

Dokumentation NAO Arm-Kontrollmodul mit Wiimote



Eine Projektarbeit von:

van Thinh Le 19195

Inhaltsverzeichnis

1. Ziele der Projektarbeit.....	3
2. Voraussetzungen zum Betrieb.....	3
3. Die Wiimote.....	4
3.1 Wiimotes Beschleunigungssensor.....	4
3.2 Berechnung von Pitch und Roll.....	5
4. WiiYourself library.....	6
5. NAOs Architektur.....	7
5.1 Betriebsmöglichkeiten des Remote Moduls.....	7
5.2 Remote Module.....	7
6. Inbetriebnahme des Moduls.....	9
6.1 NAO Start.....	9
6.2 Mit Wiimote verbinden und Modul starten.....	9
7. Modulablauf.....	9
7.1 NAO Armsteuerung Bedienungsanleitung.....	10
7.2 NAO Armkomponenten.....	10
8. Modul Architektur.....	12
8.1 Aufgaben der Klassen.....	13
8.2 Steuerungsfunktion.....	13
9. Armspezifikation.....	14
10. Probleme bei der Umsetzung.....	15
10.1 Start.....	15
10.2 WiiYourself und NAO SDK.....	15
10.3 Steuerung.....	16
10.4 Coding.....	16
11. Verbesserungsmöglichkeiten.....	16
12. Tutorial zur Modulerstellung / Modulmodifikation.....	17
13. Wiimote Auswertungs-Tool.....	20
14. Fazit.....	20
15. Quellen.....	21
15.1 Bilder.....	21
15.2 Literatur.....	21

1. Ziele der Projektarbeit

Die Aufgabe bei der Projektarbeit besteht darin, eine Armsteuerung für Aldebarans NAO Roboter zu realisieren. Die beiden Roboterarme soll über die Wiimote gesteuert werden. Ziel ist es, eine möglichst intuitive Bedienung zu ermöglichen und einen Leitfaden zur Modulerstellung für spätere NAO-Projekte zu geben. An die Dokumentation wird die Anforderung gestellt, dass sie leicht verständlich ist und auch als Nachschlagewerk für spätere NAO Projekt dienen kann. Eine Weiterentwicklung durch einen anderen Studenten soll damit erleichtert werden.

2. Voraussetzungen zum Betrieb

Folgende Dinge werden zusätzlich benötigt um NAOs Arme mit dem kompilierten Modul ansteuern zu können (Gestetet mit untenstehenden Versionsnummern) :

- Windows 7
- pthreadVCE2.dll (muss im selben Pfad wie die Executables liegen bzw. Im Debug Verzeichnis des Projektes)
- Bluetoothfähiger Windows-Rechner der die Wiimote erkennt
- Wiimote Controller
- WiiYourself Vers. 1.15 (Wiimote Library)
- Roboter NAO
- (AldeBaran SDK Vers. 1.3.17 für Simulationstests)

Um mit WiiYourself zu entwickeln oder um das Armsteuerungs-Modul zu bearbeiten:

- aktuelle Version von Cmake
- MS Visual Studio 2008
- WinDDK 7.1
- Python 2.6.4
- Microsoft Windows SDK for Windows 7

Das Modul für die Nao-Armsteuerung wurde auf Windows 7 programmiert und getestet. Es wird Python benötigt, da das SDK für den NAO darauf basiert. WiiYourself ist eine C++ Library, die mit der Wiimote kommunizieren kann.

Damit WiiYourself beim debuggen unter Visual Studio auf die Wiimote zugreifen kann, wird zusätzlich die Windows SDK und das Windows Driver Development Kit (WinDDK) gebraucht. Das liegt daran, dass Windows die Wiimote über Bluetooth als Human Interface Device (HID) erkennt.

Das Windows SDK und das WinDDK wird allerdings nur auf Coding Ebene gebraucht, um das Nao-Armkontrollmodul bei Bedarf umzuschreiben oder um ein ganz neues Modul anzulegen. Für die Ausführung des kompilierten Moduls werden sie nicht benötigt. Wie die Einbindung funktioniert wird im Kapitel 12 erklärt.

3. Die Wiimote

Für die Steuerung der Arme werden die Parameter Pitch und Roll der Wiimote ausgelesen. Pitch gibt den Anstellwinkel der Wiimote an. Roll gibt die Neigung nach links und rechts in Grad an. Die Parameter Roll und Pitch werden über die Beschleunigungsdaten der Wiimote berechnet. Wie diese Beschleunigungsdaten aufgefasst werden, wird im nächsten Punkt erklärt.



Abbildung 1: Pitch



Abbildung 2: Roll

3.1 Wiimotes Beschleunigungssensor

Der Beschleunigungssensor der Wiimote liefert 3 Werte zurück, die die Beschleunigung der Wiimote in der Einheit g angeben ($g = 9,81 \text{ m/s}^2$). In dem unteren Bild wäre $x = 0$, $y = 0$ und $z = 1$, da die Erdbeschleunigung auf der z Achse der Wiimote liegt. Würde die Wiimote senkrecht nach oben zeigen ergäben sich folgende Werte:

$$x = 0, y = -1 \text{ und } z = 0,$$

Das heißt die Erdbeschleunigung wirkt auf -Y. Wenn die Wiimote im freien Fall wäre, so wären alle 3 Werte auf 0. Folgende Zeichnung zeigt die Beschleunigungsachsen der Wiimote:

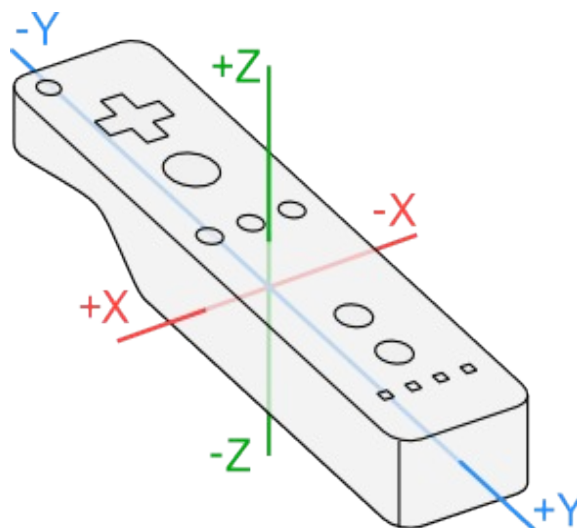


Abbildung 3: Beschleunigungsachsen der Wiimote

3.2 Berechnung von Pitch und Roll

Formel zur Pitch-Berechnung:

$$pitch = -\arcsin(y_1) \cdot \frac{180}{\pi}$$

Formel zur Roll-Berechnung:

$$roll = \text{atan2}(y_1, z_1) \cdot \frac{180}{\pi}$$

Erklärung zur Pitch-Berechnung

Die Werte x_1 , y_1 und z_1 sind hierbei nicht als die Beschleunigung zu verstehen sondern als Orientierungswerte. Zunächst werden die ursprünglichen Beschleunigungswerte x , y und z als Vektor aufgefasst. Es wird daraufhin die Länge dieses Vektor berechnet, um damit diesen Vektor zu normalisieren.

$$\text{Neuer normalisierter Vektor} = \begin{pmatrix} x/l \\ y/l \\ z/l \end{pmatrix}$$

Zusätzlich wird mit Hilfe von z_1 unterschieden in welchem Quadranten die Wiimote sich befindet. Dadurch lassen sich auch Bewegungen über 90° erkennen. Am Einheitskreis wird die Berechnung grafisch verdeutlicht:

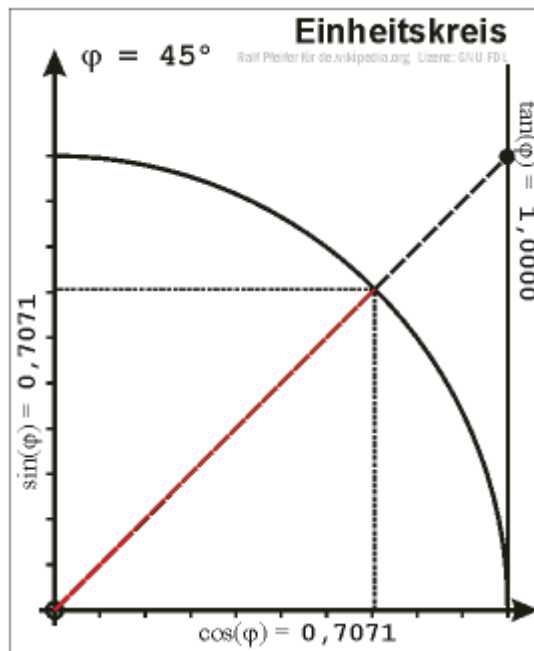


Abbildung 4: Der Winkel phi entspricht Pitch

Die trigonometrische Funktion für `arcsin()` und die Funktion `atan2()` (Arkustanges mit 2 Argumenten) aus der Math Bibliothek liefern ein Bogenmaßwert zurück. Deshalb muss diese noch in Grad umgewandelt und das Vorzeichen bei Pitch umgedreht werden, da -Y per Definition nach vorne zeigt.

Wichtig: Damit Roll richtig angezeigt wird, muss in der `wiimote.cpp` folgende Zeile auskommentiert werden: `roll = (x < 0)? -180 - roll : 180 - roll;`

4. WiiYourself library

Die WiiYourself library besteht aus einer Hauptkomponente und eine Reihe von Header-Dateien, die keine Dokumentation haben. Um die Logik zu verstehen, müssen Code-Kommentare zu Rate gezogen werden. Neben WiiYourself gibt es noch andere Bibliotheken wie die WiimoteLib (.Net Framework). Jedoch wurde WiiYourself gewählt, da die Bibliothek von Haus aus die Berechnung von Pitch und Roll mitbringt (siehe Punkt Berechnung von Pitch und Roll).

Die Hauptkomponente `wiimote` (bestehend aus `wiimote.cpp` und `wiimote.h`) enthält die Logik um sich mit der Wiimote zu verbinden und Daten von der Wiimote zu lesen. Die `wiimote_common.h` definiert Flags, die die Statusänderungen des Wiimote angeben. Es werden nur Status-Änderungen der Wiimote definiert, nicht die Buttonänderungen auf der Wiimote selbst. Zum Beispiel, wenn eine Wiimote mit WiiYourself verbunden worden ist oder wenn die Beschleunigung des Wiimote geändert wurde. Ebenso wenn ein Nunchuk oder andere Zusatzgeräte wie die Erweiterung Motion Plus angeschlossen wurde, und Tasten auf diesen gedrückt wurden. Diese Status-Angaben werden in der Enumeration `state_change_flags` definiert.

```
// flags & masks that indicate which part(s) of the wiimote state have changed
enum state_change_flags{
    ....
}
```

Die `wiimote_state.h` definiert Structs mit Variablen, die den Zustand der Wiimote speichern. Das bedeutet Variablen für Buttons, Beschleunigungsstatus oder Variablen, die den Zustand der LEDs auf der Wiimote angeben. Ferner werden Enumerations festgelegt, die zum Beispiel den Buttonstatus festlegen. Dies geschieht über ein Bit-Pattern, wobei jedes Bit eine Taste auf der Wiimote darstellt. Durch Abfrage des Bit-Patterns wird festgestellt, welche Tasten aktuell gedrückt werden. Die Abfrage welche Zusatzgeräte (z.B. Nunchuk) an die Wiimote angeschlossen sind funktioniert auch über Bit-Patterns.

```
// wiimote_state (contains the Wiimote and Extension data and settings)
struct wiimote_state {
    ....
}
```

5. NAOs Architektur

Im Grunde besteht die NAO Architektur aus einem Server-Modulsystem. Dabei wird der Server als NAOqi oder Mainbroker bezeichnet. Die Software im NAO-Roboter besteht aus einem kleinen Linux-System mit NAOqi (= dem Kern) und weiteren interne Module von Aldebaran (z.B Motion). Bei der Entwicklung für NAO werden also Module programmiert, die vom Mainbroker geladen werden. Es werden 2 Architekturen unterschieden, wie Module realisiert sein können.

1. sichere Architektur mit Remote-Module
2. unsichere Architektur mit dynamischen libraries

Dynamische libraries sind unsicher, weil sie direkt im NAO abgelegt werden. Das heißt NAOqi greift direkt auf das Modul bzw. die library zu. Stürzt jetzt NAOqi aufgrund eines Fehlers durch die library ab, kann der Roboter hinfallen und einen Schaden dadurch haben. Der Vorteil bei dynamischen libraries ist, dass sie schneller sind, als remote Module. Für die Projektarbeit wurde die sichere Architektur mit remote Module gewählt und diese soll im Folgenden näher erläutert werden.

5.1 Betriebsmöglichkeiten des Remote Moduls

Es gibt folgende zwei Möglichkeiten wie ein NAO Remote Modul betrieben werden kann.

1. Lokal: Das heißt ein lokaler NAOqi wird gestartet (auf ip 127.0.0.1). Über den Simulator Choregraphe, der mit dem lokalen NAOqi verbunden werden sollte, kann das Modul dann getestet werden
2. Live: Das Modul verbindet sich mit dem NAOqi auf dem Roboter. Der Computer auf dem sich das Modul befindet wird dabei an einem Router angeschlossen, der per WLAN mit dem NAO-Roboter verbunden ist

5.2 Remote Module

Remote-Module können vom Modulgenerator erstellt werden (siehe Kapitel "Modulerstellung"). Diese erstellt neben dem eigentlichen Modul / Module zusätzlich einen Server, auch Broker genannt. Der Sicherheitsaspekt von einem Remote Modul liegt darin, dass bei einem kritischen Fehler nur der eigene Broker abstürzt und nicht der Mainbroker auf dem NAO-Roboter. Um ein Modul aufrufen zu können, muss dieses Modul beim Broker registriert sein. Dies geschieht in der Methode:

```
ALCALL int _createModule( ALPtr<ALBroker> pBroker ){...}
```

Wenn ein Modul registriert ist, ist es in der Lage, Funktionen von anderen Remote Modulen aufzurufen. Außerdem kann es auf die internen Module von NAO mit Hilfe von Proxy-Klassen über remote calls zugreifen. Folgendes Bild verdeutlicht den Einsatz von Remote Modulen:

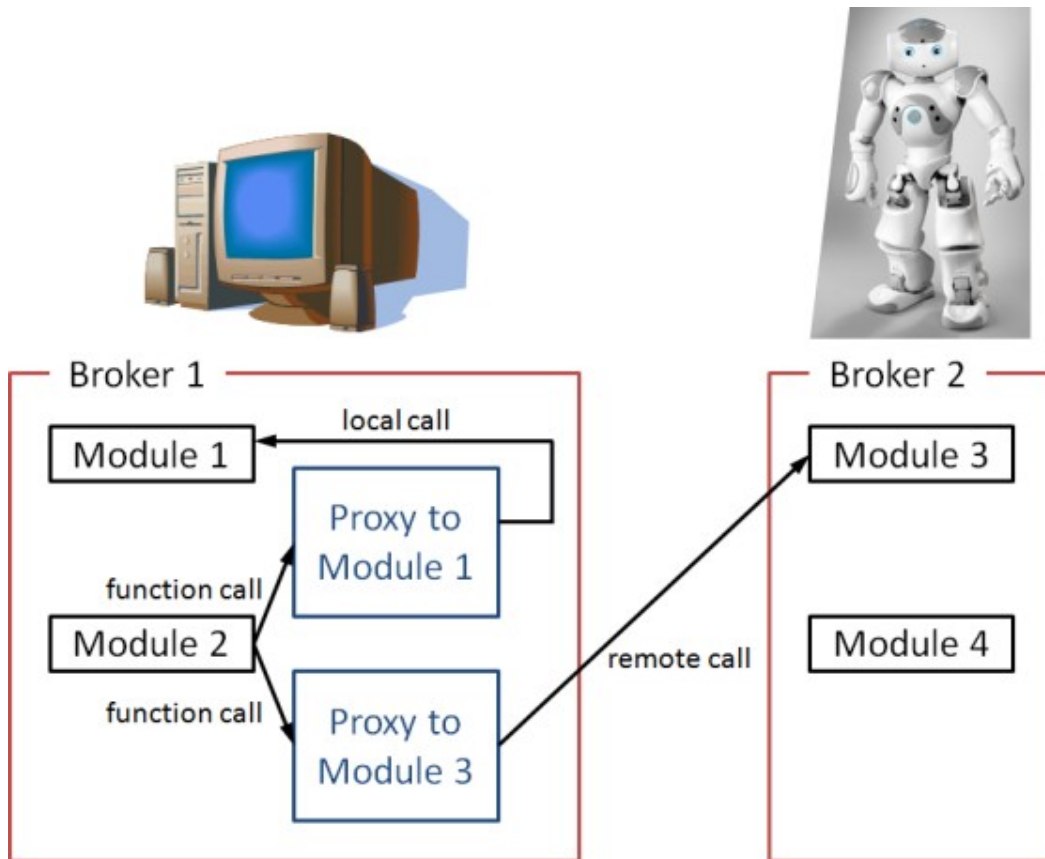


Abbildung 5: Remote Module Darstellung

Für viele interne Module gibt es Proxyklassen (z.B. `ALMotionProxy` für das Modul `ALMotion`). Interne Module, die keine äquivalente Proxyklasse haben können die allgemeine Proxyklasse `ALProxy` verwenden. Um zum Beispiel Funktionen vom `ALFrameManager` (dient dazu Choregraphie Projekte zu laden) aufzurufen, wird folgender Code gebraucht:

```
ALPtr<ALProxy> framemanager = getParentBroker()->getProxy( "ALFrameManager" );
string projectID = framemanager->call<string>
("newBehaviorFromFile",string("/srv/ftp/upload/standUp.xar"), string( "" ));
```


6. Inbetriebnahme des Moduls

6.1 NAO Start

Um den NAO zu starten muss der Knopf auf seiner Brust haltend gedrückt werden, bis NAOs Augen aufleuchten. Der NAO ist fertig hochgefahren, wenn seine Augen dauerhaft weiß leuchten. Nach dem Start hat der Roboter keine Festigkeit (Alle Stiffness Werte des NAO sind 0). Das heißt er würde sich nicht bewegen, wenn Bewegungsbefehle an ihn geschickt werden, da die Motoren aus sind. Der Roboter der Hochschule hat die IP: 192.168.2.100 . Alternativ kann auch der Simulator Choregraphe gestartet und mit NAOqi verbunden werden, wenn das Modul lokal betrieben sein soll.

6.2 Mit Wiimote verbinden und Modul starten


1. Bluetooth Software starten und nach neuen Geräten suchen
2. Die Tasten 1 und 2 auf der Wiimote gleichzeitig drücken (immer wieder kurz drücken bis das Modul connectet hat, um im Listening-Modus zu bleiben)
3. die Wiimote sollte in der Liste der zu suchenden Geräte auftauchen
4. Keinen Kopplungscode eingeben. Deshalb z.B unter Windows 7 Option 'Ohne Code koppeln' auswählen
5. naoarmmodule-local.exe (lokal mit Choregraphe und NAOqi) oder naoarmmodule-remote.bat(Remote auf NAO: ruft "naoarmmodule-remote.exe -pip 192.168.2.100" auf). Falls das Modul die Wiimote nicht findet, sollte diese neu gestartet werden. Falls die normale Executable nicht funktionieren sollten die Debug Executable benutzt werden. Diese gibt die Fehlermeldung aus und beendet sich nicht sofort

Problem: Es kann sein, dass die Wiimote nicht gefunden wird. Wenn der Fall auftritt, sollte das Gerät wieder aus der Bluetooth Liste entfernt werden und beim nächsten Connect-Versuch "automatische Treibersuche von Windows überspringen" unter Windows 7 gewählt werden.

7. Modulablauf

1. Wenn das Modul mit dem NAO-Roboter verbunden ist, initialisiert es auch den Aufstehvorgang des NAO
2. Modul wird geladen und die Arme des Naos werden in die Ausgangstellung gebracht, nachdem eine Verbindung zum NAOqi aufgebaut wurde
3. WiimoteListener wird gestartet und wartet auf die Verbindung einer Wiimote
4. Wenn eine Wiimote identifiziert wurde, läuft der Polling-Vorgang des WiimoteListeners los, um Wiimote Daten zu lesen. Wenn ein Update der Armkomponenten erfolgen soll, wird eine Funktion aus dem Modul NaoArmControl aufgerufen.
5. Der Polling Vorgang endet und das Modul wird sauber beendet, wenn die Home Taste gedrückt wird

Wie bringe ich den NAO in seine liegende Ausgangsstellung (eingeschaltet)?

Unter Choregraphie gibt es den Button  um alle Motoren des NAOs auszuschalten. Alternativ funktioniert auch ein Aufruf der Funktion `setBodyStiffness(0)`, die das Gleiche bewirkt. Ganz wichtig ist aber, dass den NAO vorher hingelegt wird, da er sonst umfällt.

7.1 NAO Armsteuerung Bedienungsanleitung

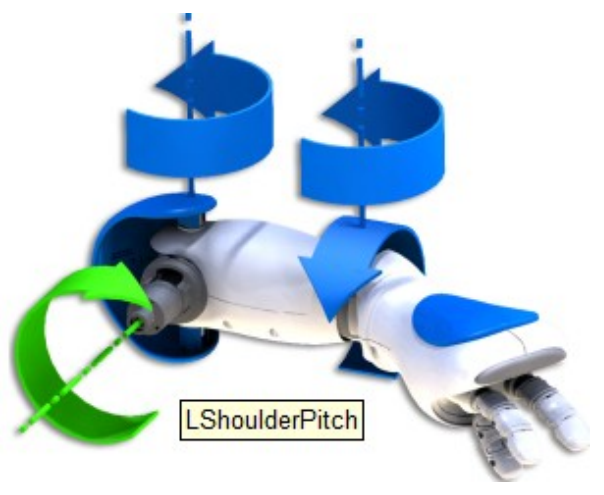
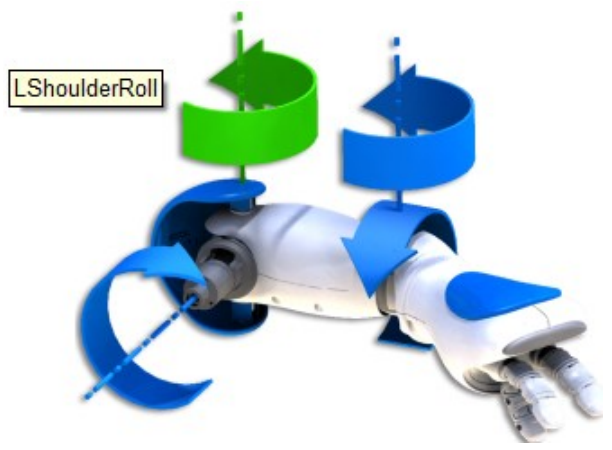
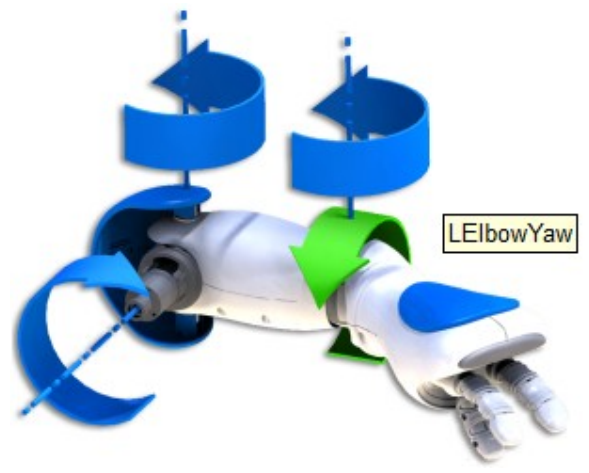
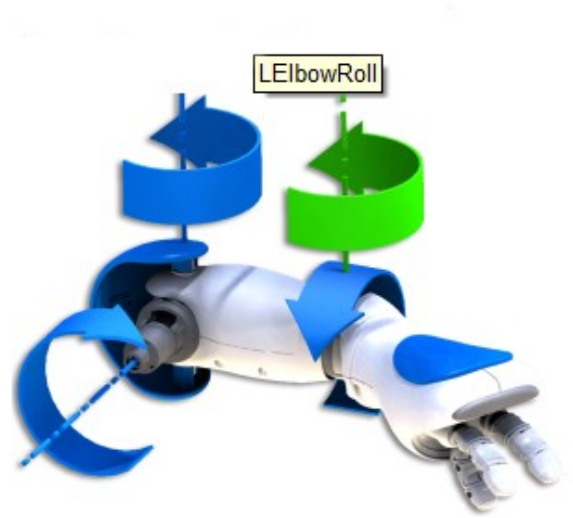
In der folgenden Tabelle wird die Bedienung der Arme gezeigt. Was hinter den Armparameter stecken, zeigen die unteren Bilder (L steht für left) im Punkt Nao Armkomponenten:

Bedienung	Taste
ShoulderPitch	Auf- und Abwärtsbewegung der Wiimote
ShoulderRoll	Neigen des Nunchuk nach links / rechts
ElbowRoll	Auf- und Abwärtsbewegung der Wiimote
ElbowYaw	Left / Right Taste (Steuerkreuz)
WristYaw	Links- oder Rechtsdrehung der Wiimote
Wechsel auf rechten Arm	2
Wechsel auf linken Arm	1
Aktivieren der Bewegungserkennung bei ShoulderPitch / ElbowRoll, ShoulderRoll	A gedrückt halten
Aktivieren der Bewegungserkennung bei WristYaw	B gedrückt halten
Wechseln zwischen ShoulderPitch / ShoulderRoll	Pfeil-Unten (Standard Oberarm)
Hände aufmachen	Nunchuk-Joystick vor bewegen
Hände zumachen	Nunchuk-Joystick nach hinten bewegen
Verbindung trennen	Home-Taste

7.2 NAO Armkomponenten

Die Ansteuerung der NAO Arme funktioniert über die Wiimote. Damit werden folgende 6 Komponenten der Arme bedient, die im Folgenden grafisch veranschaulicht werden:

- Schulterkomponente hoch/runter
- Schulterkomponente vor/zurück
- Armkomponente gestreckt/gewinkelt
- Unterarmdrehung
- Handdrehung
- Hände öffnen/schließen



8. Modul Architektur

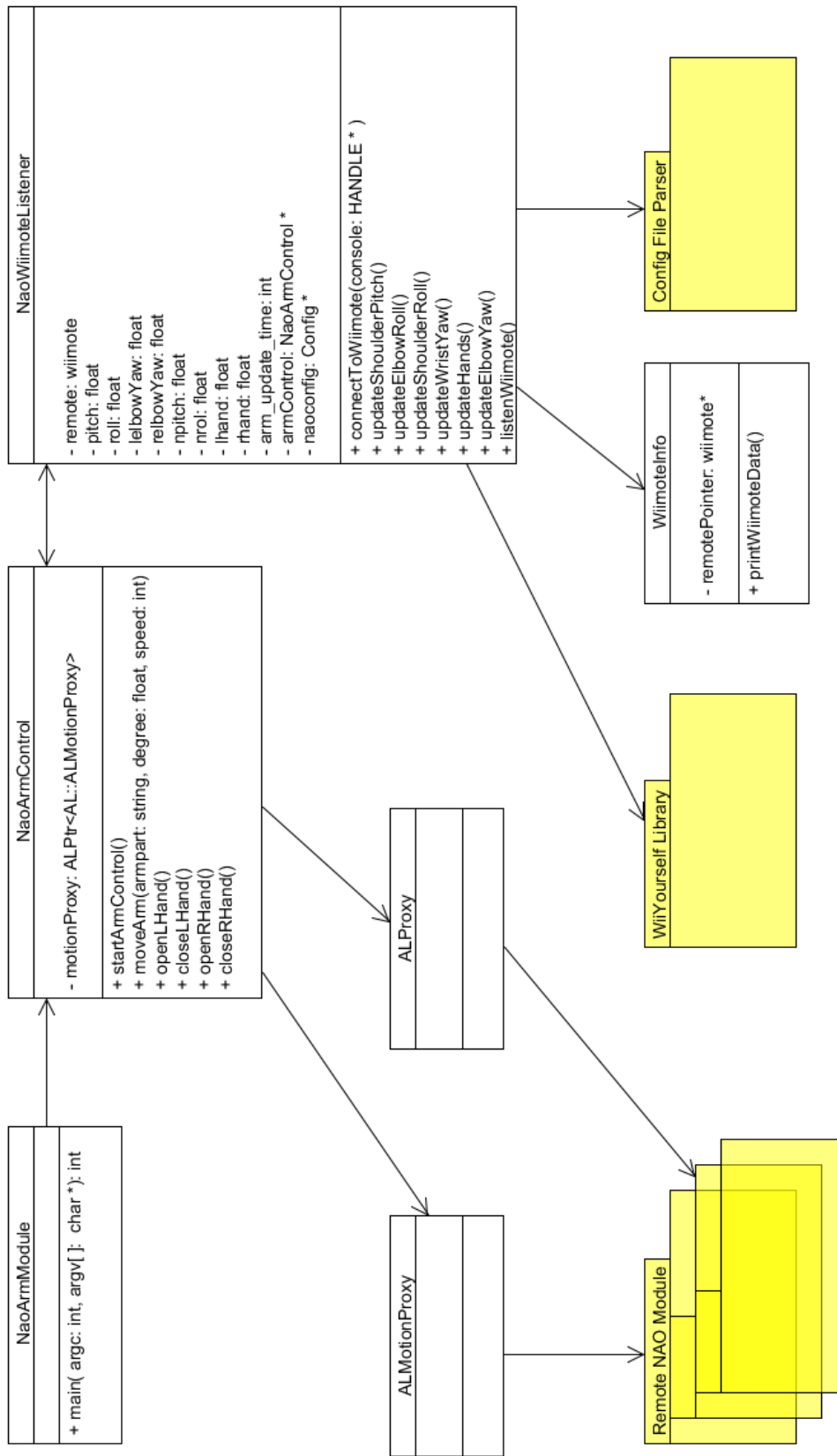


Abbildung 6: Modul Klassendiagramm

8.1 Aufgaben der Klassen

1. NaoArmModule

Dies ist die Startklasse. Hier wird der Broker gestartet und das Modul registriert. Außerdem ist hier die Logik zum sauberen Beenden des Moduls integriert

2. NaoArmControl

Diese Klasse ist das eigentliche Modul. Sie benutzt Proxyklassen um auf die Remote Module zugreifen zu können. Es wird auf das Modul ALMotion zugegriffen, um die Arme zu steuern und auf das Modul ALFrameManager, um Choreographie Projekte aufrufen zu können (für Aufstehvorgang)

Der Rumpf der oberen 2 Klassen wurde vom Modulgenerator erstellt.

3. NaoWiimoteListener

Diese Listenerklasse stellt die Verbindung mit der Wiimote her und greift mit Polling auf die Wiimotedaten zu. Armwechsel und Handzustände (auf und zu) werden in einem Callback behandelt. Schließlich greift die Klasse mit den update-Funktionen auf Methoden der NaoArmControl Klasse zu, um Armkomponenten zu steuern. Werte für Bewegungsgeschwindigkeit und update-Zeit für ShoulderPitch, ShoulderRoll und ElbowRoll werden mit Hilfe des Config File Parsers aus einer Config-Datei gelesen und an NaoArmControl übergeben. Die Logik dieser Klasse ist zum Teil aus der Demo Klasse von WiiYourself entnommen. Der verwendete Config File Parser ist ein externes C++-Modul

4. WiimoteInfo

Die WiimoteInfo Klasse gibt Daten der Wiimote formatiert auf der Konsole aus. Bis auf ein paar Positionsänderungen, ist der Ausgabecode von der Demo Klasse von WiiYourself entnommen worden

8.2 Steuerungsfunktion

Eines der Schwierigkeiten bei der Programmierung war die passende Wahl der Funktion für die Armsteuerung. Es gab drei Funktionen für eine Aufgabe, die mit unterschiedlichen Einstellungen getestet wurden. Funktionsnamen und Ergebnisse werden unten aufgeführt:

void setAngle (string pJointName, float pAngle)

Ergebnis: Bewegungen sind zu schnell und werden zu stark ausgeführt. Funktion ist ungeeignet

void gotoAngle (string pJointName, float pAngle, float pDuration, int pInterpolationType)

Ergebnis: pDuration gibt die Dauer in Millisekunden an, in der die Bewegung ausgeführt werden soll. Die Steuerung einer einzigen Armkomponente mit dieser Funktion funktioniert gut. Allerdings werden Bewegungen bei vielen Arm update-Aufrufen sehr verzögert ausgeführt. Bei kleine pDuration ist das Problem genau dasselbe wie bei setAngle()

void gotoAngleWithSpeed (string pJointName, float pAngle, int pSpeedPercent, int pInterpolationType)

Ergebnis: Es hat sich gezeigt, dass diese Funktion am besten geeignet ist. Die Variable pSpeedPercent hat den Wertebereich (1-100). Für jede Armkomponente ist es möglich, die Bewegungsgeschwindigkeit in der armControl.conf anzupassen. Durch Tests wurde festgestellt, dass 20% subjektiv am Geeignetsten ist.

9. Armspezifikation

Armkomponente / Parameter für <i>pJointName</i>	Streckbereich in Grad	Streckbereich in Bogenmaß
RShoulderPitch	Von -119.5 bis 119.5	Von -2.0857 bis 2.0857
RShoulderRoll	Von -76 bis 18	Von -1.3265 bis 0.3142
RElbowYaw	Von -119.5 bis 119.5	Von -2.0857 bis 2.0857
RElbowRoll	Von 2 bis 88.5	Von 0.0349 bis 1.5446
RWristYaw	Von -104.5 bis 104.5	Von -1.8238 bis 1.8238
RHand	Auf- und Zumachen	Auf- und Zumachen

Armkomponente	Streckbereich in Grad	Streckbereich in Bogenmaß
LShoulderPitch	Von -119.5 bis 119.5	Von -2.0857 bis 2.0857
LShoulderRoll	Von -18 bis 76	Von 0.3142 bis 1.3265
LElbowYaw	Von -119.5 bis 119.5	Von -2.0857 bis 2.0857
LElbowRoll	Von -88.5 bis -2	Von -1.5446 bis -0.0349
LWristYaw	Von -104.5 bis 104.5	Von -1.8238 bis 1.8238
LHand	Auf- und Zumachen	Auf- und Zumachen

10. Probleme bei der Umsetzung

10.1 Start

Die Schwierigkeit bei der Durchführung der Projektarbeit bestand zunächst darin eine geeignete Wiimote Bibliothek zu finden. Ein guter Überblick liefert:

<http://wiibrew.org/wiki/Wiimote/Library>

WiiYourself wurde gewählt, weil die Bibliothek bereits Pitch und Roll liefert. Die Kompilierung der Demo-Dateien konnte aber nicht durchgeführt werden und Visual C++ brachte keine klare Fehlermeldung. Erst durch eine Vertiefung in das Problem wurde klar das WinDDk und das Microsoft Windows SDK benötigt wird, außerdem war es an dieser Stelle auch wichtig in welche Reihenfolge diese SDKs eingebunden werden müssen.

Visual C++ brachte keine klare Fehlermeldung. Es hat eine Weile gebraucht bis erkannt wurde, dass zusätzlich WinDDK und Microsoft Windows SDK benötigt wird und wie diese in welcher Reihenfolge einzubinden sind (siehe Tutorial zur Modulerstellung).

10.2 WiiYourself und NAO SDK

Nicht ganz so einfach war es auch, die WiiYourself Demo und Bibliothek ohne Dokumentation und nur anhand der Kommentare zu verstehen. Vor allem die Berechnung von Pitch und Roll sowie die Reaktionen des Beschleunigungssensors waren nicht einfach zu verstehen. Deshalb wurde auch zusätzlich ein Tool entwickelt, um die Wiimote Beschleunigungsdaten festzuhalten und auszugeben (siehe Punkt Wiimote Auswertungs-Tool)

Für NAOqi gab es eine Dokumentation. Diese war jedoch unübersichtlich und mangelhaft. Zum Beispiel, war es sehr schwer, herauszufinden, wie das selbstgeschriebene Modul mit Broker sauber beendet werden kann. Zu diesem Thema liefert die Dokumentation wenig Hilfestellung und auch die Modulbeispiele konnten nicht weiterhelfen. Um das Programm sauber zu beenden, muss das Modul erst vom lokalen Broker gelöst werden, dann den lokalen Broker vom NAOqi lösen und zuletzt muss die `removeBroker()` Funktion durch den Broker-Manager aufgerufen werden. Diese Lösung entstand in Eigenrecherche.

Die Unübersichtlichkeit zeigt sich unter anderem dadurch, dass es keine Klassen API gibt wie sie beispielsweise in Java existiert. Alle Module sind im Bereich "API Modules" beschrieben. Es mangelt aber zum einen an Beispielen und zum anderen sind wichtige Klassen wie `ALBroker` und `ALModule` ganz woanders zu finden. Auch stehen max. und min. Werte für Armkomponenten weder bei den Funktionen selber noch wie vermutet unter dem Link "Motion" in der Red Documentation, sondern unter Hardware->Kinematics. Eine Verlinkung oder eine Suche (ist in späteren SDK Version eingebaut) wäre hilfreich gewesen.

10.3 Steuerung

Ebenfalls war es schwierig eine intuitive Bedienung zu ermöglichen. Der ursprüngliche Wunsch mit der Wiimote mehrere Freiheitsgrade gleichzeitig zu steuern funktioniert theoretisch. In Wirklichkeit ist diese Lösung aber nicht intuitiv und der User ist eher verwirrt beim Nachvollziehen der Bewegung. Alle Freiheitsgrade zu steuern wäre technisch mit der Wiimote schwierig zu realisieren, da sie wenige Parameter zur Positionserkennung hat und empfindlich ist. Deshalb besteht die Armsteuerung aus einem Kompromiss von Wiimote-Positionserkennung und Buttonsteuerung.

10.4 Coding

Da die Funktion Stand-up funktional nicht in der SDK implementiert ist, wurde versucht Choregraphie Projekte im Modul direkt aufzurufen.

Für die Interpretation von Choregraphie Projekten ist das Modul ALFrameManager verantwortlich. Wichtig ist es vor dem Aufruf der Choregraphie Projekten die Motoren des NAOs zu starten (`setBodyStiffness(1)`), da er sonst nicht reagiert. Ein Choregraphie Projekt wird nicht wie erwartet lokal geladen und abgespielt, sondern muss auf dem Roboter abgelegt werden. Dies geschieht über Choregraphie selbst (Option "File Transfer"). Wenn das Modul lokal ausgeführt wird, ist das nicht nötig.

Normalerweise werden bei einem File Transfer die Dateien in `/home/nao` abgelegt (laut Forumlink: http://academics.aldebaran-robotics.com/index.php?option=com_kunena&Itemid=14&func=view&catid=67&id=3164#3168).

Bei dem NAO-Roboter der HS liegen die Uploads allerdings in `/srv/ftp/upload/`. Erst mit Hilfe des Moduls ALFileManager konnte der richtige Pfad gefunden werden.

Es gab auch Bugprobleme mit WiiYourself (siehe **Wichtig** bei Punkt "Berechnung von Pitch und Roll") und den Config File Parser:

Parameter-Bug: Diese Zeile in `config.cpp`

```
while (*envp) {
```

durch diese ersetzen

```
while (envp && *envp) {
```

Außerdem wurde die `config.cpp` modifiziert, dass sie Fehlermeldungen in einer `MessageBox` ausgibt.

11. Verbesserungsmöglichkeiten

Wie oben im Punkt "10.3 Steuerung" erwähnt ist es schwierig die Arme nur mit Hilfe der Wiimote zu steuern. Deswegen gibt es folgende Verbesserungsmöglichkeiten:

Zum einen könnte die Abtastung der Wiimote durch den Zusatz Motion PLUS gesteigert werden, zum anderen könnte man zur Lageberechnung der Wiimote zusätzlich die IR-

Kamera einsetzen, die bis zu 4 Infrarotquellen erkennt. Allerdings wird dafür zusätzliche Software wie die Wii Sensor Bar oder Vergleichbares gebraucht. Damit lassen sich theoretisch auch Bewegungen nach links und rechts in einem bestimmten Rahmen erkennen.

Eine andere Alternative wäre, anstelle der Wiimote eine ganz andere Technik einzusetzen. Der Einsatz des Playstation Move-Motion-Controller oder des Kinect Systems von Microsoft wären gute Lösungen. Die Hardware gibt es allerdings noch nicht sehr lange und Libraries um die Hardware anzusprechen sind noch in der Entwicklung oder eventuell nicht ausgereift. Besonders Kinect ist für Ansteuerung der Arme geeignet, da die Kinect die Bewegungen mit Hilfe von 2 Infrarot-Tiefensensoren erfasst. Damit können theoretisch alle Freiheiten der Arme erfasst werden. Es gibt aber auch Ansätze, die Wiimote und Kinect kombinieren. Das Ergebnis gibt es unter folgendem Link:

<http://www.youtube.com/watch?v=TmTW61MLm68>

12. Tutorial zur Modulerstellung / Modulmodifikation

Dieser Abschnitt beschreibt die Modulerstellung für NAOqi SDK vers. 1.3.17

1. Zuerst sollte der ModuleGenerator.exe im Aldebaran SDK Verzeichnis gestartet werden

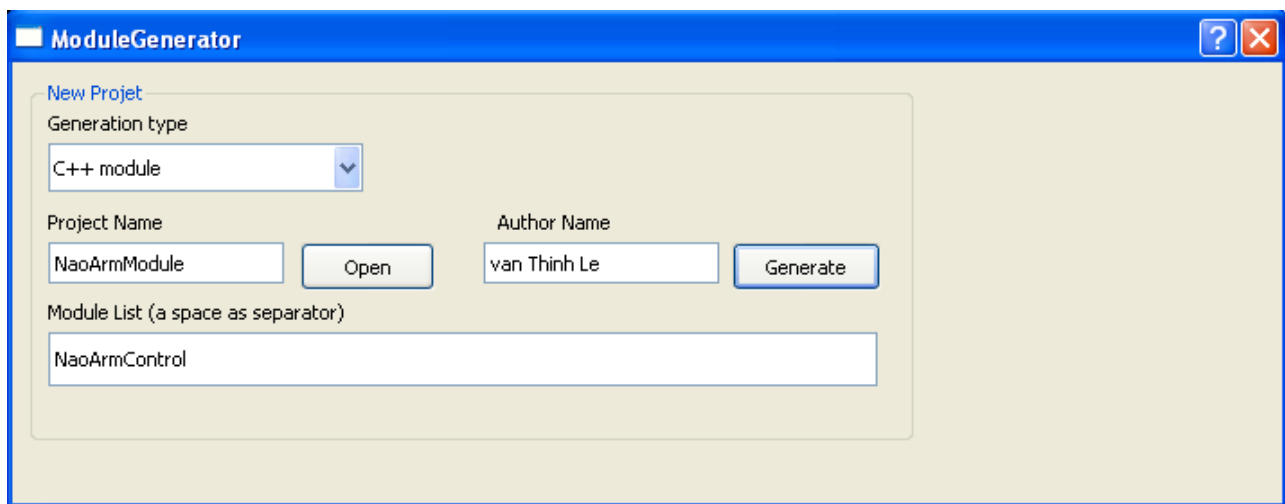


Abbildung 7: Modulgenerator Screenshot bei NaoArmModule Erstellung

2. Einstellung: Generation Type "C++ module"
Es werden 2 C++ Module (cpp-Datei + Header-Datei) erstellt. Der Projektname, der angegeben wird, ist zugleich der Name für das Startmodul mit der main()-Funktion. Dieses Modul sollte unverändert bleiben. Im Feld Module List werden die eigentlichen Module angegeben, die generiert werden und mit NAOqi kommunizieren sollen. Die Namen der Module und des Projektnamens dürfen nicht übereinstimmen
3. Auf Generate klicken. Es sollte eine Erfolgsmeldung in Form eines Dialogs kommen mit dem Hinweis, dass die Quellen generiert wurden

- Um aus den generierten Quellen ein Visual Studio Projekt zu machen wird Cmake gebraucht. Source Code Verzeichnis ist das Verzeichnis das vom Modulgenerator erzeugt wurde. Das Verzeichnis für die Binaries sollte das gleiche Verzeichnis sein

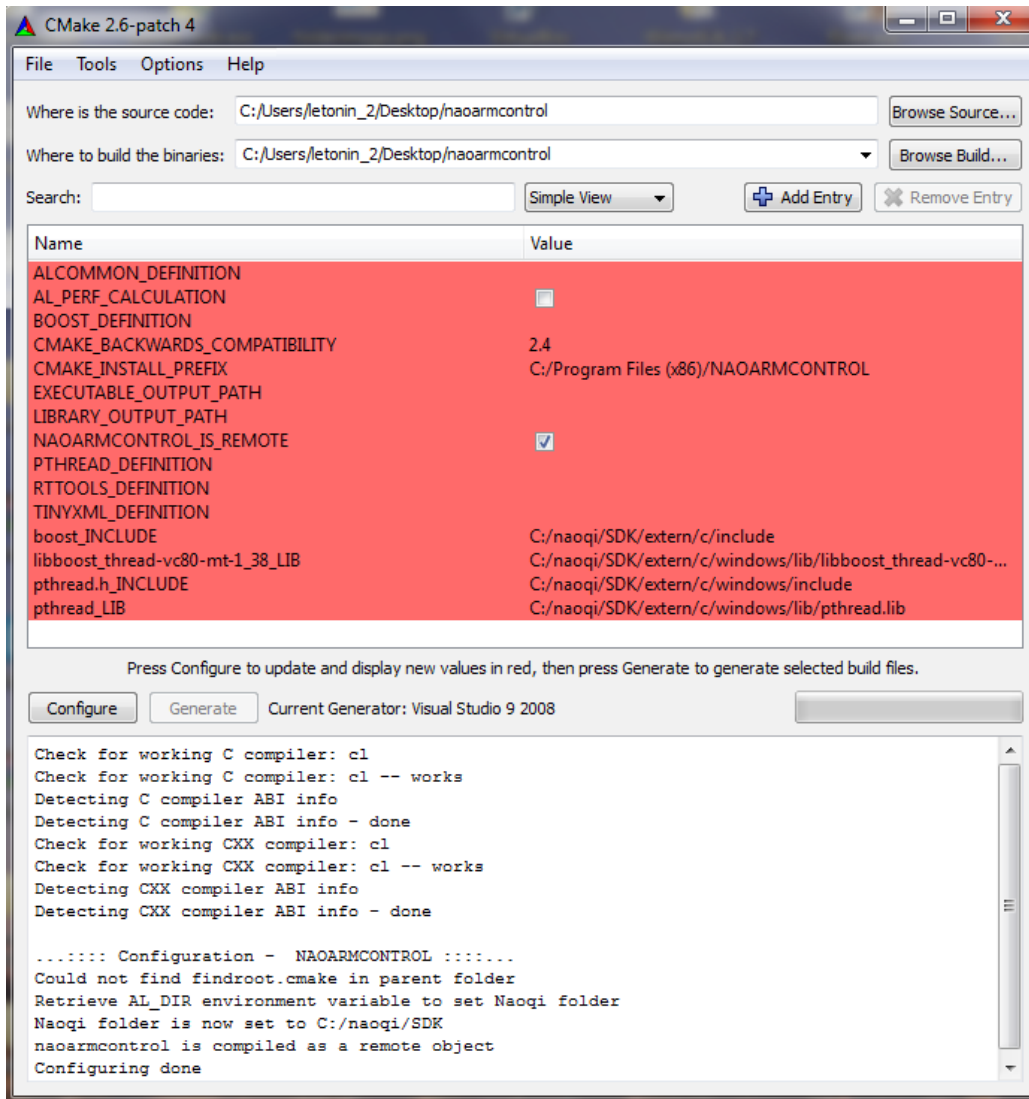


Abbildung 8: Cmake Zustand nach erstem Configure Vorgang

- Welche Optionen beim Configure Vorgang gewählt werden sollen ist NAO SDK abhängig. Daher sollte die Sektion "Programming" in der Red Documentation zu Rate gezogen werden. Nach dem ersten Configure Vorgang sieht CMake aus wie in Abb. 8. "Configure" sollte wieder geklickt werden und die roten Punkte sollten dann weiß werden
- Die Projekt Sourcen werden generiert, wenn auf "Generate" gedrückt wird. Es ist jetzt möglich das Projekt mit Visual Studio zu öffnen. Wenn das Projekt gestartet werden soll wird bei NAOqi 1.3 noch zusätzlich die Bibliothek pthreadVCE2.dll benötigt. Diese muss heruntergeladen und in das src/Debug Verzeichnis kopiert werden.

- 7. Zusätzlich muss die MS Windows SDK und die WinDDK in Visual Studio angegeben werden, wenn ein Modul bearbeitet werden soll (siehe Abbildung 9 und 10)

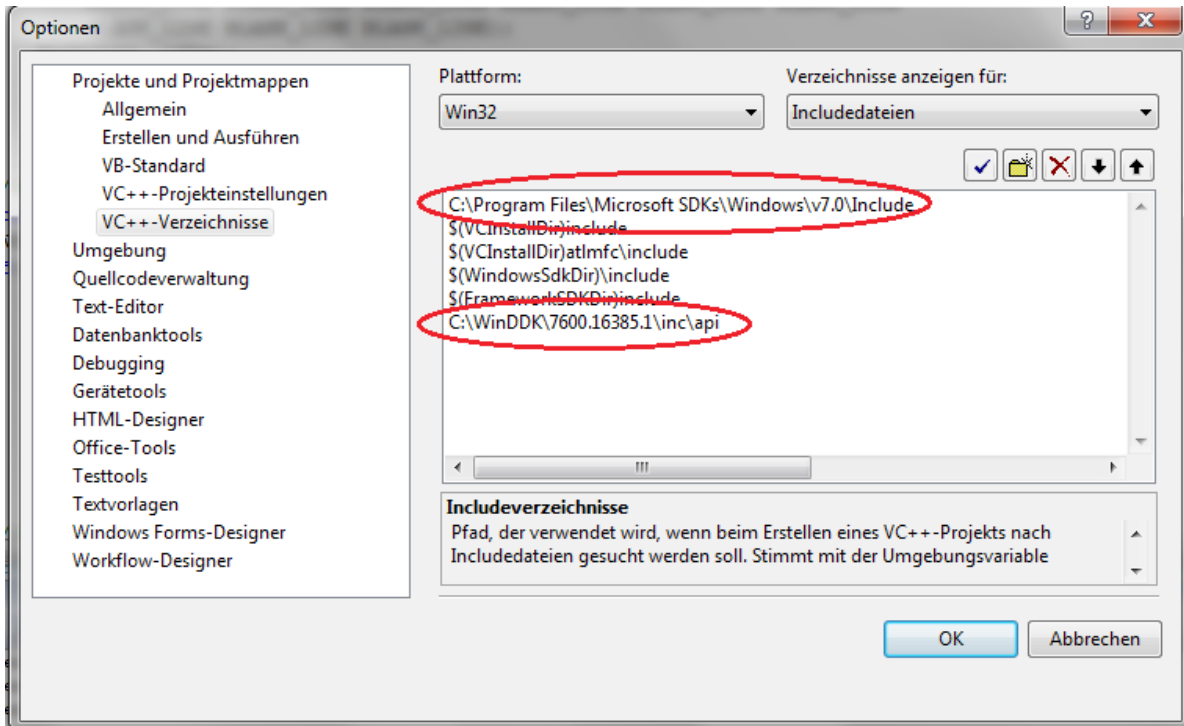


Abbildung 9: zusätzlich benötigte Include Dateien

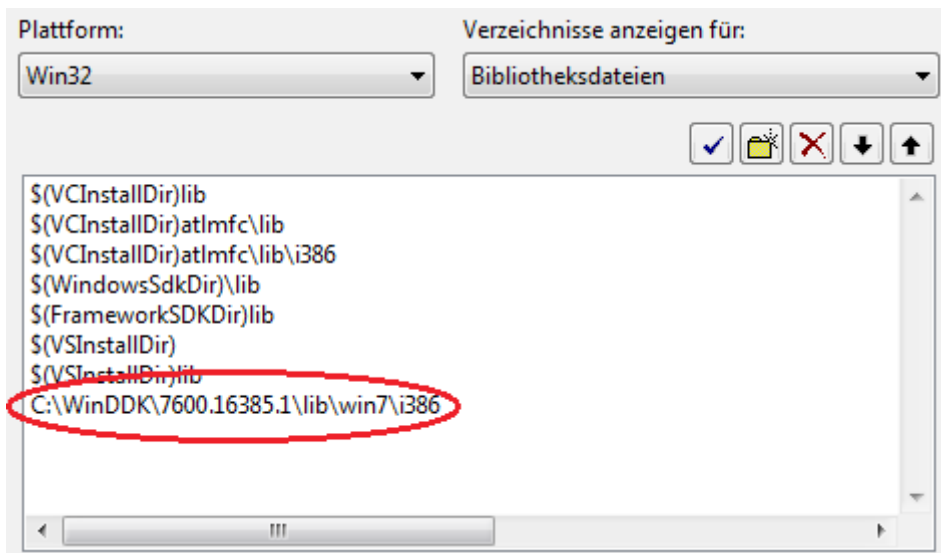


Abbildung 10: Bibliotheken von WinDDK müssen eingebunden werden

- 8. Nach dem das Modul gestartet wurde, sollte im NAOqi eine Info-Meldung kommen, dass ein neuer Broker registriert wurde. Normalerweise sucht der Broker NAOqi auf 127.0.0.1 . Um stattdessen eine Verbindung zum NAO herzustellen, muss beim Start noch " --pip Roboter-IP-Adresse " mitgegeben werden

13. Wiimote Auswertungs-Tool

Das Wiimote Auswertungs-Tool ist eine von mir modifizierte Demo Version von WiiYourself. Es dient dazu, Beschleunigungswerte in einem bestimmten Intervall festzuhalten und auszugeben. Hinzu werden Max. und Min. Werte für X, Y und Z in dem Intervall ausgegeben. Bei jedem Schleifendurchlauf (Polling) werden die Beschleunigungswerte in einer verketteten Liste so lange gespeichert, bis auf den Button A für die Ausgabe gedrückt wird. Nach der Ausgabe fängt die Liste wieder von Anfang an zu sammeln.

Das Tool hat den Zweck, Beschleunigungsänderungen nachzuvollziehen. Für eine Messung sollte am Anfang A gedrückt werden, um die Liste zu leeren. Dann kann die zu messende Bewegung ausgeführt werden. Danach wieder auf A drücken, um das Ergebnis zu erhalten.

14. Fazit

Die Programmierung des NAO Roboters macht sehr viel Spaß. Allerdings war der Weg bis dahin ziemlich lang und schwierig. Es gab sehr oft Phasen, in denen ich lange Zeit nicht weiterkam (siehe Projektschwierigkeiten). Im Gegensatz zu vielen anderen Projektarbeiten, in der die Technik vorgegeben war und das Ziel klar definiert war, ist das bei dieser Projektarbeit nicht so. Die Aufgabe war "Mache eine Software mit dem man mit der Wiimote die Arme des NAO steuern kann". Dann stand ich vor den Problemen: Wie soll eine intuitive Steuerung aussehen? In welche Programmiersprache mach ich das am besten? Mache ich ein Remote Modul oder eine dynamische Library? Welche Wiimote Library soll ich nehmen? Welche Armsteuerungsfunktion beim ALMotion Modul soll ich nehmen? Welche Geschwindigkeitsparameter sind am geeignetsten? In welche SDK Version soll ich programmieren? Etc.

Es waren sehr viel Eigenrecherche und viele Versuche erforderlich, bis ich auf eine angemessene Lösung gekommen bin. Am meisten Spaß hat dabei die Versuchphase der Armsteuerungsfunktionen gemacht. Dabei vergisst man mal schnell die Zeit und sitzt auch gern bis Mitternacht in der Hochschule.

15. Quellen

15.1 Bilder

<http://www.robotshop.com/Images/xbig/fr/robot-humanoide-nao-edition-academique-v3plus-aldebaran.jpg> (Zugriff 25.01.2011)

http://images4.wikia.nocookie.net/_cb20100909212157/cmrobobits2010/images/thumb/6/66/Broker_Proxy.png/559px-Broker_Proxy.png (Zugriff 18.01.2011)

http://wiibrew.org/wiki/File:Wiimote_axis2.png (Zugriff 26.02.2011)

http://www.techfresh.net/wp-content/uploads/2006/12/wiimote_comparison.jpg
(Zugriff 05.03.2011)

Nao Armkomponenten Bilder sind Screenshots aus Choregraphie

15.2 Literatur

http://de.wikipedia.org/wiki/Human_Interface_Device (Zugriff 24.12.2010)

<http://wiibrew.org/wiki/Wiimote#Accelerometer> (Zugriff 10.01.2011)

<http://wiibrew.org/wiki/Wiimote/Library> (Zugriff 10.11.2010)

Config File Parser: <http://www.codeproject.com/KB/files/config-file-parser.aspx>
(Zugriff 22.02.2011)

NAO Kinect+Wiimote Video: <http://www.youtube.com/watch?v=TmTW61MLm68>

Software Downloads, NAO SDK Dokumentation und Forum:

<http://academics.aldebaran-robotics.com> (Zugriff 01.11.2010 - 28.02.2011)

WiiYourself Library: <http://wiiyourself.gl.tter.org/> (Zugriff 10.11.2010)