu weit links? → drehe Kopf ein Stück nach rechts“ implementiert. Nach ein wenig Ausprobieren funktionierte diese Lösung zwar, allerdings ohne „intelligentes“ oder lernendes Verhalten irgendeiner Art. Nach diesem Prinzip lässt sich dennoch einiges erreichen, so kann man den Roboter auf diese Weise durchaus quer durch einen Raum manövrieren lassen, bis er sich in einem bestimmten Abstand und Winkel zur roten Kugel befindet. Dies wurde getestet und funktionierte, erfüllte jedoch leider nicht den Anspruch an lernendes oder intelligentes Verhalten, ganz abgesehen von den geplanten Verfahren des Reinforcement Learning.

Daher zielten die nächsten Schritte direkt darauf ab zunächst irgendetwas mit Lernalgorithmen umzusetzen. Es stellte sich heraus, dass dies nicht so einfach wie erhofft realisierbar war. Dafür gibt es mehrere Gründe, einer davon betrifft die Normalisierung von Sensordaten durch PyBrain. Um generell flexibel zu sein bietet das Framework die Möglichkeit, Eingabe und Ausgabe der Lernverfahren zu normalisieren, also auf einen festen Wertebereich abzubilden. Das erspart interne Abhängigkeiten von Eigenschaften der externen Umgebung, die für jeden speziellen Fall angepasst werden müssten. Einer der einfachsten durchgeführten Versuche hatte das Ziel, einen Roboterarm einfach in eine bestimmte Position zu bringen. Bereits hier zeigten sich seltsame Effekte, wahrscheinlich zurückzuführen auf eben genannte Normalisierung. Dabei wurden immer wieder gleiche Abläufe durchgeführt, aus denen keinerlei neue Information generiert werden kann. Erst das Deaktivieren der Normalisierung der Sensordaten durch PyBrain brachte eine Änderung und danach war der Algorithmus in der Lage zumindest für ein einzelnes Gelenk die gewünschte Position zu finden und zu lernen, wie dieses ausgerichtet sein soll.

---

<sup>61</sup> Dieser Versuch war einer der ersten von mir mit künstlichen neuronalen Netzen durchgeführten. Es ist zu erwarten, dass deutlich bessere Ergebnisse möglich sind.

Grundsätzlich ist es nicht zu empfehlen auf die Normalisierung zu verzichten, bei deren Deaktivierung muss man sich selbst darum kümmern, dass passende Werte und Wertebereiche verwendet werden. Die genauen Ursachen weshalb sich die Normalisierung derartig auswirkte konnten leider nicht ermittelt werden. In den betreffenden Zeilen von PyBrain konnte kein Fehler gefunden werden.

Einige Versuche zielten darauf ab, die ursprünglich geplante Greifbewegung zu realisieren, verwertbare Ergebnisse wurden jedoch dabei bisher leider nicht erreicht. Manuelle Versuche zeigten, dass es für den Nao äusserst schwierig ist Kugeln sicher zu greifen. Einfacher ist dies beispielsweise mit einem kleinen Gummwürfel, dabei wird allerdings die Lokalisierung des Objektes bedeutend komplizierter, da ein Würfel, im Gegensatz zu einer Kugel nicht „invariant bezüglich Rotation“ ist und somit dessen exakte Ausrichtung, bzw. deren Erkennung, eine tragende Rolle spielt. Ausserdem zeigte sich im Laufe der Bearbeitung, dass die Modellierung der Hand des Nao im Simulator Webots bisher unzureichend ist. Optisch sieht die Hand gut aus, auch das Öffnen und Schließen funktioniert zuverlässig. Entweder versagt die Kollisionserkennung oder die Physikengine von Webots ist nicht ausgereift genug. Jedenfalls ist es nicht möglich, dass die Finger des Roboters auf realistische Art ein Objekt „umgreifen“, stattdessen greifen die Finger durch das Objekt. Es kommt kein richtiger Kontakt zustande und erfolgreiches Greifen ist auch manuell kaum möglich. Daher wurde die Komplexität der Aufgabenstellung reduziert und gegen Ende der Bearbeitungsphase konzentrierten sich die Versuche darauf, mit Hilfe von Lernverfahren den Kopf des Nao so auszurichten, dass die Kugel sich in der Bildmitte befindet. Nach anfangs ziemlich chaotisch wirkenden Versuchen konnte zum Ende hin tatsächlich noch erreicht werden, dass dies mit Hilfe des in [PGPE] beschriebenen Lernverfahrens, wenigstens in Grundzügen, funktionierte.

Der Roboter war am Ende in der Lage bereits nach 10-20 Versuchen den Kopf so zu bewegen, dass sich die Kugel zumindest vorübergehend in der Bildmitte befand. Leider stoppt die Bewegung dann nicht sondern der Kopf wird weitergedreht. Dies lässt sich zwar einfach unterbinden indem man bei Erreichen des Ziels abbricht. Das Abbrechen einer Bewegung bei Erreichen eines „Zielzustands“ sollte jedoch grundsätzlich auch erlernbar sein.

Bei der Durchführung der Versuche wurden verschiedenste Variationen getestet, darunter verschiedene Belohnungsfunktionen, verschiedene Steuerkommandos und verschiedene Versuchsparameter.

Mit Versuchsparameter sind hier zum Beispiel die Anzahl der Einzelaktionen einer Aktionskette oder der Ausgangszustand gemeint. Beim Lernen ist das eigentliche Ziel nicht, aus einem bestimmten Zustand einen Zielzustand zu erreichen, sondern eher aus einem beliebigen Zustand durch Ausführung möglichst (weniger) optimaler Aktionen. Dafür wurde versuchsweise nach jedem abgeschlossenen Versuch ein zufälliger Ausgangszustand erzeugt und mit diesem erneut begonnen.

Dieses Vorgehen ließ kaum noch direkte Fortschritte erkennen und wurde daher vorerst nicht weiter verfolgt. Es ist zu erwarten, dass dieses Lernziel erheblich längere Versuchsreihen als mit fixem Ausgangszustand erfordert. Langfristig sollte man sich auch mit dieser Thematik beschäftigen, ansonsten ist durch den Einsatz der Lernverfahren kein bedeutender Vorteil zu erwarten und statische<sup>62</sup> Lösungen können vergleichbare Ergebnisse mit deutlich weniger Aufwand erreichen.

---

<sup>62</sup> Gemeint sind fest einprogrammierte Regeln wie auf der vorherigen Seite beschrieben

Eine Variante der verwendeten Belohnungsfunktion prüfte zum Beispiel nur ob der Zielzustand erreicht ist oder nicht, und gab nur in diesem Fall eine Belohnung zurück. Eine andere verwendet den Abstand zum Zielzustand um eine Bewertung zu finden, je näher, umso höher ist die Belohnung. Eine weitere Variante merkt sich den letzten Abstand und vergleicht den aktuellen mit diesem. Eine Verringerung bewirkt eine Belohnung, eine Vergrößerung hat eine Belohnung von 0 oder gar eine Bestrafung<sup>63</sup> zur Folge.

Auch die Steuerung kann man auf verschiedene Weise gestalten, man kann direkte Winkel vorgeben. Damit kann der Nao innerhalb von wenigen Aktionen komplexe und größere Bewegungen durchführen. Eine Alternative verwendet keine direkten Winkel, sondern kleine Schritte. Damit ist es nötig viele ähnliche Kommandos in Folge abzusetzen um eine größere Bewegung durchzuführen. Subjektiv war es mit kleinen Schritten einfacher erkennbare Ergebnisse in kurzer Zeit zu bekommen.

Weitere Variationen betrafen die Parameter der Lernumgebung selbst. Es wurden einige der in PyBrain verfügbaren Verfahren getestet, neben dem erwähnten PGPE wurden weitere Methoden kurzzeitig ausprobiert, jedoch nicht vertieft da auch deren Ergebnisse nicht bedeutend besser waren.

Darüber hinaus wurden testweise auch interne Parameter des PGPE-Algorithmus variiert, ohne damit jedoch Verbesserungen zu erreichen.

Man kann in PyBrain die Art der Aktivierungsfunktion der Ausgabeschicht von verwendeten künstlichen neuronalen Netzen anpassen. Verfügbar sind mindestens: Sigmoid, Softmax und Tangens Hyperbolicus. Die verwendete Funktion hat deutlichen Einfluss darauf, auf welche Weise der den (bisher nicht untersuchte) Zustandsraum durchsucht wird. Welche Funktion für welchen Versuch am besten funktioniert konnte bisher nicht endgültig geklärt werden.

Für die meisten durchgeführten Versuche war es nicht erforderlich in der simulierten Welt einen Reset durchzuführen, diese also in den Ausgangszustand zu bringen. Dies ist nur dann erforderlich, wenn der Roboter mit seiner Umgebung interagiert. Da der Nao bei den meisten Versuchen nur seinen eigenen Kopf bewegte, reichte es meist aus den Kopf in eine definierte (auch zufällige) Position zu bringen bevor der nächste Durchlauf gestartet wurde. Eine Möglichkeit für einen vollständigen Reset konnte jedoch gefunden werden. Entweder kann man dafür den Supervisor der Webots PRO Version benutzen, oder man setzt eine Library ein, die es erlaubt den Mauszeiger zu steuern und gezielt Klicks auszulösen. Damit „klickt“ man einfach den entsprechenden Reset-Button der Webots-Oberfläche. Dafür muss allerdings der Simulator auf demselben Rechner laufen, auf dem auch der eigene Code gestartet wird. Alternativ kann man einen entsprechenden kleinen Server schreiben, der über das Netzwerk gesteuert werden kann und auf dem Rechner mit dem Simulator läuft.

---

<sup>63</sup> Negative Belohnung

## 7 Ergebnisse

Dieses Kapitel reflektiert die durchgeführten Aufgaben und die dabei gesammelten Erfahrungen nochmals in zusammengefasster Form. Zunächst bezogen auf die bearbeiteten Themen, anschließend in Bezug auf die geplanten Teilaufgaben. Abschließend folgt das Fazit mit Ausblick.

### 7.1 Auswertung

Die Themen der Arbeit lassen sich rückblickend in 4 größere Bereiche aufteilen. Zum einen die Evaluierung der Nao Plattform als Ganzes, die Entwicklung der Steuerungsmodule, Robotersimulation sowie Machine Learning.

#### 7.1.1 Evaluierung der Nao Plattform

Ein Ziel dieser Arbeit war es einen möglichst umfassenden Überblick der Nao Plattform zu gewinnen. Es wurden einige positive, allerdings auch negative Erkenntnisse gesammelt.

Positiv ist vor allem das Design der Plattform insgesamt. Dieses ist gut durchdacht und erlaubt die zukünftige Erweiterung mit Fähigkeiten jeglicher Art. Dementsprechend bietet die Plattform von Haus aus bereits einen großen Funktionsumfang und die Grundlagen für viele denkbare Versuche stehen zur Verfügung.

Die Möglichkeiten des Development Kits sind umfangreich und erlauben das Programmieren von Funktionen entweder so, dass diese entweder auf dem Roboter direkt, oder mit Hilfe eines leistungsstarken PCs berechnet werden können. Die dafür nötige Kommunikationsschnittstelle wird über eine herkömmliche Netzwerkverbindung bereitgestellt.

Die meisten Schwächen der Plattform werden hauptsächlich durch die verwendete Hardware verursacht. Wenn man bedenkt was, auf einzelne Aufgaben spezialisierte, Roboter an Entwicklungs- und finanziellem Aufwand erfordern, wird nachvollziehbar, dass bei in Serie produzierten, möglichst universellen Robotern immer Abstriche gemacht werden müssen. Schade ist, dass beim Nao besonders die Umsetzung der Roboterhand davon betroffen ist und diese nur bedingt für fortgeschrittene Aufgaben geeignet ist, da sie nur über 3 Finger, die nicht unabhängig voneinander bewegt werden können, verfügt.

Neben den Einschränkungen der Hand fiel vor allem auf, dass die Gelenke praktisch über keine Federung verfügen und damit jegliche Krafteinwirkung die Mechanik direkt belastet. Das ist nicht unbedingt ein großer Nachteil, macht sich aber zum Beispiel beim Laufen bemerkbar.

Des Weiteren zeigte sich, dass vor allem die Motoren der Schultern und Knie relativ stark belastet werden. Daher neigen diese recht schnell zum Überhitzen und bei der Planung von Versuchen sollte versucht werden diese Gelenke, wann immer möglich, zu entlasten.

Damit kann die Plattform für viele verschiedene Versuche eingesetzt werden solange die Hardware des Roboters die Grundvoraussetzungen für die Bewegungsabläufe erfüllt. Für einen Einstieg in die Welt der Robotik ist der Nao auf jeden Fall geeignet und die Arbeit mit diesem erlaubt es die Anforderungen an komplexere Aufgaben zu erkennen und abzuschätzen.

### 7.1.2 Steuerungsmodule

Die entwickelten Softwaremodule konnten die zur Bearbeitung von Versuchen des maschinellen Lernens nötigen Funktionen bereitstellen und wurden im Laufe der Zeit, je nach Bedarf weiterentwickelt. Eine vollständige Unterstützung aller Funktionen des Nao war für diese Arbeit nicht notwendig, und so wurden bisher nur die erforderlichen implementiert. Grundsätzlich ist die dabei entstandene Architektur frei erweiterbar.

Der Einsatz dieser Schnittstelle zum Roboter macht dessen Steuerung deutlich bequemer, so muss etwa nicht jeder Proxy „manuell“ initialisiert werden. Dies wird durch die jeweiligen Module durchgeführt und erlaubt eine Kapselung der Komplexität des NAOqi SDK. Das Verbinden zum Nao und dessen anschließende Steuerung kann damit mittels weniger einzelner Kommandos durchgeführt werden.

Die Module sind sicherlich nicht 100% ohne Fehler und einzelne Teile des Designs sollten möglicherweise verbessert und vervollständigt werden. Da das Entwickeln einer kompletten Schnittstelle jedoch kein direktes Ziel dieser Arbeit war, wurde der Schwerpunkt der Bearbeitung in Richtung von Lernversuchen verlagert, nachdem die gewünschte Grundfunktionalität implementiert war.

Denkbar ist natürlich auch der Verzicht auf eine solche Kapselung, je nach Aufgabenstellung ist es nicht erforderlich den Zugriff zum Nao zu vereinfachen. Um die Plattform kennenzulernen erschien es dennoch sinnvoll solche Module selbst zu schreiben und dabei Erfahrungen mit dem NAOqi SDK zu sammeln.

Die Steuerung des Roboters mittels eines Gamepads erlaubt es hardwarebedingte Schwierigkeiten des Roboters zu erkennen und einzuschätzen ob bestimmte Versuche überhaupt erfolgreich umgesetzt werden können.

### 7.1.3 Robotersimulation

Versuche aus dem Bereich des maschinellen Lernens erfordern in aller Regel umfangreiche Versuchsreihen und das wiederholte Durchführen von Versuchen mit möglichst identischen Ausgangsbedingungen. Da dies in der Realität mit hohem Aufwand verbunden ist, ist eine Möglichkeit zur Simulation eine fast unumgängliche Voraussetzung.

Der getestete Simulator ist nicht ausschließlich zur Simulation von Naos entwickelt, vielmehr soll mit dieser Software eine universelle Umgebung bereitgestellt werden, für die auch eigene Robotermodelle erstellt werden können. Das erlaubt folglich auch die Erweiterung und Verbesserung des bereits vorhandenen Nao Modells. Der Support des Herstellers reagierte prompt auf Anfragen und machte einen sehr bemühten Eindruck. Das lässt erwarten, dass es mit entsprechendem Aufwand möglich ist, etwaige Unzulänglichkeiten des simulierten Roboters zu beseitigen.

Im Rahmen dieser Arbeit wurde mit Trial-Versionen von Webots gearbeitet, der Erwerb einer EDU Lizenz durch die Hochschule, wäre vermutlich eine lohnende Investition. Falls größere und komplexere Versuchsreihen durchgeführt werden sollen bleibt zu erwägen ob sich nicht sogar der Erwerb der PRO Version lohnt. Alternativ bleibt die Möglichkeit den Ausgangszustand über die gra-

fische Oberfläche wiederherzustellen bestehen. Dazu muss ein Framework<sup>64</sup> benutzt werden, das die Steuerung der Maus des PCs durch Programme erlaubt. Versuchsweise wurde diese Möglichkeit erfolgreich eingesetzt.

#### 7.1.4 Machine Learning

Zu den durchgeführten Lernversuchen ist in erster Linie zu sagen, dass man idealerweise über umfangreiche Erfahrungen mit Methoden des maschinellen Lernens verfügen sollte bevor verwertbare Ergebnisse erwartet werden können. Diese Erfahrung muss selbstverständlich nicht zwingend im Zusammenhang mit einem echten Roboter gesammelt werden. Vielmehr geht es um das grundlegende Verständnis der Funktionsweise der einzusetzenden Methoden. Diese Erkenntnis war zu erwarten und bedeutet für Projekte in diese Richtung, dass entweder erfahrene Ansprechpartner verfügbar sein sollten oder ein beträchtlicher Teil der aufgewendeten Zeit für die Einarbeitung in das Thema Machine Learning eingeplant werden muss. Das Thema Machine Learning ist zu umfangreich als, dass innerhalb von wenigen Tagen ein umfassender Überblick über die nötigen Erfahrungen gesammelt werden und konkrete Entscheidungen für bestimmte Methoden oder Versuchsdetails getroffen werden könnten.

So hat sich gezeigt, dass die anfänglich geplanten Teilaufgaben auch aufgrund von fehlenden Erfahrungen im Bereich Machine Learning sehr schwierig umzusetzen sind. Nichts desto trotz sollte man sich davon nicht entmutigen lassen und sich grundsätzlich auch an ambitionierten Aufgaben versuchen. Dabei können viele wertvolle Erfahrungen gesammelt werden, auch wenn eine vollständig erfolgreiche Umsetzung dann häufig nicht erwartet werden kann.

Grundsätzlich kann man zu maschinellem Lernen sagen, dass möglichst genaue Kenntnis von Details der geplanten Verfahren und eine umfassende Analyse der jeweiligen Problemstellung notwendig sind um die Komplexität des Lösungsansatzes möglichst gering halten zu können. Zum Beispiel kann es sinnvoll sein zu lernende Bewegungen nicht immer aus vielen einzelnen zusammen zu setzen, sondern möglicherweise eine andere Darstellung zu suchen und dadurch die Anzahl der Parameter deutlich zu reduzieren. Trajektorien wären eine mögliche Alternative, welche die Komplexität der Beschreibung von Bewegungen deutlich reduzieren kann.

Die Erweiterung bestehender Lernverfahren ist natürlich möglich, allerdings nicht einfach und ohne vollständiges Verständnis der Ausgangsbasis von vornherein zum Scheitern verurteilt. In diese Richtung sollte man sich entsprechend erst dann wagen, wenn die ursprüngliche Methode (für passende Aufgaben) erfolgreich angewendet werden konnte.

---

<sup>64</sup> Beispielsweise „pywin32“, verfügbar unter: <http://sourceforge.net/projects/pywin32/files/pywin32/>

### 7.1.5 Auswertung der geplanten Teilaufgaben

Zu guter Letzt bleibt die Auswertung der anfangs geplanten Teilaufgaben.

Obwohl insgesamt viel Zeit investiert und umfangreiche Versuche durchgeführt wurden, konnten nicht alle Aufgaben bearbeitet werden. Die Gründe hierfür liegen zu einem großen Teil darin, dass die Ziele zu hoch gesetzt waren. Da die geplanten Ziele grundsätzlich der Einarbeitung in die Plattform dienen sollten ist dies nicht weiter tragisch. Um eine breite Evaluierung der Nao Plattform in der verfügbaren Zeit durchführen zu können wurden vor allem zu Beginn nicht alle Versuche vertieft und schlechten Ergebnisse von Methoden aus dem Bereich der künstlichen Intelligenz und der Objekterkennung nicht auf den Grund gegangen. Mit stärkerem Fokus auf einzelne Teilbereiche wäre dies jedoch möglich und sinnvoll gewesen.

Künftige Projekte sollten die Ziele daher am besten etwas niedriger ansetzen und sich auf die Lösung von einzelnen Teilaufgaben konzentrieren. Sollte dabei ein Ziel „zu schnell“ erreicht werden, können Teilaufgaben jederzeit erweitert und vertieft werden. Bei der Arbeit mit dem Nao entstehen viele Ideen, die unterschiedlichste Lösungsansätze erlauben. Zu hohe Ziele können nach erfolglosen Versuchen zu Frustration führen, was vor allem die Arbeit im Team erschweren kann. Dies ist jedoch grundsätzlich so und nicht speziell für den Bereich Robotik oder maschinelles Lernen gültig.

Die abschließende Auswertung der Teilaufgaben (des maschinellen Lernens) muss leider festhalten, dass diese nur ansatzweise gelöst werden konnten. Die Gründe dafür wurden den entsprechenden Stellen genannt. Der grundsätzliche Anspruch der Arbeit war jedoch nicht die Lösung der Teilaufgaben, sondern das Sammeln von Erfahrungen und Grundlagen. Diese wurden hier beschrieben und dürften für weitere Projekte mit dem Nao verwertbar sein.

Die gesammelten Erfahrungen erlauben die Einschätzung, dass die Teilaufgaben grundsätzlich zwar lösbar sind, im Detail jedoch die Grenzen der Nao Plattform erreichen. Eine Einschätzung der Grenzen und Einschränkungen der Nao Plattform sollte ist mit den Ausführungen dieser Arbeit möglich.

## 7.2 Fazit und Ausblick

In erster Linie ist zu erwähnen, dass die Arbeit mit dem Nao in aller Regel Spaß machte. Die meisten Probleme oder Abweichungen zur anfänglichen Planung sind weniger auf Unzulänglichkeiten der Nao Plattform zurückzuführen, als vielmehr entweder auf Einschränkungen anderer Software oder schlicht zu hoch gesetzte Ziele. Besonders für die eigentlich anvisierten Lernverfahren sollte man sehr viel Zeit einplanen und die entsprechenden Lernaufgaben dürfen nicht zu anspruchsvoll sein. Mit Versuchen dazu könnte man sicherlich ohne Weiteres nochmals das Doppelte und mehr an Zeit verbringen.

Grundsätzlich wurde jedoch eine Basis geschaffen, die es erlaubt den Nao und das Framework PyBrain miteinander zu verwenden und entsprechende Lernverfahren anzuwenden oder zu entwickeln. Bei der Entwicklung dieser Basis wurden viele Erfahrungen mit dem Nao gesammelt und die Formulierung der Ziele dieser Arbeit würde mit im Nachhinein sicher etwas anders ausfallen.

Aufgrund des Umfangs der Thematik ist eine gewisse Einarbeitung unumgänglich. Das sollte, falls man noch keine Erfahrungen mit maschinellem Lernen, Robotern oder Robotersimulatoren besitzt, fest eingeplant werden. Andererseits bedeutet dies aber auch, dass man durch ähnliche Projekte viele verschiedenen Themen untersuchen und dabei entsprechend breites Wissen dabei sammeln kann.

Künftige Projekte können auf den bearbeiteten Grundlagen aufbauen und zum Beispiel die entwickelte Schnittstelle zum Nao vervollständigen, weiterentwickeln und verbessern. Ebenso kann der vorhandene Quellcode natürlich auch einfach nur dazu benutzt werden um die Funktionsweise von Methoden für den Zugriff über das NAOqi SDK nachzuschlagen und ansonsten unabhängig davon mit eigenem Code weiterzuarbeiten.

Da sich herausgestellt hat, dass die Hand des Nao nur sehr einfaches Greifen erlaubt, sollte bei der Definition von Aufgabenstellungen insbesondere darauf geachtet werden, dass der Nao selbst am Ende nicht zum Hindernis wird. Da der Nao beim RoboCup erfolgreich als Standardplattform eingesetzt wird, liegt es auf der Hand Projekte in diese Richtung zu orientieren und zum Beispiel das Greifen und Werfen, wie es in dieser Arbeit geplant war, durch „kicken“ mit dem Fuß zu ersetzen. Im Bereich Fußball gibt es viele Situationen, die eine für maschinelles Lernen geeignete Aufgabenstellung bieten. Denkbar wären unter anderem Elfmeterschießen, Zweikämpfe, Pässe, Steuerung eines Torwarts, und viele mehr.

Im Zuge eines Masterstudiums an der HdM werde ich als Ansprechpartner für weitere Projekte zur Verfügung stehen können und eventuell selbst in Zukunft nochmals mit dem Nao das ein oder andere Projekt bearbeiten oder die hier vorbereiteten Ansätze ausbauen.<sup>65</sup>

---

<sup>65</sup> Bei Bedarf kann ich entweder über (db054@hdm-stuttgart.de) oder (davidbertram@gmx.de) kontaktiert werden.



## Anhang A: Inhaltsbezogene Anhänge

Hier finden sich Ergänzungen, Hintergründe, Quellenauszüge die in dieser Arbeit angesprochen, jedoch nicht vertieft wurden und dennoch interessante Zusatzinformationen enthalten.

### A.1 RoboCup - Objective

Auszug aus der offiziellen Internetseite<sup>66</sup> des RoboCup Projekts vom 4.September 2011:

#### *Objective*

##### ***Pushing the State-Of-The-Art***

*It is our intentions to use RoboCup as a vehicle to promote robotics and AI research, by offering publicly appealing, but formidable challenge. One of the effective ways to promote engineering research, apart from specific application developments, is to set a significant long term goal. When the accomplishment of such a goal has significant social impact, it is called the grand challenge project. Building a robot to play soccer game itself do not generate significant social and economic impact, but the accomplishment will certainly considered as a major achievement of the field. We call this kind of project as a landmark project. RoboCup is a landmark project as well as a standard problem.*

##### ***The Dream***

*We proposed that the ultimate goal of the RoboCup Initiative to be stated as follows:*

*By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA, against the winner of the most recent World Cup.*

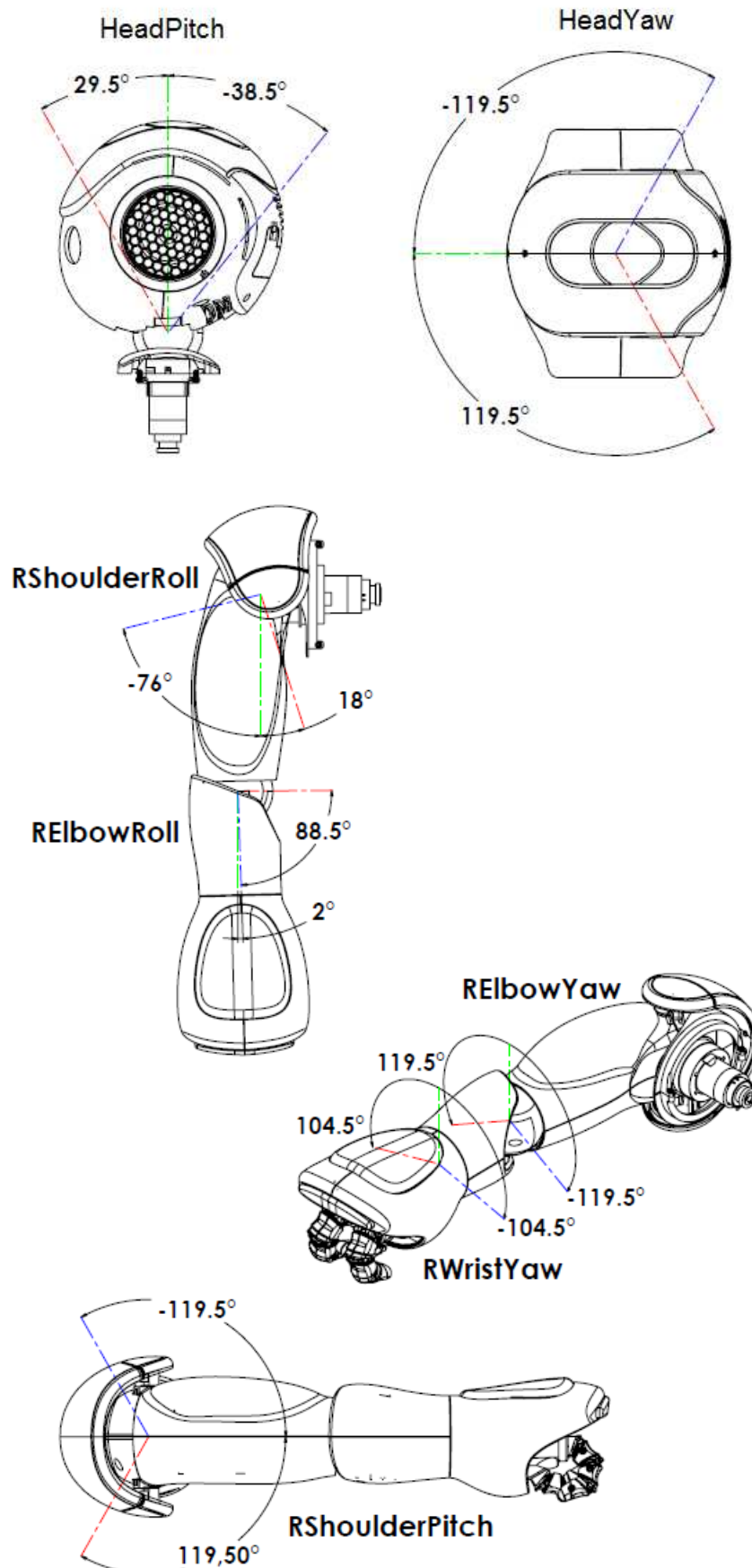
*We propose that this goal to be the one of the grand challenges shared by robotics and AI community for next 50 years. This goal may sounds overly ambitious given the state of the art technology today. Nevertheless, we believe it is important that such a long range goal to be claimed and pursued. It took only 50 years from the Wright Brother's first aircraft to Apollo mission to send man to the moon and safely return them to the earth. Also, it took only 50 years, from the invention of digital computer to the Deep Blue, which beat human world champion in chess. We recognize, however, that building humanoid soccer player requires equally long period and extensive efforts of broad range of researchers, and the goal will not be met in any near term.*

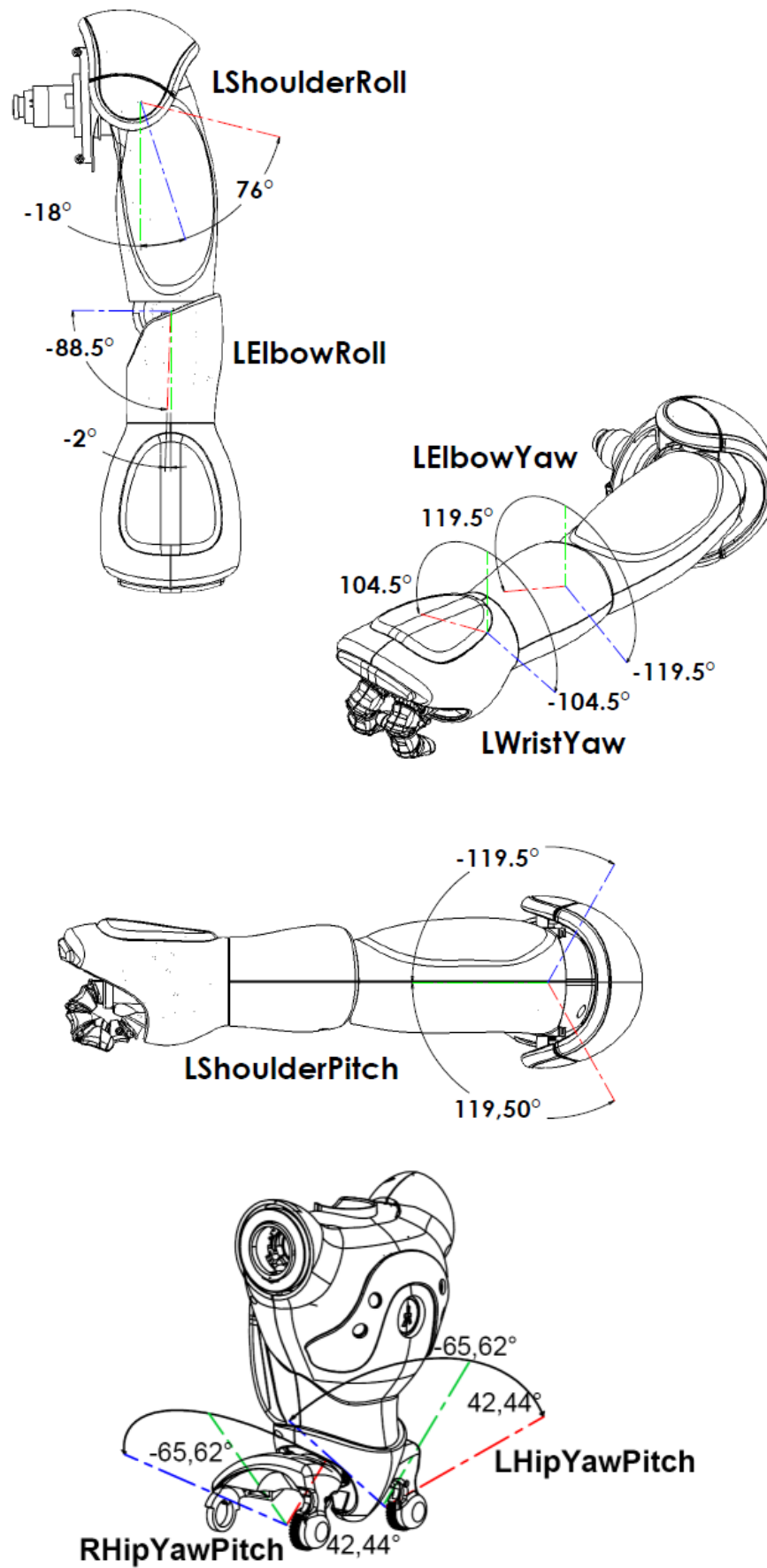
...

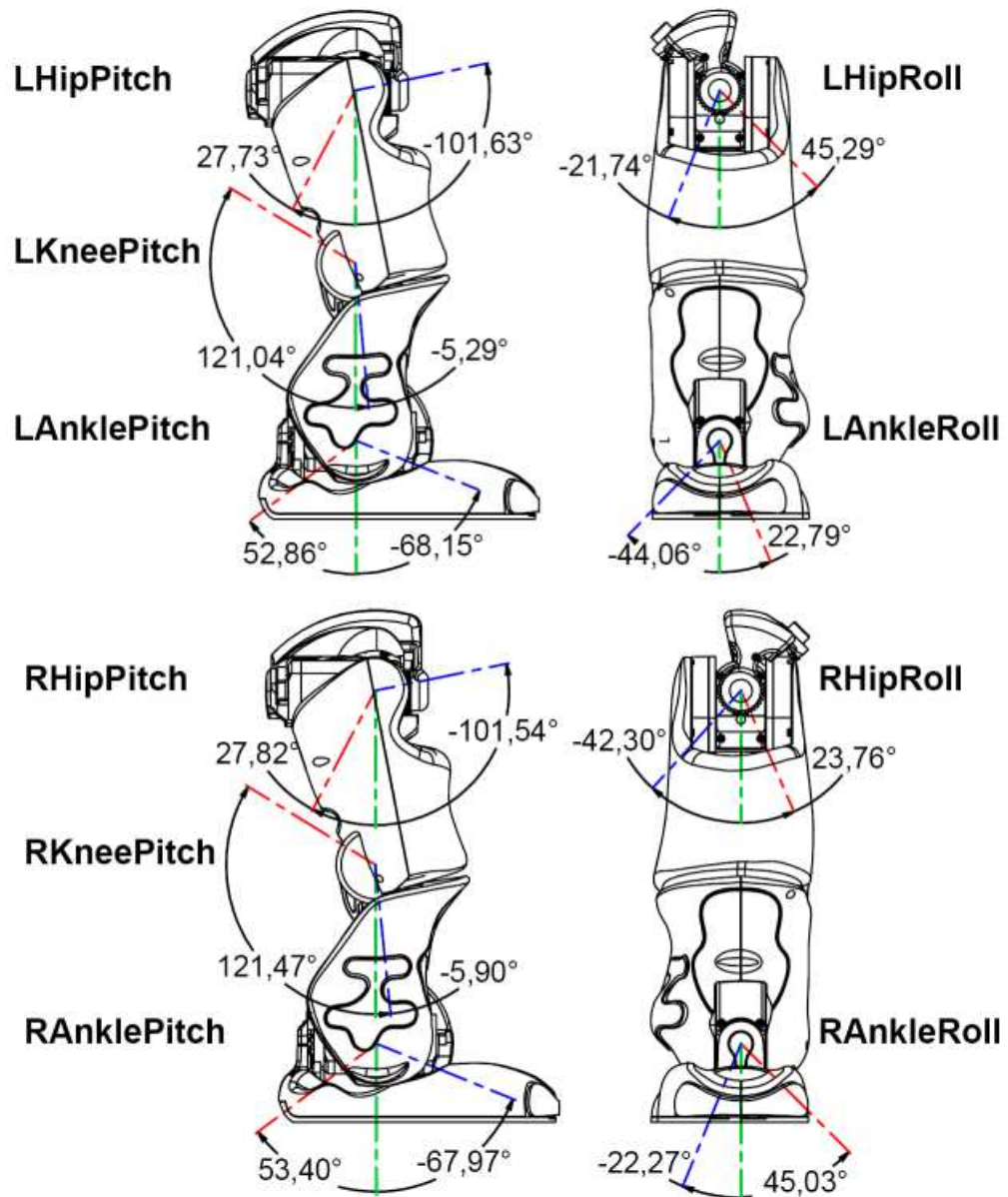
---

<sup>66</sup> <http://www.robocup.org/about-robocup/objective/>

## A.2 Grafiken der Gelenke

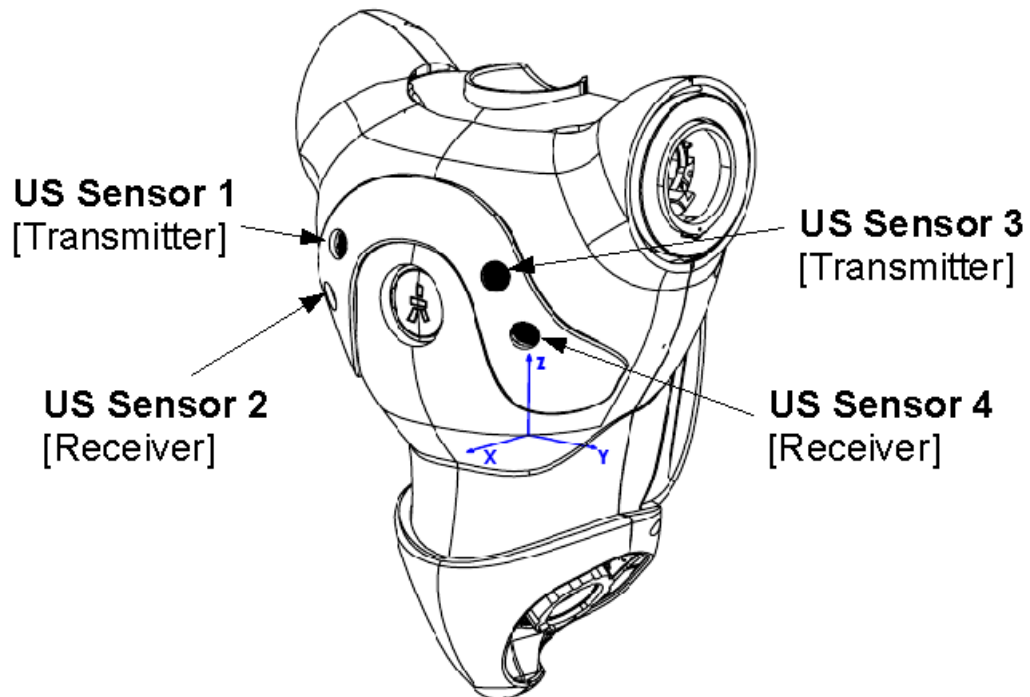






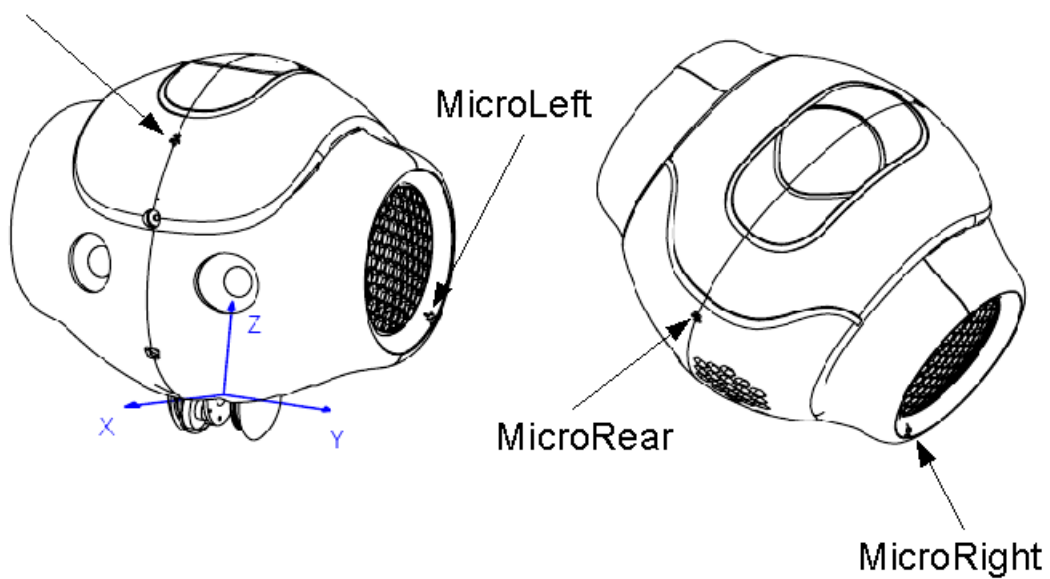
### A.3 Grafiken der Sensoren

#### Ultraschallsensoren:

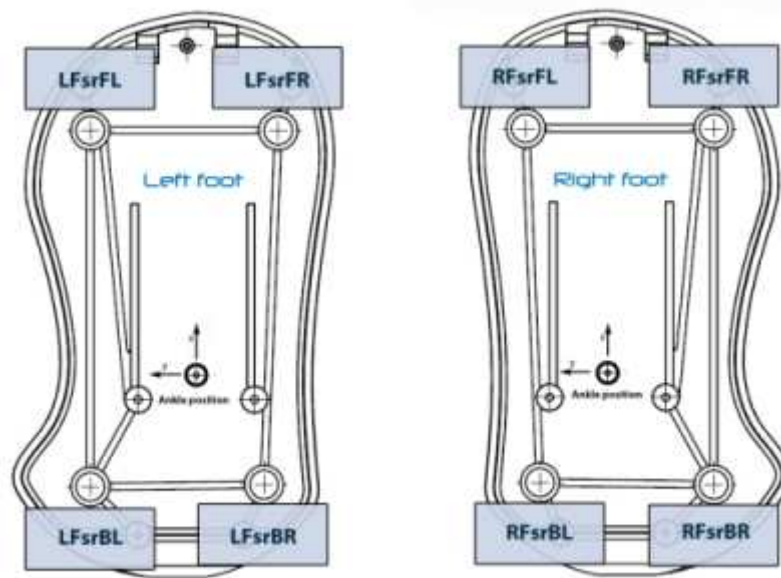


#### Mikrophone:

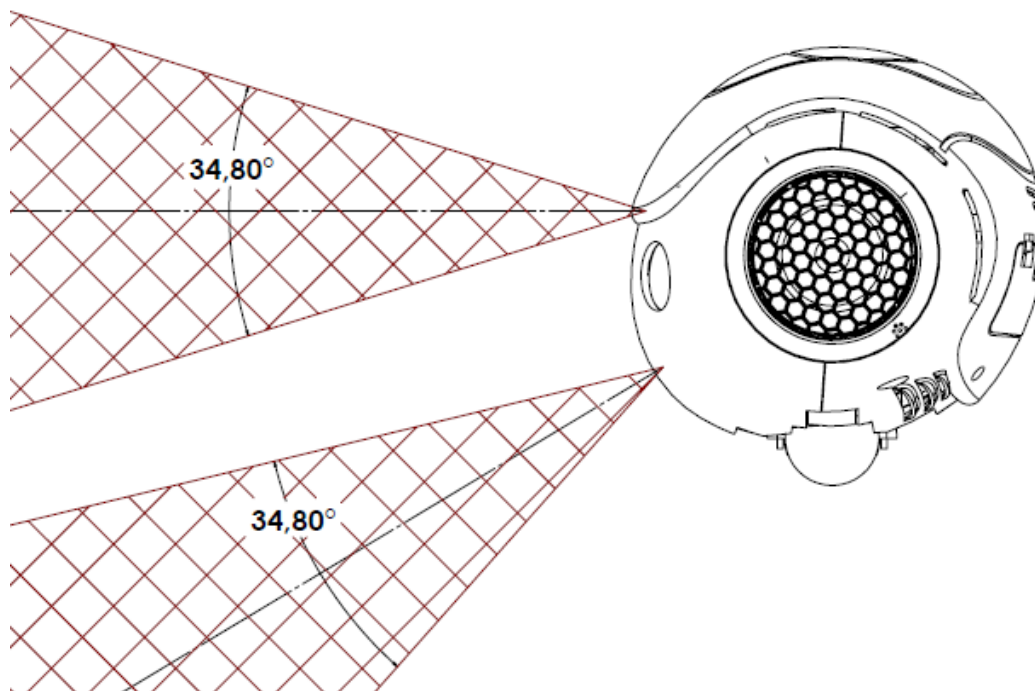
MicroFront



## Druckempfindliche Sensoren der Füße:



## Kameras:



## Anhang B: Überblick der entwickelten Softwaremodule

Dieser Anhang gibt einen Überblick zu den entwickelten Software Modulen. Zunächst ein Diagramm, das den Aufbau und das Zusammenspiel der einzelnen Klassen veranschaulichen soll. Anschließend folgt eine kurze Beschreibung dieser Klassen.

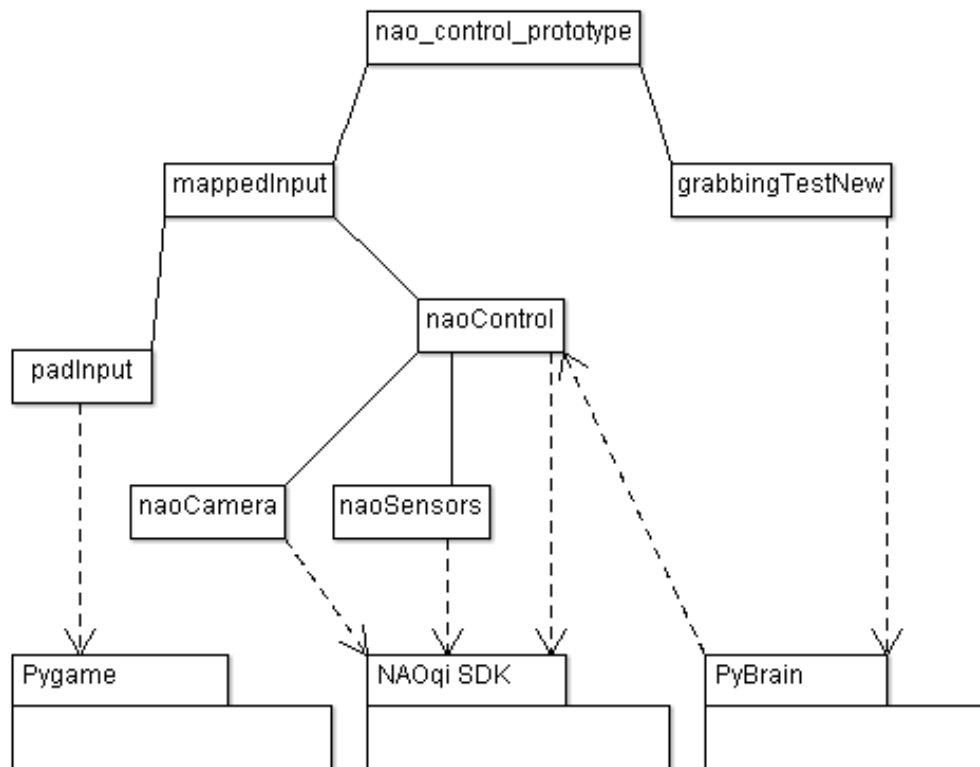
Details und die genaue Funktionsweise einzelner Teile werden hier nicht beschrieben, tiefergehende Fragen können per E-Mail<sup>67</sup> gestellt werden.

Die Module stellen kein fertiges Framework zur Einbindung des Nao in Python dar. Das ist auch nicht notwendig, da über das NAOqi SDK der komplette Funktionsumfang bereitsteht. Das Ziel der Module besteht vielmehr darin, bestimmte Funktionen derart zu kapseln, dass deren Verwendung möglichst einfach ist. Der Quellcode kann dafür verwendet werden die Funktionsweise von Methoden und des NAOqi SDK als Ganzes nachzuvollziehen und zu verstehen.

Einzelne Klassen und Methoden funktionieren im beigefügten Code (noch) nicht oder sind nur als Überbleibsel früherer Versuche übrig geblieben. Diese machen das Nachvollziehen des Quellcodes nicht einfacher, wurden allerdings nicht eigens entfernt. Damit ist der Code als Ganzes sicherlich kaum durchschaubar. Im Folgenden werden die wichtigsten Methoden und deren Verwendung nur ansatzweise skizziert. Die im Code enthaltenen Kommentare sind ebenfalls nicht vollständig und entsprechen nicht an allen Stellen exakt dem aktuellen Code. Die Entwicklung eines Frameworks das hohen Ansprüchen genügt müsste im Rahmen eines eigenständigen Projekts umgesetzt werden.

---

<sup>67</sup> Siehe Fußnote 65 in Kapitel 7.2



Dieses Diagramm enthält nur die wichtigsten Module und soll nur die grobe Funktionsweise veranschaulichen. Im oberen Teil erkennt man die selbstgeschriebenen Module, im unteren Teil sind die verwendeten externen Libraries und dargestellt.

- **naoControl**

Diese Klasse enthält, neben den beiden folgenden, die interessantesten Codeteile. Ein Objekt dieser Klasse wird mit Hilfe der entsprechenden IP-Adresse und dem Port dazu verwendet eine Verbindung zu einem Nao aufzubauen. Die Klasse stellt Methoden zur Steuerung des Roboters bereit. Praktisch sieht ein Steuerkommando für einen Roboter der unter dem Namen *nao* eingebunden wurde dann folgendermaßen aus:

```
nao.moveStraight(0.5)
```

- **naoCamera**

Stellt den Zugriff auf die Kameras bereit. Ein Objekt dieser Klasse wird automatisch beim Herstellen einer Verbindung zum Nao erzeugt und ist über die Klasse *naoControl* verfügbar. Mittels zweier Methoden können Bilder, in für PIL<sup>68</sup> oder OpenCV passendem Format abgefragt werden.

```
nao.camera.getPILImage()
nao.camera.getCVImage()
```

<sup>68</sup> Python Imaging Library, genaueres unter: <http://www.pythonware.com/products/pil/>



- **naoSensors**

Zugriff auf die (bisher implementierten) Sensoren des Nao erhält man über ein Objekt von diesem Typ. Analog zur vorigen Klasse wird dieses automatisch erzeugt und eine Abfrage der Gelenksensoren des rechten Arms kann mit folgendem Kommando durchgeführt werden.

```
nao.sensors.getRightArmValues()
```

- **grabbingTestNew**

Dieses Modul wurde verwendet um die verschiedenen Versuchsumgebungen mit PyBrain zu erstellen und Versuche durchzuführen.

- **padInput**

Hier werden Methoden zum Zugriff auf das Gamepad mit Hilfe von Pygame bereitgestellt und diese Klasse wird im Prinzip ausschließlich von der Klasse *mappedInput* verwendet.

- **mappedInput**

Dieses Modul läuft in einem eigenen Thread und fragt in einer Endlosschleife Eingabewerte vom Modul *padInput* ab. Hier können einzelne Funktionen des Roboters bzw. des Moduls *naoControl* mit den Eingabemöglichkeiten des Gamepads verknüpft werden.

- **nao\_control\_prototype**

Dieses Modul startet das Programm und erzeugt ein Objekt vom Typ *mappedInput*, zur manuellen Steuerung, oder vom Typ *grabbingTestNew* um Versuche zum maschinellen Lernen durchzuführen.

Diese kurze Beschreibung reicht selbstverständlich nicht aus um den Code umfassend zu verstehen, soll jedoch auch nur einen Überblick vermitteln. Eine eingehende Dokumentation wäre erst dann sinnvoll, wenn die Module soweit fertiggestellt sind, dass sie eine vollständige und fehlerfreie Schnittstelle zum Nao bieten können.

## Literaturverzeichnis

- [Aldebaran] **Community-Seite von Aldebaran**  
(2011): <http://users.aldebaran-robotics.com>
- [Alpaydin] **Maschinelles Lernen**  
Ethem Alpaydin (2004), ISBN 978-3-486-58114-0
- [BK] **Learning OpenCV**  
Gary Bradski und Adrian Kaehler (2008), ISBN 978-0-596-51613-0
- [Ertel] **Grundkurs Künstliche Intelligenz**  
Wolfgang Ertel (2008), ISBN 978-3-8348-0783-0
- [Maucher] **Skript – Machine Learning**  
Dr. Johannes Maucher (2009),  
[http://www.hdm-stuttgart.de/~maucher/Lecture\\_ML\\_SS09.html](http://www.hdm-stuttgart.de/~maucher/Lecture_ML_SS09.html)
- [MPGPE] **Multimodal Parameter-exploring Policy Gradient**  
Frank Sehnke, Alex Graves, Christian Osendorfer, Jürgen Schmidhuber  
[www6.in.tum.de/Main/Publications/Sehnke2010c.pdf](http://www6.in.tum.de/Main/Publications/Sehnke2010c.pdf)
- [Patterson] **Künstliche neuronale Netze: Das Lehrbuch**  
Dan Patterson (1996), ISBN 3-8272-9531-9
- [PGPE] **Parameter-exploring Policy Gradients**  
Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters  
und Jürgen Schmidhuber (TU München, Manno-Lugano, Max-Planck Institut  
Tübingen, 2010)  
[www.idsia.ch/~juergen/nn2010.pdf](http://www.idsia.ch/~juergen/nn2010.pdf)
- [RN] **Künstliche Intelligenz: Ein moderner Ansatz**  
Stuart Russell und Peter Norvig (2003), ISBN 978-3-8273-7089-1
- [SB] **Reinforcement Learning: An Introduction**  
Richard S. Sutton und Andrew G. Barto (The MIT Press, 1998)  
ISBN 978-0-2621-9398-6
- [Watkins] **Learning from Delayed Rewards**  
Watkins (Cambridge, 1989): gesichtet 2011  
[http://www.cs.rhul.ac.uk/~chrisw/new\\_thesis.pdf](http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf)
- [Wikipedia] **Wikipedia**  
(2011): <http://de.wikipedia.org>

Die folgenden Quellen sind nicht im Text referenziert und daher getrennt aufgeführt.

- [BORG]      **BORG – The RoboCup@Home team of the University of Groningen  
Team Description Paper**  
Jorge Dávila Chacón, Tim van Elteren, Bas Hickendorff, Herke van Hoof, Eric Jansen, Simon knuijver, Ciarán Lier, Paul Neculoiu, Aleke Nolte, Christof Oost, Valentina Richthammer, Florin Schimbinschi, Marten Schutten, Amirhosein Shantia, Ron Snijders, Egbert van der Wal, Dr. Dijn van der Zant
- [DPSC]      **Depth Perception with a Single Camera**  
Jonathan R. Seal, Donald G. Bailey, Gourab Sen Gupta (2005)
- [ILTHM]     **Imitation Learning and Transferring of Human Movement and Hand  
Grabbing to Adapt to Environment Changes**  
Stephen Al-Zubi, Gerald Sommer
- [Kühne]     **Universalitätsbeweise – Seminar zu Künstlichen Neuronalen Netzen und  
Fuzzy-Logik**  
Sven Kühne
- [LDsMI]     **Learning Depth from Single Monocular Images**  
Ashutosh Saxena, Sung H. Chung, Andrew Y. Ng
- [Nilsson]    **Learning Strategies for Mid-Level Robot Control: Some Preliminary  
Considerations and Experiments**  
Nils J. Nilsson (2000)
- [RC]        **Robot Catching**  
Marcia Riley, Christopher Atkeson
- [RLARM]     **Reinforcement Learning to Adjust Robot Movements to New Situations**  
Jens Kober, Erhan Oztop, Jan Peters
- [RLSRWR]    **Reinforcement Learning on a Simple Real Walking Robot**  
Michel Tokic, Wolfgang Ertel, Hans-Peter Radtke, Johar Akmal, Walter Krökel
- [RMAX]      **R-MAX – A General Polynomial Time Algorithm for Near-Optimal  
Reinforcement Learning**  
Ronen I. Brafman, Moshe Tennenholtz (2002)
- [Sanchez]    **Artificial Vision in the Nao Humanoid Robot**  
Tomás Gonzáles Sánchez (Universitat Rovira I Virgili, 2009)
- [Thorne]     **Introduction to Computer Vision in Python**  
Brian Thorne (University of Canterbury, 2009)
- [Ultsch]      **A Neural Network learning Relative Distances**  
Alfred Ultsch

- [VANHRC]    Using visual attention in a Nao humanoid to face the RoboCup any-ball challenge**  
Juan F. Garcia, Francisco J. Rodriguez, Francisco Martín, Vicente Matellán
- [WEBOTS]    Webots User Guide – release 6.3.4**  
Cyberbotics Ltd. (2011)

## Erklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

---

Ort, Datum

---

Unterschrift