FIDUCIA

DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

Entwicklung einer Gestiksteuerung mittels Kinect für den humanoiden Roboter Nao

Studienarbeit

für die Prüfung zum

Bachelor of Engineering

des Studienganges Informationstechnik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

Lukas Essig
und
Michael Stahlberger

Abgabedatum 12. Mai 2014

Bearbeitungszeitraum	24 Wochen
Matrikelnummer	8898018, 1367912
Kurs	TINF11B3
Ausbildungsfirma	Fiducia IT AG Karlsruhe
Betreuer	Herr Prof. Hans-Jörg Haubner

Erklärung

Gemäß §5 (3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 22. September 2011.

Ich habe die vorliegende Studienarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

_____ _____

Ort Datum Unterschrift

_____

Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

DHBW	Dualen Hochschule Baden-Württemberg

Kapitel 1

Einleitung

In dieser Einleitung wird das zentrale Thema dieser Arbeit beleuchtet, beschrieben in welchem Umfeld die Arbeit entworfen wurde und strukturell erläutert, wie die Zusammenarbeit im Projekt stattgefunden hat. Zudem wird aufgezählt, wie sich diese Arbeit inhaltlich aufbaut.

1.1 Ziel der Arbeit

Zentrales Thema der Arbeit ist die Interaktion zwischen Mensch und Roboter. Dabei soll ein Modell des humanoiden Roboters Nao durch die Gestik seines Benutzers gesteuert werden. Der Roboter ahmt die Bewegungen nach, die der Benutzer zuvor ausgeführt hat. Zudem sollen bestimmte Steuergesten implementiert werden, die den Roboter in verschiedene Modi versetzen, in denen er gleiche Gesten auf verschiedene Weise interpretiert. Die Gestik-Erkennung soll mit Hilfe eines Xbox Kinect Moduls erfolgen. Von dieser Stereokamera werden bestimmte Gesten aufgezeichnet und über eine selbst implementierte Software verarbeitet und an den Nao-Roboter weitergeleitet, der diese dann nachahmt. Dabei ist darauf zu achten, dass keine Bewegungen von Nao ausgeführt werden, die ihm mechanisch oder elektronisch Schaden zufügen können.

Das Vorgehen ist folgendermaßen strukturiert: Nach der Einarbeitung in die Materie Nao - Roboter und Kinect - Kamera soll die Wahl einer geeigneten Programmiersprache getroffen werden. Daraufhin müssen zwei verschiedene Anwendungen konzeptioniert und implementiert werden. Erstens die Erkennung der Gesten mittels Kinect und zweitens das Empfangen der Gesten und die kinematische Umsetzung durch den Roboter. Zwischen diesen beiden Anwendungen muss eine geeignete Kommunikationsschicht entworfen und implementiert werden. **To do** (1)

Der erste große Meilenstein ist die Übertragung der Armbewegung des Menschen auf den Roboter. Darauf folgend soll versucht werden auch Kopf- und Torsobewegungen zu übertragen. Je nach dem wie der zeitliche Fortschritt nach diesen beiden Meilensteinen ist, kann auch die Steuerung der Füße und somit das Laufen implementiert werden.

1.2 Inhalt

Grundlagen (Roboter, DOF,) -¿ Kinect -¿ Nao -¿ Realisierung (Prototypen, Architektur, Umsetzung) -¿ Fazit(was hat geklappt, was fehlt,)

To do (2)

1.3 Umfeld & Strukturelles

Diese Studienarbeit mit dem Thema *Entwicklung einer Gestiksteuerung mittels Kinect für den humanoiden Roboter Nao* wurde während zwei Theoriephasen an der Dualen Hochschule Baden-Württemberg (DHBW) am Standort Karlsruhe von Michael Stahlberger und Lukas Essig durchgeführt. Betreut wurde die Arbeit durch Herrn Prof. Hans-Jörg Haubner, sowie Herrn Michael Schneider.

Dieses Dokument wurde von den beiden Autoren in LaTeX verfasst. Dies hat den Vorteil, dass sehr leicht gleichzeitig an der Ausarbeitung geschrieben werden kann, ohne dass Konflikte dabei auftreten.

Als Programmiersprache wurde C# gewählt, da sowohl das Kinect-Modul als auch der Nao-Roboter über eine Schnittstelle verfügen, die diese objektorientierte Sprache unterstützt. Dies macht es einfacher, die beiden Programme später zu verknüpfen.

Sowohl die LaTeX Dokumentation, als auch der Source-Code sind mit Hilfe von Git[1] in einem Repository versioniert worden. Damit lässt sich genau verfolgen, wer wann eine Änderung durchgeführt hat und zudem lässt sich auch eine vorherige Versionen wiederherstellen. Das Repository wurde auf der Internetplattform *GitHub* gespeichert. Damit ist es möglich, von jedem Rechner auf das Repository und somit die Dateien und Dokumente zuzugreifen. *GitHub* bietet den Vorteil, dass die Zusammenarbeit an Projekten noch zusätzlich durch Management-Funktionen unterstützt werden. In diesem Fall wurden davon hauptsächlich zwei Funktionen benutzt. Einmal die *Issue*- Funktion, mit der Anforderungen, Aufgaben, Bugs etc. eingetragen und einem Bearbeiter zugeordnet werden können.

[1] Software zur Versionsverwaltung von Dateien und Verzeichnissen

Zusätzlich wurde noch die *Milestone*-Funktion genutzt, mit der Projekt - Meilensteine erstellt werden können. Diese geben an, welche Funktion bzw. Anforderung bis wann erledigt sein sollten. Die Issues sind dann einem Meilenstein zugeordnet und wenn ein Issue geschlossen wird, ist sehr einfach dargestellt, zu wie viel Prozent der Meilenstein erreicht ist.

Kapitel 2

Kinect

Im folgenden Kapitel wird der Aufbau und die Funktion des Kinect Sensors erläutert.

2.1 Kinect Hardware

Microsoft brachte im November 2010 mit dem Produkt Xbox Kinect eine neues Produkt auf den Markt. Mit diesem war es möglich, ihr eigenes Produkt Xbox 360 über eine 3D-Kamera zu steuern und Spiele zu spielen. Mit diesem Produkt konnte erstmals ein preiswerter Sensor verwendet werden, um 3D-Bilddaten auszuwerten. Im Folgenden wird auf die Hardware des Sensors näher eingegangen:

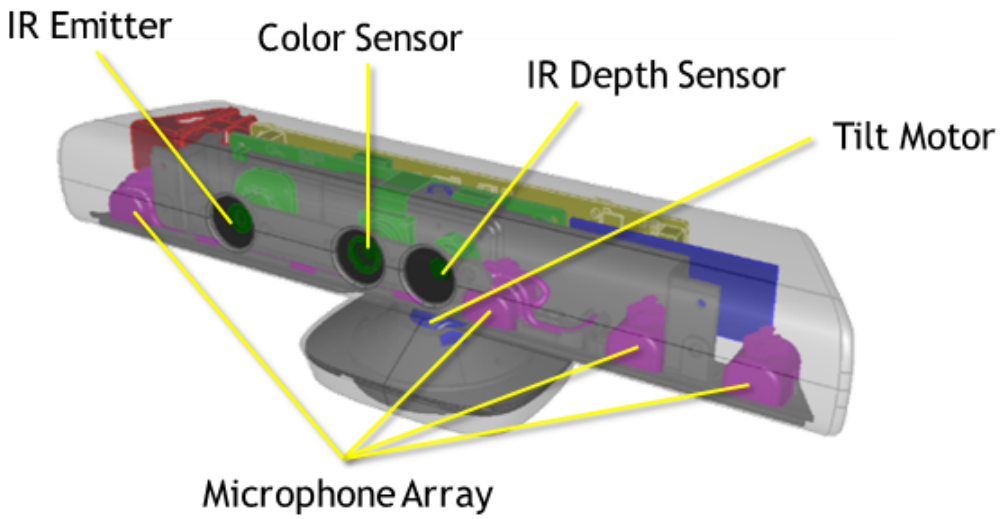

Abbildung 2.1: Aufbau Kinect [**?**]

[**?**]
To do (3) **To do** (4) **To do** (5)

Hardware ========

Kippbarer Kopf

Infrarot Projektor

Zwei Kameras

4 Mikrofone -¿ Für dieses Projekt irrelevant (evtl. Sprachkommandos? Kinect Lib unterstützt Grammatik!!)

Auflösung der Kameras

color camer = max resolution of 1280 x 960 depth camera = max resolution of 640 x 480.

2.2 Kinect Software

(Essenz auf dem, was Kinect für mich erledigt theorie, mathematische Berechnung tiefenstream usw)

Da das Produkt Xbox Kinect bereits vor einigen Jahren auf den Markt kam, hatten die Entwickler Zeit, um ein SDK zu entwickeln, was alle wichtigen Programmfunktionen bereits enthält. Dies macht es einem Entwickler relativ leicht eine Anwendung zu erstellen, die bestimmte Kinect-Funktionen bereitstellt.

Die Hardware liefert folgende Werte:

- Audiosignale inkl. Richtungswert des Microphonarrays
- RGB Bildsignal der Kamera
- Tiefensignal

Für dieses Projekt sind die Audiodaten jedoch irrelevant. Wichtig sind primär die Bilddaten inklusive Tiefenwerte. Das Kinect SDK bietet bereits standardmäßig Zugriff auf diese Werte. Dabei können im Programm bestimmte Proxyobjekte registriert und abgefragt werden. Für dieses Projekt ist das Skeletonobjekt von großer Bedeutung. Anhand von Vergleichsmustern erzeugt die Software -nicht Kinect!- einen sog. Skeletonstream, der versucht die Position eines menschlichen Skeletts im Kinect Koordinatenraum abzubilden. [**?**]

2.2.1 Der Skeletonstream

Damit die Bewegungen eines Benutzers mittels Kinect erkannt und im Programm verarbeitet werden können, werden im Tiefenstream bestimmte Bereiche definiert und mit bestehenden Mustern verglichen. Somit können die einzelnen Gelenke

des menschlichen Skeletts (Joints) erkannt und in Zusammenhang gebracht werden. Als Entwickler können diese dann in einem Skelettobjekt mit der erkannten Genauigkeit ausgelesen und verwendet werden. Für die Erkennung der Armbewegungen werden Schulter-, Ellenbogen-, Hand- und Hüftjoints benötigt (Siehe Realisierung).

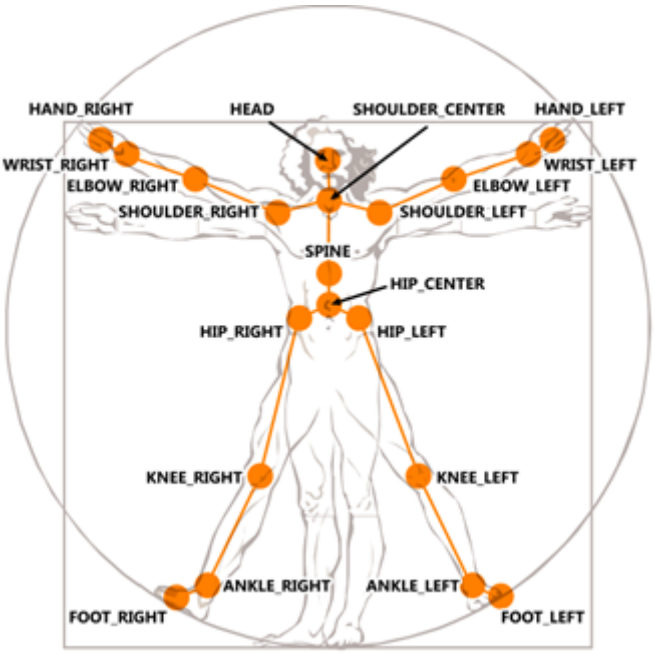

Abbildung 2.2: Kinect Joints [**?**]

(Technischer Hintergrund der Berechnung -¿ Bild mit verschiedenen Farben für SKelettbereiche, aber nicht öffentlich, da Microsoft Geheimnis)

(Bild Kinect Koordinatenraum) (Bild mehrere User im Kinectraum)

To do (6) **To do** (7) **To do** (8) **To do** (9)

Dieser Stream berechnet anhand der Tiefendaten und der RGB-Daten werte für ein Menschliches Skelett:

-Armwinkel -Positionen -Kinectraum...

DELME [**?**]

2.3 Kinect SDK

(Essenz auf der praktischen Anwendung des SKDs -¿ Methoden usw.)

Nach dem Erscheinen der Hardware war diese auch schnell in Entwicklerkreisen gefragt. Doch Microsoft selbst gefiel dies zunächst nicht, denn der Konzern befürchtete, dass Cheater sich an ihren Spielen zu schaffen machen würden. So gab es zunächt kein SDK von Microsoft selbst. Die Open Source Gemeinde jedoch

erkannte das Potential des Produktes schneller und entwickelte (eine Schnittstelle? zu) OpenNI, einem Framework, das die Auswertung von 3D-Sensordaten verschiedenster Hersteller unterstützt. Dieses Framework bietet somit durch seine Plattformunabhängigkeit die Möglichkeit unterschiedliche Betriebssysteme mit unterschiedlichen 3D-Sensoren zu kombinieren. [**?**] Microsoft zog nach und gab am 17. Juni 2011 die freie Beta Version des Microsoft SDKs frei. Somit hatte nun jeder Entwickler freien Zugang zu allen Kinectfunktionen, die auch von Microsoft selbst bisher genutzt wurden. Einer der Vorteile des Kinect SDKs besteht darin, dass die Skelett-Erkennung (siehe Kapitel 2.2 Kinect Software) ohne initiale Pose möglich ist, was im Gegensatz zum OpenNI-Framework steht. [**?**] Da für dieses Projekt die Plattformunabhängigkeit nicht relevant ist, sowohl aber die Skeletterkennung, wurde sich für das Microsoft SDK entschieden.

2.3.1 Funktionen des SDKs

Methoden des SDKs

To do (10)

Kapitel 3

Nao

Nao ist ein humanoider Roboter des französischen Herstellers *Aldebaran Robotic*. Die erste Version des Roboters wurde 2006 vorgestellt und unter anderem mit einem Investment durch Intel weiterentwickelt. Es gibt den Roboter in verschiedenen Ausführungen mit verschieden verbauten Sensoren und kostet ungefähr 10.000 Euro. [**?**]

Im folgenden Kapitel wird der Aufbau und die Funktionen der Hardware, sowie die zugehörige Software zur Bedienung und Programmierung des Nao vorgestellt.

3.1 Nao Hardware

Der in dieser Arbeit verwendete Nao ist vom Typ *NAO V3+, V3.2*. Dieser Typ ist 573.2 mm hoch, 290 mm tief und 273.3 mm breit. In Abbildung 3.1 *Übersicht Nao V3.2* sind alle Sensoren und Aktoren aufgeführt.

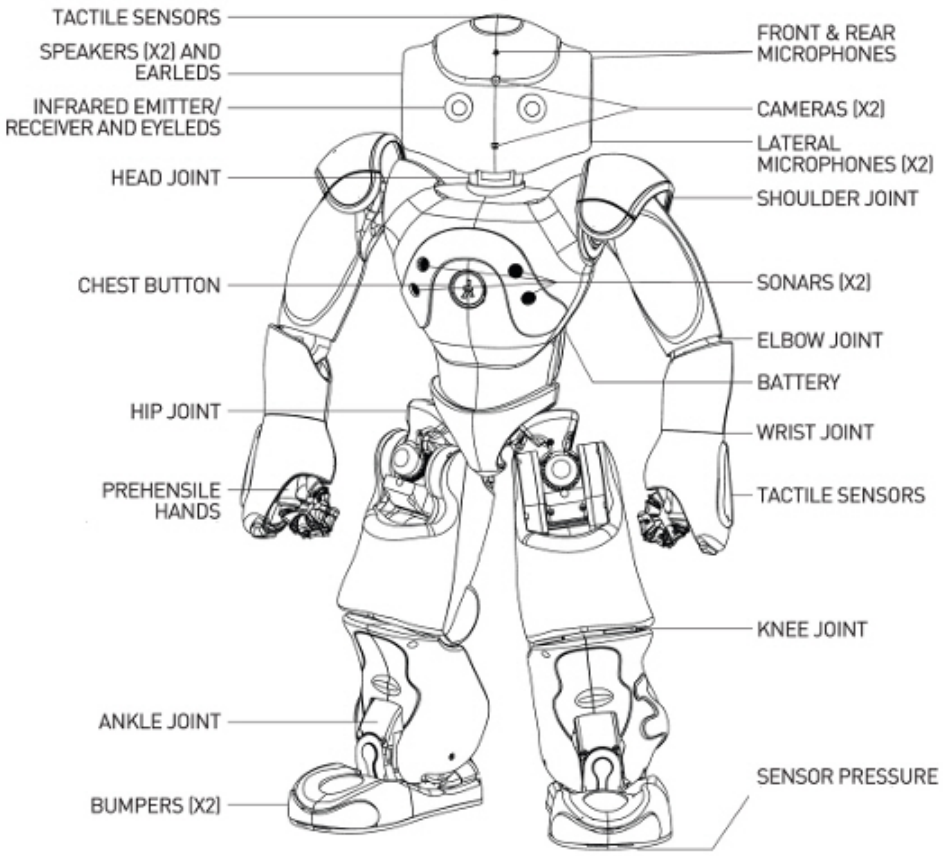

Abbildung 3.1: Übersicht Nao V3.2

Gelenke

Nao besitzt im Kopf, den beiden Armen, dem Becken und den beiden Beinen jeweils mehrere Gelenke (*Joint*, siehe Bild 3.1). Damit ist eine umfangreiche Bewegung in alle Richtungen der drei Achsen möglich. Der Kopf lässt sich in Z-Richtung drehen und in Y-Richtung neigen, damit Nao auch räumlich sehen bzw. Objekte verfolgen kann. Die Arme besitzen die gleichen Gelenke wie in einem menschlichen Körper. Dazu gehören Schulter-, Ellenbogen- und Hangelenk. Das Schultergelenk dient dazu, den Arm zu heben/senken und ihn zu öffnen bzw. zu schließen. Das Drehen und Öffnen/Schließen des Unterarms geschieht durch das Ellenbogengelenk in Kombination mit dem Hangelenk. Die Finger der Hand können nur als Ganzes geöffnet bzw. geschlossen werden. Das Beckengelenk wird dazu benutzt, den Torso von Nao nach vorne oder hinten zu neigen. Die Beine bestehen aus drei Gelenken: Einem Hüft-, einem Knie- und einem Fußgelenk. Die Bewegungsfreiheit der Beine ähnelt dem des menschlichen Beins, wobei keines der Gelenke in Z-Richtung gedreht werden kann.

Die Bestimmung der Gelenkpositionen erfolgt über einen magnetischen Drehwinkelgeber mit einer Auflösung von 12 Bit. Das macht beispielsweise bei einem Wert von 4096 pro Umdrehung eine Präzision von 0.1 Grad.

Aktoren
In Nao sind vier verschiedene Typen von Motoren verbaut. Diese unterscheiden sich im wesentlichen in ihrer maximalen Anzahl an Drehungen pro Minute, dem Drehmoment und der Drehzahlrückstellung. Dies ist wichtig, da nicht jedes Gelenk und der zugehörige Aktor mit der gleichen Masse belastet wird.

Elektronik & Sensoren
Das Herz von Nao ist dessen Motherboard mit einer x86 AMD CPU mit 500MHz. Der Arbeitsspeicher mit 256MB RAM und die 2GB Flash-Speicher befinden sich zusammen mit dem Prozessor im Kopf. Die Batterie mit rund 30Wh hält für die aktive Nutzung (viele Bewegungen und Sensoraktivitäten) ca. 60min und die normale Nutzung ca. 90min.

Links und Rechts am Kopf befinden sich jeweils ein Lautsprecher und ein Mikrofon. Zusätliche Mikrofone sind am Kopf auch noch vorne und hinten angebracht. Damit ist es Nao möglich, ein Geräusch zu lokalisiern und gegebenenfalls dahin zu folgen. Um gleichzeitig die Ferne und die Nähe visuell zu verarbeiten wurde über und unter den Augen jeweils eine VGA - Kamera mit einer Auflösung von 640x480 Pixeln installiert. Die Augen selbst dienen zur Erkennung von Infrarotlicht, wobei auch hier in jedem Auge jeweils ein Sensor verbaut ist.

Auf der Brust von Nao befinden sich Ultraschallsensoren zur Distanzermittlung (je 2 Emitter und Empfänger). Diese haben eine Auflösung von 1cm und eine Erkennungsweite von 0.25m bis 2.55m. Unter 0.25m erkennt Nao nur noch, dass ein Objekt im Weg ist, aber nicht wie weit es entfernt ist.

Sensoren zur Kontakterkennung befinden sich auf dem Kopf, dem Brustbutton, auf und neben den Händen, sowie vorne an den Füßen. Unter den Füßen befinden sich zudem noch piezoresistive Drucksensoren mit einem Arbeitsbereich von 0 bis 25 Newton. Damit lässt sich unter anderem erkennen, ob Nao nur auf einem Bein oder auf unebenen Untergrund steht.

To do (11)

3.2 Nao Software

Choreographe

Webots

3.3 Nao SDK

Naoqi (architektur, modell) embedded

verfügbare sprachen

vorstellung c#

Kapitel 4

Realisierung

In diesem Kapitel wird näher auf das Zusammenspiel von Sensorik (Kinect) und Aktorik (Nao) eingegangen. Zunächst wird die Idee der Softwarearchitektur vorgestellt. Im Anschluss daran, wird auf die entworfenen Algorithmen näher eingegangen.

To do (12) **To do** (13)Wahl Bibliothek, warum?...) **To do** (14) **To do** (15) **To do** (16)

4.1 Architektur

Die Architektur der Anwendung kann grob in zwei Teilbereiche gegliedert werden, den Sensorteil und den Aktorteil. Der Sensorteil ist dafür zuständig, die entsprechenden Gesten des Benutzers zu ermitteln und zu verarbeiten. Der Aktorteil der Anwendung ist dafür zuständig die Werte entsprechend auf die Wertebereiche des Nao-Koordinatensystems zu transformieren und diese dem Roboter zu übermitteln.

4.1.1 Interfaces

Als Schnittstelle zwischen der Erkennung und der Ausführung der Armpositionen wird essenziell die Methode *updateAngles* vom Interface *ISkeletonAngles* verwendet. Diese stellt alle benötigten Winkel wie Shoulder Pitch, Shoulder Roll, Ellbow Roll, Elbow Yaw bereit.

4.1.2 Klassendiagramm

4.1.3 Sequenzdiagramm: Winkelerkennung

To do (17)

4.2 Kinect Armerkennungs-Algorithmus

Ein wesentlicher Teil des Programmes besteht darin, die Gesten einer Person, die sich vor dem Kinect-Sensor befindet, zu erkennen und in die passenden Winkelwerte für den Roboter umzurechnen. Hierfür wird mathematisch gesehen das Kreuzprodukt der passenden Knochen errechnet, um die passenden Winkel zur Steuerung des Nao-bots zu erhalten.

Um einen Arm des Roboters zu steuern, bedarf es folgender Winkel:

- Shoulder Pitch
- Shoulder Roll
- Elbow Roll
- Elbow Yaw

4.2.1 Shoulder Pitch

Der Shoulder Pitch des menschlichen Skelettes kann durch die Joints **Hip, Shoulder und Elbow** errechnet werden. Dabei werden zwei Vektoren aufgespannt. Ein Vektor verbindet den Punkt *Hip* mit dem *Shoulder* Punkt, der andere Vektor stellt den Oberarm dar und verbindet den Punkt *Shoulder* mit dem Punkt *Elbow*. Dabei alle Joints nur von einer Seite des Körpers, also z.B. *HipRight, ShoulderRight, ElbowRight* extrahiert.

4.2.2 Shoulder Roll

Der Shoulder Roll Winkel kann nicht mit drei aneinander liegenden Gelenken ermittelt werden. Deshalb benutzt man jeweils zwei Hilfs-Vektoren, die im Skelett nicht anatomisch verbunden sind. Der erste Vektor wird durch die Hüfte aufgespannt, dies betrifft die Joints *HipRight* und *HipLeft*. Der andere Vektor wird analog zum Oberarm aufgespannt, dies betrifft die Joints *Shoulder* und *Elbow*.

4.2.3 Elbow Roll

Dieser Winkel entspricht der Beugung des Ellenbogens. Dafür notwendig sind die Joints *Shoulder*, *Elbow* und *Hand*. Dabei entsprechen die Vektoren dem Oberarm (Shoulder-Elbow) und dem Unterarm (Elbow-Hand) des menschlichen Skelettes.

4.2.4 Elbow Yaw

asd

Nach der Erstellung dieser Vektoren wird eine Methode aufgerufen, die das Kreuzprodukt der Vektoren errechnet und somit den jeweiligen Winkel zurück liefert.

```
public static float getAngle(Vector3D a, Vector3D b,
   Vector3D c)
{
     Vector3D bone1 = a - b;
     Vector3D bone2 = c - b;

     bone1.Normalize();
     bone2.Normalize();

     float dotProduct =
        (float)Vector3D.DotProduct(bone1, bone2);

     return (float)Math.Acos(dotProduct);
}
```

Listing 4.1: Ermittlung des Winkels mithilfe von drei Punkten

Da die Methode auch mit vier Gelenken (Shoulder Roll, Elbow Yaw) durchgeführt werden kann, existiert auch folgende Methode:

```
public static float getAngle(Vector3D a, Vector3D b,
   Vector3D x, Vector3D y)
{
     Vector3D bone1 = a - b;
     Vector3D bone2 = x - y;

     bone1.Normalize();
     bone2.Normalize();

     float dotProduct =
        (float)Vector3D.DotProduct(bone1, bone2);
```

```
    return (float)Math.Acos(dotProduct);
}
```

Listing 4.2: Ermittlung des Winkels mithilfe von vier Punkten

To do (18)

4.3 Prototyp

Anhand der obigen Überlegungen wurde der erdachte Algorithmus zunächst anhand eines Prototyps implementiert. Dieser sollte zunächst dazu dienen, das Microsoft Kinect SDK näher kennen zu lernen**To do** (19). Der erste Prototyp erfüllte folgende Funktionen:
-Kinect SDK Library -(Connect-Disconnect von Kinect) -Winkelerkennung des Benutzers mit Anzeige in Grad -Anzeige des Kamerabildes mit Gelenkkennzeichnung von Kopf und Armen

To do (20)

Kapitel 5

Ausblick

Schluss

5.1 Weiterführende Anwendungsgebiete

Es stellt sich nun die Frage, welchen technischen Nutzen die Teleoperation innerhalb der Gesellschaft finden könnte. Zunächst gibt es ein großes Anwendungsgebiet in der Medizin....

To do...

- ☐ 1 (p. 1): @Lukas Zwei Anwendungen oder eher eine Anwendung?
- ☐ 2 (p. 2): Inhalt der Studienarbeit
- ☐ 3 (p. 4):

  -Aufbau: -Microphonarray zweck=richtung d. Geräusch -IR Sensor = Deep Stream -RGB Kamera = Bild -Rauminformationen für Software -Vorteil: Günstig, leicht zu entwickeln, da SDK vorhanden
- ☐ 4 (p. 4): Hardware Bilder
- ☐ 5 (p. 4): erklärungen zu Bilder
- ☐ 6 (p. 6): SDK Funktionen erklären: skelton, mehrere user, deep stream, color stream
- ☐ 7 (p. 6): Screenshot von Deep Stream (Kinect Studio)
- ☐ 8 (p. 6): Unterschied Microsoft SDK und Freie Implementierung OpenNI
- ☐ 9 (p. 6): Bild von Kinect-Koordinatensystem -¿ Bezug auf Nao
- ☐ 10 (p. 7): evtl. vergleich SDK OpenNI?
- ☐ 11 (p. 10): vllt. einzelne Bilder rein?
- ☐ 12 (p. 12): Wahl Programmiersprache (warum? -¿Kinect und Nao gleich, Visual Studio)
- ☐ 13 (p. 12): (
- ☐ 14 (p. 12): Erklärung Webots, Choreograph, Nao Verbindung, Libraries...

- ☐ 15 (p. 12): Danach -¿ echter Nao im Laber -¿ Unterschied zur Simulation
- ☐ 16 (p. 12): Laborbedingungen Steuerungen, Vor- und Nachteile der Bedienung
- ☐ 17 (p. 12): Architekturidee: Input, Verarbeitung(evtl. Filter-¿Probleme Ruckeln?), Output,
- ☐ 18 (p. 12): Methode beschreiben
- ☐ 19 (p. 12): getrennt zusammen?
- ☐ 20 (p. 15): Screenshot von Prototypprogramm