**Koppelung einer Bewegungsentwicklung**

**für humanoide Roboter**

**mit der Bewegungserkennung**

**von Personen durch die Kinect**

**Studienarbeit**

für die Prüfung zum

Bachelor of Engineering

des Studienganges Informatik

Studienrichtung Informationstechnik

an der Dualen Hochschule Baden-Württemberg Karlsruhe

von

**Dennis Alles**

und

**Karolin Edigkaufer**

12.05.2014

Matrikelnummer 3934520 (Dennis Alles)

6589515 (Karolin Edigkaufer)

Kurs TINF11B3

Betreuer Prof. Dr. Hans-Jörg Haubner

Michael Schneider

Ehrenwörtliche Erklärung

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

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

Ort, Datum Unterschrift

Ort, Datum Unterschrift

Abbildungsverzeichnis

[Abb. 1: Die Sensoren der Kinect [8] 4](#_Toc386030058)

[Abb. 2: Skeleton mit 20 Punkten [9] 5](#_Toc386030059)

[Abb. 3: Kinect Developer Toolkit 7](#_Toc386030060)

[Abb. 4: Die Lage der Motoren beim Nao [14] 9](#_Toc386030061)

[Abb. 5: Die Hardwarekomponenten des Nao [15] 10](#_Toc386030062)

[Abb. 6: Nao Bibliothek nutzen 11](#_Toc386030063)

[Abb. 7: Code Map 14](#_Toc386030064)

[Abb. 8: Programmablaufplan – erster Teil 17](#_Toc386030065)

[Abb. 9: Programmablaufplan – zweiter Teil 19](#_Toc386030066)

Codeverzeichnis

[Code 1: Thread 21](#_Toc386030067)

[Code 2: Thread 2 22](#_Toc386030068)

Abkürzungsverzeichnis

DHBW Duale Hochschule Baden-Würrtemberg

GUI Graphical User Interfce

SDK Software Development Kit

STA Singlethreaded Apartment

VGA Video Graphics Array

Inhaltsverzeichnis

[Ehrenwörtliche Erklärung III](#_Toc386030072)

[Abbildungsverzeichnis IV](#_Toc386030073)

[Codeverzeichnis V](#_Toc386030074)

[Abkürzungsverzeichnis VI](#_Toc386030075)

[Inhaltsverzeichnis VII](#_Toc386030076)

[1 Einleitung 1](#_Toc386030077)

[1.1 Motivation 1](#_Toc386030078)

[1.2 Aufgabenumfeld 1](#_Toc386030079)

[1.3 Aufgabenbeschreibung und Ziele 2](#_Toc386030080)

[1.4 Vorgehensweise 2](#_Toc386030081)

[2 Grundlagen 3](#_Toc386030082)

[2.1 Kinect 3](#_Toc386030083)

[2.1.1 Sensoren der Kinect 3](#_Toc386030084)

[2.1.2 SDK 6](#_Toc386030085)

[2.1.3 Kinect Developer Toolkit 6](#_Toc386030086)

[2.2 Humanoide Roboter: Nao 8](#_Toc386030087)

[2.2.1 Nao 8](#_Toc386030088)

[2.2.1.1 Hardware 8](#_Toc386030089)

[2.2.1.2 Software 10](#_Toc386030090)

[2.2.1.3 Nao SDK 10](#_Toc386030091)

[2.2.1.4 NaoQi 11](#_Toc386030092)

[2.2.1.5 Webots für Nao 11](#_Toc386030093)

[2.3 Programmiersprache 11](#_Toc386030094)

[2.4 Threads – Karo 11](#_Toc386030095)

[2.5 Winkelberechnung 12](#_Toc386030096)

[3 Konzeption/Planung 13](#_Toc386030097)

[4 Umsetzung 14](#_Toc386030098)

[4.1 Programmstruktur / Architektur - Dennis 14](#_Toc386030099)

[4.1.1 Programmablaufplan einer Bewegungsnachahmung 17](#_Toc386030100)

[4.1.2 Warum Threads? 20](#_Toc386030101)

[4.2 Nao – Dennis 22](#_Toc386030102)

[4.2.1 Zufällige Bewegung ausführen 22](#_Toc386030103)

[4.2.2 Bewegung wiederholen 24](#_Toc386030104)

[4.2.3 Speicherung Winkel vom Nao 24](#_Toc386030105)

[4.3 Kinect – Karo 24](#_Toc386030106)

[4.3.1 Berechnung Winkel Kinect 24](#_Toc386030107)

[4.3.2 Vergleich Winkel Kinect <-> Nao 24](#_Toc386030108)

[4.3.3 Schwierigkeitsgrade 24](#_Toc386030109)

[4.3.4 Timer 24](#_Toc386030110)

[4.4 GUI - Karo 24](#_Toc386030111)

[4.5 Test (Zunächst mal noch nicht) 25](#_Toc386030112)

[5 Fazit (Zunächst mal noch nicht) 26](#_Toc386030113)

[Literaturverzeichnis IX](#_Toc386030114)

# Einleitung

Sowohl Roboter als auch die direkte Interaktion zwischen Anwender und Computer liegen voll im Trend in der Informatik. Die starre Programmierung eines Rechners wird es zwar immer geben, aber oftmals rückt das Thema den Anwender selbst mit einzubeziehen mehr und mehr in den Vordergrund. Hierbei liegt die große Herausforderung in der Wahrnehmung und Nachahmung von menschlichen Aktionen [1].

Die Studienarbeit beschäftigt sich mit dem System Kinect von Microsoft und einem humaoiden Roboter Nao. Die Kinect ist eine Hardware zur Gestik- und Mimik Erkennung.

## Motivation

Die Motivation ist es den Roboter in einer spielerischen Umgebung mit dem Menschen zu verknüpfen, um so auch manche Berührungsängste mit dieser neuartigen Technik zu nehmen. Zudem kann Nutzern auf diese Weise auch gezeigt werden, welche Möglichkeiten es gibt, Roboter zu nutzen, die bisher noch nicht in Betracht gezogen worden sind. So ist es zum Beispiel denkbar, dass gewisse Bewegungsabläufe, wie das Erlernen von Schwimmbewegungen, in Zukunft über so ein Verfahren bereits trocken gelernt werden können.

## Aufgabenumfeld

Diese Studienarbeit entsteht an der Dualen Hochschule Baden Württemberg (DHBW) und wird von zwei Studenten gemeinsam geschrieben. Zur Verfügung steht ein Labor mit einem Nao und einem Rechner, auf dem alle relevanten Dokumente und Programme genutzt werden können. Zudem können die Programme auch auf den privaten Rechnern installiert werden, so dass mit Hilfe von Simulationssoftware auch von zu Hause aus gearbeitet werden kann.

## Aufgabenbeschreibung und Ziele

Ziel der Arbeit ist es, ein Spiel zu programmieren, bei dem sowohl ein Roboter als auch das Kamerasystem Kinect zum Einsatz kommt. Dem Roboter Nao werden verschiedene Bewegung beigebracht, die er auf Befehl automatisiert ausführen kann.

Die Kinect ist dafür zuständig, die Bewegungen des davorstehenden Spielers aufzunehmen.

Der Ablauf des programmierten Spiels kann in drei Schritte unterteilt werden. Im ersten Schritt führt Nao eine der programmierten Bewegungen vor. Sobald die Roboterbewegung vollendet ist, beginnt Schritt zwei, in dem die Kinect die Bewegung des Spielers aufnimmt. Gleichzeitig läuft eine festgelegte Zeit herunter. Die Aufgabe des Spielers ist es innerhalb der Zeit vom Roboter durchgeführte Bewegung möglich ähnlich nachzuahmen. Nach dem Ablauf der Zeit beginnt Schritt drei, bei dem die getätigte Benutzerbewegung mit der programmierten Roboterbewegung vom System verglichen wird.

Dieser Ablauf mit anschließender Auswertung wird in einem spielerischen Umfeld dargestellt und auf einem Monitor angezeigt.

## Vorgehensweise

Die geplante Vorgehensweise sieht vor zu Beginn dem Nao verschiedene einfache Bewegungen einzuprogrammieren. Anschließend wird die Kinect mit einem Skeleton Stream programmiert und die Winkel am Ende einer Bewegung berechnet, die bei den Bewegungen entstehen. Diese sollen dann mit den Winkel des Roboters verglichen und gewertet werden. Das Ergebnis soll in einer übersichtlichen Graphical User Interface (GUI) dem Nutzer gezeigt werden.

Sofern dieses einfache Beispiel reibungslos funktioniert, werden die Bewegungen des Roboters umfangreicher und komplizierter und es sollen Zwischenwerte der Winkel verglichen werden.

# Grundlagen

Dieses Kapitel schildert die für das Verständnis dieser Arbeit notwendigen Grundlagen. Zu diesem Zweck wird in Kapitel 2.1 die „Kinect-Kamera“ und in Kapitel 2.2 der Roboter NAO vorgestellt.

## Kinect

Die Kinect ist eine Hardware aus dem Hause Microsoft in Zusammenarbeit mit dem Unternehmen PrimeSense. Sie wurde zunächst für die Steuerung der Spielekonsole Xbox360 entwickelt. Die Kinect setzt wie auch die Nintendo Wii oder Sonys Move auf den Körpereinsatz des Benutzers. Der große Unterschied liegt aber darin, dass der Benutzer keinen Controller in der Hand halten muss. Bei der Kinect wird der ganze Körper zum „Controller“. Die Kinect-Kamera ist eine Leiste aus vielen Sensoren (siehe Abb. 1) und erkennt so die Bewegungen der davorstehenden Personen [2], [3].

Nachdem die Kinect so erfolgreich für die Spielekonsole verkauft wurde, ist sie seit Februar 2012 auch für Windows erhältlich. Wie Microsoft selbst schreibt, erhält der Windows-Computer durch die Kinect Augen, Ohre und auch ein Gehirn. Bei der Bestellung der Hardware wird zusätzlich ein Software Development Kit (SDK) mitgeliefert, was nichts anderes ist als eine Sammlung von Werkzeugen und Anwendungen, um eine Software zu erstellen. Das Windows-SDK soll also die Software-Entwicklung für Kinect-Anwendungen erleichtern. Des Weiteren befindet sich eine Dokumentation für die Kinect im Windows-Umfeld im Angebot [4], [5].

### Sensoren der Kinect

In der ca. 20 cm breiten Kinect sind ein Infrarot-Projektor, eine Infrarot-Kamera, eine Farb-Kamera sowie vier Mikrofone verbaut. Damit die Sensoren perfekt auf den Benutzer ausgerichtet werden können, befindet sich zwischen der Kameraleiste und dem Standfuß ein Motor. Mithilfe dieses Motors kann die Sensoren-Leiste um 27° geneigt werden.

Das Sehfeld der Kinect-Sensoren beträgt im Horizontalen 57° und im Vertikalen 43°. Aus diesem Grund muss ein Spieler auch einen Mindestabstand haben, sodass der ganze Körper erkannt werden kann. Laut Herstellerangaben können die Sensoren der Kinect bei einem Abstand von 1,2 bis 3,5 Meter zuverlässig arbeiten. Bei der Xbox sollte der Abstand zwischen Spieler und Kameraleiste bei einem Spieler ca. 1,8 Meter und bei zwei Spieler ca. 2,4 Meter betragen. Bei Anwendungen, die nur den Oberkörper des Spielers benötigen ist dann entsprechend ein geringerer Abstand möglich [6], [7].



Abb. : Die Sensoren der Kinect [8]

Durch die beiden Infrarot-Komponenten, den Infrarot-Projektor und die Infrarot-Kamera, werden die Tiefendaten erzeugt. Sie werden also benötigt um die dritte Dimension hinzuzufügen. Die Grundlage für die 3D-Erfassung ist das strukturierte Licht. Zunächst wird von dem Infrarot-Projektor eine definierte Punkt-Matrix in den Raum gesendet. Die Infrarot-Kamera nimmt das Muster des Infrarot-Lichts wieder auf und gibt die Daten an den internen Chip weiter. Von ihm werden dann aus den Kameradaten die räumlichen Koordinaten ausgerechnet.

Bei dem Infrarot-Projektor handelt es sich um einen Klasse 1-Laser mit einer Wellenlänge von 830 nm. Da wir Menschen nur eine Wellenlänge von ca. 380 nm bis ca. 780 nm wahrnehmen können, liegt der Laserstrahl der Kinect im nicht-sichtbaren Spektrum. Außer dass sie nicht sichtbar sind, hat es noch den wichtigen Vorteil, dass sie selbst bei einem Einfall ins Augeninnere des Benutzers nicht schädlich sind.

Die Farbkamera ist für die Aufnahme von 2D-Bildern zuständig. Die Kamera besitzt hierbei eine Video Graphics Array (VGA)-Auflösung von 640x480 Pixeln.

Die vier Mikrofone werden für eine Audioortung benutzt. Somit lässt sich ein Sprachbefehl eines Benutzers genau zuordnen. Die Mikrofone der Kinect ermöglichen bei einem Einsatz mit einer Xbox, das Xbox-Spiel per Sprache zu starten oder von vorne zu beginnen.

Das Kamerasystem ist mithilfe des Infrarot-Sensors in der Lage Menschen von leblosen Objekten zu unterscheiden. Für jeden erkannten Spieler, der vor der Kinect steht, wird ein Skelett mit 20 Gelenken erstellt.



Abb. : Skeleton mit 20 Punkten [9]

Mithilfe dieser 20 Punkte (auch Joints genannt) können die Körperteile eines Spielers unterschieden und so Bewegungen verarbeitet werden. Außerdem ist es möglich die Gesichter von verschiedenen Personen zu unterscheiden und Aussage über die Mimik zu treffen [5], [10].

### SDK

Zunächst erschien die Kinect im November 2010 als zusätzliche Xbox-Hardware ohne Pläne für PC-Treiber. Viele Leuten wollten die Kinect aber in Verbindung mit einem PC bedienen können, worauf das Unternehmen Adafruit Industries eine Prämie von 1000 US-Dollar für den ersten funktionieren PC-Treiber anbot. Microsoft dachte, dass es sich bei den Programmierungen um Hacking-Versuche im Sinne von Raubkopien oder Cheaten handelt und kündigte an dagegen gerichtlich vorzugehen. Später bemerkte Microsoft, dass die meisten Entwickler das System nicht hacken wollen, sondern nur den Zugriff auf die Sensordaten.

Das Resultat aus dieser Entwicklung ist, dass es nun zwei verschiedene SDKs gibt. Eins davon ist das offizielle SDK von Microsoft, was im Juni 2011 ausgeliefert wurde und das andere ist ein quelloffenes SDK. Diese beiden SDKs stehen in Konkurrenz zueinander und die Anwendungen des einen SDKs sind inkompatibel mit dem andern SDK [11].

### Kinect Developer Toolkit

Im SDK enthalten ist auch das Kinect Developer Toolkit. Es ist also wie das SDK selbst auch ein Werkzeugsatz, der für Kinect-Entwickler gedacht ist. Das englischsprachige Toolkit kann auf der Homepage von Microsoft heruntergeladen und danach auf dem eigenen Rechner installiert werden [12]. Das Toolkit liegt nun in den eigenen Programmen und kann auch wie eine normale Anwendung gestartet werden.

Das Toolkit bringt verschiedene Codebeispiele und weitere Entwicklungsressourcen mit.

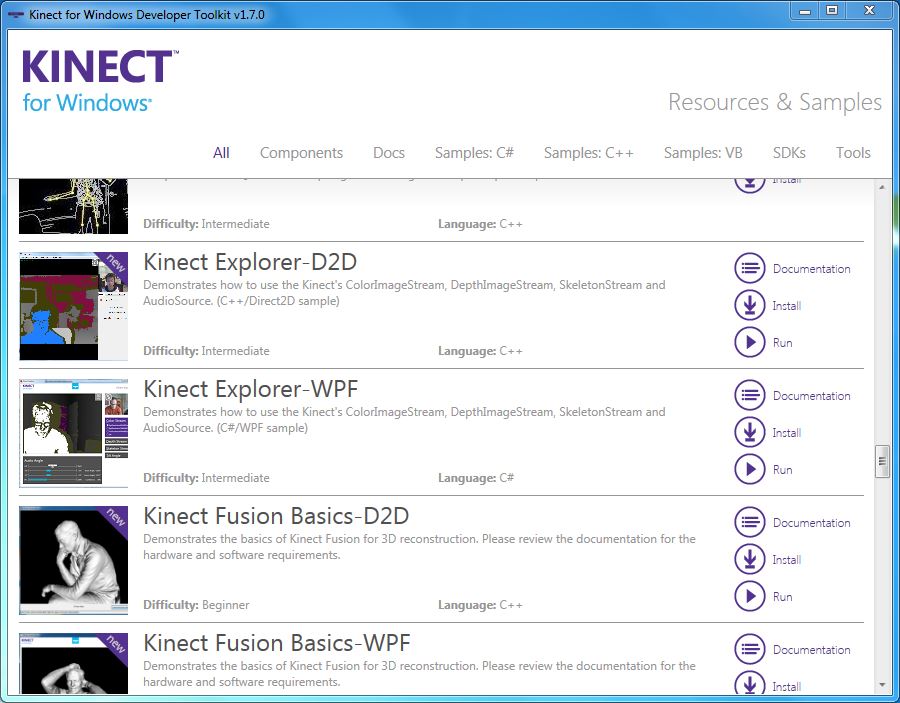


Abb. : Kinect Developer Toolkit

Auf dem Screenshot des Developer Toolkits sind verschiedene Programmbeispiele zu sehen. Jedes dieser Beispiele ist mit einem Bild und einer kurzen Beschreibung erklärt. Darüber hinaus ist die Schwierigkeit sowie die verwendete Programmiersprache aufgezeigt. Zu jedem Programmbeispiel gibt es auf der rechten Seite drei Buttons:

* Documentation
* Install
* Run

Bei Aufruf der Dokumentation wird ein neues Browserfenster geöffnet mit den Informationen des aufgerufenen Beispiels. Es wird also automatisch die Seite von Microsoft mit dem entsprechenden Bericht geöffnet.

Um ein Beispielprogramm starten zu können (run) muss es zunächst installiert werden.

Diese Erweiterung bringt eine Vielzahl zusätzlicher Codebeispiele. Das für die GUI erforderliche Modul hört auf den Namen Microsoft.Kinect.Toolkit und ist in die Projektmappe zu integrieren und mit notwendigen Referenzen zu versehen. Der Code der .xaml-Datei beginnt mit dem Einbinden des für das Toolkit zuständigen Namespaces:

Skeleton-Tracking…

Mit Bilder

## Winkel

### Roll-Nick-Gier-Winkel

### Winkelberechnung

## Humanoide Roboter

Ein humanoider Roboter ist ein Roboter, der dem Mensch nachempfunden ist und dessen Bewegungen die eines Menschen ähneln. Genutzt werden humanoide Roboter zumeist in der Servicerobotik, zum Beispiel als Haushaltshilfe für häufig anfallende Aufgaben. Staubsaugen, Spülmaschine ausräumen oder Toilette putzen sind Aufgaben, die in Zukunft von Robotern ausgeführt werden könnten. Zahlreiche Unternehmen und Einrichtungen befassen sich derzeit mit der Entwicklung und dem Testen von humanoiden Robotern. Darunter sind auch namenhafte Firmen und Universitäten, wie das Karlsruher Institut für Technologie, das Frauenhofer Institut, Toyota und Google [11] [12] [13].

### Nao

Nao ist ein 58cm großer humanoider Roboter der Firma Aldeberan Robotics aus Frankreich und der humanoide Roboter der für die Programmierung genutzt wurde.

Der erste Prototyp vom Nao erschien 2005, seit dem sind mehr als 5000 Naos verkauft worden. Diese werden fast ausschließlich in Bildungseinrichtungen genutzt. [16]

Der Roboter hat die Fähigkeiten sich zu bewegen, Menschen/Gegenstände wahrzunehmen, zu hören und zu sprechen. Dies ist durch die verschiedenen Komponenten wie Mikrophonen und Entfernungssensoren möglich.

#### Hardware

Der humanoide Roboter Nao in der aktuellen Version 4 hat einen integrierten Akku, welcher ca. eine Stunde genutzt werden kann und etwa fünf Stunden zum vollen Aufladen benötigt.

Er ist außerdem mit einem Intel „ATOM Z530“ Prozessor ausgestattet mit 512 KB Cache Speicher und 1 GB RAM. Der eingebaute Flashspeicher hat Platz für 2 GB Daten.

Zur Kommunikation zwischen Computer und Roboter verfügt Nao über Ethernet und Wireless Fidelity. Über seine Internet Protokoll (IP)-Adresse und den Port 9559 erreicht man den Nao. Durch Eingabe der IP in den Browser, kann man noch einige Einstellungen, den Nao betreffend, vornehmen.

Für diese Ausarbeitung wurde Nao in der Version 3.2 bzw. 3+ verwendet. Die Unterschiede liegen in im verwendeten Prozessor (x86 AMD GEODE 500 MHz CPU) und dem Speicher (256MB SCRAM/2GB Flashspeicher). Im Folgenden betrachten wir nur die Version 3.2, da diese die Grundlage für unser Programm darstellt.

Wie in Abb. 4 zu sehen, hat Nao 25 eingebaute Motoren, um die Bewegungen so humanoid als nur möglich aussehen zu lassen. Sechs davon sind für Armbewegungen auf der linken Seite, sechs für Armbewegungen auf der rechten Seite, fünf für Hüftbewegungen, drei für Beinbewegungen der linken Seite, drei für Beinbewegungen der rechten Seite und zwei für den Kopf zuständig. Dabei gibt es vier unterschiedliche Arten von Motoren. Für die Motoren HeadYaw, Head Pitch, ShoulderPitch, ShoulderRoll, EllbowYaw und EllbowRoll warden Motoren des Typs zwei verwendet und simulieren ein Kugelgelenk. Typ-Drei Motoren werden lediglich im Motor WristYaw genutzt, da dort die Drehrichtung der Hand beeinflusst wird. Die Motoren des Typs vier sind ausschließlich bei den Händen zu finden, da diese die Finger nachbilden. Und alle Motoren, die die Hüfte und Beine betreffen sind Motoren des Typs eins und lassen sich in zwei Richtungen (vor und zurück) ändern. [17]



Abb. : Die Lage der Motoren beim Nao [14]

Neben den genannten Aktoren hat der Nao auch einige Sensoren, die es ihm ermöglichen Hindernisse zu erkennen, die Winkeleinstellungen zu messen und vieles mehr.

Die Sensoren kann man in folgende Kategorien einteilen:

* Kontaktsensoren
* Force Sensing Resistoren
* Inertialsensoren
* Inkrementalgeber (rotierend)
* Sonarsensoren

Die Kontaktsensoren bestehen aus dem Brustknopf, bei dessen Drücken Nao seinen Namen, den Akkuladestand und seine IP-Adresse verkündet, den beiden Bumpern je Fuß, die registrieren, ob der Roboter mit seinem Fuß einen Gegenstand berührt und aus taktilen Sensoren. Taktile Sensoren reagieren auf Berührungen. Drei dieser taktilen Sensoren sind beim Nao auf seinem Kopf zu finden und je drei Sensoren befinden sich an jeder Hand.

Ein Force Sensing Resistor ist ein Sensor der die einwirkende Kraft bzw. den Druck misst. Vier dieser Sensoren befinden sich an jedem Fuß des Nao, um bei falscher Krafteinwirkung rechtzeitig zu warnen.

Inertialsensoren sind Sensoren zur Messung von Beschleunigungen und Drehraten. Drei davon befinden sich im Torso des Naos.

Der Nao hat 32 Inkrementalgeber eingebaut und nutzt diese zur Erfassung von Winkelveränderungen und Drehrichtungen.

Zwei Sender und zwei Empfänger für Sonarsensoren sind dem Nao an der Brust eingebaut. Diese dienen zum Messen von Entfernungen im Bereich von 0,25m und 2,55m.

Zudem hat er zwei Kameras, vier Mikrofone und einen Lautsprecher im Kopf integriert, sodass er mit seiner Umwelt interagieren kann.

Die Hardwarekomponenten sind wie folgt verteilt:

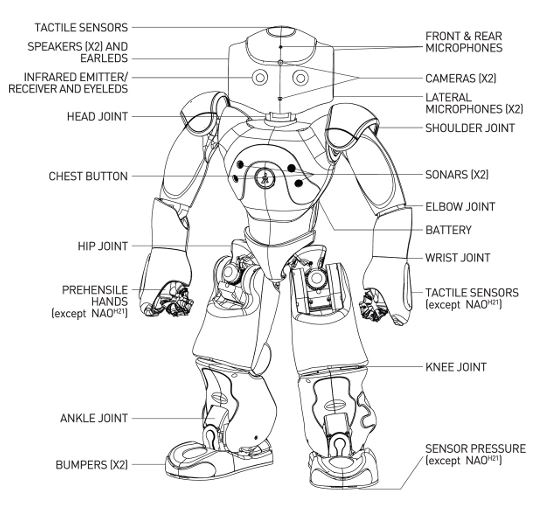


Abb. : Die Hardwarekomponenten des Nao [15]

#### Software

Auf dem Nao läuft ein Linux Embedded Betriebssystem, dem OpenNAO, welches die Software NAOqi ausführt (siehe Abb. 6). OpenNAO ist ein 32-Bit Linux Dristribution, angelehnt an Gentoo.

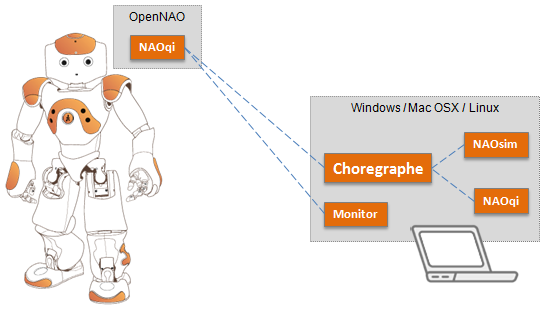


Abb. : Nao Software

Um den Nao zu programmieren, gibt es die Desktop Software „Choreographe“, welche vom Hersteller mitgeliefert wird. Das ist eine visuelle Programmiersprache und hat ebenfalls das Programm NAOqi implementiert. Webots ist die Simulation von einem Nao, früher NAOsim genannt. Bevor ein Stück Programmcode auf dem Nao ausgeführt wird, kann man sich so vergewissern, dass das Programm das tut, von dem man ausgeht und ob dies eventuell zu Problemen auf dem Nao führen kann. So kann verhindert werden, dass der Roboter versucht Bewegungen auszuführen, für die seine Motoren nicht geeignet sind oder ob der Roboter eventuell schaden nimmt, zum Beispiel durch Umfallen.

In der Simulationswelt ist es dabei auch möglich Hindernisse einzubauen, um so nicht nur die Motoren, sondern auch die Sensoren zu testen.

Im Choreographen, wie in Abb. 7 zu sehen, ist es möglich, vorgespeicherte Bewegungen auszuführen, z.B. hinsetzen, hinlegen, gerade hinstellen, Nao Texte sprechen zu lassen und gewisse Abläufe auf ihm zu speichern, sodass diese auch nach Neustarten des Roboters noch verfügbar sind.

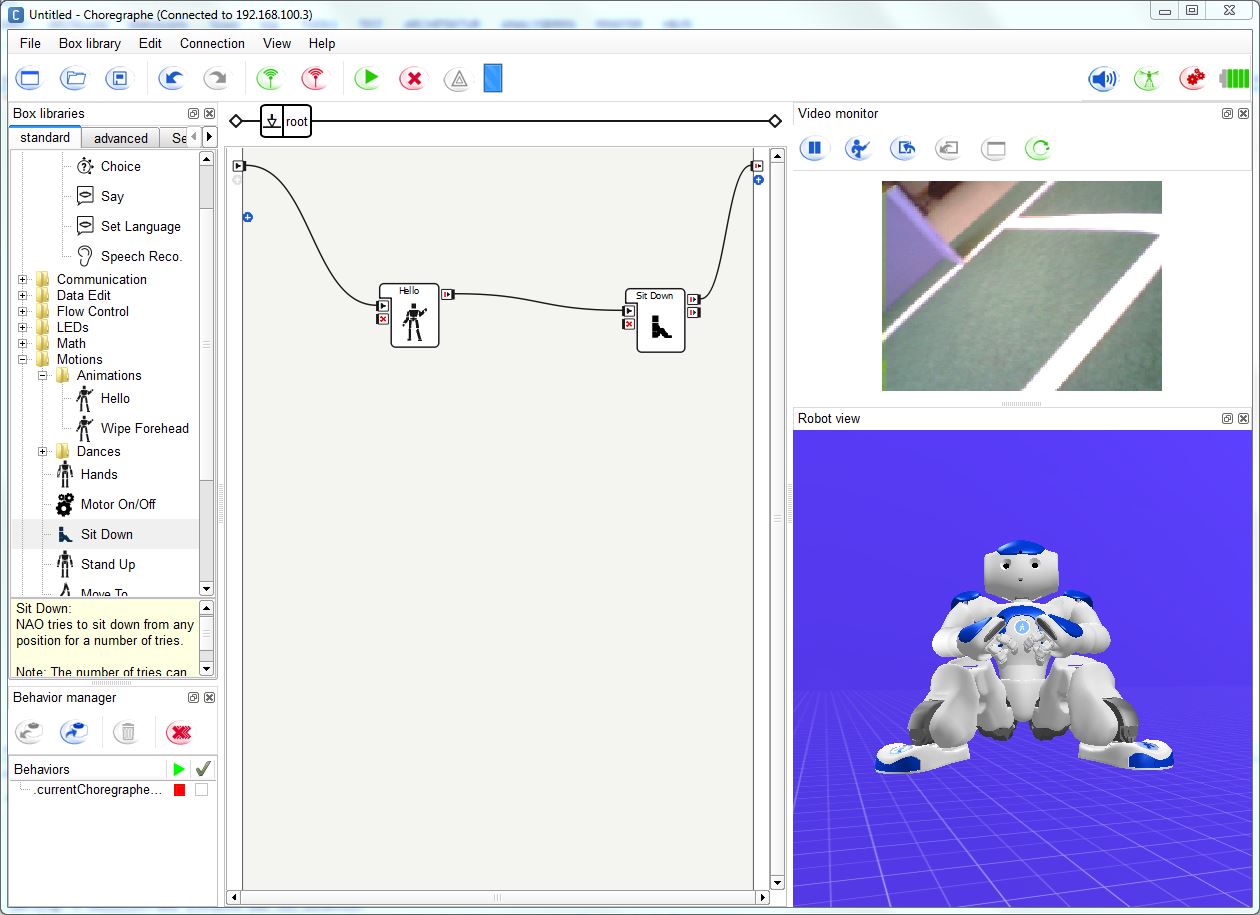


Abb. : Choreographe

Choreographe ist zudem eine der vorhandenen SDKs. Nao kann direkt mit Python und C++ angesprochen werden, über eine Programmierschnittstelle versteht Nao über andere SDKs aber auch Java, MathLab und .NET Sprachen, wie C#, wie man in Abb. 7 zu sehen ist.

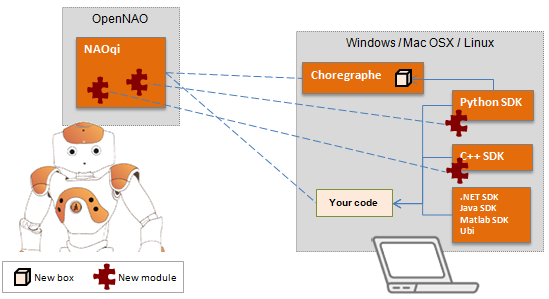


Abb. : Nao Programmiersprachen

Dafür bringt Nao seine eigene Bibliothek für andere Programmiersprachen mit. Dies funktioniert in Visual Studio wie in Abb. 8 zu sehen ist. In diesem Beispiel wird der Nao den Text „Hello World from c sharp“ sprechen.



Abb. : Nao Bibliothek nutzen

TextToSpeechProxy ist eines der Klasse, die in den Bibliotheken integriert sind und ermöglicht es Texte zu definieren, die Nao bei Ausführung des Codes spricht. Andere Klassen sind MotionProxy für frei zu definierende Bewegungen mittels Winkelangabe in Randiant und RobotPostureProxy um vorgefertigte Bewegungen auszuführen. [19]

### Andere humanoide Roboter

Neben dem Roboter Nao gibt es auch andere humanoide Roboter, wie beispielsweise den Roboter Romeo, der ebenfalls von der Firma Aldebaran Robotics hergestellt wird und den bereits genannten Eigenentwicklungen einiger wissenschaftlichen Institute. Da während der Ausarbeitung lediglich mit dem Roboter Nao gearbeitet wurde, werden die anderen humanoiden Roboter nicht weiter betrachtet.

## Entwicklungsumgebung

Microsoft empfiehlt die Nutzung von C++, C# oder Visual Basic für die Kinect, da es für diese Programmiersprachen vorgefertigte Klassen im SDK gibt, auf welche man zurückgreifen kann. Für den Roboter Nao kann man wie neben C++ unnd Python ebenfalls C# nutzen.

Da sowohl die Kinect, als auch der Nao mit C# arbeiten (können), ist auch die Anwendung in C# programmiert. So ist es möglich, dem Nao Daten zur Ausführung zu senden und hat diese dann im gleichen Format vorliegen, wie sie später von der Kinect aufgezeichnet und an die Anwendung gesendet werden. Es ist also keine Konventierung der Daten im Programm mehr nötig.

## Threads

# Konzeption/Planung

Zu Beginn wird die Entwicklungsumgebung eingerichtet, inklusive der SDKs von Kinect und Nao. Dann werden dem Nao einige Bewegungen einprogrammiert, wobei es sich zu Beginn um einfache Bewegungen handelt, damit alle Komponenten auf ihre Funktionstüchtigkeit getestet werden können. Außerdem sollen die entsprechenden Winkel der durchgeführten Bewegung ausgelesen und an eine Klasse übermittelt werden. Diese Klasse ist dann auch dafür zuständig, dass die Winkel der Nao-Bewegung mit den Winkel, welche die Kinect aufgezeichnet hat, verglichen werden.

Die Kinect bringt einige fertige Programmierungen mit, unter anderem eine Programmierung, in der die Skelette von Personen sowie deren Entfernung erkannt wird. Dieses Programm soll als Grundlage dienen, mit dem der Nutzer seine Bewegungen selbst sieht und das Programm die Winkel der Nutzer errechnen kann.

Es bleibt zu überlegen ob verschiedene Schwierigkeitsstufen darüber abgebildet werden, dass es unterschiedlich schwere Bewegungen zum Nachmachen gibt, die stetig komplizierter und/oder länger werden oder über die prozentuale Übereinstimmung der beiden Bewegungen bzw. derer Winkel.

Da für den Menschen kleinere Unterschiede in den Winkeln nicht sichtbar sind, wird beim Vergleich dieser ein Filter eingesetzt werden. Dieser Filter rundet die empfangenen Werte und schließt so Unterschiede im nicht wahrnehmbaren Bereich aus.

Zunächst werden lediglich die Winkel des Starts und Endes der Benutzer-Bewegung gemessen und mit den gespeicherten Werten des Naos verglichen. Sofern dieser Mechanismus funktioniert sollen die Winkel der Bewegungen an weiteren Zwischenpunkten verglichen werden. Dies verhindert, dass der Spieler Bewegungen abkürzen und somit schummeln kann.

Um die Applikation auf ihre Funktionsfähigkeit zu testen, wird jede der Bewegungen zehn mal hintereinander durchgeführt und die Rate ermitteln, wie viele Bewegungen anerkannt werden. Auf diesen Ergebnissen aufbauen werden die entsprechenden Filter angepasst.

# Umsetzung

In diesem Kapitel wird auf die komplette Entwicklung eingegangen. Im ersten Unterkapitel wird die Struktur des kompletten Programms erläutern. Da die Struktur sich größtenteils in zwei Teilgebiete einordnen lässt, wird in den beiden darauffolgenden Unterpunkten die Nao- und Kinect-Programmierung erläutert. Danach werden die Themen GUI und Test beschrieben.

## Programmstruktur / Architektur - Dennis

Das Spiel *SpielNaoKinect* wird mit der in Abb. 7 gezeigten Code Map beschrieben. Die einzelnen Klassen, die man auf der Abbildung sieht, können zusätzlich noch aufgeklappt werden, sodass auch die verwendeten Variablen und Methoden sichtbar sind.

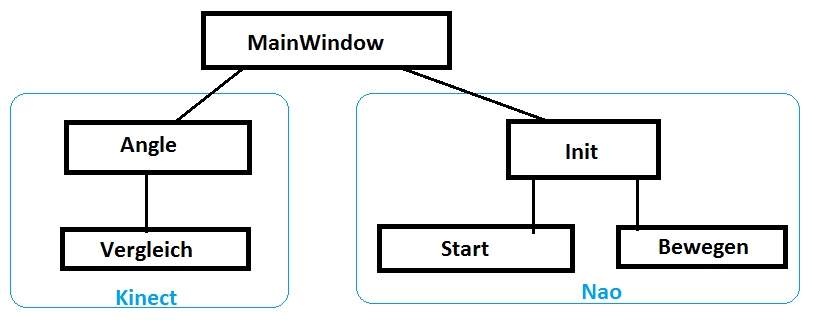


Abb. : Code Map

***MainWindow*:**

Die Klasse *MainWindow* ist der Einstiegspunkt des Programms und für die visuelle Darstellung der Anwendung zuständig. Außerdem dient die Klasse als Brücke für den Kinect- und Nao-Teil. Zunächst erfolgt eine Überprüfung der Kinect-Sensoren, also ob die Kinect angeschlossen ist und ob ein Bild dargestellt werden kann. Wenn die Überprüfung erfolgreich verlief, werden die Klassen *Angle* und *Init* initialisiert. Außerdem wird die Methode Initialisierung von der Klasse *Init* aufgerufen. Bei diesem Aufruf wird die IP-Adresse und der Port vom Nao übergeben. Die IP-Adresse und der Port sind hart im Code verankert, jedoch muss bei einer Änderung einer der beiden Werte lediglich eine Stelle im Code angepasst werden.

Die Oberfläche mit all ihren Funktionen wird in Kapitel 4.4 beschrieben.

***Init* (Nao-Bereich):**

Diese Klasse hat drei Methoden:

* *Initialisierung*Die Methode *Initialisierung* wird nur einmalig beim Start des Programms aufgerufen und ist wie der Name ahnen lässt für die Initialisierung eines MotionProxys, RobotPostureProxy und TextToSpeechProxys zuständig. Des Weiteren werden die Klassen *Start* und *Bewegen* initialisiert und anschließend die Methode *Startposition* von *Start* aufgerufen.
* *Bew\_Winkel*Diese Methode wird aufgerufen, wenn Nao eine Bewegung vorführen soll. Je nach dem ob Nao eine neue Bewegung vormachen soll oder die bereits ausgeführte Bewegung wiederholen soll wird eine vorgegebene Methode in der Klasse *Bewegung* aufgerufen.
* *Bew\_Ausgangsposition*Hierbei wird die Methode *Ausgangsposition* der Klasse *Bewegen* aufgerufen.

***Start* (Nao-Bereich):**

Diese Klasse hat nur die eine Methode *Startposition*. In dieser Methode bewegt sich Nao in die Startposition StandInit. Außerdem sagt Nao parallel dazu „Herzlich Willkommen zum Spiel“.

***Bewegen* (Nao-Bereich):**

*Bewegen* ist für die Durchführung einer Roboterbewegung zuständig. Alle Methoden dieser Klasse werden entweder von *Bewegen* selbst oder von *Init* aufgerufen.

Wird beispielsweise die Methode *Ausgangsposition* aufgerufen, so geht Nao wieder in seine Standardposition StandInit, wie bei Spielstart.

Wenn eine neue Bewegung ausgeführt werden soll, dann wird eine Bewegung per Zufall ausgewählt und der Roboter führt diese vorgegebene Bewegung mittels des erzeugten MotionProxys aus.

***Angle* (Kinect-Bereich):**

Diese Klasse kommt währenddessen der Benutzer die Bewegung von Nao nachmachen soll in einer while-Schleife zum Einsatz. Die while-Schleife, die eine bestimmte Zeit (in unserem Fall 10 Sekunden) ausgeführt wird, ruft jedes Mal die Methode *Berechnen* auf. In dieser Methode werden die aktuellen Joints von der vor der Kinect stehenden Person berechnet. Mithilfe dieser Joints können danach die Winkel bestimmt werden und es erfolgt das Weiterreichen dieser Werte an die Klasse *Vergleich*.

***Vergleich* (Kinect-Bereich):**

In dieser Klasse gibt es für jeden Winkel der vergleicht wird eine extra Methode:

* Achsel\_links\_roll
* Achsel\_rechts\_roll
* Achsel\_rechts\_pitch
* Achsel\_links\_pitch
* Ellenbogen\_rechts\_roll
* Ellenbogen\_lins\_roll

In allen Methoden erfolgt zunächst eine Anpassung der übergebenen Kinect-Winkel, sodass die Winkel von dem Spieler und des Naos übereinstimmen. Letztendlich erfolgt der Abgleich, ob die beiden Winkel nahezu identisch sind.

### Programmablaufplan einer Bewegungsnachahmung

Dieses Kapitel erläutert den programminternen zeitlichen Ablauf, wenn der Spieler eine Bewegung nachmachen möchte. Der Programmablaufplan wurde in zwei Bilder gesplittet, sodass die zwei wichtigen Teile getrennt voneinander betrachtet werden können.

Neben jeder Aktion ist in roter Farbe der Klassenname notiert, sodass die Zuordnung zwischen der Aktion und einer konkreten Klasse nochmals verdeutlicht wird.

Bei den beiden Abbildungen wird zusätzlich auf die gesamte Thread-Struktur eingegangen. In dunkelblauer Farbe ist das Starten eines Threads gekennzeichnet bzw. die Überprüfung ob ein Thread fertig ist. In hellblauer Farbe sind die Namen angegeben. Sodass es ersichtlich ist, was in einem Thread passiert sind auch alle Aktionen eines Threads in hellblau umrandet.

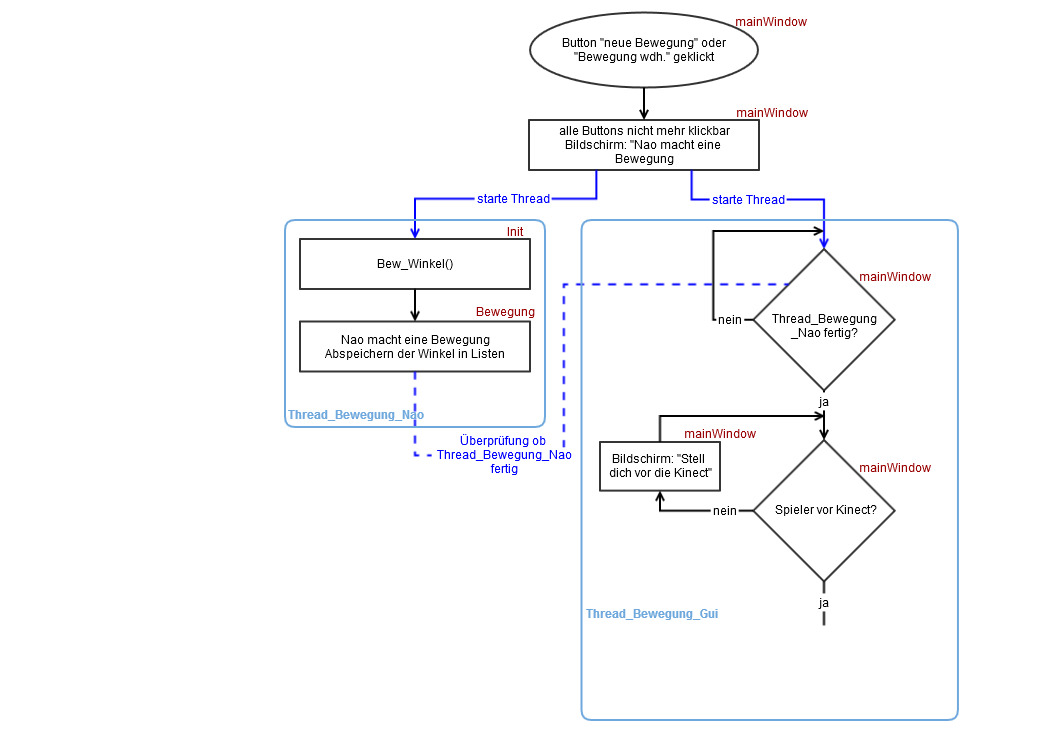


Abb. : Programmablaufplan – erster Teil

Wenn ein Spieler eine Bewegung von Nao nachmachen möchte, muss auf der GUI entweder auf den Button „neue Bewegung“ oder „Bewegung wdh.“ geklickt werden. Dieser Klick ist auch gleichzeitig der Start des internen Ablaufs. Auf der grafischen Oberfläche des Spiels erscheint nun der Text „Nao macht eine Bewegung“ und alle Buttons sind nicht mehr anklickbar.

Danach werden zwei Threads (Thread\_Bewegung\_Nao und Thread\_Bewegung\_GUI) gestartet.

Im Thread\_Bewegung\_Nao wird die Methode Bew\_Winkel() aus der Klasse *Init* aufgerufen und dann in der Klasse *Bewegung* die Nao-Bewegung ausgeführt. Währenddessen die Bewegung ausgeführt wird, werden verschiedene Winkel in Listen gespeichert. Wenn die Bewegung vom Nao fertig ist, beendet sich automatisch dieser Thread.

Thread\_Bewegung\_GUI ist zwar schon gestartet, wartet aber in einer while-Schleife bis der parallel-laufende Thread beendet ist und erst dann erfolgt der weitere Ablauf. Im nächsten Schritt wird überprüft, ob die Kinect einen Spieler erkennt. Falls dies nicht zutrifft, erscheint auf dem Bildschirm „Stell dich vor die Kinect“ und es wird gewartet bis ein Spieler erkannt wird. Ist nun ein Spieler erkannt, erfolgt der weitere Ablauf, der im zweiten Teil nun erläutert wird.

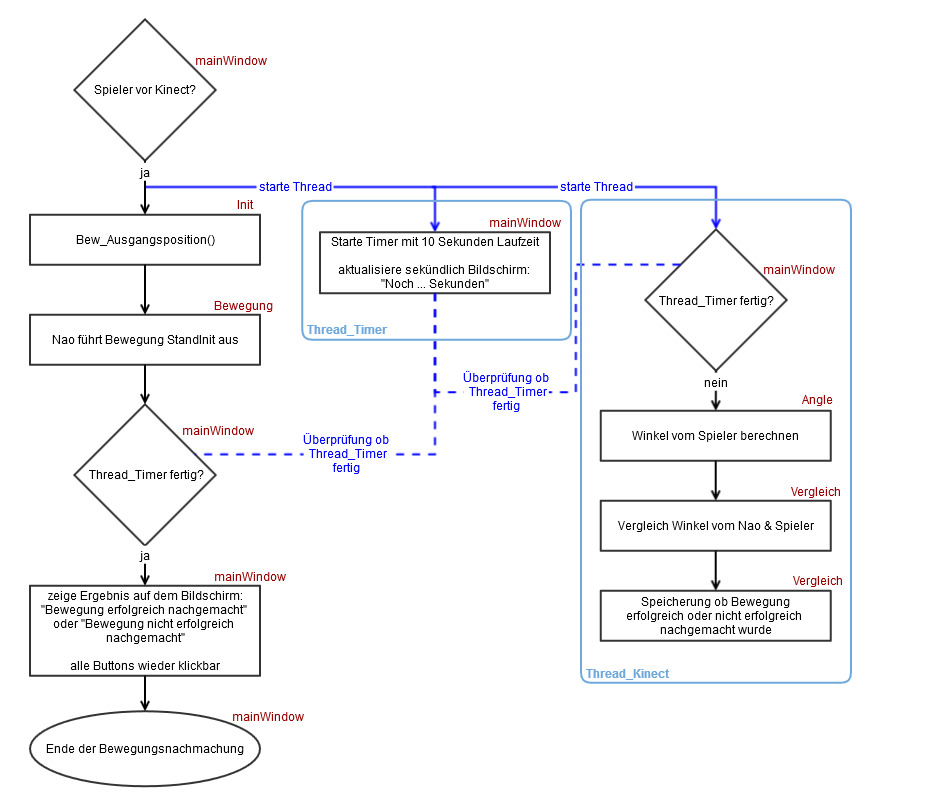


Abb. : Programmablaufplan – zweiter Teil

Bis zum Anfang dieser Abbildung wurde die Bewegung vom Nao ausgeführt, die Winkel dieser Bewegung gespeichert und es wurde ein Spieler von der Kinect erkannt.

Nun werden erneut zwei Threads (Thread\_Timer und Thread\_Kinect) gestartet. Ab diesem Zeitpunkt werden nun drei Aktionen parallel ausgeführt.

In der Klasse *Init* wird die Methode Bew\_Ausgangsposition() aufgerufen, woraufhin Nao wieder in seine Ausgangsposition StandInit geht. An dieser Stelle Nao wieder in seine Ausgangsposition zu stellen hat den Vorteil, dass jede Bewegung mit derselben Ausgangsposition startet. Nach der durchgeführten Bewegung kommt eine Schleife, die solange ausgeführt wird, bis der Thread\_Timer fertig ist.

Parallel dazu wird in Thread\_Timer (mittlere Spalte der Abbildung) ein Timer mit einer Laufzeit von 10 Sekunden gestartet. In diesem Thread wird dann auch immer die grafische Oberfläche aktualisiert, auf der dann die verbleibende Dauer des Timers angezeigt wird.

Die dritte Parallelität (Thread\_Kinect; rechte Spalte der Abbildung) wird solange ausgeführt, bis der Timer fertig ist. In diesem Thread wird in der Klasse *Angle* die Winkel des Spielers berechnet und die Werte an die Klasse *Vergleich* übergeben. Dort werden dann die Winkel vom Nao, die in Listen vorliegen, mit den aktuellen Spieler-Winkeln verglichen. Der letzte Schritt dieses Threads ist die Speicherung, ob die Bewegung erfolgreich oder nicht erfolgreich nachgemacht wurde.

Sobald der Timer fertig ist, erfolgt die weitere Abarbeitung des Hauptzweigs (linke Spalte der Abbildung). Es werden die vorherig gespeicherten Werte angeschaut und dementsprechend ein Text „Bewegung erfolgreich nachgemacht“ oder eben „Bewegung nicht erfolgreich nachgemacht“ angezeigt. Der letzte Schritt ist nun die Buttons wieder zu aktivieren, sodass der Benutzer im nächsten Schritt eine neue Aktion auswählen kann. Somit ist die komplette Bewegungsnachahmung inklusive der Auswertung vollendet.

### Warum Threads?

Im Programmcode werden, wie bei den beiden den Abbildungen XXX und XXX zu sehen, an zwei verschiedenen Stellen Threads verwendet.

Threads werden im Allgemeinen benötigt um mehrere Verarbeitungsschritte gleichzeitig auszuführen, sodass mehr als eine Operation ausgeführt werden kann.

Da ein Timer inklusive einer Oberflächenanpassung und die Bewegungserkennung des Spielers inklusive eines Winkelvergleich parallel ablaufen soll, wird ein Thread benötigt.

In Thread\_Timer wird ein Timer heruntergezählt und sekündlich die GUI angepasst. Während diesem Zeitraum soll gleichzeitig die Kinect die Bewegungen des Spielers aufnehmen, die aktuellen Winkel speichern und diese mit den gespeicherten Roboterwinkeln vergleichen (siehe Abb. 9).

**1 Thread[] thread = new Thread[2];**

**2 thread[0] = new Thread(new ThreadStart(Thread\_Timer));**

**3 thread[1] = new Thread(new ThreadStart(Thread\_Kinect));**

**4 thread[0].SetApartmentState(ApartmentState.STA);**

**5 thread[1].SetApartmentState(ApartmentState.STA);**

**6 thread[0].Start();**

**7 thread[1].Start();**

Code : Thread

In der ersten Zeile von Code 1 wird ein Threadarray mit einer Größe von 2 erzeugt. In Zeile 2 wird dem nullten Element des Arrays der Thread „Thread\_Timer“ zugewiesen. In der nächsten Zeile erfolgt das gleiche mit dem ersten Element des Arrays und „Thread\_Kinect“.

Sodass Threads auch Zugriff auf die anderen parallel laufenden Threads haben, müssen sie vom Typ Singlethreaded Apartment (STA) sein. Dies geschieht in Zeile 4 und 5. Ein Apartment ist ein logischer Container, in dem alle Objekte (in diesem Codebeispiel Thread\_Timer und Thread\_Kinect) gespeichert werden, welche dieselben Anforderungen an den Threadzugriff haben. Somit können beide Threads Aufrufe des anderen Threads empfangen. Außerdem hat dies den Vorteil, dass die Ressourcen threadsicher verwendet werden.

Im nächsten Schritt (Zeile 6 und 7) werden beide Threads mit der Start()-Methode gestartet und nun kann die parallele Abarbeitung von **private void Thread\_Timer(){ … }** und **private void Thread\_Kinect(){ … }** beginnen [17], [18].

An manchen Stellen ist es wichtig zu wissen, ob ein Thread bereits beendet ist oder noch ausgeführt wird. Hierzu betrachten wir die andere Stelle im Code, bei der die Threads Thread\_Bewegung\_Nao und Thread\_Bewegung\_Gui zum Einsatz kommen (siehe Abb. 8). Beide Threads werden, wie oben beschrieben, gestarten und laufen nun parallel. Jedoch soll Thread\_Bewegung\_Gui erst weitere Aktionen ausführen, sobald Thread\_Bewegung\_Nao fertig ist.

**1 private void Thread\_Bewegung\_Gui()**

**2 {**

**3 //…**

**4 while (thread[0].IsAlive);**

**5 //…**

**6 }**

Code : Thread 2

In diesem Codestück ist Zeile 4 für das Warten verantwortlich. Der Code wird bis zur vierten Zeile parallel zum Code von Thread\_Bewegung\_Nao ausgeführt. Der Befehl „thread[0].IsAlive“ gibt wahr zurück, wenn der Thread\_Bewegung\_Nao aktiv ist. Also wird in der while-Schleife solange gewartet bis der Thread nicht mehr ausgeführt wird. Erst wenn dies eintrifft wird auch der Code unterhalb Zeile 4 abgearbeitet.

## Nao – Dennis

### Zufällige Bewegung ausführen

Insgesamt gibt es XXX vorgegebene Bewegungen für Nao. Wenn der Benutzer eine neue Bewegung nachmachen möchte, wird folgender Code ausgeführt:

**1 private int Bewegungsnummer;**

**2 public void Bewegung\_erzeugen()**

**3 {**

**4 Random r = new Random();**

**5 Bewegungsnummer= r.Next(1, 11);**

**6 Bewegung();**

**7 }**

**8 public void Bewegung()**

**9 {**

**10 //…**

**11 switch (Bewegungsnummer)**

**12 {**

**13 case 1:**

**14 //Bewegung 1**

**15 break;**

**16 case 2:**

**17 //Bewegung 2**

**18 break;**

**19 //…**

**20 case 10:**

**21 //Bewegung 10**

**22 break;**

**23 }**

21 }

In der ersten Zeile wird eine int-Zahl Bewegungsnummer definiert. Von außen wird die Methode „Bewegung\_erzeugen()“ aufgerufen, welche eine zufällige Zahl bestimmt um somit eine zufällige Bewegung auszuwählen. In Zeile 4 wird die Zufallsklasse initialisiert und anschließend eine zufällige Zahl zwischen eins und zehn in der Variable Bewegungsnummer gespeichert. Danach wird in die Methode „Bewegen()“ gesprungen. Ab Zeile 11 erfolgt dann eine switch-Anweisung mit der Übergabe der vorherigen Zufallszahl Bewegungsnummer. Wurde beispielsweise durch die Random-Funktion Bewegungsnummer=6 gesetzt, dann wird auch die sechste Bewegung ausgeführt.

### Bewegung wiederholen

Möchte der Spieler, dass Nao nochmals die letzte Bewegung vorführt, muss er den Button „Bewegung wdh.“ betätigen. Dieser Fall ist ähnlich zu dem Fall aus Kapitel 4.2.1, bei dem eine zufällige Bewegung ausgeführt wurde. Da sich die int-Zahl Bewegungsnummer seit dem letzten Aufruf nicht mehr geändert hat, ist der Ablauf einfach und es muss nur die Methode „Bewegung()“ ausgeführt werden (vergleiche Abbildung XXX). Die Bewegungsnummer ist noch identische und deshalb wird nochmals die identische Bewegung von Nao ausgeführt.

### Speicherung Winkel vom Nao

## Kinect

### Berechnung Winkel Kinect

### Vergleich Winkel Kinect <-> Nao

### Schwierigkeitsgrade

Die Schwierigkeitsgrade sind mit der Übereinstimmung realisiert.

### Timer

## GUI

Die GUI wie in Abb. 13 zu sehen besteht aus der Ansicht von der Kincet, in der zu sehen ist, wie der Nutzer die Bewegung nachmacht. Dabei wird sein Skelett schematisch dargestellt. Auf der rechten Seite befinden sind drei Buttons:

* „Neue Bewegung“
* „Bewegung wdh“
* „Neuer Spieler“

Die Buttons „Bewegung wdh“ und „Neuer Spieler“ sind ausgegraut, bis die erste Bewegung abgeschlossen ist.

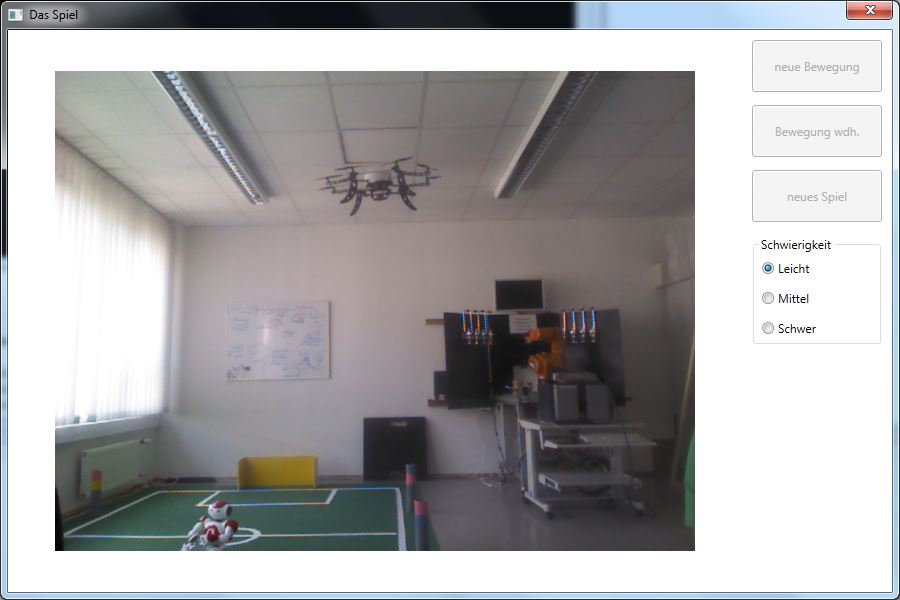


Abb. : GUI zum Spiel

Während eine neue Bewegung vom Nao ausgeführt wird oder die letzte Bewegung wiederholt wird, ist in der GUI der Text „Der Roboter macht eine Bewegung vor“. Dieser wechselt nach Abschluss der Bewegung auf „Bitte mache die Bewegung des Roboters so gut wie möglich nach, falls du sie noch einmal sehen möchtest klicke auf ‚Bewegung wdh‘“.

Während der Spieler die Bewegung nachmacht, geht der Roboter in die Ausgangsposition zurück und in der rechten oberen Ecke des Bildes läuft ein Timer von 10 Sekunden ab. In dieser Zeit hat der Spieler die Möglichkeit die Bewegung nachzumachen. Je nach Korrektheit der nachgemachten Bewegung erhält der Spieler Punkte, welche oberhalb der Buttons zu sehen sind.

Durch Klicken auf „Neuer Spieler“ werden die bisher gesammelten Punkte auf Null gesetzt und der gleiche oder ein anderer Spieler kann versuchen, eine höhere Punktzahl zu erreichen.

## Test

Jede Bewegung wurde 20 mal vor der Kinect durchgeführt. Die beste Bewegung war das Heben der rechten Hand, wobei die Bewegung 18 mal als erfolgreich anerkannt wurde. Am schlechtesten wurde die Bewegung … mit 8 Bewegungen erkannt.

# Fazit

Das folgt am Ende.

Literaturverzeichnis

|  |  |
| --- | --- |
| [1] | Karlsruher Institut für Technologie, „Imitationslernen in der Robotik,“ 2012. [Online]. Available: http://www.hyperraum.tv/tag/autonome-roboter/. [Zugriff am 15 01 2014]. |
| [2] | Wikipedia, „Kinect,“ 10 01 2014. [Online]. Available: http://de.wikipedia.org/wiki/Kinect. [Zugriff am 15 01 2014]. |
| [3] | Microsoft, „Kinect für Windows,“ [Online]. Available: http://www.microsoft.com/en-us/kinectforwindows/. [Zugriff am 15 01 2014]. |
| [4] | Golem, „SDK - Software Development Kit,“ [Online]. Available: http://www.golem.de/specials/sdk/. [Zugriff am 15 01 2014]. |
| [5] | J. Roßberg, „Alle Informationen zur neuen Bewegungssteuerung,“ 01 11 2010. [Online]. Available: http://www.gamepro.de/xbox/spiele/xbox-360/kinect-adventures/artikel/kinect,46324,1967600.html. [Zugriff am 15 01 2014]. |
| [6] | Computer Bild, „Technische Details zur Bewegungserkennung,“ Computer Bild, 30 06 2010. [Online]. Available: http://www.computerbild.de/artikel/cbs-News-Xbox-360-Kinect-3D-Kamera-technische-Details-5405436.html. [Zugriff am 01 04 2014]. |
| [7] | Microsoft, „Problembehebung bei der Bewegungserkennung,“ Microsoft, [Online]. Available: http://support.xbox.com/de-DE/xbox-360/kinect/body-tracking-troubleshoot. [Zugriff am 01 04 2014]. |
| [8] | ifixit.com, „Kinect ohne Gehäuse,“ 05 01 2012. [Online]. Available: http://wiki.zimt.uni-siegen.de/fertigungsautomatisierung/index.php/Datei:S910310\_abb2.png. [Zugriff am 10 03 2014]. |
| [9] | Microsoft, „Tracking Users with Kinect Skeletal Tracking,“ [Online]. Available: http://msdn.microsoft.com/en-us/library/jj131025.aspx. [Zugriff am 10 03 2014]. |
| [10] | Steffen, „ Einsatzmöglichkeiten einer 3D-Kamera in der Produktionstechnik am Beispiel der Kinect-Kamera,“ 13 02 2013. [Online]. Available: http://wiki.zimt.uni-siegen.de/fertigungsautomatisierung/index.php/Einsatzm%C3%B6glichkeiten\_einer\_3D-Kamera\_in\_der\_Produktionstechnik\_am\_Beispiel\_der\_Kinect-Kamera. [Zugriff am 01 04 2014]. |
| [11] | T. Hanna, in *Microsoft Kinect - Programmierung des Sensorsystems*, Heidelberg, dpunkt.verlag GmbH, 2013, p. 4. |
| [12] | Microsoft, „Downloads Kinect for Windows,“ Microsoft, [Online]. Available: http://www.microsoft.com/en-us/kinectforwindowsdev/Downloads.aspx. [Zugriff am 13 04 2014]. |
| [13] | N. Demuth, „Die Welt,“ 27 08 2013. [Online]. Available: http://www.welt.de/wissenschaft/article119430009/Mit-solchen-Maschinen-werden-wir-alleine-sein.html. [Zugriff am 05 03 2014]. |
| [14] | Die Zeit, „zeit.de,“ 04 12 2013. [Online]. Available: http://www.zeit.de/wirtschaft/unternehmen/2013-12/google-roboter-android-erfinder-andy-rubin. [Zugriff am 05 03 2014]. |
| [15] | Frauenhofer Institut, „Frauenhofer Institut - Humaniode Roboter,“ [Online]. Available: https://www.iosb.fraunhofer.de/servlet/is/5093/. [Zugriff am 05 03 2014]. |
| [16] | Aldebaran Robotics, „Aldebaran Robotics Nao Software Documentation,“ [Online]. Available: https://community.aldebaran-robotics.com/doc/1-14/family/robots/motors\_robot.html. [Zugriff am 06 03 2014]. |
| [17] | Aldebaran Robotics, „Aldebaran Robotics,“ [Online]. Available: http://www.aldebaran-robotics.com/en/Discover-NAO/Key-Features/hardware-platform.html. [Zugriff am 06 03 2014]. |
| [18] | Microsoft, „ApartmentState-Enumeration,“ Microsoft, [Online]. Available: http://msdn.microsoft.com/de-de/library/system.threading.apartmentstate.aspx. [Zugriff am 09 04 2014]. |
| [19] | Galileo Computing, „Multithreading mit der Klasse Thread,“ Galileo Computing, 2010. [Online]. Available: http://openbook.galileocomputing.de/visual\_csharp\_2010/visual\_csharp\_2010\_11\_002.htm#mj5eadcfd43437a62e48102a560638faea. [Zugriff am 09 04 2014]. |

Alle Abbildungen, Tabellen, o.ä., die keine Literaturangabe enthalten, sind eigene oder firmenintern verwendete Darstellungen.