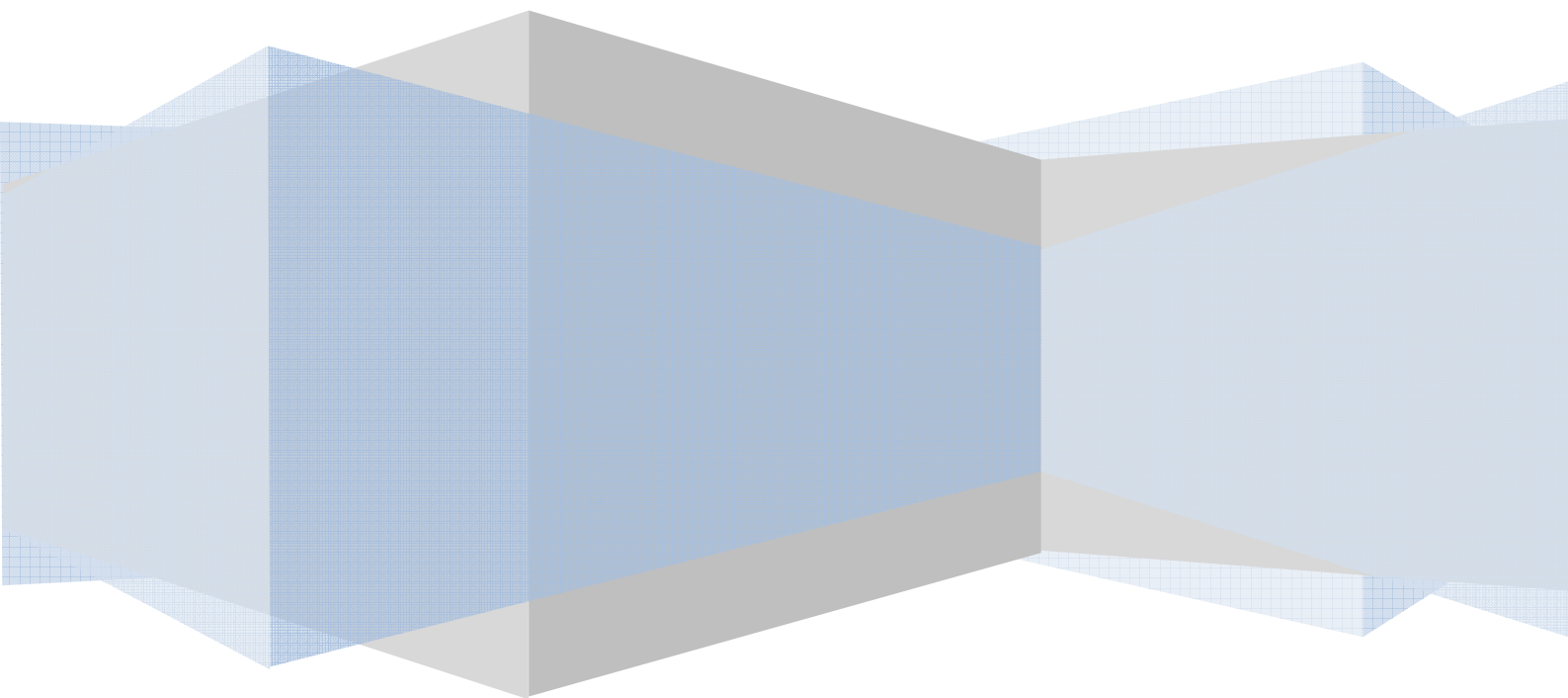


VS-Möbel

Unterrichtspraxis mit dem Lego NXT-Baustein

Programmieren mit NXC 2.Auflage

Frank Engeln



Unterrichten mit dem NXT und der Programmiersprache NXC. Warum NXC? NXC-Dateien kann man in jedem Editor erstellen. „Not eXactly C“ (NXC) ist eine Sprache zur Programmierung des Lego Mindstorms NXT in C-ähnlicher Syntax. Besonders bei der Lösung komplexer Probleme bietet die Programmierung des Lego-Roboters in NXC gegenüber der Benutzung der mitgelieferten Mindstorms-Software vor allem den Vorteil hoher Flexibilität. Das Programm ist kostenlos und kann auch von Schülern zu Hause kostenlos heruntergeladen, schnell und einfach installiert werden. Die Syntax ähnelt der von C bzw. C++. Ein Übergang zu anderen Systemen mit höheren Programmiersprachen wird hierdurch erleichtert.

Das Handbuch ist eine aus der Praxis entstandene Konzeption zur Einführung in das LEGO® MINDSTORMS® Education NXT System für Schüler und Lehrer.

© 2008, 2011 von Frank Engeln,
Bursibantstrasse. 3, D-48429 Rheine Germany
Alle Rechte vorbehalten.

Autor: Frank Engeln

Printed in Tauberbischofsheim

VS-Möbel
Hochhäuser Straße 8
97941 Tauberbischofsheim

Inhalt

BricxCC installieren.....	4
Kommentare.....	9
Zeichnen und schreiben	10
Erste Töne	12
Musik, oder ... wie bringe ich dem NXT die Noten bei?.....	14
Töne und Klänge in der Physik.....	14
Eine einfache Ampelanlage.....	17
Motoren.....	20
Programmanweisungen	21
Durchmesser, Radius und Umfang.....	22
Motoren II.....	24
Lösungen von Seite 25 und 26.....	27
Wiederholungen	28
Konstante.....	29
Konstante II	30
Variable	31
Variable II	32
Zufallszahlen	33
Steuerbefehle	34
Sensoren.....	37
Mit mehreren Tasks arbeiten.....	55
Subroutinen.....	57
Aufgabenblatt.....	58
Bluetooth	59
Programm mit Fehlern.....	63
Auflösung nächste Seiten	64
Auflösung von Seite 52/53.....	65
Beispielprogramme.....	67
NXC- Befehle im Überblick	68
Roboter Beispiel – Soccer	72
FAQ.....	74
Links	79

NXC

NXC ist eine Programmiersprache für das Lego Mindstorms NXT System. Die Sprache hat eine C ähnliche Syntax. NXC wird z. Z. ständig weiterentwickelt, daher können auf der Webseite verschiedene Versionen heruntergeladen werden.

Benötigte Programme unter Windows:

- BricxCC (<http://sourceforge.net/projects/bricxcc/>) Das Programm ist kostenlos und hat eine gepackte (*.zip) Größe von 2,7 Mb. In dem Download (test_release.zip) unter <http://bricxcc.sourceforge.net/> ist der NBC Compiler enthalten.

Die Installation:

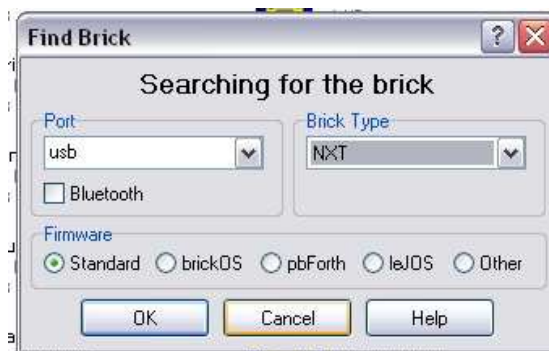
Zuerst muss man BricxCC herunterladen und ganz normal installieren. Die Datei kann entpackt auf andere Rechner kopiert werden und ist damit funktionsfähig.

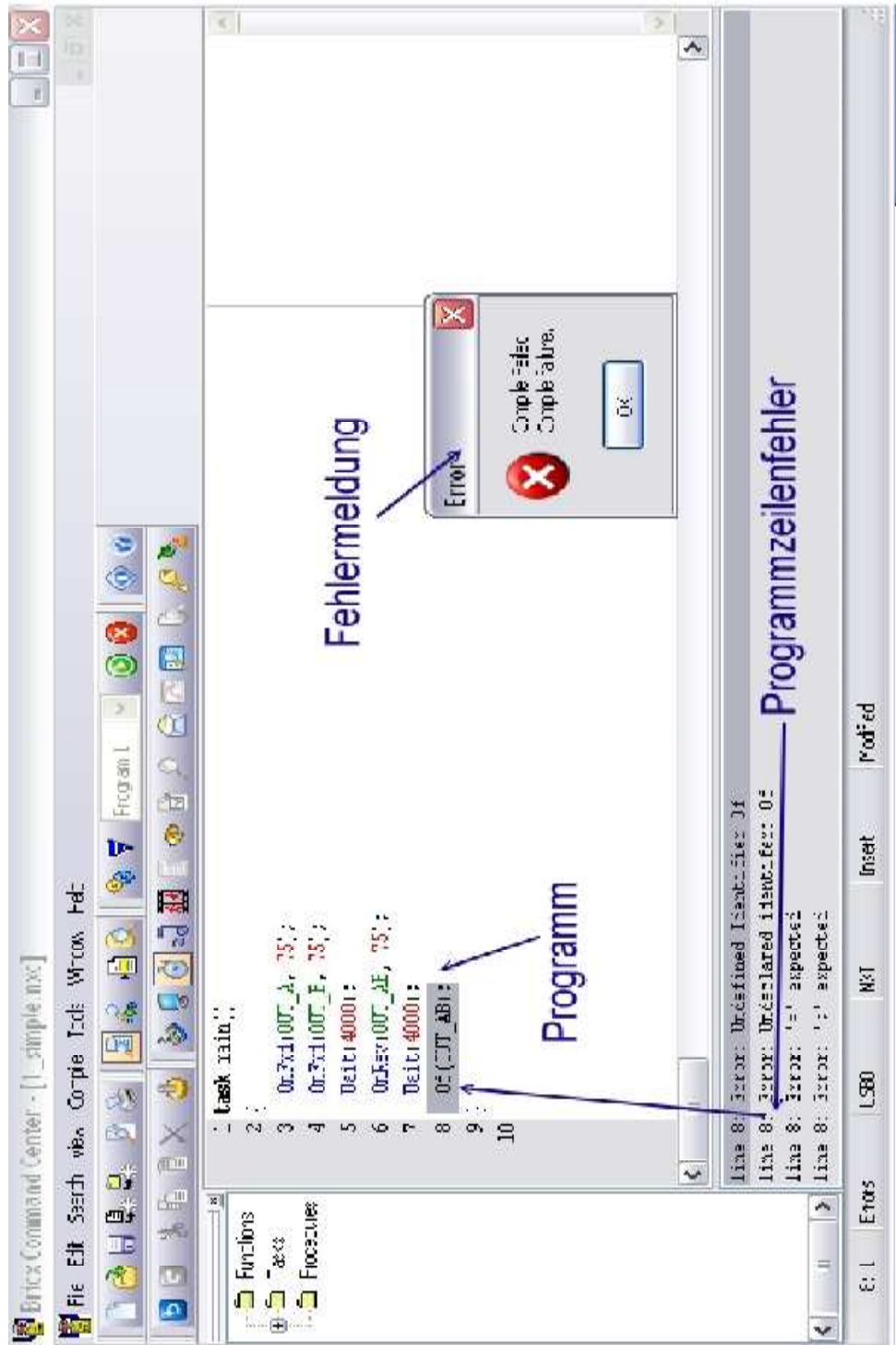
Nun ist man auch schon fertig mit der Installation und kann zum erstellen eines Programms BricxCC starten.

Die Auswahl beim Start von BricxCC

Beim Start von BricxCC muss man einige Angaben machen. Man muss die Felder folgendermaßen ausfüllen:

- Port: USB
- Brick Type: NXT
- Firmware: Standard





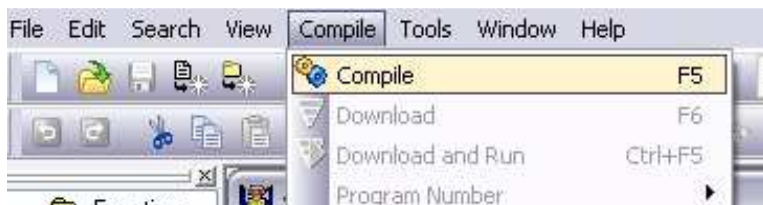
Programm erstellen:

Die Software erstellt man nun im BricxCC und speichert sie anschließend als NXC File (*.nxc). Das Programmsymbol sieht so aus:



Das Kompilieren:

Um die Software zu Kompilieren klickt man einfach auf die beiden Zahnräder in der Symbolleiste oder drückt F5.



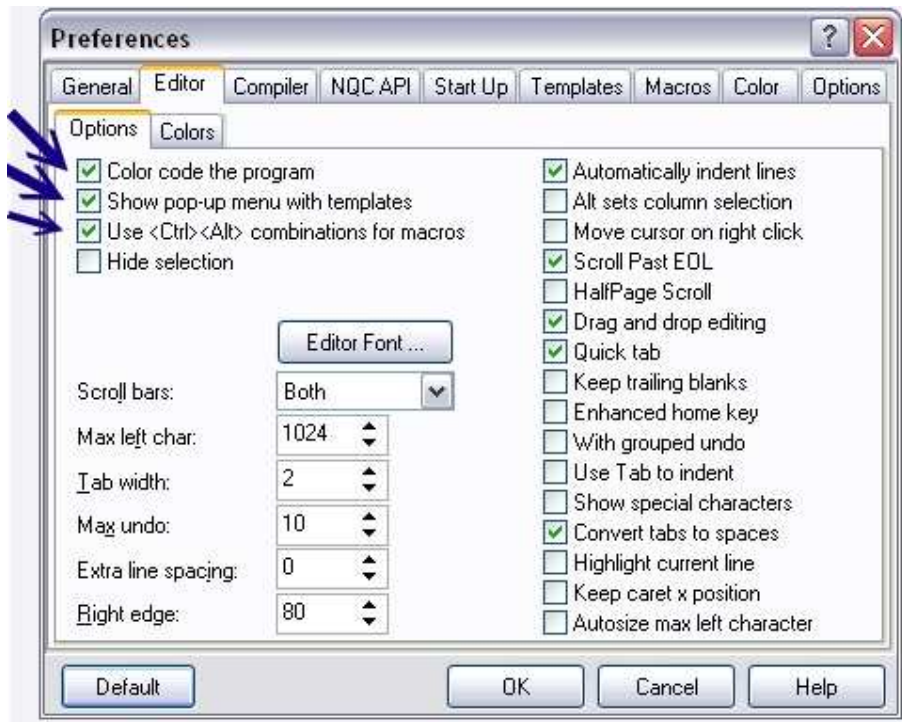
Das Transferieren auf den NXT:

Hierfür klickt man einfach auf das blaue Dreieck in der Symbolleiste oder drückt F6.

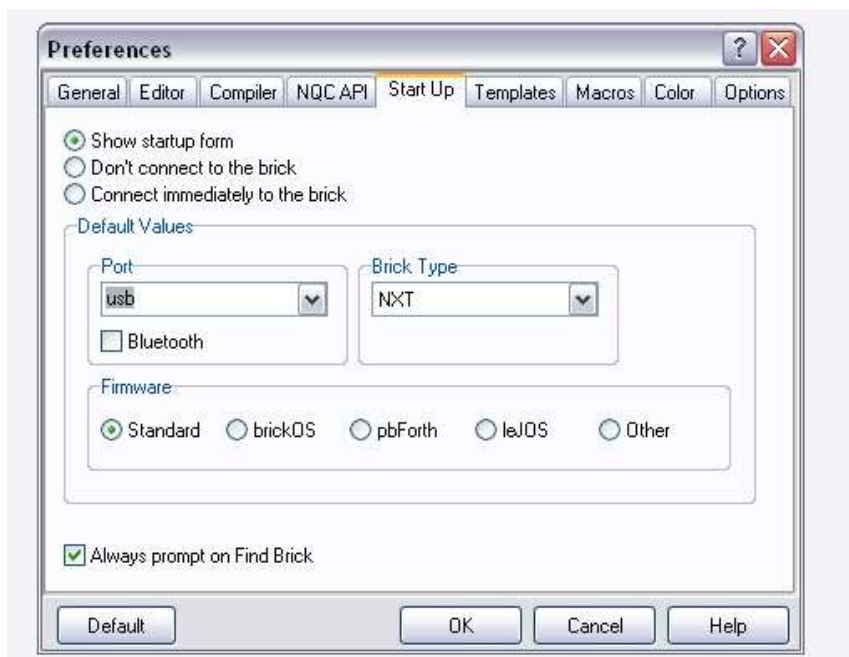


Die Tastaturkürzel und das Template-Fenster sind leider nicht in der Voreinstellung von BricxCC aktiviert. Das muss noch nachgeholt werden, indem man unter „Edit –Preferences – Editor“ die Häkchen für „show pop-up menu with templates“ und „USE <CTRL> <ALT> combinations for macro“ setzt.

Ebenfalls ist zu kontrollieren, ob „Color code the programm“ eingeschaltet ist. Dieses erleichtert das Lesen der Programme.



Um nicht bei jedem Start den Port und Brick auswählen zu müssen, kann man unter „Start Up“ eine Voreinstellung machen.



Programme können mit BricxCC auf drei verschiedene Arten erstellt werden.
Man . . .

1. schreibt das Programm Buchstabe für Buchstabe, Zeichen für Zeichen, oder man
2. nutzt das „Templatefenster“ (Vorlagenfenster), indem man sich den gewünschten Befehl per Mausklick aussucht, der dann im Programmfenster erscheint (2), oder man
3. gibt den Programmtext über Tastaturkürzel ein (der schnellste Weg!). Diese beziehen sich in der Regel auf die englischsprachige Beschreibung der Funktion.

Tastaturkürzel	Ausgabe (Befehle)	Bedeutung
<Ctrl><Alt><Shift>T	task "name"() { "statements" }	Aufgabe (task) festlegen - „name“ = main - „statements“ = weitere Programmeingaben
<Ctrl><Alt>a	OUT_A	Ausgang (Motor) A
<Ctrl><Alt>b	OUT_B	Ausgang (Motor) B
<Ctrl><Alt>c	OUT_C	Ausgang (Motor) C
<Ctrl><Alt>f	OnFwd();	Ausgang (Motor) auf vorwärts (forward) stellen
<Ctrl><Alt>r	OnRev();	Ausgang (Motor) auf rückwärts (rewind) stellen
<Ctrl><Alt>o	Off();	Ausgang (Motor) ausstellen (off)
<Ctrl><Alt>w	Wait();	Wartezeit (Wait) festlegen 1000 = 1 Sekunde

KOMMENTARE

Material: NXT Baustein, Übertragungskabel, PC

- Aufgabe:**
1. Erstellt ein Programm und schreibt Kommentare in der Form wie es unten zu sehen ist.
 2. Versuche einmal ein Bild aus Kommentare zu zeichnen.
 3. Macht einen Urheber oder Programmierhinweis

Kommentare

In NXC gibt es zwei Arten von Kommentaren. Die Erste kommt von der Programmiersprache C und beginnt mit `/*` und endet mit `*/`.

Man kann eine Zeile oder mehrere Zeilen Kommentieren. Das ist wichtig für die Dokumentation im Programm oder für den Urheber des Programms, sowie Hinweise. Der Compiler ignoriert Kommentare

```
/* Das ist ein Kommentar */
```

Bsp.:

```
/*
      " " "
      ( o o )
  ---ooO--- ( _ ) ---Ooo---
*/
```

Die zweite Art ist `//` und dieser endet mit Beginn einer neuen Zeile.

`//` ein Zeilen Kommentar

Bsp.:

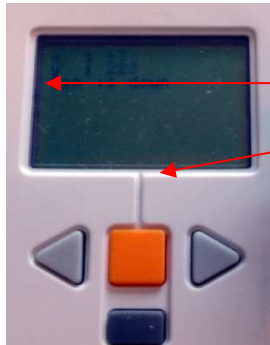
```
// Dieses Programm soll eine Hilfe sein.....
```

```
/******Frank Engeln*****/
```

ZEICHNEN UND SCHREIBEN

Material: NXT Baustein, Übertragungskabel, PC

Der NXT hat ein schwarz-weißes Display mit der Auflösung 100*64. Das LC Display ermöglicht die Anzeige von Texten und beispielsweise erfassten Messwerten.



Den Standort der Ausgaben gibt man mit x und y Koordinaten an (x von links, y von unten).

Für die y Koordinaten kann man auch folgendes benutzen
LCD_LINE1,
LCD_LINE2, bis LCD_LINE8. (von oben gezählt)

// Programm zum Ausgabe auf das Display

```
task main()
{
  TextOut(0, LCD_LINE2, "Hallo Welt"); //Gibt bei x=0 und y=2. Zeile von oben
  Wait(5000);
}
```

// Programm zur Ausgabe von Zahlen auf das Display

```
task main()
{
  NumOut(0, LCD_LINE8, 8); // Gibt bei x=0 und in der 8. Zeile von oben die Zahl 8 aus
  Wait(5000);
}
```

TextOut(x, y, Nachricht, clear=false)

NumOut(x, y, Wert, clear=false)

Die ersten beiden Angaben bezeichnen die Position, an der die Textausgabe beginnt. Der dritte Parameter ist der eigentliche Text. Der vierte Parameter ist optional und legt fest, ob das Display vor der Ausgabe gelöscht werden soll, oder nicht. Wenn man den Parameter weglässt, dann wird nicht gelöscht.

ClearScreen() löscht den Bildschirminhalt

Aufgabe: 1. Lasse dich von dem NXT mit deinem Namen in der Mitte des Displays begrüßen. Speicher die Aufgabe unter "hallo.nxc" ab.

Es gibt auch eine Reihe von weiteren Befehlen, mit denen man recht aufwändige Zeichnungen erstellen kann.

GraphicOut(x, y, filename, clear=false)

gibt eine Bitmapdatei aus, die sich auf dem Roboter befindet

CircleOut(x, y, Radius, clear=false)

zeichnet einen Kreis mit dem Mittelpunkt (x,y) und dem angegebenen Radius

LineOut(x1, y1, x2, y2, clear=false)

zeichnet eine Linie vom Punkt (x1,x2) zum Punkt (x2,y2)

PointOut(x, y, clear=false)

zeichnet einen einzelnen Punkt

RectOut(x, y, Breite, Höhe, clear=false)

zeichnet ein Rechteck mit der linken oberen Ecke in in (x,y) und den angegebenen Abmessungen

Bsp.:

LineOut(10, 100, 50); // Zeichnet eine Linie von x1=0 y1=10 zu x2=100 y2=50

CircleOut(20, LCD_LINE6, 15); // Zeichnet einen Kreis um x=20 in der 6.Zeile mit dem Radius 15

Mittlerweile gibt es auch viele Spiele für den NXT, die in NXC geschrieben wurden:

<http://gansweith.freehostia.com/nxtgames.html>

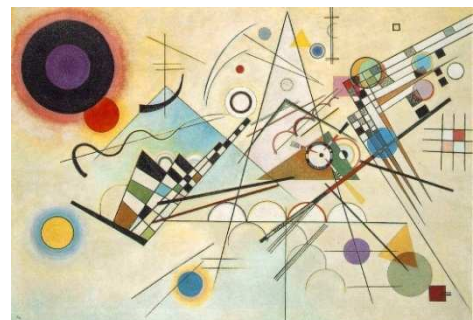
Wer mehr Grafiken oder anderes erstellen möchte, dem sei das Tool nXRICeditV2 nahegelegt.

Ein Tool zum Erstellen oder Bearbeiten von RIC Dateien für Mindstorms NXT

http://ric.dreier-privat.de/Docu/index_ger.htm

Aufgabe: 1. Entwerfe Grafiken – sei ein kleiner Kandinsky. Speicher die Aufgabe unter "kandinsky.nxc" ab.

Zur Information: *Kandinsky* war ein Künstler des Expressionismus und vor allem der abstrakten Kunst. Er stammte aus einer wohlhabenden Teehändlerfamilie aus Moskau.



ERSTE TÖNE

Material: NXT Baustein, Übertragungskabel, PC

Der **Morsecode** oder **Morsekode** ist ein Verfahren zur Übermittlung von Buchstaben und Zeichen. Dabei wird ein konstantes Signal ein- oder ausgeschaltet.

Der Code verwendet drei Symbole, die *Punkt* (·), *Strich* (–) und *Pause* () genannt werden, gesprochen als *Dit*, *Dah* und „Schweigen“. Genauer gilt Folgendes:

- Ein *Dah* ist üblicherweise dreimal so lang wie ein *Dit*.
- Die Pause zwischen zwei gesendeten Symbolen ist ein *Dit* lang.
- Zwischen Buchstaben in einem Wort wird eine Pause von *Dah* eingeschoben.
- Die Pause zwischen Wörtern beträgt sieben *Dits*.
-

--- --- ··· / ··· --- ---
M O R S E (space) C O D E

Lateinische Buchstaben Buchstabe im Code

A · –	J · – – –	S ···
B – ···	K – · –	T –
C – · – ·	L · – ···	U · · –
D – · ·	M – –	V ··· –
E ·	N – ·	W · – –
F · · – ·	O – – –	X – · · –
G – – ·	P · – – ·	Y – · – –
H ····	Q – – · –	Z – – · ·
I · ·	R · – ·	

Aufgabe: 1. Erstelle ein neues Programm. Speichere dieses mit dem Namen „morsen.nxc“ ab.

2. Was bedeutet: · – · · · – – · – – – / – · – · – – – ?

Um einen Ton erklingen zu lassen benötigt man die Anweisung:

PlayTone();

Bsp. :

```
task main()
{
  PlayTone(440, 200); // SpieleTon(Frequenz (Kammerton a1), Dauer (2sec.))
  Wait(500);
}
```

3. Schreibe das folgende Programm ab, speichere es und übertrage es auf den NXT Baustein. Starte das Programm.

```
Task main()
{
  PlayTone(440,200); PlayTone(440,75); Wait(200); //Pause
  PlayTone(440,200); PlayTone(440, 75); PlayTone(440, 75); PlayTone(440,200);
  Wait(200); PlayTone(440,200); PlayTone(440, 75); PlayTone(440,200);
  PlayTone(440, 75);
  Wait(400);
  PlayTone(440,200); PlayTone(440, 75); PlayTone(440, 200); PlayTone(440, 75);
  Wait(200);
  PlayTone(440, 200); PlayTone(440, 200); PlayTone(440,200); Wait(200);
  PlayTone(440,200); PlayTone(440, 75); PlayTone(440,75); Wait(200);
  PlayTone(440, 75); Wait(200);
}
```

4. Was ist die Bedeutung? Notiere Punkte und Striche und Übersetzte. Was muss geändert werden, damit es noch mehr nach „Morsen“ klingt?

5. Programmiere deinen Namen im Morsecode. Schreibe dazu erst deinen Namen in Punkte, Strichen und Pausen auf. Schreibe nun das Programm und speichere es unter „morseNamen.nxc“ ab. Spiele es nun deinem Tischnachbarn vor und lasse ihn den Morsecode übersetzen.

MUSIK, ODER ... WIE BRINGE ICH DEM NXT DIE NOTEN BEI?

Material: NXT Baustein, Übertragungskabel, PC

- Aufgabe:**
1. Erstelle ein neues Programm. Speichere dieses mit dem Namen „meineMusik.nxc“ ab.
 2. Schreibe das Programm Bsp. a) ab, speicher es, kompiliere es (F5) und übertrage es auf den NXT (F6). Probiere nun das Beispiel b).
 3. Ändere die Töne ab, erfinde Melodien.
 4. Versuche dich als Komponist und erfinde Musikstücke. Speichere diese ab.

Eine Warnung noch vorweg, C ist *case sensitive*, es kommt also sehr darauf an, ob man etwas groß oder klein schreibt.

Töne und Klänge in der Physik

Töne und Klänge sind die Grundbausteine der Musik. Physikalisch gesehen ist ein Ton eine Sinusschwingung, charakterisiert durch die Parameter Frequenz (Tonhöhe = Schallschwingungen pro Sekunde) und Amplitude (Lautstärke). Das menschliche Ohr nimmt Schallschwingungen ab einer Frequenz von etwa 16 Hz (Einheit Hertz, 1 Hz = 1 Schwingung pro Sekunde) als sehr tiefen Ton wahr. Der höchste wahrnehmbare Ton liegt bei Kindern und Jugendlichen bei etwa 20000 Hz (20 kHz) und lässt mit höherem Alter nach. Als Bezugspunkt für die Tonfrequenzen in der Musik ist der Kammerton a^1 mit einer Frequenz von 440 Hz festgelegt.

Bsp. a) :

```
task main()
{
  PlayTone(440, 400); // SpieleTon(Frequenz (Kammerton  $a^1$ ), Dauer (4 sec.))
  Wait(500);
}
```

Um einen Ton erklingen zu lassen, kann man auch den Befehl :

PlayToneEx(Frequenz, Dauer, Lautstärke, Wiederholung) nutzen. Bei „Wiederholung“ gibt es die Möglichkeit für wahr oder falsch, also *true* oder *false*.

Bsp. b):

```
task main()
{
  PlayToneEx(262,400,3,FALSE); Wait(500);
  PlayToneEx(294,400,2,FALSE); Wait(500);
  PlayToneEx(262,1600,8,FALSE); Wait(2000);
}
```

Übersicht der Frequenzen:

Ton	Frequenz								
	1.	2.	3.	4.	5.	6.	7.	8.	9.
B	247	494							
A#	233	466							
A	220	440							
G#		415							
G		392							
F#		370							
F		349							
E		330							
D#		311							
D		294							
C#		277							
C		262							

Wenn aus dem Kammerton A (fett gedruckt) ein tiefes A werden soll, so wird die Frequenz halbiert: 220 Hz.

Soll ein hoher Ton erklingen, so verdoppelt man den Wert.

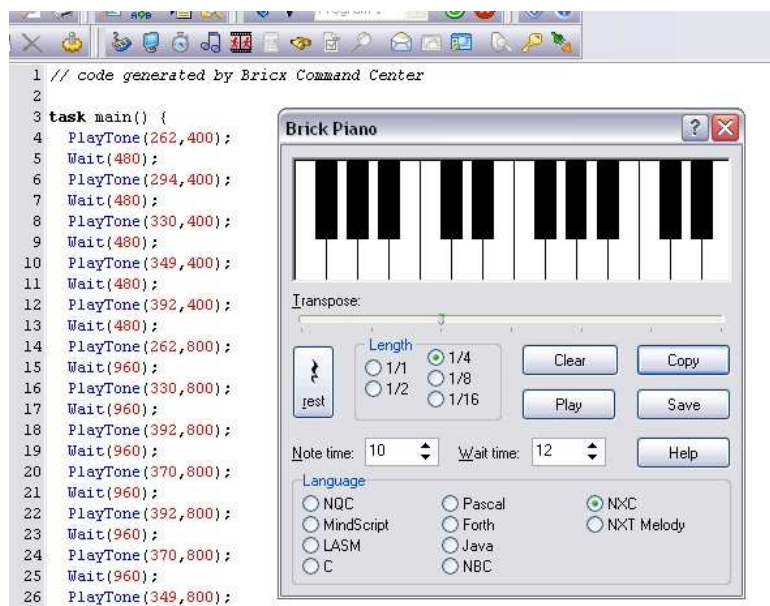
A 440 880 1760 3520 7040 14080

Der NXT wartet nicht, bis der Ton zu Ende gespielt ist . Wenn du also eine Tonfolge für ein Lied programmierst, füge besser Wait (); - Befehle hinzu, die etwas länger sein sollten, als die Tondauer!

Aufgabe: Fülle die Tabelle bis zur 9. Stufe aus.

Im BricxCC kann man unter "Tools / Brick Piano" direkt auf die Tasten drücken und ein Lied spielen. Als Ergebnis bekommt man z.B. diesen NXC-Code:

```
task main() {
  PlayTone(262,400);
  Wait(480);
  PlayTone(294,400);
  Wait(480);
  PlayTone(330,400);
  Wait(480);
  ...
```



Aufgabe: Spiele eine Melodie (NXT muss online sein) und speicher diese als *musik.nxc* ab (save). Öffne die Datei und sieh dir den Quelltext an.

EINE EINFACHE AMPELANLAGE

Material: NXT Baustein, Übertragungskabel, PC, 3 Lampen, Farbbausteine (rot, gelb, grün)

Aufgabe: 1. Erstelle eine Ampelschaltung wie in dem Bild unten links. Speichere dieses mit dem Namen „Ampel.nxc“ ab.

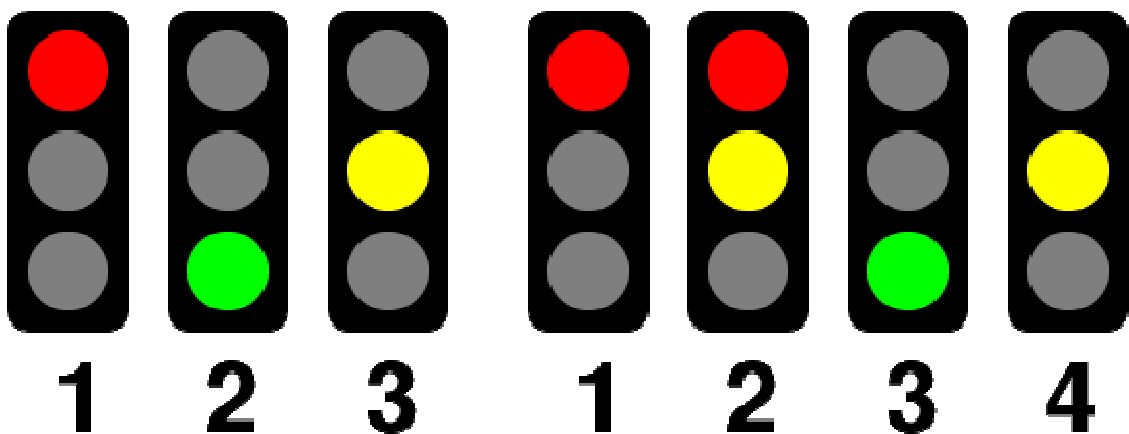
Ein NXC Programm hat immer diese Grundstruktur:

```
task main()
{
  „Anweisungen“;
}
```

Nach jeder Anweisung muss ein Semikolon folgen.

Eine normale europäische Lichtzeichenanlage steuert den Verkehr dabei mit Hilfe der drei Signalfarben Grün, Gelb und Rot. Zur Regelung des Verkehrs werden diese Farben einzeln oder in Kombination angezeigt. Die Reihenfolge (auch *Signalfolge* oder *Farbbildfolge* genannt) solch einer Lichtzeichenanlage ist dabei immer:

- Grün: Der Verkehr ist freigegeben
- Gelb: Auf nächstes Signal warten
- Rot: Keine Einfahrerlaubnis



Aus: <http://de.wikipedia.org>

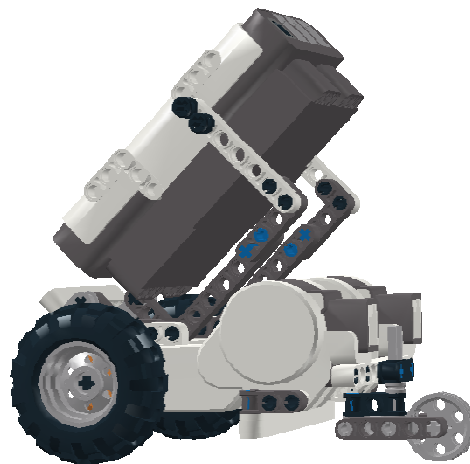
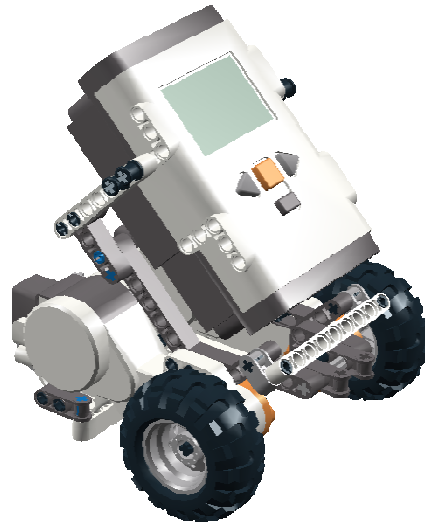
Varianten der Standard-Lichtzeichenanlage in verschiedenen Ländern
 In einzelnen Ländern sind noch zusätzliche Farbkombinationen zugleich oder hintereinander möglich:

Land	Ampelphase	Bemerkung
Deutschland, Großbritannien, Österreich, Ungarn, Schweiz, Polen, Norwegen, Russland,	Rot-Gelb (gleichzeitig): Zwischen rot und grün (Siehe Bild rechts)	<i>Achtung, gleich wird die Erlaubnis zur Fahrt gegeben</i>
Schweden	Grün-Gelb:	<i>Achtung, es wird gleich rot</i>
Belgien, Dänemark, Frankreich, Griechenland, Irland, Italien, Luxemburg, Niederlanden, Australien, Taiwan, Brasilien und den Vereinigten Staaten.	Grün folgt direkt auf Rot	-----
Vereinigte Staaten, Südafrika, Taiwan	Rotes Blinklicht	<i>Stopp! Anhalten, dann langsam weiterfahren, wenn Kreuzung frei.</i>
Österreich, Litauen, der Türkei, Mexiko und Israel	Grünes Blinklicht am Ende der Grünphase	<i>Achtung, es wird gleich gelb gezeigt</i>
China	Grünes Blinklicht am Ende der Grünphase	<i>Achtung, es wird gleich rot gezeigt</i>

Aufgabe: 2. Erstelle die verschiedenen Ampelschaltungen der unterschiedlichen Länder. Speichere diese mit dem Namen z.B. „Ampel_Deutschland.nxc“ ab.

Für die nächsten Aufgaben brauchst du einen einfachen Roboter mit 2 Motoren an Ausgang A und C.

Beispiel :



MOTOREN

Für die Ansteuerung der Motoren gibt es natürlich eine Vielzahl von Funktionen. Der NXT verfügt über drei Motorausgänge mit den Bezeichnungen A, B und C. Für diese Ausgänge sind passende Konstanten definiert:

- OUT_A
- OUT_B
- OUT_C

Da man häufiger mehrere Motoren gleichzeitig ansteuern möchte, sind auch andere Kombinationen möglich:

- OUT_AB
- OUT_AC
- OUT_BC
- OUT_ABC

Ein einfaches Programm könnte also folgendermaßen aussehen:

```
// Programm Motoren an
```

```
task main()
{
  OnFwd(OUT_AC, 50);
  Wait(500);
  Off(OUT_AC);
}
```

Hier werden die Motoren A und C eingeschaltet und zwar mit 50% der eigentlichen Leistung.



PROGRAMMANWEISUNGEN

Material: NXT Modell, Übertragungskabel, PC

Dieses Programm hat 3 Anweisungen.

```
task main()
{
  OnFwd(OUT_A,50);
  Wait(1000);
  Off(OUT_A);
}
```

„OnFwd(OUT_A, 50);“ = Diese Anweisung sagt dem Roboter den Motor, der an Ausgang A angeschlossen ist mit einer Vorwärtsdrehung zu starten. Die Geschwindigkeit des Motors ist 50%.

„Wait(1000);“ = Diese Anweisung sagt dem Roboter, dass er für 1 Sekunde warten soll.

„OnFwd(Port[s], Leistung)“ Für *Leistung* ist eine Zahl zwischen -100 und 100 anzugeben. Negative Werte bedeuten dabei eine Umkehr der Drehrichtung.

„Off(OUT_A);“ = Den Motor A abbremsen

Aufgaben:

1. Tippe das unten stehende Programm ab.
2. Schreibe neben jede Zeile, was das Programm macht
3. Beseitige alle Fehler im Programm
4. Kompiliere und übertrage das Programm
5. Speicher das Programm unter deinem Namen ab.
6. Drucke das Programm aus.

// Der Roboter soll.....

// Name und Datum

```
task main()          // _____
{
  OnFwd(OUT_AC, 75); // _____
  Wait(3000);        // _____
  OnRev(OUT_AC, 75); // _____
  Wait(3000);        // _____
  Off(OUT_AC);       // _____
}
```

Aufgaben: 1. Lasse den Roboter vorwärts fahren für 3 Sekunden mit einer Leistung von 80 %. Die

Die Motoren sind an A und C angeschlossen. Ändere die Werte und probiere den Roboter

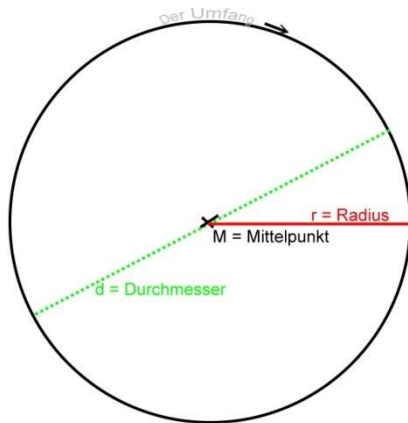
fahren zu lassen.

2. Der Roboter soll sich um 90° nach rechts drehen. Speichere die Ergebnisse unter „drehung90.nxc“ ab.

DURCHMESSER, RADIUS UND UMFANG

Radius, Durchmesser und Umdrehung. Diese drei Faktoren sind sehr wichtig in der Robotik. Soll ein Roboter besonders schnell fahren oder soll er eine Strecke mit Hindernissen (Unebenheiten auf dem Boden) meistern, ist es ein Fahrzeug für draußen (Off-Road)?

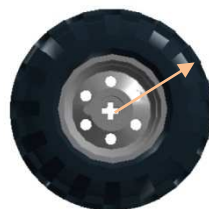
Als **Radius** (von lat. *radius*; „Strahl“) (deutsch: **Halbmesser**) bezeichnet man in der Geometrie den Abstand zwischen dem Mittelpunkt *M* eines Kreises und der Kreislinie.



Der Radius *r* entspricht dem halben Durchmesser *d*. Zum Kreisumfang *U* verhält sich der

Radius wie folgt: $r = \frac{U}{2\pi}$.

Bei dem NXT Rad ist das:



Der **Durchmesser** (griech. *Diameter*) ist die Entfernung zwischen den Schnittpunkten eines Kreises mit einer Geraden, die dessen Mittelpunkt schneidet.

Der Umfang ist die Strecke, die das Rad bei einer Umdrehung zurücklegt. Den Umfang errechnet man mit der Formel:

// Der Durchmesser x Pi = der Umfang

Pi (π) ist eine mathematische Konstante, eine Kreiszahl mit dem unendlichen Wert:
 3,14159 26535 89793 23846 26433 83279 50288 41971 69399 37510 58209 74944 59230
 78164 06286 20899 86280 34825 34211 70679 ...

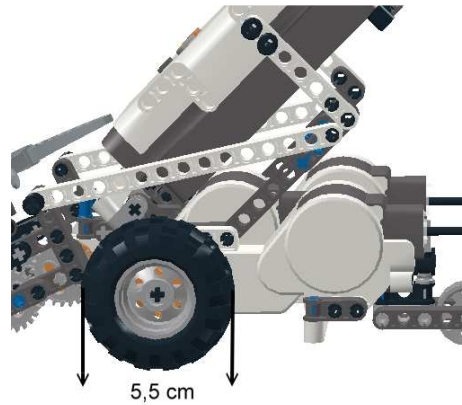
Der Umfang des Rades bzw. Reifen bedeutet abgewickelt auf die Straße den zurückgelegten Weg.

Bsp.: Ein Legorad mit einem Durchmesser von 3,18 cm hat einen Umfang von 9,99 cm.

Nach einer Umdrehung hat das Rad also 9,99 cm zurückgelegt. Jetzt ein paar Aufgaben zum Üben. Die richtige Radgrößenwahl kann manchmal einen Wettbewerb entscheiden!

Aufgaben:

1. Du hast diesen Roboter:
 Der Durchmesser des Rades beträgt 5,5 cm.



Wie weit ist der Roboter gefahren, wenn sich das Rad 8 mal gedreht hat?

_____ cm

2. Du bereitest dich auf einen Wettkampf vor. Die Aufgabe lautet: Der Roboter soll eine Strecke von 10 m so schnell es geht fahren. Du hast 4 Räder mit folgenden Durchmessern zur Auswahl:

2,86 cm , 3,18 cm , 5,5 cm und 7,94 cm.

Wie viele Umdrehungen muss dein gewähltes Rad machen?

Erstelle eine Tabelle in der du die Umdrehungen vergleichen kannst.

Durchmesser	Formel	Umdrehungen
2,86 cm	_____ · π =	_____
3,18 cm		
5,5 cm		
7,94 cm		

Wenn der Roboter eine bestimmte Streckenlänge fahren soll, er das aber nicht exakt so macht, kann es verschiedene Gründe dafür geben. Manchmal hängt es vom Antrieb deines Roboters, dem Ladezustand der Batterien oder des Akku und von der Art der Oberfläche ab, auf welcher der Roboter läuft.

Hier hilft es den Durchschnitt zu errechnen.

Aufgabe:

1. Nimm deinen Roboter und lasse ihn 3 Mal drei Sekunden vorwärts fahren.

Markiere mit Kreide einen Punkt auf den Reifen und zähle die exakten Umdrehungen.

Was stellst du fest?



d= 5,5 cm

Berechne nun den Durchschnitt.

Ergebnis 1 + Ergebnis 2 + Ergebnis 3

3

Der Roboter legt in 3 Sekunden im Durchschnitt eine Strecke von _____ cm zurück.

MOTOREN II

Material: NXT Modell (2 Motoren (OUT_A und OUT_C)), Übertragungskabel, PC

Wer eine Strecke genau fahren möchte, benötigt eine exakte Kontrolle der Motoren. Hier zeigen sich die Vorzüge von z.B. NXC. Alle notwendigen Funktionen sind in der Software vorhanden, werden aber von der grafischen Oberfläche (NXT-G) nicht genutzt. Für die Steuerung der Motoren gibt es sehr viele unterschiedliche Funktionen.

Hier eine Liste:

- Off(Ausgang) Schaltet den angegebenen Ausgang bzw. Motor ab und bremst
- Coast(Ausgang) Schaltet den Ausgang A ab und lässt den Motor auslaufen
- Float(Ausgang) Alias für Coast
- OnFwd(Ausgang, Leistung) Starten vorwärts
- OnRev(Ausgang, Leistung) Starten rückwärts
- OnFwdReg(Ausgang, Leistung, Modus) Motor(en) geregelt vorwärts starten
- OnRevReg(Ausgang, Leistung, Modus) Motor(en) geregelt rückwärts starten
- OnFwdSync(Ausgänge, Leistung, Versatz) Motoren synchronisiert vorwärts starten
- OnRevSync(Ausgänge, Leistung, Versatz) Motoren synchronisiert vorwärts starten

Aufgabe: 1. Schreibe ein kleines Programm, wobei der Roboter mit einer Leistung von 75% 2 Sekunden vorwärts fahren, bremsen, 5 Sekunden warten und dann 2 Sekunden rückwärts fahren soll. Nun soll der Roboter stoppen aber nicht bremsen. Mache eine Markierung, von wo du gestartet bist. Was stellst du fest? Beschreibe dein Ergebnis und erkläre den Unterschied von *Off* und *Float*.

2. Schreibe ein Programm mit „OnFwdSync(OUT_AC, 50,0)“, „OnRevSync(OUT_AC, 50, 90)“ und „OnFwdSync(OUT_AC, 50, -50)“. Lasse deiner Fantasie freien Lauf. Beobachte den Roboter und verändere die Werte.

Für *Leistung* ist eine Zahl zwischen -100 und 100 anzugeben. Negative Werte bedeuten dabei eine Umkehr der Drehrichtung. *Vorsicht:*

Diese Vereinbarung scheint nicht hundertprozentig zu stimmen. Geht man z.B. mit dem Wert von +100 aus weiter hoch bis +127, so gibt es kaum Veränderungen. Bei +128 kehrt sich aber die Drehrichtung der Motoren um. Ein Roboter, der sich vorher rechts herum gedreht hat, dreht sich nun links herum.

Für *Versatz* wird ebenfalls eine Zahl zwischen -100 und 100 verlangt. Dabei werden für 0 beide Motoren synchronisiert (geeignet z.B. für gerade Fahrt), bei 100 sowie -100 genau entgegengesetzt gedreht (jeweils andere Richtung, z.B. zum Drehen auf der Stelle). *Vorsicht:* Bei Werten knapp unterhalb von +50 bewegt sich nur einer der beiden Motoren, der andere bleibt nahezu regungslos.

Motoren II

Der nächste Befehl erlaubt es die Motoren zu synchronisieren, er bietet sich für normale gerade Fahrtstrecken an:

OnFwdReg(Ausgang, Leistung, RegMode) wie OnFwd, aber mit geregelter Synchronisation

Es gelten folgenden Angaben für RegMode

OUT_REGMODE_IDLE, keine Regulation

OUT_REGMODE_SPEED, Regelt die Geschwindigkeit eines Motors

OUT_REGMODE_SYNC, Synchronisiert die Drehung der Motoren

Interessant sind die Möglichkeiten der Funktion OnFwdReg, die das folgende Programm zeigt.

```
// Programm Motoren-Synchronisation

task main()
{
  OnFwdReg(OUT_AC, 50, OUT_REGMODE_IDLE );
  Wait(4000);

  PlayTone(440, 200); Wait(1000);
  OnFwdReg(OUT_AC, 50, OUT_REGMODE_SPEED );
  Wait(4000);

  PlayTone(440, 200); Wait(100);
  OnFwdReg(OUT_AC, 50, OUT_REGMODE_SYNC );
  Wait(4000);

  Off(OUT_AC);
}
}
```

Aufgabe: Schreibt das obere Programm ab, kompiliert es und überträgt es auf den NXT bzw. euren Roboter. (Speichern nicht vergessen) Startet das Programm und haltet eines der Räder gut fest. Was stellt ihr fest, wenn die einzelnen Funktionen starten?

- Beim OUT_REGMODE_IDLE

- Beim OUT_REGMODE_SPEED

- Beim OUT_REGMODE_SYNC

LÖSUNGEN VON SEITE 25 UND 26

Seite 25

/*Aufgabe: 1. Schreibe ein kleines Programm, wobei der Roboter mit einer Leistung von 75% 2 Sekunden vorwärts fahren, bremsen, 5 Sekunden warten und dann 2 Sekunden rückwärts fahren soll. Nun soll der Roboter stoppen aber nicht bremsen. Mache eine Markierung, von wo du gestartet bist. Was stellst du fest? Beschreibe dein Ergebnis und erkläre den Unterschied von *Off* und *Float*.*/

```
task main()
{
  OnFwd(OUT_AC, 75);
  Wait(2000);
  Off(OUT_AC);
  Wait(1000);
  OnFwd(OUT_AC, 75);
  Wait(2000);
  Float(OUT_AC);
}
```

Off schaltet den Motor aus und bremst, Float lässt den Motor auslaufen. Der Roboter steht nicht genau an seinem Ausgangspunkt.

/*Aufgabe 2 Ein Programm mit „OnFwdSync(OUT_AC, 50,0)“, „OnRevSync(OUT_AC, 50, 90)“ und „OnFwdSync(OUT_AC, 50, -50)“ */

```
task main()
{
  OnFwdSync(OUT_AC, 50, 0);
  Wait(1000);
  OnFwdSync(OUT_AC, 50, -50);
  Wait(1000);
  OnRevSync(OUT_AC, 50, 90);
  Wait(1000);
  Off(OUT_AC);
}
```

Seite 26

Wenn man dieses Programm startet und eines der Räder am Roboter mit der Hand gut festhält, dann sind die unterschiedlichen Möglichkeiten sehr gut zu spüren.

- Beim OUT_REGMODE_IDLE ist kein besonderer Effekt zu bemerken.
- Beim OUT_REGMODE_SPEED stellt man fest, dass der Motor kräftiger gegen den Widerstand angeht.
- Beim OUT_REGMODE_SYNC hält das andere Rad auch an, bis man wieder los lässt.

WIEDERHOLUNGEN

Material: NXT Modell, Übertragungskabel, PC

Schleifen.

Soll ein Roboter z.B. ein Quadrat fahren, so muss er vorwärts fahren, sich um 90⁰ drehen, vorwärts fahren, sich um 90⁰ drehen, vorwärts fahren, sich um 90⁰ drehen und vorwärts fahren, und sich um 90⁰ drehen. Um den Programmiercode zu vereinfachen benutzt man Schleifen.

repeat() wird zur Schleifenbildung genutzt. Die Zahl hinter der repeat- Anweisung in Klammern (), zeigt an, wie oft etwas wiederholt werden muss. Die Anweisungen,

Bsp.:

```
/* Der Roboter fährt hin und her - Kommentare nicht vergessen! */
```

```
task main()
{
    repeat(2)
    {
        OnFwd(OUT_AC,80);
        Wait(400);
        OnRev(OUT_AC,80);
        Wait(400);
    }
    Off(OUT_AC);
}
```

Was passiert? Der Roboter fährt vor und zurück, vor und zurück.

- Aufgabe:**
1. Der Roboter soll nun ein Quadrat fahren. Speichere dieses mit dem Namen „quadrat.nxc“ ab.
 2. Der Lego-Roboter soll nun 4 Mal ein Quadrat fahren. Speichere dieses mit dem Namen „quadrat4.nxc“ ab.
 3. Der Roboter soll nun eine spiralförmige Vorwärtsfahrt machen. Wie kann diese Spirale aussehen? Mache dir erst eine Zeichnung und überlege mit deinem Tischnachbarn. Speicher das Programm unter „spirale.nxc“ ab.

KONSTANTE

Material: NXT Modell, Übertragungskabel, PC

Wenn der Roboter sich um 90^0 drehen soll, er das aber nicht exakt so macht, kann es verschiedene Gründe dafür geben. Manchmal hängt es vom Antrieb deines Roboters, dem Ladezustand der Batterien oder des Akku und von der Art der Oberfläche ab, auf welcher der Roboter läuft. Anstatt, dieses im Programm zu ändern, ist es einfacher, einen Namen für diese Zahl zu verwenden. Konstante Werte (= immer gleichbleibende Werte) können in NXC festgelegt werden.

#define legt die Art der Konstanten fest (definiert die Konstante)
Eine Konstante wird im BricxCC rot dargestellt!

Bsp.:

```
#define FAHRZEIT 400  
#define DREHUNG 150
```

```
task main()  
{  
    OnFwd(OUT_AC);  
    Wait(FAHRZEIT);  
    OnRev(OUT_C);  
    Wait(DREHUNG);  
    Off(OUT_AC);  
}
```

Die ersten beiden Zeilen definieren die Konstanten. Diese können während des gesamten Programmablaufs verwendet werden.

Gründe die für die Verwendung von Konstanten sprechen:

1. Dein Programm wird übersichtlicher und lesbarer
2. Du kannst den Konstanten logische Namen geben
3. Bei größeren Programmen sind die Werte leichter zu finden, da die Konstanten zu

Beginn

des Programms festgelegt werden können, als irgendwo zwischen den vielen anderen Anweisungen.

Aufgabe: 1. Ändere deine Programme von Seite 28. Setze am Anfang Konstante ein. Speichere dieses mit dem Namen „quadratKonst.nxc“ ab.

Wenn der Roboter kein schönes Quadrat gefahren ist, verändere die Werte in den ersten beiden Zeilen.

KONSTANTE II

Springen wir zurück zur Musik.

```
/* Programm zu Ausgabe eines Liedes  
geschrieben von Frank Engeln 2008 */
```

Gegeben sind die Konstanten:

```
#define c6 1048  
#define d6 1175  
#define e6 1319  
#define f6 1397  
#define g6 1568
```

Schreibe nun folgendes Stück:

```
g6-e6-e6  
f6-d6-d6  
c6-d6-e6-f6-g6-g6-g6
```

```
#define c6 1048  
#define d6 1175  
#define e6 1319  
#define f6 1397  
#define g6 1568
```

```
task main()  
{  
  PlayTone(g6, 400); Wait(500);  
  PlayTone(e6, 400); Wait(500);  
  PlayTone(e6, 800); Wait(900);  
  PlayTone(f6, 400); Wait(500);  
  PlayTone(d6, 400); Wait(500);  
  PlayTone(d6, 800); Wait(900);  
  PlayTone(c6, 400); Wait(500);  
  PlayTone(d6, 400); Wait(500);  
  PlayTone(e6, 400); Wait(500);  
  PlayTone(f6, 400); Wait(500);  
  PlayTone(g6, 400); Wait(500);  
  PlayTone(g6, 400); Wait(500);  
  PlayTone(g6, 800); Wait(900);  
}
```

Aufgabe: 1. Schreibe das Programm ab. Um welches Lied handelt es sich? Kannst du es zu Ende schreiben?

VARIABLE

Eine Variable ist mit einem kleinen Behälter vergleichbar, in dem du Dinge – vor allem Zahlen – speichern kannst, die du später wieder an unterschiedlichen Stellen im Programm verwenden oder auch ändern möchtest.

Um neue Variablen des Typ *int* anzulegen benutzt man folgenden Befehl:

```
int name;
```

Diese Variable ist jetzt mit "name" abrufbar.

Der Variablentyp *int* entspricht einer Ganzzahl – einer Zahl ohne Bruchteil (Ganzzahlen werden auch als natürliche oder ganze Zahlen bezeichnet).

int = integer = eine einfache fortlaufende positive oder negative Ganzzahl.

Zurück zu dem Spiral-Programm - „spirale.nxc“ – von der Aufgabe 3, Seite 28.

Eine mögliche Lösung:

```
#define DREHUNG 250
```

```
int fahrzeit; //hier klein geschrieben um zwischen Konstanter und Variabler zu unterscheiden
```

```
task main()
```

```
{
```

```
  fahrzeit = 200; //weist der Variablen einen Startwert zu
```

```
  repeat(10) //du weißt noch? Repeat=wiederholen
```

```
    {
```

```
      OnFwd(OUT_AC);
```

```
      Wait(fahrzeit); //die Variable wird hier verwendet
```

```
      OnRev(OUT_C);
```

```
      Wait(DREHUNG);
```

```
      fahrzeit += 50; //der Wert der „fahrzeit“ wird bei jedem Durchgang um 50 erhöht
```

```
    }
```

```
  Off(OUT_AC);
```

```
}
```

Aufgabe: 1. Schreibe das Programm ab. Ändere die Werte. Speichere dieses mit dem Namen „spiraleVariab.nxc“ ab.

Hinweis: Dateinamen können beim NXT maximal 15 Zeichen plus 3 Zeichen für die Endung lang sein!

VARIABLE II

Noch einige Informationen zu den Variablen.

Die Angaben der Sekundenwerte wird wie folgt errechnet:

fahrzeit=200; // 200 / (geteilt durch) 200 = 2 Sekunden

Neben addieren (+=) zu einer Variablen können wir einer Variable auch mit einer Zahl multiplizieren (*=), subtrahieren (-=) und durch das Verwenden von /= teilen.

Beim Teilen wird das Ergebnis zur nächsten Zahl gerundet. Zur Erinnerung:

int = ganze Zahlen

Es können pro Zeile auch mehrere Variablen definiert werden.

```
int ersterWert;  
int zweiterWert, dritterWert;  
  
task main()  
{  
    ersterWert = 10;  
    zweiterWert = 20 * 5;  
    dritterWert = zweiterWert;  
    dritterWert /= ersterWert;  
    dritterWert -= 1;  
    ersterWert = 10 * (dritterWert + 3);  
}
```

Aufgaben:

1. Schreibe das Programm ab.
2. Welches Ergebnis ist zu erwarten?
4. Wie kann man das Ergebnis auf das Display ausgeben?
Speichere dieses mit dem Namen „rechenVariab.nxc“ ab.

5. Lass deinen Roboter nun „tanzen“ und spielen. Mindestens 30 Sekunden lang soll er sich bewegen, drehen und Musik spielen. Speichere dieses mit dem Namen „tanzen.nxc“ ab.

Zufallszahlen

Was macht der Roboter eigentlich, wenn wir ihm nicht genau sagen was er machen soll?

Der Roboter soll Dinge machen, die wir nicht vorhersehen können. Dafür benutzt man Zufallszahlen wie bei einem Würfelspiel.

Das nächste Beispiel verwendet Zufallszahlen und lässt den Roboter zufällige Bewegungen machen.

```
int fahrzeit, drehung;

task main()
{
  repeat(10)
  {
    fahrzeit = Random(7000);
    drehung = Random(1000);
    OnFwd(OUT_AC, 50);
    Wait(fahrzeit);
    OnRev(OUT_A, 50);
    Wait(drehung);
  }
}
```

Das Programm definiert zwei Variable und diesen werden Zufallszahlen zugewiesen.

Random(7000); heißt = irgendeine Zahl (ein Wert) zwischen 0 und 7000. Diese Zahlen verändern sich ständig. Die Angaben sind Angaben in Sekunden. Die Fahrzeit kann 7 Sekunden oder weniger sein.

Man kann auch direkt „Wait(Random(7000))“ schreiben. Dieses ist aber umständlicher.

- Aufgabe:**
1. Schreibe das Programm ab. Kompiliere und übertrage es auf deinen Roboter.
 2. Schreibe ein eigenes Programm und lasse den Roboter „Freestyle-tanzen“.

STEUERBEFEHLE

In NXC gibt es die **While-Schleife** als Kontrollstruktur. Sie dient dazu, eine Abfolge von Anweisungen mehrfach auszuführen, solange eine Bedingung erfüllt ist. Diese Bedingung wird geprüft, bevor die Anweisungsfolge abgearbeitet wird. Es kann also auch sein, dass die Abfolge gar nicht ausgeführt wird. Wenn die Bedingung ständig erfüllt ist, dann wird die Schleife zur Endlosschleife. Wenn deutsche Befehlswoorte gewählt werden, dann heißt die While-Schleife meist *Solange-Schleife*.

Zurück zu den Zufallszahlen. Wir haben hier eine „repeat“-Anweisung vorgegeben. Macht man an der Stelle eine „while“-Anweisung, so wird das Programm solange ausgeführt, bis der Wert nicht mehr „true“ also der Wahrheit entspricht.

Aufgabe: Öffne dein Programm (s.u.) und ersetze „repeat(10)“ mit „while(true)“. Was stellst du fest?

```
int fahrzeit, drehung;

task main()
{
  repeat(10)
  {
    fahrzeit = Random(7000);
    drehung = Random(1000);
    OnFwd(OUT_AC, 50);
    Wait(fahrzeit);
    OnRev(OUT_A, 50);
    Wait(drehung);
  }
}
```

Die „if“-Anweisung ähnelt der „while“-Anweisung.

Eine **bedingte Anweisung** ist in NXC eine Anweisung, die nur unter einer bestimmten Bedingung ausgeführt wird. Sie ist einer der wichtigsten Bestandteile der Programmierung, da durch sie ein Programm auf unterschiedliche Zustände und Eingaben reagieren kann.

```
if (Temperatur > 20)           //wenn die Temperatur größer als 20 (Grad) ist, dann
  schalte die                 Heizung aus.
    HeizungAusschalten();
else                           // ansonsten schalte die Heizung ein
  HeizungEinschalten();
```

Wenn die Bedingung zwischen den Klammern () zutrifft, wird der Teil zwischen den Klammern { } durchgeführt. Andernfalls wird der Teil zwischen den Klammern { } nach dem Wort „else“ durchgeführt.

Einer Variablen kann ein Wert zugewiesen werden. Werte können auf unterschiedliche Weise unterschieden werden:

== ist genau gleich
< kleiner als
<= kleiner oder gleich
> größer als
>= größer oder gleich
!= ungleich

Bsp.: Wenn der Roboter (das Rad) 5 Umdrehungen gemacht hat, dann mache, ansonsten fahre rückwärts.

Du kannst die Bedingungen mit && untereinander kombinieren, was "und" bedeutet, oder mit ||, was "oder" bedeutet.

Den |-Strich erreichst du mit der Tastenkombination „Alt Gr“ und „><“

Hier sind einige Beispiele für Bedingungen:

true immer wahr
false nie wahr
ttt != 3 wahr, wenn ttt ungleich 3 ist
(ttt >= 5) && (ttt <= 10) wahr, wenn ttt zwischen 5 (inclusive) und 10 (inclusive) liegt
(aaa == 10) || (bbb == 10) wahr, wenn aaa oder bbb (oder beide) genau 10 sind

Beachte bitte, dass, die „if“- Anweisung aus zwei Teilen besteht. Die eine Hälfte wird sofort nach der „if“- Bedingung ausgeführt, wenn die Bedingung zutreffend ist, und die andere Hälfte wird nach dem „else“ ausgeführt, wenn die Bedingung falsch ist. Das Schlüsselwort „else“ und die Anweisungen nach ihm müssen aber nicht unbedingt sein. Sie können weggelassen werden, wenn es nichts gibt zu tun, falls die Bedingung falsch ist.

Aufgabe: Ändere das Programm auf Seite 33 so ab, dass der Roboter sofort stoppen soll, wenn eine Zufallszahl der Fahrzeit kleiner gleich 50 ist.

In NXC gibt es eine Kontrollstruktur namens **Do-while-Schleife**, die es ermöglicht, dass abhängig von einer gegebenen Booleschen Bedingung ein Codeabschnitt wiederholt ausgeführt wird.

Der Code innerhalb des Rumpfes wird zuerst ausgeführt, dann wird die Bedingung ausgewertet. Falls die Bedingung wahr ist, so wird der Code innerhalb des Rumpfes erneut ausgeführt. Dies wiederholt sich, bis die Bedingung falsch wird.

```
do
{
  Befehle;
}

while (Bedingung);
```

Beispiel:

Der Roboter soll irgendwie für 30 Sekunden herumfahren.

```
int fahrzeit, drehung; gesamtzeit;

task main()
{
  gesamtzeit = 0;
  do
  {
    fahrzeit = Random(7000);
    drehung = Random(1000);
    OnFwd(OUT_AC, 50);
    Wait(fahrzeit);
    OnRev(OUT_A, 50);
    Wait(drehung);
  }
  while (gesamtzeit < 30000);
  Float(OUT_AC);
}
```

Noch einmal zur Erklärung:

Die **while**- Anweisung prüft die Bedingung innerhalb der Klammern () vor dem Abarbeiten der Befehle innerhalb der Klammern { }, während die **do**- Anweisung dies erst danach tut. Das heißt, dass die Befehle { } nach der **while**- Anweisung unter Umständen nie abgearbeitet werden, nämlich dann, wenn die Bedingung () nie zutrifft.

Bei der **do**- Anweisung werden die Befehle { } zumindest einmal abgearbeitet, da die Bedingung () ja erst danach überprüft wird.

Aufgabe: Dein Roboter soll für mindestens 15, höchstens aber für 30 Sekunden „tanzen“. Er soll sich dabei zufällig nach links und rechts drehen.

SENSOREN



SENSOREN – UMDREHUNGSSENSOR

In jedem Motor ist ein Rotationssensor eingebaut. Das Rotationssignal beinhaltet zwei Werte:

1. die Drehrichtung des Motors
2. die Anzahl der Drehimpulse.

Der NXT-Baustein erhält 360 Drehimpulse. Das entspricht 360° für eine Umdrehung.

Für Fortgeschrittene:

Die Winkelstellung eines Motors am Anschluss *Port* kann jederzeit, also auch während der Drehung, mit int **MotorTachoCount**(*Port*) abgefragt und mit **ResetTachoCount**(*Port*) auf 0 zurückgesetzt werden.

ResetRotationCount(*Port*) zurücksetzen

ResetBlockTachoCount(*Port*) zurücksetzen

ResetAllTachoCounts(*Port*) setzt *alle* Rotationszähler zurück auf 0, entspricht also dem Aufruf aller **Reset...Count**()

Will man seinen Roboter in eine genau festgelegte Strecke fahren lassen, oder der Roboter soll in einem bestimmten Winkel seine Richtung ändern, so nutzt man diese Befehle:

RotateMotor(*Ausgang, Leistung, Winkel*)

RotateMotorEx(*Ausgang, Leistung, Winkel, Versatz, bSync, bStop*)

Die Motoren drehen sich also solange, bis sie einen bestimmten Drehwinkel erreicht haben.

Für *bSync* und *bStop* sind true oder false einzusetzen. Für *bSync* bedeutet true, dass die Motoren an den entsprechenden Ausgängen mit dem angegebenen *Versatz* synchronisiert werden sollen (s.25). Ist *bStop* auf true gesetzt, so wird der Motor nach der Rotation abgebremst (vgl. die Funktion **Off**()), ansonsten werden sie lediglich auslaufen gelassen (vgl. die Funktion **Coast**()).

Die Motor-Befehle im ersten Teil setzen die Motoren nur in Gang, das Abschalten der Motoren erfolgt dann in Abhängigkeit von Sensor-Werten, oder der Zeit. Bei den folgenden Motor-Befehlen wird ein Drehwinkel vorgegeben, Dieses Verhalten ist z.B. dann nützlich, wenn man eine genau festgelegte Strecke zurücklegen möchte, oder der Roboter seine Richtung um einen bestimmten Winkel ändern soll.

Für recht genaue Rotationen des Roboters wird man in der Regel **RotateMotorEx** nutzen.

RotateMotorEx(OUT_BC, 30, 180, 100, true, true)

bewirkt eine Rechtsdrehung des Roboters um 90° . Die Genauigkeit der Bewegung hängt aber auch von äußeren Faktoren, wie dem Untergrund und der exakten Montage der Räder ab, zuviel darf man hier also nicht erwarten.

Aufgabe: Auf Seite 28 hast du gelernt ein Quadrat zu fahren. Ändere das Programm so ab, dass der Roboter ein Quadrat fährt mit Hilfe von „**RotateMotorEx**()“.



SENSOREN – ULTRASCHALLSENSOR

Delphine benötigen ihn zur Navigation, Kommunikation, Lokalisierung von Fischen, Fledermäuse zur Navigation und Lokalisierung von Beutetieren. Der Ultraschallsensor.

Mit **Ultraschall (US)** bezeichnet man Schall mit Frequenzen, die oberhalb des vom Menschen wahrgenommenen Bereiches liegen. Das umfasst Frequenzen zwischen etwa 20 kHz und 1...10 GHz. Der Vorteil eines Ultraschallsensors besteht darin, dass diese Sensoren ohne eine Abtastung des zu vermessenden Objektes materialunabhängig funktionieren. Der Ultraschallsensor ist der einzige digitale Sensor, er besitzt einen integrierten Mikrocontroller, der alle Ultraschallmesswerte zum NXT-Baustein per I2C-Kommunikation sendet.

Wie funktioniert das?

Der Ultraschallsensor eruiert den Abstand zu Gegenständen, indem er 12 Signalstöße bei 40 kHz aussendet und dann die Zeit misst, bis er eine Reflektion des ausgesandten Signals empfängt. Die Zeit zwischen der Emission und dem Empfangen des reflektierten Stoßes ist proportional zum Abstand zwischen Objekt und Sensor. Der messbare Distanzbereich liegt zwischen 0 cm und 255 cm und kann daher durch ein einziges Byte kodiert werden. Der Ultraschallsender benötigt die gelieferten 9 Volt, daher nimmt die Messgenauigkeit mit abnehmender Batteriespannung ab.

Um ihn als solchen zu erkennen benutzt man folgende Funktion:

```
SetSensorLowspeed(IN_1); // Setzt IN_1 als Ultraschallsensor
```

Der Befehl zum Auslesen sieht diesmal etwas anders aus:

```
ultraschall = SensorUS(IN_1); // liest den Wert des Ultraschallsensors in Port 1 aus.
```

Aufgabe: Erkläre was das folgende Programm macht. Schreibe die Erklärung auf die Linie. Tippe das Program ab und übertrage es auf deinen Roboter. Achte darauf, dass der Sensor und die Motoren am richtigen Ausgang/Eingang angeschlossen sind.

```
task main() // _____
{
  SetSensorLowspeed(IN_1); // _____
  while(true) // _____
  {
    OnFwd(OUT_AC,50); // _____
    while(SensorUS(IN_1)>15); // _____
    Off(OUT_AC); // _____
    OnRev(OUT_AC, 75); // _____
    Wait(500); // _____
  }
}
```

Aufgabe: Schreibe nun ein Programm für eine Alarmanlage. Sobald ein Hindernis in 25 cm Nähe des Sensors kommt, soll ein lautes Signal ertönen.



SENSOREN – LICHTSENSOR

Der Lichtsensor ist ein analoger Sensor.

Es gibt zwei verschiedene Arten (Modus) den Lichtsensor zu nutzen:

- im Reflected Light Modus wird eine integrierte LED aktiviert, welche das Messen von Helligkeitswerten auch bei schwachem Umgebungslicht möglich macht und
- den Ambient Light-Modus (ohne LED).

Die Empfindlichkeit des Sensors ist bei frontalem Lichteinfall und bei Licht mit verhältnismäßig langer Wellenlänge (rotes und infrarotes Licht) am größten. Je grösser der Winkel des Lichteinfalls, desto schlechter wird das Licht vom Sensor detektiert.

Der Lichtsensor erkennt keine Farben, kann diese aber „erahnen“.

Das sehen wir:



-

das sieht der Sensor:



Er misst die Helligkeit im Bereich von 0 bis 100%.

Angesprochen wird der Sensor mit:

```
SetSensorLight(IN_2); // Setzt IN_2 als Lichtsensor
```

Die Werte des Sensors werden mit folgendem Befehl ausgelesen:

```
licht = Sensor(IN_2); // liest den Helligkeitswert von IN_2 aus
```

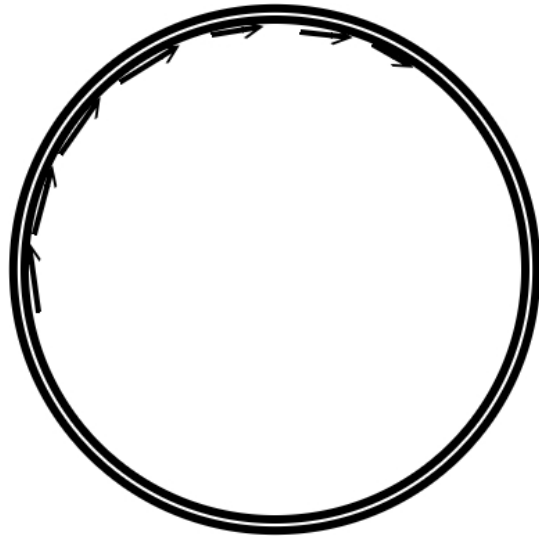
Die Variable „licht“ ist vorher zu erstellen.

Wir benötigen auch die Testauflage mit der schwarzen Linie, die mit dem NXT Satz ausgeliefert wird und einen Roboter mit Lichtsensor. Dieser sollte möglichst nah über den Boden angebracht werden. Das Grundprinzip der „Linie folgen“ ist, dass der Roboter versucht, auf dem Rand der schwarzen Linie zu bleiben und sich von der hellen Umgebung abwendet.

```

#define GRENZWERT 40
task main()
{
SetSensorLight(IN_2);
OnFwd(OUT_AC, 50);
while (true)
{
if (Sensor(IN_2) > GRENZWERT)
{
OnRev(OUT_C, 50);
Wait(100);
until(Sensor(IN_2) <= GRENZWERT);
OnFwd(OUT_AC, 50);
}
}
}

```



#define GRENZWERT 40 ist ein Vergleichswert für die Helligkeit. Diese muss auf das Umgebungslicht angepasst werden. Der Sensor liegt an Port 2 an und der Roboter fährt erst einmal vorwärts. In der Schleife fragt der Sensor ab, wann immer der Wert größer als 40 (=heller Untergrund) ist, schalte den Motor C in den Rückwärtsgang, bis der Helligkeitswert wieder unter 40 gesunken ist (=dunkler Untergrund). Der Roboter fährt im Uhrzeigersinn.

- Aufgabe:**
1. Schreibe ein Programm wobei der Roboter einer schwarzen Linie folgt, aber gegen den Uhrzeigersinn fährt.
 2. Baue einen Roboter, der sich nur innerhalb eines mit dunklen Rand begrenzten Feldes bewegt.
 3. Schreibe ein Programm, welches verhindert dass dein Roboter über die Tischkante hinausfährt und damit vom Tisch fällt.

Soll die interne Lichtquelle nicht aktiviert sein, so muss man sie mittels

```
SetSensorType(S3, SENSOR_TYPE_LIGHT_INACTIVE);
```

ausgeschaltet werden.

LICHTMESSUNG MIT DEM NXT

Aufgabe: Schreibe das folgende Programm ab, führe das Programm aus und ermittle die Sensorwerte für die Farbe rot, blau und gelb. Trage die Ergebnisse in die Tabelle ein und vergleiche eure Werte (Sensor an Port 2). Die Werte werden auf dem Display des NXT ausgegeben.

```
int wert;
sub display() //definiert ein Unterprogramm "display"
{
    TextOut(0, 20, "Messwert :");
    NumOut(50, 20, wert);
    return;
}
task main()
{
    SetSensorLight(IN_2);
    while(true)
    {
        wert = Sensor(IN_2);
        display(); //hier wird das Unterprogramm aufgerufen
    }
}
```

Farbe	rot	blau	gelb	weiß	schwarz
Wert					

SENSOREN – TASTSENSOR



Der Tastsensor wird analog betrieben. Dieser passive Sensor hat eine sehr geringe Stromaufnahme.

Der Tastsensor des NXT erkennt nur zwischen 2 Zuständen.

- 0 = nicht gedrückt
- 1 = gedrückt

Angesprochen wird der Sensor durch:

```
SetSensorTouch(IN_3); // Setzt IN_3 als Tastsensor
```

Übersetzt aus dem Engl.: Set = Setze; Sensor = Sensor; Touch = berühren

Zum Auslesen verwenden wir wieder folgenden Befehl:

```
tastsensor = Sensor(IN_3); // liest den Wert des Sensors IN_3 aus
```

Die Variable tastsensor ist zu erstellen.

Aufgabe: SchlieÙe den Tastsensor an Port 3. Der Roboter soll **so lange** geradeausfahren, **bis** er gegen ein Hindernis stößt. Versuche das Programm zu erarbeiten, indem du den Text unten durcharbeitest.

- Schreibe ein neues Programm.
- Speichere es unter dem Namen „tastsensor.nxc“ ab.
- Definiere den Sensortyp an Port 3.
- Der Roboter soll nun vorwärtsfahren, **so lange bis** der Taster gedrückt wird.
- Der Befehl „**until**“ (englisch = so lange bis) ist sehr leistungsfähig. Der Programmablauf wird an dieser Stelle unterbrochen, bis die Bedingung innerhalb der Klammern () zutrifft.
- Die Bedingung ist, das der Wert von Sensor_3 == 1 sein muss. Das bedeutet, dass der Sensor gedrückt wurde. Wird der Sensor nicht gedrückt ist der Wert = 0. Trifft der Roboter auf einen Gegenstand und der Tastsensor wird gedrückt, die Bedingung wird also wahr, wird das Programm fortgesetzt und der Roboter soll stoppen.

Wie sieht der Quelltext aus? Schreibe das Programm.

```
task main ( )  
  
{  
  
Programm mit 4 Zeilen  
  
}
```



SENSOREN – GERÄUSCHSENSOR

Der Geräuschsensor, Tonsensor oder Soundsensor ist dem Lichtsensor in der Handhabung sehr ähnlich. Dieser analoge Sensor misst Schalldruck entweder in dB oder in dBA. Es können Werte im Bereich von 55 dB bis 90 dB gemessen werden. 90 dB haben Autohupen oder sind LKW-Fahrgeräusche. Dieser Sensor kann also nur hören wie laut etwas ist.

Der Zusatz A gibt an, dass die unterschiedlichen Tonfrequenzen ähnlich dem menschlichen Hörempfinden unterschiedlich bewertet werden, d.h. mittlere Frequenzen (Spitze bei 2 kHz) werden stärker berücksichtigt. Der allgemeine Frequenzbereich reicht von 20 Hz bis 18 kHz.

Schalldruckpegel sind extrem schwierig zu messen, also werden die Sensor-Messwerte auf dem MINDSTORMS NXT in Prozent angezeigt [%].

- 4-5% ist wie ein leises Wohnzimmer
- 5-10% Menschen die in einem Abstand miteinander sprechen
- 10-30% ist normales Gespräch - nah an dem Sensor - oder Musik, die mit einer „normalen“ Zimmerlautstärke gespielt wird
- 30-100% ist das Schreien von Menschen oder Musik, die mit hoher Lautstärke gespielt wird.

Um ihn als Geräuschsensor zu definieren benutzt man folgende Anweisung:

```
SetSensorSound(IN_4); // Setzt IN_4 als Geräuschsensor
```

Um die Werte des Sensors abzurufen geht man folgendermaßen vor:

```
tensensor = Sensor(IN_4); // liest den Geräuschwert von IN_4 aus
```

Die Variable „tensensor“ ist vorher zu erstellen.

Bsp.:

```
#define LAUTSTAERKE 40
```

```
#define TONSENSOR SENSOR_4
```

```
task main()
```

```
{
```

```
SetSensorSound(IN_4);
```

```
    while(true){
```

```
        until(TONSENSOR > LAUTSTAERKE);
```

```
        OnFwd(OUT_AC, 50);
```

```
        Wait(500);
```

```
        until(TONSENSOR > LAUTSTAERKE);
```

```
        Off(OUT_AC);
```

```
        Wait(500);
```

```
    }
```

```
}
```

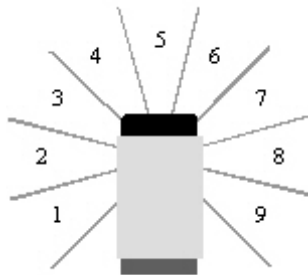
- Aufgaben:**
1. Nutze den NXT als Messgerät. Lege eine Tabelle mit verschiedenen Geräuschen und der Lautstärke in dB.
 2. Schreibe ein Programm wobei der Roboter 5 Sekunden lang vorwärts fahren soll, wenn ein Mal laut geklatscht wird.

Auflösung Programm Tastsensor Seite 42

```
task main ()
{
  SetSensor(IN_3, Sensor_TOUCH); //oder SetSensorTouch(IN_3)
  OnFwd(OUT_AC, 50);
  Until (Sensor_3 == 1);
  Off(OUT_AC);
}
```

SENSOREN – ANDERE SENSOREN

Die neuen Sensortypen findet man in der BricxCC Oberfläche unter dem Punkt DIRECT CONTROLLER.



Am Beispiel des **Infrarot-Sucher Sensor** von der Firma Hitechnic soll gezeigt werden, wie andere Sensoren angesprochen werden. Der HiTechnic-Infrarotdetektor ermöglicht das Registrieren und Orten von Infrarot-Lichtquellen.



Um ihn als Sensor zu definieren benutzt man folgende Anweisung:
SensorUS(IN_1)

Um den IR „Seeker“ zu testen, stecke den Anschluss in Port 1 und wähle auf dem NXT Baustein „View“, Ultraconic cm, Port 1. Wenn du den Infrarotsensor bewegst, ändern sich die Werte auf dem Display von 1-9. Auf der Grafik siehst du, welcher Bereich welchen Wert hat. 0 heißt, dass der Sensor kein Signal empfängt. Beim Roboterfußball sendet der spezielle Ball vom RoboCupJunior Infrarotstrahlen. Der Sensor kann diese empfangen. Vorsicht: Sonnenlicht beeinflusst die Werte.

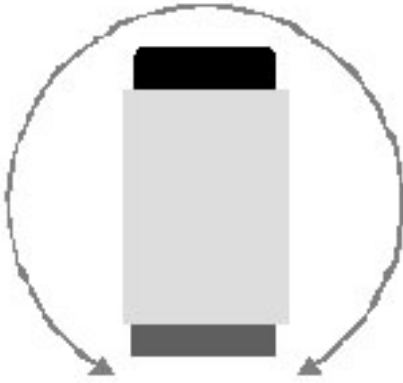
```
task main()
{
  SetSensorLowspeed(IN_1);
  OnFwd(OUT_BC, 70);

  while(true)
  {
    while(SensorUS(IN_1)<=4);
    Off(OUT_C);
    OnFwd(OUT_B, 100);
  }
}
```

```

Wait(100);
}
}

```



Der **Gyro-Sensor** ist von der Art her wie der Schall-Sensor. Er gibt einen Wert zwischen -360 und $+360^\circ$ aus. Dieser Wert wird alle $0,1$ Sekunden aktualisiert. Er gibt aber nicht die Neigung an sondern die Drehgeschwindigkeit. Dieser Wert gibt nur an, um wie viel $^\circ$ sich der Sensor in 1 Sekunde dreht. Wenn dieser Wert also bei 0° , in der nächsten Messung bei 50° und dann wieder bei 0° liegt, dann hat sich der NXT um $50^\circ/10$ (Wegen 10 Messungen pro Sekunde) um 5° gedreht.

Ein Gyrosensor wird so eingesetzt:

Bsp.:

```

#define GYRO_OFFSET 4

task main()
{
  SetSensorHTGyro(S1);
  until(false)
  {
    NumOut(0, LCD_LINE1, SensorHTGyro(S1,GYRO_OFFSET), true);
    Wait(100);
  }
}

```

Angaben zum Einsatz der Sensoren finden sich auf der Herstellerseite.

Problem: Der Sensor baut immer einen kleinen Messfehler ein. Dieser liegt bei ca $+5$. Da das aber nur ca. gesagt ist kann man ihn nicht komplett ausblenden. Das ist schade.

Der Kompass-Sensor

Der NXT Kompass-Sensor ist ein digitaler Kompass, der das magnetische Erdfeld misst und einen Wert ausgibt, der die gegenwärtige Orientierung darstellt. Die magnetische Orientierung wird zum nächsten 1° errechnet und als Zahl von 0 bis 359 zurück gegeben.

Es ist aber darauf zu achten, dass er möglichst eben eingebaut wird, sonst ergeben sich unbrauchbare Werte.

Bsp.:

```
task main()
{
  SetSensorLowSpeed(S2);
  until(false)
  {
    NumOut(0, LCD_LINE1, SensorHTCompass(S2), true);
    Wait(100);
  }
}
```

Der neue RGB Farb- und Lichtsensor

Der RGB Farbsensor kann drei Funktionen erfüllen: Als Farbsensor kann er zwischen sechs Farben unterscheiden, als Lichtsensor bestimmt er die Helligkeit von direktem Licht oder Umgebungslicht, und als Farblampe leuchtet er rot, grün oder blau.

Wie bei jedem anderen Sensor muss erst festgelegt werden, an welchem Eingang der Farbsensor angeschlossen ist. Dazu dient der Befehl:

```
SetSensorColorFull(IN_1); // Sensor ist an 1 angeschlossen
```

Der Wert des Sensor kann dann wie beim Tastsensor abgerufen werden:

```
int x = SENSOR(IN_1);
```

x kann dabei folgende Werte („Farben“) annehmen:

1 = Schwarz, 2 = Blau, 3= Grün, 4= Gelb, 5= Rot,

6= Weiß

Es können die drei LEDs des Farbsensors auch einzeln angesprochen werden und somit verschiedene



Farben erzeugt werden. Die Befehle zum ansteuern der LEDs lauten:

```
SetSensorColorRed(IN_1); //rot
SetSensorColorGreen(IN_1); //grün
SetSensorColorBlue(IN_1); //blau
SetSensorColorNone(IN_1); //alle aus!
```

Bsp.:

```
#include "NXCDefs.h"

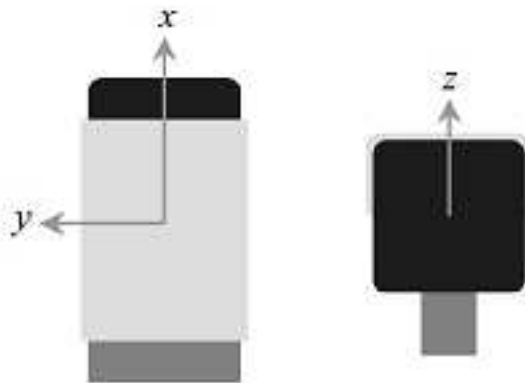
task main()
{
  // Sensor
  SetSensorColorFull(IN_1);
  while(true)
  {
    while(SENSOR_1 == 2) //blau
    {
      OnFwd(OUT_AC, 80);
    }
    while(SENSOR_1 == 5) //rot
    {
      OnRev(OUT_AC, 80);
    }
  }
}
```

(nach Daniel Braun)

Beschleunigungs- bzw. Tilt Sensor

Der HiTechnic Beschleunigungs- bzw. Tilt Sensor misst die Beschleunigung in drei Achsen. Es misst auch kippen entlang jeder Achse. Mit dem Sensor kann man die Beschleunigung des Roboters im Bereich -2g bis + 2g mit einer Skalierung von etwa 200 Zählungen pro g messen. Die Beschleunigung Messung für jede Achse wird etwa 100 mal pro Sekunde aktualisiert.

Die drei Achsen der Messung markiert sind x , y und z wie abgebildet.



Beispiele:

- Beim freien Fall im Schwerfeld der Erde beträgt die Beschleunigung $g = 9,81 \text{ m/s}^2$
- Auf der Achterbahn Silverstar im Europa-Park herrschen Vertikalbeschleunigungen von bis zu $4 g$ (40 m/s^2).

Der Beschleunigungssensor kann auch zur Neige in drei Achsen gemessen werden. Dies ist möglich, weil die Schwerkraft als Beschleunigung wahrgenommen wird.

Die aktuelle Version von NXC enthält eine API-Funktion zum Auslesen der HiTechnic Beschleunigungssensors.

```
bool ReadSensorHTAccel (const Byte-Port, int & x, int & y, int & z)
```

```
// HiTechnic Beschleunigungs-Sensor angeschlossen an Port 1
```

```
task main()
{
  int x,y,z;

  SetSensorLowspeed ( S1 );
  Wait ( 50 );
```

```

while ( true )
{
  ReadSensorHTAccel ( S1 , x, y, z);
  TextOut ( 0 , LCD_LINE1 , "x: " );
  NumOut ( 6 * 2 , LCD_LINE1 , x);
  TextOut ( 0 , LCD_LINE2 , "y: " );
  NumOut ( 6 * 2 , LCD_LINE2 , y);
  TextOut ( 0 , LCD_LINE3 , "z: " );
  NumOut ( 6 * 2 , LCD_LINE3 , z);

  Wait ( 100 );
}
}

```



Winkelsensor

Die HiTechnic Winkelsensor wird mit einer Standard LEGO Technik Kreuzachse verbunden und misst drei Rotation einer Achse besitzenden.

Diese sind

- Absolute Winkel: der Drehwinkel einer Achse von 0 Grad bis 359 Grad, Genauigkeit 1 Grad.
- Kumulierte Winkel: die kumulierte Anzahl der Grade die die Achse gemacht hat.
- Umdrehungen pro Minute (RPM): Die aktuelle Schätzung der Drehrate von 1 U / min bis 1.000 U / min.

Der Winkelsensor ist ideal für den Einbau in komplexe Modelle, wo die Winkel oder die Drehzahlüberwachung gewünscht ist. Z.B. eine Wetterstation (Windfahne) oder ein Messrad.

Beispielprogramm von Hitechnic:

```

//=====
// HiTechnic - Angle Sensor Sample Program
//
#define ANGLE S1
//=====
// ReadSensorHTAngle(port, Angle, AccAngle, RPM)
// Reads the HiTechnic Angle Sensor and returns the current:
// Angle      degrees (0-359)
// Accumulated Angle degrees (-2147483648 to 2147483647)
// RPM        rotations per minute (-1000 to 1000)

```

```

void ReadSensorHTAngle(int port, int &Angle, long &AccAngle, int &RPM)
{
    int count;
    byte cmndbuf[] = {0x02, 0x42}; // I2C device, register address
    byte respbuf[]; // Response Buffer
    bool fSuccess;
    count=8; // 8 bytes to read
    fSuccess = I2CBytes(port, cmndbuf, count, respbuf);
    if (fSuccess) {
        Angle = respbuf[0]*2 + respbuf[1];
        AccAngle = respbuf[2]*0x1000000 + respbuf[3]*0x10000+
            respbuf[4]*0x100 + respbuf[5];
        RPM = respbuf[6]*0x100 + respbuf[7];
    } else {
        // No data from sensor
        Angle = 0; AccAngle = 0; RPM = 0;
    }
}

// ReserSensorHTAnalog(port, resetmode)
// resetmode:
// HTANGLE_MODE_CALIBRATE Calibrate the zero position of angle.
// Zero position is saved in EEPROM on sensor.
// HTANGLE_MODE_RESET Reset the rotation count of accumulated.
// angle to zero. Not saved in EEPROM.
#define HTANGLE_MODE_CALIBRATE 0x43
#define HTANGLE_MODE_RESET 0x52
void ResetSensorHTAngle(int port, int resetmode)
{
    int count;
    byte cmndbuf[] = {0x02, 0x41, 0}; // I2C device, register address
    byte respbuf[]; // buffer for inbound I2C response
    cmndbuf[2] = resetmode; // Set reset code
    count=0; // 0 bytes to read
    I2CBytes(port, cmndbuf, count, respbuf);
    if (resetmode == HTANGLE_MODE_CALIBRATE)
        Wait(50); // Time to allow burning EEPROM
}

//=====
task main()
{
    int angle;
    long acc_angle;
    int rpm;

    SetSensorLowSpeed(ANGLE);
    Wait(100);

    TextOut(0, LCD_LINE1, "HiTechnic");
    TextOut(0, LCD_LINE2, " Angle Sensor");
    TextOut(0, LCD_LINE8, "");
}

```

```

while(true) {
  Wait(100);

  ReadSensorHTAngle(ANGLE, angle, acc_angle, rpm);
  TextOut(0, LCD_LINE4, "Angle :   ");
  NumOut(7*6, LCD_LINE4, angle);

  TextOut(0, LCD_LINE5, "AccAng:   ");
  NumOut(7*6, LCD_LINE5, acc_angle);

  TextOut(0, LCD_LINE6, "RPM :   ");
  NumOut(7*6, LCD_LINE6, rpm);

  if (ButtonPressed(BTNLEFT,0)) {
    while(ButtonPressed(BTNLEFT,0));
    ResetSensorHTAngle(ANGLE, HTANGLE_MODE_RESET);
  }
  if (ButtonPressed(BTNRIGHT,0)) {
    while(ButtonPressed(BTNRIGHT,0));
    ResetSensorHTAngle(ANGLE, HTANGLE_MODE_CALIBRATE);
  }
}
}
}

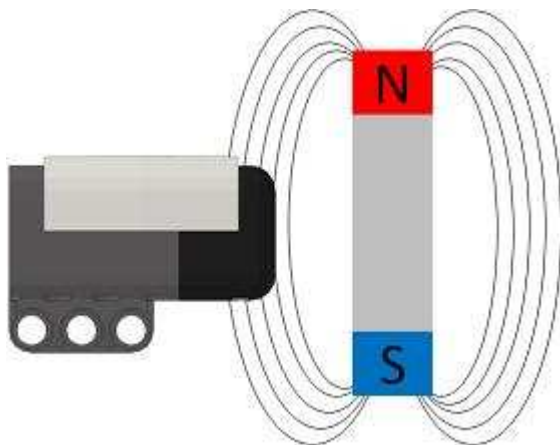
```

<http://www.hitechnic.com>

Magnetsensor

Der NXT Magnetsensor ermöglicht es mit dem Roboter Magnetfelder zu finden. Der Sensor erkennt Magnetfelder, die auf der ganzen Vorderseite des Sensors in einer vertikalen Ausrichtung sind.

Siehe Abbildung



Wenn die Ausrichtung des Magnetfeldes „Norden“ wie in der Abbildung oben ist, wird das Magnetfeld erkannt. Wird der Magnet seitlich gestellt, kann es nicht nachgewiesen werden.

Der Sensor kann bis zu etwa 300 mal pro Sekunde gelesen werden.

Test: SchlieÙe den Sensor an Ausgang 1 an und drücke „View – Ambient light – Port 1“ auf dem NXT. Halte einen Magneten vor den Sensor und beobachte, was passiert.

Der Magnetsensor hat das gleiche Interface wie der Gyro-Sensor.

```
#define SensorHTMagnet(port,offset) SensorHTGyro(port,offset)
#define SetSensorHTMagnet(port)    SetSensorHTGyro(port)

#define MAGNET IN_1

task main()
{
    int offset, magnetic_value;

    SetSensorHTMagnet(MAGNET);

    TextOut(0, LCD_LINE1, "HiTechnic");
    TextOut(0, LCD_LINE2, " Magnetic Sensor");

    TextOut(0, LCD_LINE4, "Calibrating, ");
    TextOut(0, LCD_LINE5, "keep magnets");
    TextOut(0, LCD_LINE6, "away...");
    Wait(1000);
    //Get initial offset assuming no magnetic field is present.
    offset = SensorHTMagnet(MAGNET, 0);
    TextOut(0, LCD_LINE4, " Value: ");
    TextOut(0, LCD_LINE5, " ");
    TextOut(0, LCD_LINE6, " ");

    while(true) {
        magnetic_value = SensorHTMagnet(MAGNET, offset);

        TextOut(6*7, LCD_LINE5, " ");
        NumOut(6*7,LCD_LINE5, magnetic_value);

        Wait(100);
    }
}
```

EOPD Sensor

HiTechnic-Abstandssensor, EOPD-Sensor (Electro Optical Proximity Detector)

Dieser optisch-elektronische Sensor kann Objekte bis zu einer Entfernung von 20cm sowie kleine Distanzänderungen präzise erkennen. Damit schließt er die Lücke zwischen dem Ultraschall- und dem Berührungssensor. Durch sein pulsiertes Lichtsignal können externe Lichteinflüsse ausgeblendet werden.



In zwei Modi kann er auch auf helle und dunkle Objekte justiert werden.

Der Sensor kann mehr als 300 mal pro Sekunde eingelesen werden und reagiert schnell auf Änderungen.

Jeder EOPD läuft auf einer leicht anderen Samplingrate sodass es auch untereinander kaum Interferenzen gibt.

Der EOPD Sensor ist ein analoger Sensor wie die LEGO Lichtsensor

```
# Define EOPD IN_3 //Sensor in Port 3
```

```
task main()
{
  int eopd;
  ...
```

Auszug für die Werteausgabe auf dem NXT:

```
...
while(true) {
  // Read EOPD Sensor
  eopd = 1023-SensorRaw(EOPD);

  TextOut(0, LCD_LINE4, "eopd: ");
  NumOut(6*5, LCD_LINE4, eopd);

  iSteer = EOPD_TARGET - eopd;
  ...
```

MIT MEHREREN TASKS ARBEITEN

Bisher bestanden alle bisherigen Programme aus genau einem TASK („task main()“). Ein NXC Programm kann viele Task (engl. = Aufgabe) haben – bis zu 255. Um ein Programm ausführen zu können, muss der Task „main“ eingesetzt werden. Im „task main“ können die anderen Aufgaben aufgerufen werden. Dieses erfolgt mit dem Aufruf „Precedes()“ – alle Aufgaben werden gleichzeitig ausgeführt – oder „Follows(task1, task2...)“.

Bsp. mit einem Infrarotsensor (IR Seeker) an Port 1:

```
task ballDa() // task 1
```

```
{
  while(true)
  {
    OnFwd(OUT_BC, 100);
    while(SensorUS(IN_1)==5);
    Off(OUT_BC);
    OnFwd(OUT_BC, 70);
    Wait(5000);
  }
}
```

```
task ballSuchen() // task 2
```

```
{
  while(true)
  {
    while(SensorUS(IN_1)<=4);
    Off(OUT_C);
    OnFwd(OUT_B, 100);
    Wait(100);
  }
}
```

```
task main()
```

```
{
  SetSensorLowspeed(IN_1);
  OnFwd(OUT_BC, 70);
  Precedes(ballDa, ballSuchen); // task 1 und task 2 zusammen
}
```

Sobald ein laufender Task das Ende seines Programmcodes erreicht, gilt er als beendet. Das gesamte Programm ist beendet, wenn der letzte Task beendet wurde. Das Programm kann man auch durch „StopAllTasks()“ beenden. **Stop(true)** beendet das Programm in jedem Fall. Dabei werden alle laufenden Tasks abgebrochen.

Jeder Task kann andere Tasks starten und Funktionen aufrufen. Damit sich gleichzeitig laufende sich nicht in die Quere kommen, sollte beim Zugriff auf kritische Funktionen oder Variable der sog. Mutex-Mechanismus (MUTual EXclusion, gegenseitiger Ausschluss) verwendet werden. Ein Mutex wird wie eine *globale* Variable definiert und innerhalb eines Tasks oder einer Funktion folgendermaßen verwendet:

mutex *irgendeinName*; global definierter Mutex

task *name*()

{

Acquire(*irgendeinName*); reserviert diesen Mutex, bzw. wartet zunächst,
bis er von einem anderen Task freigegeben wird

// kritisches Codesegment

Release(*irgendeinName*); gibt diesen Mutex wieder frei, damit andere Tasks
kritischen Code ausführen können

}

Bsp.:

mutex bewegungM;

task *quadrat*()

{

while (**true**)

 {

 Acquire(bewegungM);

 OnFwd(OUT_AC, 75); Wait(1000);

 OnRev(OUT_C, 75); Wait(500);

 Release(bewegungM);

 }

}

task *sensoren_pruefen*()

{

while (**true**)

 {

if (SENSOR_1 == 1)

 {

 Acquire(bewegungM);

 OnRev(OUT_AC, 75); Wait(500);

 OnFwd(OUT_A, 75); Wait(500);

 Release(bewegungM);

 }

 }

}

task *main*()

{

 Precedes(*quadrat*, *sensoren_pruefen*);

 SetSensor(IN_1);

}

Aufgaben: 1. Schreibe beide Programme ab und ändere die Werte. Beobachte was passiert.

2. Baue verschiedene Sensoren in das Programm mit ein.

SUBROUTINEN**Unterprogramme**

Manchmal benötigt man exakt dieselbe Abfolge von Programmschritten an mehreren Stellen eines Programms. Man kann sie nun mehrmals schreiben oder einfach als Unterprogramm von der gewünschten Stelle aus aufrufen.

Beispiel:

```

sub turn_around( int pwr )    //definiert das Unterprogramm turn_around(),
{                               //welches das Ausweichmanöver durchführen
                               //soll mit dem integer für „power“ = Stärke
    OnRev(OUT_C, pwr); Wait(700);
    OnFwd(OUT_AC, pwr);
}

task main()
{
    OnFwd(OUT_AC, 75);
    Wait(1000);
    turn_around(65);           //1. von hier aus wird turn_around() aufgerufen
    Wait(2000);
    turn_around(65);           // 2.
    Wait(1000);
    turn_around(65);           //3.
    Off(OUT_AC);
}

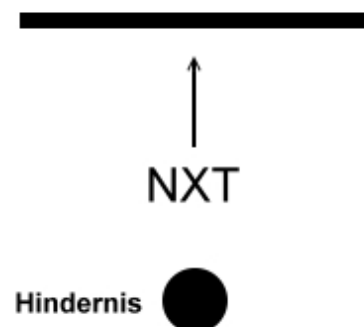
```

In diesem Programm wurde ein Unterprogramm definiert, welches den Roboter auf der Stelle drehen lässt, indem der Motor C in den Rückwärtsgang (Wait(200)) geschaltet wird, während sich der Motor A noch vorwärts dreht. Die **task** main () ruft das Unterprogramm turn_around() (turn = drehen, around = herum) dreimal auf.

Der Vorteil von Unterprogrammen (sub) ist, dass sie nur einmal im NXT gespeichert werden, was viel Speicherplatz spart.

Wichtig zu wissen ist es, dass Unterprogramme nicht von anderen Unterprogrammen aus aufgerufen werden können aber Unterprogramme von verschiedenen Tasks aufgerufen werden können. Das ist aber nicht ratsam, weil dann dasselbe Unterprogramm vielleicht gleichzeitig von mehreren Tasks aufgerufen wird, was zu unvorhersehbaren Reaktionen des Roboters führt.

Aufgabe: Der Roboter soll bis zu einer schwarzen Linie fahren, 1 Sekunde rückwärts fahren, sich dann um 180° drehen, bis zum Hindernis fahren und stehen bleiben. Variiere das Programm. Suche die passenden Sensoren und arbeite mit mehreren Tasks oder Unterprogrammen.



AUFGABENBLATT

1. Schreibe ein Programm das deinen Roboter 2 Sekunden vorwärts fahren lässt, dann 2 Sekunden stehen bleibt und wieder zurückfährt.
2. Der Roboter soll vorwärtsfahren und dabei „beschleunigen“ (immer schneller werden). Nach Erreichen der Spitzengeschwindigkeit soll der Roboter allmählich wieder langsamer werden.
3. Finde heraus, mit welchen Parametern der Roboter um 90° , 45° oder 360° gedreht werden kann.
4. Lasse deinen Roboter ein geschlossenes Sechseck durchfahren. Benutze hierzu geeignete Schleifen.
5. Lasse deinen Roboter eine immer größer werdende Spirale fahren.
6. Der Roboter soll 2 Sekunden exakt geradeaus fahren und dann zufallsgesteuert entweder um 90° nach links oder rechts fahren.
7. Der Roboter soll geradeaus fahren, bis er auf einen Gegenstand trifft und dann stoppen.
8. Der Roboter soll selbstständig durch einen Raum navigieren und dabei Gegenständen ausweichen.
9. Der Roboter soll durch einen Raum fahren, ohne irgendwo gegen zu fahren. Bei einem lauten Geräusch soll er stoppen und sich um 180 Grad drehen und die Prozedur soll von vorne beginnen (Schleife).
10. Der Roboter soll einer schwarzen Linie folgen.
11. Schreibe ein Programm, dass dein Roboter nicht vom Tisch fallen kann.
12. Der Roboter soll einen Gegenstand kreisförmig Umfahren.

BLUETOOTH

Bluetooth ist ein in den 1990er Jahren ursprünglich von Ericsson entwickelter Industriestandard für die Funkvernetzung von Geräten über kurze Distanz. Bluetooth bildet dabei die Schnittstelle, über die sowohl mobile Kleingeräte wie Mobiltelefone und PDAs als auch Computer und Peripheriegeräte (wie in unserem Fall der NXT) miteinander kommunizieren können. Hauptzweck von Bluetooth ist das Ersetzen von Kabelverbindungen zwischen Geräten.

Bluetooth ermöglicht dem NXT-Stein mit allen Geräten die Bluetooth-fähig sind, kabellos zu kommunizieren. Ausschließlich die neue MINDSTORMS Education Software wird Dank Bluetooth in der Lage sein mit mehreren NXT-Geräten gleichzeitig zu kommunizieren.

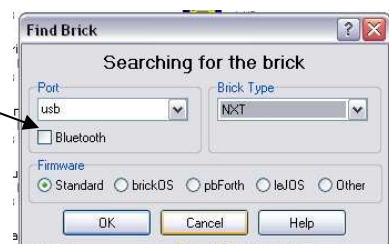
Ob Class I oder Class II ist ziemlich egal - auch wenn der NXT nur Class II hergibt. Ansonsten Bluetooth Stack auf dem PC installieren, Bluetooth-Stick in den USB Port stecken.

Es können nur Bluetooth-Verbindungen genutzt werden, die in der Datei `nxt.dat` im Programmverzeichnis von BricxCC stehen. Dort steht für jede Brick eine Zeile wie:
`BTH::HANSA11=BTH::HANSA11::00:16:53:05:92:90::11`

Im vorderen Teil steht der Bezeichner BTH für Bluetooth und der Name der Brick auf. Im hinteren Teil ist die ID der Brick zu sehen. Die Einträge in der `nxt.dat` lassen sich notfalls mit einem Editor erstellen. Die ID kann man auf der Brick finden unter *Setting -> NXT Version*,

Einfacher ist es die Datei zu löschen, den Bluetooth-Adapter zu aktivieren und einmal die Verbindung z.B. mittels der LEGO-Software aufzubauen. Wenn man dann BricxCC neu startet wird die Datei passend neu erstellt, man muss dann nur im *Find Brick* Dialog den richtigen Port auswählen. (Quelle: http://debacher.com/wiki/Lego_Mindstorms)

Programm starten und Bluetooth wählen.



Bluetooth und NXT funktioniert nicht mit jedem Handy. Auf www.lego.com gibt es ein Programm zum herunterladen und eine Liste mit den kompatiblen Handys.

Daniele Benedettelli hat eine Bibliothek geschrieben, die das Verwenden von Bluetooth mit NXC stark vereinfacht.

Um Bluetooth in NXC nutzen zu können, kann man sich unter http://daniele.benedettelli.com/BT_NXC.htm die „NXC Bluetooth library for Mindstorms NXT (BtLib.zip)“ herunterladen und in das Verzeichnis von BricxCC entpacken.

Dort gibt es auch zwei Programmbeispiele.

Sind mehrere NXT vorhanden können ohne einen diese über Bluetooth kommunizieren.

Wenn nicht, funktioniert ein Austausch auch zwischen NXT und dem PC.

Zuerst müssen zwei oder mehr NXTs (bzw. NXT mit PC) miteinander (drahtlos)

verbunden werden. Dies erfolgt über das NXT-Menü. Nur wenn die NXTs über das NXT-Menü verbunden wurden lassen sich Daten zwischen Ihnen austauschen.

Der NXT, der Daten senden soll wird Master genannt. Der NXT der diese empfängt, Slave. Ein

Master kann mit bis zu 3 Slaves gleichzeitig Verbunden sein. Hierfür stehen die Kanäle 1,2

und 3 zur Verfügung. Der Slave wird immer auf Kanal 0 mit dem Master verbunden.

Zusätzlich können die versendeten Nachrichten an 10 verschiedene Mailboxen (Briefkästen) adressiert werden.

(BT_CONN (constant)) kontrolliert, ob der Slave korrekt auf Kanal 1 mit dem Master verbunden ist.

Mit der Funktion: **BluetoothStatus(conn)** wird die Nachricht „erstellt“ und versendet mit: **SendRemoteString(conn,queue,string)**.

Eingehende Nachrichten vom Slave werden mit: **ReceiveRemoteString(queue,clear,string)** empfangen und auf dem Display ausgegeben.

Sollte eines der Programme beendet oder der NXT ausgeschaltet werden, gehen die Nachrichten des anderen verloren.

Um die Bibliothek zu nutzen, muss die Bibliothek eingebunden werden:

```
#include " NXCDefs.h"
```

```
#include " BTlib.nxc"
```

```
//MASTER
#define BT_CONN 1
#define INBOX 1
#define OUTBOX 5

sub BTCheck(int conn)
{
    if (!BluetoothStatus(conn)==NO_ERR)
    {
        TextOut(5,LCD_LINE2,"Error");
        Wait(1000);
        Stop(true);
    }
}
```

```

task main()
{
    string in, out, iStr;
    int i = 0;
    BTCheck(BT_CONN); //überprüfe die Verbindung
    while(true)
    {
        iStr = NumToStr(i);
        out = StrCat("M",iStr);
        TextOut(10,LCD_LINE1,"Master Test");
        TextOut(0,LCD_LINE2,"IN:");
        TextOut(0,LCD_LINE4,"OUT:");
        ReceiveRemoteString(INBOX, true, in);
        SendRemoteString(BT_CONN,OUTBOX,out);
        TextOut(10,LCD_LINE3,in);
        TextOut(10,LCD_LINE5,out);
        Wait(100);
        i++; //Wert hochzählen
    }
}

```

Das Programm des Slaves ist ähnlich dem des Masters, mit der Ausnahme, dass es die `SendResponseString(queue,string)` nutzt, anstatt des `SendRemoteString` Befehls, da der Slave nur an den Master auf Kanal 0 – Nachrichten schicken kann.

```

//SLAVE
#define BT_CONN 1
#define INBOX 5
#define OUTBOX 1

sub BTCheck(int conn)
{
    if (!BluetoothStatus(conn)==NO_ERR)
    {
        TextOut(5,LCD_LINE2,"Error");
        Wait(1000);
        Stop(true);
    }
}

task main()
{
    string in, out, iStr;
    int i = 0;
    BTCheck(0); //check master connection
    while(true)
    {

```

```

    iStr = NumToStr(i);
    out = StrCat("S",iStr);
    TextOut(10,LCD_LINE1,"Slave Test");
    TextOut(0,LCD_LINE2,"IN:");
    TextOut(0,LCD_LINE4,"OUT:");
    ReceiveRemoteString(INBOX, true, in);
    SendResponseString(OUTBOX,out);
    TextOut(10,LCD_LINE3,in);
    TextOut(10,LCD_LINE5,out);
    Wait(100);
    i++;
  }
}

```

(nach Benedetelli)

```

// Hauptprogramm um die Bibliothek BTLib zu testen
#include "NXCDefs.h"
#include "BTLib.nxc"
#define BT_CONN 1 //Pfad wo der Slave verbunden ist
#define MAILBOX 0

sub masterBTCheck(int conn)
{
  string cStr;
  cStr = NumToStr(conn);
  cStr = StrCat("on line",cStr);
  if (!BTCommCheck(conn))
  {
    TextOut(5,LCD_LINE2,true,"*Connect Slave");
    TextOut(5,LCD_LINE3,false,cStr);
    TextOut(60,LCD_LINE3,false,"from");
    TextOut(5,LCD_LINE4,false,"this NXT menu");
    Wait(3000);
    Stop(true);
  }
}

task main()
{
  string in, out, iStr;
  int i = 0;
  masterBTCheck(BT_CONN);
  TextOut(10,LCD_LINE1,true,"Master Test");
  TextOut(5,LCD_LINE2,false,"IN:");
  TextOut(5,LCD_LINE4,false,"OUT:");
}

```

```

while(true)
{
    iStr = NumToStr(i);
    out = StrCat("M",iStr);
    in = BTReceiveMessage(BT_CONN, MAILBOX, TRUE);
    TextOut(5,LCD_LINE3,false,"");
    TextOut(5,LCD_LINE3,false,in);
    BTRSendMessage(BT_CONN,MAILBOX,out);
    TextOut(5,LCD_LINE5,false,"");
    TextOut(5,LCD_LINE5,false,out);
    Wait(100);           //Pause (muss nicht sein)
    i++;
}
}

```

PROGRAMM MIT FEHLERN

Aufgabe:

Schreibe das Programm ab und korrigiere die 4 Fehler.

Um das Programm zu testen brauchst du einen Roboter mit einem Lichtsensor. Der Roboter soll einer Linie folgen.

```

#define SpeedSlow 50
#define SpeedFast 100

int SV;
int LoopCount;

sub FollowLine(int loopTime)
{
    int Threshold1=600;
    int Threshold2=630;
    int theSpeed;
    long t

    // Sensortypen und Modus einstellen
    SetSensorType(IN_3, IN_TYPE_LIGHT_ACTIVE);
    SetSensorMode(IN_3, IN_MODE_RAW);

    // Wiederholung starten
    t = CurrentTick() + loopTime;
    while (t > CurrentTick())

        // den Wert des Lichtsensors lesen
        SV = SensorRaw(IN_3);

```

```
// Geschwindigkeit für den Motor 1 setzen
if (SV < Threshold2)
    OnFwd(OUT_A, SpeedFast);
else
    OnFwd(OUT_A, SpeedSlow);

// Geschwindigkeit für den Motor 2 setzen
if (SV > Threshold1)
    OnFwd(OUT_B, SpeedFast);
else
    OnFwd(OUT_B, SpeedSlow);

// Sensorenwert auf das Display ausgeben
// NumOut(0, LCD_LINE1, false, SV);

    LoopCount++;
}
// 10 Sekunden Wiederholung ist gemacht
return;
}

task main()
{
    string lcStr;
    string svStr;
    string msg;

    // Subroutine aufrufen
    FollowLine(10000);

    // output Ergebnisse
    lcStr = NumToStr(LoopCount);
    svStr = NumToStr(SV);
    msg = svStr + " - " + lcStr;
    TextOut(0, LCD_LINE1, msg);

    // beide Motoren aus
    Off(OUT_AB)

    // zeige dem Benutzer die letzte Nachricht
    wait(10000);
}
```

Auflösung nächste Seiten

Auflösung von Seite 63/64

```
#define SpeedSlow 50
#define SpeedFast 100

int SV;
int LoopCount;

sub FollowLine(int loopTime)
{
  int Threshold1=600;
  int Threshold2=630;
  int theSpeed;
  long t; // 1. FEHLER Semikolon fehlt

  // Sensortypen und Modus einstellen
  SetSensorType(IN_3, IN_TYPE_LIGHT_ACTIVE);
  SetSensorMode(IN_3, IN_MODE_RAW);

  // Wiederholung starten
  t = CurrentTick() + loopTime;
  while (t > CurrentTick())
  { // 2. FEHLER geschweifte Klammer fehlt
    // den Wert des Lichtsensors lesen
    SV = SensorRaw(IN_3);

    // Geschwindigkeit für den Motor 1 setzen
    if (SV < Threshold2)
      OnFwd(OUT_A, SpeedFast);
    else
      OnFwd(OUT_A, SpeedSlow);

    // Geschwindigkeit für den Motor 2 setzen
    if (SV > Threshold1)
      OnFwd(OUT_B, SpeedFast);
    else
      OnFwd(OUT_B, SpeedSlow);

    // Sensorenwert auf das Display ausgeben
    // NumOut(0, LCD_LINE1, false, SV);
```

```
LoopCount++;
}
// 10 Sekunden Wiederholung ist gemacht
return;
}

task main()
{
  string lcStr;
  string svStr;
  string msg;

  // Subroutine aufrufen
  FollowLine(10000);

  // output Ergebnisse
  lcStr = NumToStr(LoopCount);
  svStr = NumToStr(SV);
  msg = svStr + " - " + lcStr;
  TextOut(0, LCD_LINE1, msg);

  // beide Motoren aus
  Off(OUT_AB); // 3. FEHLER Semikolon fehlt

  // zeige dem Benutzer die letzte Nachricht
  Wait(10000); // 4. FEHLER "Wait" - erster Buchstabe groß
}
```

BEISPIELPROGRAMME

Kleine Beispiele zum Abtippen und testen.

1) -----

```
int a=40;

task main ()
{
  TextOut(0, 60, "Test");
  Wait(2000);
  NumOut(a/2, a-10, a);
  Wait(10000);
}
```

2) -----

```
int a;
task main ()
{
  SetSensorLowspeed (S1);
  while (true)
  {
    a = SensorUS (S1);
    ClearScreen();
    TextOut (0, 20, "x-tilt");
    NumOut (20, 10, a);
  }
}
```

3) -----

```
int a = 50;

void George(int y, byte b, string msg)
{
  TextOut(10, y*b, msg);
  y = y*b+a;
  SetSoundFrequency(y);
}

task main()
{
  string foo = "Test";
  int a=10;
  byte c=3;
  George(a, c, foo);
}
```

NXC- BEFEHLE IM ÜBERBLICK

Hier siehst du eine Liste aller Statements, Bedingungen, Befehle und Ausdrücke.

Statements sind Schlüsselworte, die ein Programm anweisen etwas auszuführen.

Statement	Beschreibung
while (Bedingung) Anweisung	Führe <i>Anweisung</i> nie oder solange aus, wie die <i>Bedingung</i> zutrifft
do Anweisung while (Bedingung)	Führe <i>Anweisung</i> einmal oder solange aus, wie die <i>Bedingung</i> zutrifft
until (Bedingung) Anweisung	Führe <i>Anweisung</i> nie oder solange aus, bis die <i>Bedingung</i> zutrifft
repeat (Wert) Anweisung	Wiederhole die <i>Anweisung</i> entsprechend dem Wert in Klammern
if (Bedingung) Anweisung 1	Führe <i>Anweisung 1</i> aus, wenn die <i>Bedingung</i> zutrifft
if (Bedingung) Anweisung 1 else Anweisung 2	Führe <i>Anweisung 1</i> aus, wenn die <i>Bedingung</i> zutrifft, andernfalls führe <i>Anweisung 2</i> aus
start task_name;	Programm/Task starten
stop task_name;	Programm/Task stoppen
return	Kehre zur aufrufenden Stelle zurück

Bezeichner werden für Variable, Aufgabe, Funktion und Subroutinenamen benutzt. Der erste Buchstabe eines Bezeichners muss ein Groß- oder Kleinbuchstabe oder der Unterstrich sein ('_'). Restliche Zeichen können Buchstaben, Zahlen und ein Unterstrich sein. Einige mögliche Bezeichner sind für den Gebrauch in der NXC Sprache selbst reserviert. Diese reservierten Worte sind Anrufschlüsselwörter und können möglicherweise nicht als Bezeichner verwendet werden. Eine komplette Liste von Schlüsselwörtern:

<code>__RETURN__</code>	<code>false</code>	<code>safecall</code>
<code>__STRRETVAL__</code>	<code>for</code>	<code>short</code>
<code>__RETVAL__</code>	<code>goto</code>	<code>sign</code>
<code>__TMPBYTE__</code>	<code>if</code>	<code>start</code>
<code>__TMPWORD__</code>	<code>inline</code>	<code>stop</code>
<code>__TMPLONG__</code>	<code>int</code>	<code>string</code>
	<code>long</code>	<code>struct</code>
<code>abs</code>	<code>switch</code>	
<code>asm</code>	<code>task</code>	<code>case</code>
<code>bool</code>	<code>true</code>	<code>char</code>
<code>break</code>	<code>typedef</code>	<code>void</code>
<code>byte</code>	<code>unsigned</code>	<code>while</code>
<code>const</code>	<code>until</code>	<code>repeat</code>
<code>continue</code>	<code>priority</code>	<code>return</code>
<code>default</code>	<code>mutex</code>	<code>sub</code>
<code>do</code>		
<code>else</code>		

Variablen

Alle Variablen in NXC sind von den folgenden Arten:

Type Name	Information
bool	8 bit unsigned
byte, unsigned char	8 bit unsigned
unsigned int	16 bit unsigned
unsigned long	32 bit unsigned
long	32 bit signed
short, int	16 bit signed
char	8 bit signed
mutex	Spezielle Art benutzt für exklusiven Codezugang
string	Array von byte
struct	Benutzer definiertes structure types
Arrays	Arrays irgendeiner Art

„signed“ bedeutet, dass der Datentyp *char* wie der numerische Bereich -128..127 behandelt wird, das ASCII-Zeichen 255 wird also (numerisch) als -1 interpretiert; demgegenüber bedeutet „unsigned“, dass „char“ wie der Bereich 0..255 verwendet wird, das ASCII-Zeichen 255 wird also auch numerisch als 255 interpretiert.

Anweisungen

Es gibt neun verschiedene Zuweisungsoperatoren. Der grundlegendste Operator, '=', weist einfach den Wert des Ausdrucks der Variablen zu. Die anderen Operatoren ändern den Variablen-Wert auf irgendeine andere Art wie die Tabelle zeigt.

Operator	Tätigkeit
=	Weist der Variablen einen Ausdruck zu
+=	Addiere den Ausdruck zur Variablen hinzu
-=	Subtrahiere den Ausdruck zur Variablen
*=	Multipliziere die Variable durch den Ausdruck
/=	Dividiert die Variable durch den Ausdruck
%=	Weist einer Variablen den Modulo-Wert zu
&=	Bitweises UND Ausdruck in die Variable

=	Bitweises ODER Ausdruck in die Variable			
^=	Nur Bitweises ODER in die Variable			
=	Setzt die Variable zu einem absoluten Wert			
+-=	Setzt die Variable zum Zeichen (- 1, +1.0) des Ausdrucks			
	<table border="1"> <tr> <td>Satzvariable zur</td> <td>bf-res</td> <td>utf8</td> </tr> </table>	Satzvariable zur	bf-res	utf8
Satzvariable zur	bf-res	utf8		

Bedingungen

Bedingung	Bedeutung
true	Immer wahr
false	Immer falsch
Ausdruck 1 == Ausdruck 2	Prüfe, ob die <i>Ausdrücke</i> gleich sind
Ausdruck 1 != Ausdruck 2	Prüfe, ob die <i>Ausdrücke</i> nicht gleich sind
Ausdruck 1 < Ausdruck 2	Prüfe ob <i>Ausdruck 1</i> kleiner als <i>Ausdruck 2</i> ist
Ausdruck 1 <= Ausdruck 2	Prüfe ob <i>Ausdruck 1</i> kleiner oder gleich <i>Ausdruck 2</i> ist
Ausdruck 1 > Ausdruck 2	Prüfe ob <i>Ausdruck 1</i> größer als <i>Ausdruck 2</i> ist
Ausdruck 1 >= Ausdruck 2	Prüfe ob <i>Ausdruck 1</i> größer oder gleich <i>Ausdruck 2</i> ist
Bedingung 1 && Bedingung 2	Logisches UND der beiden <i>Bedingungen</i> Nur dann wahr, wenn beide <i>Bedingungen</i> wahr sind
Bedingung 1 Bedingung 2	Logisches ODER der beiden <i>Bedingungen</i> Wahr, wenn eine der beiden <i>Bedingungen</i> wahr ist

Mathematische Operatoren

Operator	Beschreibung	Bezug	Einschränkung	Beispiel
abs()	Absoluter Wert	n/a		abs(x)
sign()	Vorzeichen	n/a		sign(x)
++	Addition	left	Nur Variablen	x++ oder ++x
--	Subtraktion	left	Nur Variablen	x-- oder --x
-	Vorzeichenwechsel	rechts	Nur Konstanten	-x
~	Bitweise Verneinung	rechts		~123
!	Logische Negation			!x
*	Multiplikation	links		x * y
/	Division	links		x / y
%	Modulo (ganzzahliger Rest)	links		123 % 4
+	Addition	links		x + y
-	Subtraktion	links		x - y
<<	Left shift	links		x << 4
>>	Right shift	links		x >> 4
<, >	Relationale Operatoren	links		x < y
<=, >=				
==, !=	gleich, nicht gleich	links		x == 1
&	Bitweises UND	links		x & y
^	Bitweises XOR	links		x ^ y
 	Bitweise ODER	links		x y
&&	Logisches UND	links		x && y
 	logisches ODER	links		x y
?:	bedingter Wert	n/a		x == 1 ? y : z

ROBOTER BEISPIEL – SOCCER

Der hier abgebildete Roboter stammt vom Robocup. Der Roboter besteht aus dem NXT, zwei Motoren, 1 Kompass-Sensor der Firma HiTechnic (oben rechts), 1 IR Seeker (Infrarotsensor) der Firma HiTechnic, 1 Tastsensor und einem Lichtsensor.

Der Roboter wurde mit NXC programmiert.

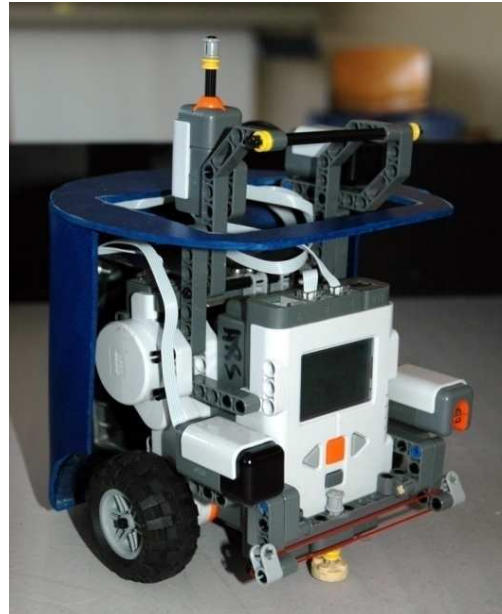
Beispielprogramm (nicht original):

```
task ballDa()
{
while(true)
{
OnFwd(OUT_BC, 100);
while(SensorUS(IN_1)==5);
Off(OUT_BC);
OnFwd(OUT_BC, 70);
Wait(5000);
}
}

task ballSuchenLinks()
{
while(true)
{
while(SensorUS(IN_1)<=4);
Off(OUT_C);
OnFwd(OUT_B, 100);
Wait(100);
}
}

task ballSuchenRechts()
{
while(true)
{
while(SensorUS(IN_1)>=6);
Off(OUT_C);
OnRev(OUT_B, 100);
Wait(100);
}
}

task main()
{
SetSensorLowspeed(IN_1);
OnFwd(OUT_BC, 70);
Precedes(ballDa, ballSuchenLinks,
ballSuchenRechts);
}
```





Das Gehäuse ist aus Plastik, geformt mit Heißluft. Die obere Platte mit Heißkleber auf die Rundung gesetzt. 2 Aluminiumwinkel verbinden das Gehäuse mit dem Roboter.

Der Taster hat die Funktion zum Starten. Das Gummiband ist der „Schussapparat“. Die Kabel stammen von der Firma www.mindsensors.com . Der Vorteil: Die Kabel sind flexibler und besser zu verlegen.

FAQ

Wo kann ich die aktuelle NXC Version herunterladen und auf meinem Computer(Betriebssystem Windows XP) erfolgreich installieren.

<http://bricxcc.sourceforge.net/nbc>

Wenn ich den Befehl so eingebe:

```
task main()
{
  OnFwd(OUT_AC, 100);
}
```

, dann laufen die Motoren ca. 1 Sekunde und das ganze Programm ist beendet.

Komisch. Normalerweise sollten sich die Motoren doch unendlich drehen?

„task main“ startet die Motoren, wartet aber nicht solange, bis sie wieder ausgeschaltet werden, sondern beendet sich direkt danach selbstständig. Damit wird dann alles beendet, auch die Motoren.

Also laufen die Motoren grundsätzlich unendlich lang, aber nur solange, wie auch der „task main“ läuft.

Was bedeutet dieses #include "NXCDefs.h"?

#include "Dateiname"
fügt den Inhalt einer Datei an der Stelle ein, wo der include Befehl verwendet wird.

Bsp.:

```
Datei "NXCDefs.h"
=====
// Definition A
// Definition B
// Definition C
=====
```

Datei 2: "MeinProgramm.nxc"

```
include "NXCDefs.h" // kopiert die folgenden 3 Zeilen aus obiger Datei
// Definition A
// Definition B
// Definition C
```

In den neuen NXC Programm Versionen ist die NXCDefs.h Datei bereits enthalten. Den Inhalt der inkludierten Datei sieht man aber als Anwender nicht.

Ich habe ein Problem mit dem US-Sensor. Ich habe folgendes Testprogramm geschrieben:

Code:

```
#include "NXCDefs.h"

task main()
{ while(TRUE){
  SetSensorLowspeed(IN_2);
  int ultra = Sensor(IN_2);
  if(ultra < 15)
  {
    OnFwd(OUT_A, 30);
  }
  else
  {
    Float(OUT_A);
  }
}
}
```

Komischerweise startet der Motor aber nicht. Ich habe auch schon unterschiedliche Werte (20, 30) gewählt und auch < durch > ersetzt... jedoch ohne Erfolg.

Wenn ich im NXT unter View den US Sensor teste, zeigt er mir jedoch die passenden cm-Werte an...

```
int ultra = SensorUS(IN_2); // so muss es heißen
```

Kann es sein, dass NXC nur bis +- 32.768 zählen/rechnen kann? Also insgesamt nur bis 65.536 ?

Das sind doch nur 16 Bit!

Mach ich was falsch, oder habe ich was übersehen mit dem Zahlenbereich?

Der NXT kann doch 32 Bit!

Er könnte doch bis 4.294.967.296 zählen/rechnen tun können

Insgesamt also, +- 2.147.483.648

Dann hast Du die Variable als Integer (*int x*) oder vielleicht überhaupt nicht deklariert. Deklariere mit *long x* oder *unsigned long x*, aber auch nur dann, wenn Du es wirklich brauchst.

Mich würde interessieren, warum man nicht dieselbe Methode gleichzeitig von 2 verschiedenen Threads aus benutzen kann, wenn man keine Referenzvariablen benutzt.
(Ich dachte, alle Variablen werden bei einem Methodenaufruf neu auf den Stack gelegt und sind damit Thread unabhängig!)

Bsp:

```
int add(int a, int b) // nicht: int add(int &a, int &b)
{
return a+b;
}
```

John Hanson sagt, dass alle Variablen intern global verwaltet werden und einen Stack gibt es nicht.

Methoden können entweder als inline oder mit *safecall threadsafe* gemacht werden. (inline kopiert den Inhalt der Methode an die Stellen im Programm, wo die Methode aufgerufen wird. Bei *safecall* schützt der Compiler die Methode über ein *Mutex*.)

Hier ein Programm von Hitechnic in NXC, um den hauseigenen Kompass-Sensor zu kalibrieren (ungeprüft, unkorrigiert und ohne Gewähr)

Code:

```
#include "NXCDefs.h"

#define COMPASS_PORT IN_1

int count;
int heading;
byte cmndbuf[]; // buffer for outbound I2C command
byte respbuf[]; // buffer for inbound I2C response

task main()
{
SetSensorLowspeed(COMPASS_PORT); // set sensor port 1 to low speed serial (I2C)

while (TRUE) // run forever til stop button pressed
{

// Go into cal mode if orange button pressed

if(ButtonState(BTN4))
{
ArrayInit(cmndbuf, 0, 3); // set the buffer to hold 2 values
cmndbuf[0] = 0x02; // set write to channel
cmndbuf[1] = 0x41; // to set write address
cmndbuf[2] = 0x43; // to write "C"
count=0; // no bytes to read
I2CBytes(COMPASS_PORT, cmndbuf, count, respbuf);
while(ButtonState(BTN4));

Wait(100);
```

```
ClearScreen();
TextOut(20, LCD_LINE1, "Calibrating");

while(ButtonState(BTN4)==0);

ArrayInit(cmndbuf, 0, 3); // set the buffer to hold 2 values
cmndbuf[0] = 0x02; // set write to channel
cmndbuf[1] = 0x41; // to set write address
cmndbuf[2] = 0x00; // to write 0
count=0; // no bytes to read
I2CBytes(COMPASS_PORT, cmndbuf, count, respbuf);
while(ButtonState(BTN4));

Wait(100);
}

// Read from Compass

ArrayInit(cmndbuf, 0, 2); // set the buffer to hold 2 values
cmndbuf[0] = 0x02; // set write to channel
cmndbuf[1] = 0x42; // to set read address
count=2; // 6 bytes to read
I2CBytes(COMPASS_PORT, cmndbuf, count, respbuf);
heading=respbuf[0]*2+respbuf[1]; // Calculate the heading

// Display heading

ClearScreen();
NumOut(20, LCD_LINE1, heading);
Wait(12);

}
}
```

Was sind eigentlich die Unterschiede zwischen nbc und nxc?

NBC ist eine Assembler-artige Programmiersprache (Next Byte Codes) und NXC eine C-artige Sprache (Not exactly C).
Beides sind Open Source Sprachen und eine "bessere" Alternative als NXT-G.
Beide sind kostenfrei.

Gibt es in NXC eine Möglichkeit „Klassen“ (ähnlich wie in Delphi) zu schreiben und zu verwenden?

Nein, in NXC gibt es keine, aber in NXJ gibt es Klassen.

Kann man auch Funktionen, also „voids“ gleichzeitig ausführen?

Nein, nur „tasks“ können parallel laufen.

Wofür sind die Klammern nach dem Tasknamen? Ich hab mal gehört, dass man da eine Variable reinschreiben kann ist das richtig und wofür?

Code:

```
void fahren(int tempo) {
    OnFwd((AC),tempo);
}

task main() {
    fahren(50);
}
```

So wie oben kann man es nutzen. Ist jetzt keine sehr nützliche Funktion, aber zeigt wie es geht.

Generell kann man mit zusätzliche Prozeduren, Funktionen(=Prozedur mit Rückgabewert) und Tasks ein Programm besser strukturieren und kann dann statt viele einzelne Befehle immer wieder aufrufen zu müssen, einfach nur noch die Prozedur starten, wo diese Befehle schon drin stehen.

Ein übergebener Parameter in Klammern, macht die Prozedur dann noch vielseitiger einsetzbar, wenn dies notwendig ist. Ohne Parameter macht einer Prozedur eigentlich immer das Gleiche. Mit Parameter führt sie Berechnung oder Aktionen aus, in Abhängigkeit vom Parameter. Hier ändert sich die Geschwindigkeit in Abhängigkeit vom Parameter.

Wichtig ist immer, die Prozedur/Funktion/Task muss vor dem Aufrufen definiert werden, also darüber stehen. Bei einem Parameter muss man auch immer einen Typ angeben, was für eine Art Wert die Prozedur erwartet - in diesem Fall einen *int* (zahlen) Wert.

Ich möchte gerne den NXT 2.0 Farbsensor (nicht der von HiTechnic!) als Lichtsensor verwenden. Den Befehl als Farbsensor kenne ich (SetSensorColorFull(IN_X);),allerdins den Befehl nicht, um den Lichtwert, und nicht den Farbwert von 1-6 abzurufen. Welcher Befehl ist das?

Mit dem Befehl SetSensorColorRed(IN_X) bekommt man einen ähnlichen Wert wie mit dem Lichtsensor, nur von 0 bis 1000.

LINKS

http://bricxcc.sourceforge.net/nbc/	Offizielle NXC-Webseite
http://bricxcc.sourceforge.net/	Offizielle BricxCC-Webseite
http://lukas.internet-freaks.net/nxt.php	Deutsches Tutorial zur Programmierung des NXT mit NXC (auch für Linux)
http://www.ist.uni-stuttgart.de/robolab/NXC-Hilfe.pdf	Deutsche Kurzanleitung der wichtigsten NXC-Befehle
http://www.debacher.de/wiki/NXC	deutschsprachige Hilfe zur Programmiersprache NXC
www.nxt-forum.de	Das größte deutschsprachige Forum zu Lego Mindstorms
http://www.mindsensors.com	Sensoren der Firma Mindsensors
http://www.hitechnic.com	Sensoren der Fa. Hitechnic
http://robotics.benedettelli.com	Infos, Beispiele, Bluetooth (in englisch)
