# Dokumentation – NXT

Sven Schröder

20. September 2012

## Inhaltsverzeichnis



# 1 Entwurf

## 1.1 Software

Beim Entwurf der Software für unseren Teil der Aufgabe hatten wir zwei Dinge zu beachten, die mäßige Rechenleistung[1] des LEGO© Mindstorms© NXT Brick (im folgenden nur noch Brick genannt) und die durch LEGO© begrenzte Anzahl von Robotern auf maximal 4.

[1] 8-Bit ARM mit 48 MHz Takt, 64KB RAM

### 1.1.1 nxc – Not eXactly C

nxc ist die für LEGO© NXT native Programmiersprache. Obwohl die Syntax von nxc der Programmiersprache C ähnlich ist, ist sie in ihrem Umfang erheblich eingeschränkt. Aus dem Fehlen des Pointerkonzept resultiert unter anderem der Verlust auf die Speicherverwaltung direkt einwirken zu können.

### 1.1.2 nxt-python-framework

Das nxt-python-framework[2] agiert als Interface, welches die von LEGO© in dem „Bluetooth Development Kit“ veröffentlichten direkten Kommandos, nutzt um mit der Hardware zu interagieren.

Wie wird dies erreicht?
Mit LEGO© NXT ist es möglich, dass sich bei maximal vier NXTs einer zum Master erklärt. Der Master kann nun durch speziell kodierte Befehle die anderen drei NXTs fernsteuern. Dieses Verhalt macht sich das nxt-python-framework zu nutze und täuscht maximal 3 NXTs vor, dass es ein NXT-Master sei. Wenn dies geschehen ist können die NXTs von PC-Seite ferngesteuert werden.

Vorteil: Durch die Nutzung von nxt-python integriert sich die Komponente NXT-Erkunder nahtlos in das übrige System.

Nachteil: Der synchrone Start bzw. Stopp von zwei Motoren ist nur schwerlich realisierbar, da zwei Befehle benötigt würden, die nacheinander verschickt und auf NXT-Seite nacheinander ausgewertet werden würden.

Wegen dem immensen Vorteil der einfachen Integration in das Restsystem und dem Nachteil der asynchronen Ansteuerung von Motoren entstand die Idee die beiden Programmiersprachen (nxc und python) zu kombinieren.

### 1.1.3 hybrider Ansatz

Die Kombination von nxc und nxt-python wurde wie folgt realisiert. Es wurde in einfaches Kommunikationsprotokoll (siehe Abbildung 2 auf Seite 4) konzipiert durch welches der Aufruf von in nxc implementierten Funktionen durch nxt-python ermöglicht wird.

## 1.2 Idee 1 → Modell 1

### 1.2.1 Idee

Unsere erste Idee bestand im Prinzip aus zwei unabhängigen Ideen. Zum Einen wollten wir ein Fahrgestell konzipieren, das auch bei unwegsamen Gelände eine kontrollierte Bewegung des Explorer ermöglichen würde und zum Anderen wollten wir einen Sensor der schon viele Informationen über die Umgebung sammelt ohne, dass der Explorer jeden Quadratzentimeter abfahren muss.

[2] http://code.google.com/p/nxt-python/

#### 1.2.2 Konstruktion

#### 1.2.3 Test

#### 1.2.4 Pros & Cons

#### 1.2.5 Fazit

### 1.3 Idee 2 → Modell 2

#### 1.3.1 Idee

#### 1.3.2 Konstruktion

#### 1.3.3 Test

#### 1.3.4 Pros & Cons

#### 1.3.5 Fazit

### 1.4 Idee 3 → Modell 3

#### 1.4.1 Idee

#### 1.4.2 Konstruktion

#### 1.4.3 Test

#### 1.4.4 Pros & Cons

#### 1.4.5 Fazit

### 1.5 Fazit und Entscheidung

## 2 Kommunikation

### 2.1 Bluetooth

### 2.2 Kommunikationsprotokoll PC ↔ NXT

anfänglich 3-way-handshake wegen missverständnis

Kommunikation PC → NXT:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| A | ; | ID | ; | F | , | W |

Kommunikation PC ← NXT:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| A | ; | ID | ; | T | , | W | ,2 | $W_2$ |

Legende:
A = Typ der Nachricht (m = Nachricht, r = m erhalten, a = r erhalten)
F = Funktion die aufgerufen werden soll
W = Zahlwert
T = Typ der Antwort
$_2$ = optional

Abbildung 1: Kommunikationsprotokoll 3-way-handshake

### 2.3 Kommunikation mit dem MCC

## 3 Logik

Problem das der Robo blockiert, das Program aber nicht

Kommunikation PC → NXT:

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| ID | ; | F | , | W |

Kommunikation PC ← NXT:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| ID | ; | T | , | W | ,2 | $W_2$ |

Legende:
F = Funktion die aufgerufen werden soll
W = Zahlwert
T = Typ der Antwort
$_2$ = optional

Abbildung 2: Kommunikationsprotokoll

## 3.1 Explorationsalgorithmen

In der Welt der autonomen Rasenmäh- und Staubsaugroboter haben sich vier Algorithmen durchgesetzt:

1. Touch and Go[3]
2. circle
3. radar
4. Wandverfolgung

1 – 3 werden im Folgenden näher besprochen. 4 konnte aufgrund der begrenzten Anzahl an Sensoren nicht implementiert werden und bleibt deshalb außen vor.

### 3.1.1 Exploration – simple

### 3.1.2 Exploration – circle

### 3.1.3 Exploration – radar

## 3.2 GoToPoint

[3]hier simple genannt