Programmieren von LEGO Robotern mit der Sprache NXC

1. Das erste Programm

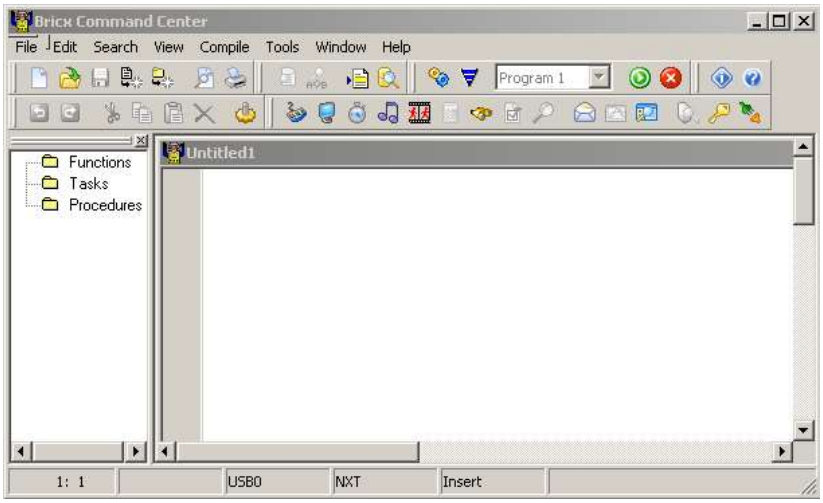
In diesem Kapitel wird Dir gezeigt, wie Du ein einfaches Programm zur Steuerung des Lego Mindstorm NXT-Roboters schreiben kannst. Der Roboter wird so programmiert, dass er für 2 Sekunden vorwärts, danach für weitere 2 Sekunden rückwärts fährt und dann anhält. Nicht sehr spektakulär, aber es wird Dich in die Grundidee der Programmierung mit der Sprache NXC (Not eXactly C) einführen und Dir zeigen, wie einfach die Programmierung ist. Aber bevor wir ein Programm schreiben können, benötigen wir zuerst einen Roboter.
Baue den Roboter Tribot aus dem Heft (Seite 8 - 22) im Lego-Mindstorm Kasten auf. Achte dabei besonders auf den richtigen Anschluss der Kabel (Motoren B und C). Dieses ist wichtig, damit dein Roboter später in die richtige Richtung fährt.

Wenn du den Roboter fertig aufgebaut hast, schließe ihn über das USB-Kabel an dem Computer an.

1.1. Das Bricx Command Center (BricxCC)

Wir schreiben unsere Programme mit dem **Bricx Command Center.** Starte das Bricx Command Center über Start > Programme > NWT > BricxCC.
Das Programm fragt Dich nach dem NXT. Schließe ihn deshalb VORHER über das USB-Kabel an und schalte den Roboter ein.
Mache die folgenden Einstellungen (siehe rechtes Fenster):
Port: usb, **Brick Type:** NXT, **Firmware:** Standard.
Klicke dann auf den **OK**-Button.

Jetzt erscheint das Programmfenster, wie auf der rechten Seite gezeigt. Das Bricx Command Center sieht wie ein Standard-Texteditor (beispielsweise Wordpad) mit dem üblichen Menü aus. Es hat Tasten zum Öffnen, Drucken und Bearbeiten von Dateien. Aber es gibt auch einige spezielle Menüs für die Erstellung (kompilieren) und den Download der Programme auf den Roboter, sowie für das Anzeigen von Informationen des Roboters. Diese kannst Du aber für diesen Moment vorerst ignorieren.



Wir gehen nun daran ein neues Programm zu schreiben. Drücke auf das weiße Icon „New File“ (leeres weißes Blatt) um ein neues Programmblatt zu erhalten.

1.2. Wir schreiben unser erstes Programm

Tippe jetzt das folgende Programm ab. Beachte dabei die Groß- und Kleinschreibung.

```
task main()
{
OnFwd(OUT_BC, 75);
Wait(2000);
OnRev(OUT_BC, 75);
Wait(2000);
Off(OUT_BC);
}
```

Das Programm sieht auf den ersten Blick etwas kompliziert aus, deshalb wird es jetzt genauer beschrieben:
Programme in NXC bestehen aus **Tasks** (engl. task = Aufgabe). Unser Programm hat eine Task namens **main** (engl. main task = Hauptaufgabe). Jedes Programm muss einen main-Task haben, dieser wird immer vom Roboter ausgeführt. Du erfährst mehr über Tasks in Kapitel 6. Ein Task besteht aus einer Reihe von Befehlen, auch Anweisung genannt. Es gibt Klammern { } um diese Befehlsblöcke, so dass klar ist, zu welchem Task sie gehören. Jede Anweisung endet mit einem Semikolon (Strichpunkt). Auf diese Weise wird deutlich, wo ein Befehl endet und wo der nächste beginnt.
Eine Task sieht in der Regel wie folgt aus:

```
task main()
{
Anweisung 1;
Anweisung 2;
…
}
```

Unser Programm hat fünf Anweisungen. Werfen wir einen Blick darauf:

OnFwd(OUT_BC, 75);
Die Anweisung OnFwd „sagt“ dem Roboter: „Schalte den Motor B und den Motor C (an Ausgang B und C: OUT_BC) ein und drehe den Motor in Vorwärtsrichtung“. Die nachfolgende Zahl 75 setzt die Geschwindigkeit des Motors auf 75% der maximalen Geschwindigkeit. Wenn sich nur ein Motor drehen soll, steht in dem Befehl anstelle von OUT_BC nur noch z. B. OUT_B.

Wait(2000);
Jetzt ist es Zeit, eine Weile zu warten. Der Befehl Wait weist an, 2 Sekunden (2000) zu „warten“. Die Zahl in der Klammer gibt die Zeit in 1 / 1000 Sekunden an. Damit kannst Du sehr genau bestimmen, wie lange das Programm zu warten hat. Hier hält das Programm exakt für 2 Sekunden an, ehe es mit der Abarbeitung des nächsten Befehls weiter voranschreiten kann bzw. der Roboter die nächste Aktion ausführt. Der Roboter führt also 2 Sekunden lang den in der Zeile vor dem Wait-Befehl stehenden Befehl aus, er fährt also 2 Sekunden lang vorwärts.

OnRev(OUT_BC, 75);
Der Befehl OnRev weist den Roboter an in umgekehrter Richtung, also rückwärts zu fahren. Auch die ersten beiden Befehle hätten auf diese Weise geschrieben werden können.

Wait(2000);
Wir warten wieder 2 Sekunden.

Off(OUT_BC);
Zum Schluss schalten wir beide Motoren mit Off aus.

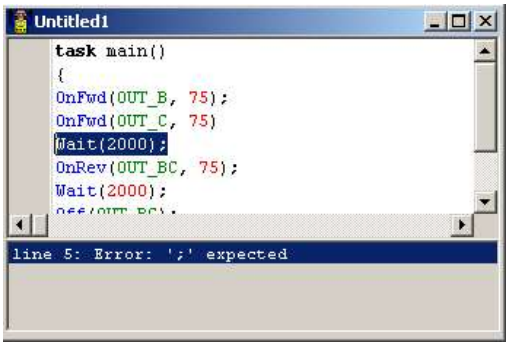
Das Programm in Kurzfassung: Es bewegt beide Motoren für 2 Sekunden vorwärts, dann rückwärts für 2 Sekunden, und schließlich schaltet es die Motoren aus.

Wahrscheinlich hast Du die Farbgebung bemerkt während Du das Programm eingegeben hast. Diese erscheinen automatisch. Die Farben sind hilfreich um Fehler im Programm zu finden.

1.3. Wir lassen das Programm ablaufen

Sobald Du ein Programm geschrieben hast, muss es zunächst kompiliert werden (das heißt, es muss in den Code, den der Roboter „verstehen“ und durchführen kann, übersetzt werden). Zum Kompilieren musst Du das Symbol mit den zwei Zahnrädchen (siehe die Abbildung oben) anklicken. Wenn es Fehler in Deinem Programm gibt, werden sie Dir angezeigt (siehe unten). Danach muss das Programm zum Roboter über das USB-Kabel übertragen werden. Dazu musst Du das Symbol mit dem blauen Dreieck anklicken. Betätigst Du diese Taste, dann wird das Programm auf den NXT übertragen. Jetzt kannst Du Dein Programm laufen lassen. Betätige dazu die grüne Run-Taste oder starte das Programm direkt vom NXT aus.
Verhält sich der Roboter wie erwartet? Wenn nicht, überprüfe das USB-Kabel und vergewissere Dich, dass die Motoren richtig angeschlossen sind. Falls das Programm läuft, kannst Du es mit der roten Stopp-Taste abbrechen.

1.4. Fehler im Programm



Beim Schreiben des Programms kann es sein, dass Du Fehler machst. Der Compiler zeigt diese Fehler am unteren Rand des Fensters an, wie in der nebenstehenden Abbildung dargestellt. Er wählt automatisch den ersten Fehler aus (hier den vergessenen Strichpunkt). Wenn es mehrere Fehler gibt, kannst du die Fehlermeldungen anklicken, um zu ihnen zu gelangen. Beachte bitte, dass oft Fehler am Anfang des Programms andere Fehlermeldungen an anderen Stellen nach sich führen können. Es ist also besser nur den ersten Fehler zu korrigieren und anschließend das Programm erneut zu kompilieren. Beachte auch, dass dir die Farben viel dabei helfen, um Fehler zu erkennen. Schreiben wir beispielsweise in der letzten Zeile „Of“ statt „Off“, wird der Befehl nicht blau hervorgehoben. Die häufigsten Fehler sind falsch gesetzte Klammern und vergessene Strickpunkte am Ende der Befehlszeile. Wenn beispielsweise die Meldung **line 5 : Error: ';' expected** erscheint, so schau doch zuerst in die Zeile zuvor, ob dort nicht der Strich-Punkt fehlt!
Es gibt auch Fehler, die nicht vom Compiler gefunden werden. Wenn wir OUT_A geschrieben hätten, wäre der Fehler unbemerkt geblieben, weil dieser Motor existiert, obwohl wir ihn nicht im Roboter benutzen. Wenn dein Roboter sich also anders verhält, als du erwartet hast, so ist ein Fehler in deinem Programm die wahrscheinlichste Ursache.

1.5. Ändern der Geschwindigkeit

Wie du vielleicht bemerkt hast, bewegt sich dein Roboter ziemlich schnell. Wir setzen die Geschwindigkeit von 75% der maximalen Geschwindigkeit auf 50% herab:

```
task main()
{
OnFwd(OUT_B, 50);
Wait(2000);
OnRev(OUT_BC, 50);
Wait(2000);
Off(OUT_BC);
}
```

1.6. Zusammenfassung

In diesem Kapitel hast du dein erstes Programm in NXC mit dem Bricx Command Center (BricxCC) geschrieben. Du solltest jetzt wissen, wie man ein Programm schreibt, wie Du es auf den Roboter übertragen kannst und wie der Roboter das Programm startet. Das Programm BricxCC bietet noch weitere Funktionen. Diese Anleitung wird sich in erster Linie mit der Programmiersprache NXC befassen. Es wird nur dann auf Merkmale von BricxCC eingegangen, wenn Du sie wirklich brauchen kannst.
Du hast außerdem schon einige wichtige Aspekte der Sprache NXC gelernt. Als erstes hast du erfahren, dass jedes Programm eine task main() hat, welche immer beim Starten des Programms als erstes aufgerufen wird. Auch hast Du die vier grundlegenden Motor-Befehle: OnFwd(), OnRev() und Off() kennen gelernt, sowie die Anweisung Wait().
Aber mit der Sprache NXC kannst du noch viel mehr tun.

2. Weitere einfache Programme

Unser erstes Programm war nicht wirklich beeindruckend. Wir versuchen es noch interessanter zu gestalten. Wir werden Schritt für Schritt komplexere Programme schreiben und dabei einige wichtige Merkmale der Programmiersprache NXC kennen lernen.

2.1. Kurven fahren

Du kannst Deinen Roboter durch Stillstand oder Umkehr der Richtung einer der beiden Motoren eine Kurve fahren lassen. Schreibe das Beispielprogramm ab, lade es auf den Roboter und starte das Programm. Der Roboter sollte ein Stück geradeaus fahren und dann eine 90-Grad Linkskurve vollziehen.

```
task main()
{
OnFwd(OUT_BC, 75);
Wait(2000);
OnRev(OUT_C, 75);
Wait(400);
Off(OUT_BC);
}
```

Vielleicht musst du den Parameter des zweiten Wait-Befehls anpassen, um eine 90-Grad-Kurve zu fahren. Dieses hängt von der Oberflächenbeschaffenheit, auf der der Roboter fährt und von dem Ladezustand der Akkus ab. Anstatt derartige Änderungen im Wait-Befehl vorzunehmen, empfiehlt es sich einen Namen für diesen Parameterwert zu nutzen. In NXC kannst Du konstante Werte festlegen, wie folgendes Beispielprogramm zeigt:

```
#define FAHREN 2000 // Fahrzeit
#define DREHEN 400  // Drehzeit
task main()
{
OnFwd(OUT_BC, 75);
Wait(FAHREN);
OnRev(OUT_C, 75);
Wait(DREHEN);
Off(OUT_BC);
}
```

Die ersten beiden Zeilen definieren wir zwei Konstanten FAHREN und DREHEN. Diese können als Parameter nun im Programm verwendet werden. Definieren von Konstanten ist aus zwei Gründen empfehlenswert:
1. Das Programm wird besser lesbar, und
2. Die Werte im Programm können leichter geändert werden.
Beachte, dass BricxCC den define-Anweisungen eine eigene Farbe – die Farbe lila - zuweist.

2.2. Schleifen

Wir wollen jetzt ein Programm schreiben, das den Roboter in einem Quadrat fahren lässt. Ein Quadrat zu fahren bedeutet: Vorwärts fahren – 90 Grad Drehung – Vorwärts fahren – 90 Grad Drehung – Vorwärts fahren – 90 Grad Drehung usw.
Um ein Quadrat zu programmieren, können wir einfach die obigen Code-Sequenz vier Mal wiederholen, aber dies kann auch viel einfacher, mit der so genannten repeat-Anweisung, programmiert werden.

```
#define FAHREN 2000
#define DREHEN 400
task main()
{
	repeat(4)                    // 4fache Wiederholung der Befehle
	{                            // in den geschweiften Klammern
	OnFwd(OUT_BC, 75);
	Wait(FAHREN);
	OnRev(OUT_C, 75);
	Wait(DREHEN);
	}
	Off(OUT_BC);
}
```

Falls dein Roboter kein schönes Quadrat gefahren ist, verändere doch die Werte für FAHREN und DREHEN in den ersten beiden Zeilen. Die Zahl (4) innerhalb der Klammern der repeat-Anweisung gibt an, wie oft der Code zwischen den geschweiften Klammern wiederholt wird. Beachten Sie, dass im obigen Programmbeispiel die define-Anweisungen wieder benutzt wurden. Dies ist zwar nicht notwendig, trägt aber zur Lesbarkeit des Programms bei.
Als letztes Beispiel lassen wir den Roboter 10mal im Quadrat fahren.
Hier ist das Programm:

```
#define FAHREN 2000
#define DREHEN 400
task main()
{
	repeat(10)
	{
		repeat(4)
		{
		OnFwd(OUT_BC, 75);
		Wait(FAHREN);
		OnRev(OUT_C, 75);
		Wait(DREHEN);
		}
	}
	Off(OUT_BC);
}
```

Das Beispielprogramm zeigt zwei repeat-Anweisungen. Eine repeat-Anseiung befindet sich innerhalb der anderen. Dies wird auch eine "verschachtelte" repeat-Anweisung bzw. allgemein „Verschachtelung“ genannt. Du kannst die Verschachtelung beliebig oft wiederholen. Es empfiehlt sich, etwas Zeit und Sorgfalt auf das Setzen der Klammern und das Einrücken der Programm-Befehle zu verwenden.

- Der Task beginnt bei der ersten (geschweiften) Klammer und endet bei der letzten.
- Die erste repeat-Anweisung beginnt bei der zweiten (geschweiften) Klammer und endet bei der Fünften.
- Die geschachtelte repeat-Anweisung beginnt mit der dritten (geschweiften) Klammer und endet mit der Vierten.

Wie Du erkennst, werden Klammern immer paarweise gesetzt. Das Code-Stück (Programmblock) zwischen den Klammern wird (zur besseren Übersicht) eingerückt.

2.3. Kommentare einfügen

Um dein Programm noch lesbarer zu gestalten, empfiehlt es sich das Programm mit Bemerkungen (Kommentare) zu versehen. Wenn Du / / in einer Zeile einfügen, wird der Rest der Zeile ignoriert und kann zur Kommentierung genutzt werden. Ein langer Kommentar kann zwischen / * und * / geschrieben werden. Das vollständige Programm könnte somit wie folgt aussehen:

```
/* 10 Quadrate
Dieses Programm veranlasst den Roboter 10 Quadrate zu fahren
*/
#define FAHREN 2000         // Zeit für die Geradeausfahrt
#define DREHEN 400          // Zeit für die 90°-Kurve
task main()
{
       repeat(10)           // 10 Quadrate fahren
       {
              repeat(4)     // Fahre die vier Ecken und Seiten
              {
              OnFwd(OUT_BC, 75);
              Wait(FAHREN);
              OnRev(OUT_C, 75);
              Wait(DREHEN);
              }
       }
       Off(OUT_BC);         // Motoren ausschalten
}
```

2.4. Zusammenfassung

In diesem Kapitel hast Du die richtige Anwendung der repeat-Anweisung und die Verwendung von Kommentaren gelernt. Auch hast Du verschachtelte Anweisungen und die Nutzung von Einrückungen gesehen. Mit dem bisher Gelernten kannst Du einen Roboter entlang aller möglichen Wege fahren lassen. Eine gute Übung für Dich: Versuche die Programme dieses Kapitels zu verändern, bevor Du mit dem nächsten Kapitel fortfährst!

3. Variablen

Variablen bilden einen sehr wichtigen Aspekt jeder Programmiersprache. Variablen sind eine Art Behälter oder Container - in denen Werte gespeichert werden können. Wir können Variablen an verschiedenen Stellen im Programm nutzen, und wir können Variablen ändern. Veranschaulichen wir die Verwendung von Variablen anhand eines Beispiels.

3.1. Eine Spirale fahren

Angenommen wir wollen das obige Programm so abändern, dass der Roboter eine Spirale fährt. Dies können wir erreichen, indem wir den Wert von FAHREN, also der Zeit in der der Roboter geradeaus fährt, zwischen jeder Drehung erhöhen. Aber wie können wir das tun? Wir haben FAHREN bisher als eine Konstante definiert, deren Wert daher nicht verändert werden kann. Statt einer Konstante benötigen wir nun eine Variable. Variablen können einfach in NXC definiert werden. Hier ist das Beispielprogramm:

```
/* Spiralen-Programm */
#define DREHEN 200  // definiert die Konstante DREHEN mit dem Wert 200
int FAHREN;         // definiert die Variable FAHREN
task main()
{
       FAHREN = 100;       // weist der Variablen den Startwert 100 zu
       repeat(50)
       {
              OnFwd(OUT_BC, 50);
              Wait(FAHREN);              // FAHREN = Variable für die Geradeausfahrt
              OnRev(OUT_C, 50);
              Wait(DREHEN);              // DREHEN = Konstante für die Drehung
              FAHREN = FAHREN + 20;      // erhöht den Wert der Variablen FAHREN um 20
       }
       Off(OUT_BC);        // Schalte nun die Motoren ab
}
```

Die interessanten Zeilen wurden mit Kommentaren versehen. Zuerst definieren wir eine Variable, indem wir das Stichwort **int** gefolgt von einem Namen wählen. (In der Regel werden Kleinbuchstaben für Variablennamen und Großbuchstaben für Konstanten vergeben). Der Name der Variablen muss mit einem Buchstaben beginnen, nachfolgend können auch Ziffern und Unterstriche verwendet werden. Andere Symbole sind nicht erlaubt. (Das gleiche gilt für Namen von Konstanten, Tasks, usw.) Das Wort **int** steht für Ganzzahl. Nur ganze Zahlen können in **int** Variablen gespeichert werden.
Der Variablen „FAHREN“ wird der Startwert 100 zugewiesen. Wird nun die Variable benutzt (wie in der Zeile 11 und Zeile 15) besitzt diese den Wert 100. Jetzt folgt die repeat-Schleife in der die Variable benutzt wird um den Roboter vorwärts fahren zu lassen. Am Ende der Schleife erhöhen wir den Wert der Variable um 20 (FAHREN = FAHREN + 20). Beim ersten Schleifendurchlauf fährt der Roboter 100ms geradeaus. Beim zweiten Durchlauf 120ms, beim dritten Durchlauf 140ms usw. Neben dem Hinzufügen von Werten zu einer Variable kann man Variablen auch

- multiplizieren *
- subtrahieren -
- teilen /
  (Beachte, dass bei der Teilung einer int-Ganzzahl das Ergebnis auf die nächste ganze Zahl gerundet wird z.B. 3,4 ist 3; 3,5 ist 4).

Du kannst auch mehrere Variablen miteinander verrechnen, sowie komplizierte Ausdrücke berechnen. Das nächste Beispiel hat keine Wirkung auf den Roboter, es dient lediglich der Veranschaulichung, wie Variablen verwendet werden können. Du musst dieses Programm nicht abtippen, es reicht, wenn Du es Dir durchliest.

```
int aa;             // Definition der Variablen aa
int bb, cc;         // gleichzeitige Definition der Variablen bb und cc
task main()
{
aa = 10;            // weise der Variablen aa den Wert 10 zu
bb = 20 * 5;        // weise bb den Wert 20 mal 5 = 100 zu
cc = bb;            // sezte cc gleich bb, cc wird also ebenfalls 100
cc = cc / aa;       // berechne cc / aa und weise das Ergenis cc zu, cc = 100/10 = 10
cc = cc - 1;        // ziehe von cc 1 ab und weise das Ergebnis cc zu, cc wird also 9
aa = 10 * (cc + 3); // berechne 10 * (cc + 3), aa wird also 10 * (9 + 3) = 120
}
```

Beachte, dass wir in einer Zeile mehrere Variablen definieren können, wie in der zweiten Zeile gezeigt. Wir könnten auch alle drei Variablen in einer Zeile definieren.

3.2. Zufallszahlen

In allen bisher aufgeführten Programmen haben wir genau definiert, was der Roboter tun soll. Aber die Dinge werden oftmals viel interessanter, wenn man nicht genau weiß was der Roboter tun wird. Wir wollen einige Zufälligkeiten in den Bewegungen des Roboters einbauen. Mit NXC kannst Du zufällige Zahlen erzeugen. Das folgende Programm verwendet Zufallszahlen, um den Roboter „beliebig“ umher fahren zu lassen. Der Roboter fährt eine zufällige Zeit geradeaus und macht anschließend eine zufällige Drehung.

```
/* Zufallsbewegung */
int FAHREN, DREHEN;        // definiert 2 Variable
task main()
{
       while(true)                 // weist der Variablen einen Startwert zu
       {
       FAHREN = Random(1000);
       DREHEN = Random(400);
       OnFwd(OUT_BC, 75);
       Wait(FAHREN);               // verwendet die Variable für die Pausenlänge
       OnRev(OUT_C, 75);
       Wait(DREHEN);
       }
}
```

Das Programm definiert zwei Variablen und weist ihnen dann Zufallszahlen zu. Random(1000) bedeutet, dass die ganzzahlige Zufallszahl zwischen 0 und 999 liegt. Jedes Mal wenn die Funktion Random aufgerufen wird unterscheiden sich die Zahlen. Mit der Verwendung der Variablen FAHREN und Wait (FAHREN) konnten wir die ungünstige Schreibweise Wait(Random(1000)) vermeiden.
Im soeben eingeführten Beispielprogramm, hast Du auch einen neuen Schleifen-Typ kennen gelernt. Anstatt der Verwendung von **repeat** haben wir die **while(true)** Anweisung benutzt. Die while-Schleife wird so lange wiederholt, wie die Bedingung zwischen den Klammern wahr ist. Eine while-Schleife mit der Bedingung true (= wahr) bricht nie ab, man spricht hier auch von einer Endlosschleife. Du kannst eine laufende Endlosschleife durch Drücken des dunkelgrauen NXT-Knopfes abbrechen. Mehr über while-Schleifen erfährst Du in Kapitel 4.

3.3. Zusammenfassung

In diesem Kapitel hast du etwas über die Verwendung von Variablen gelernt. Du kannst auch noch andere Datentypen wie int deklarieren, z.B. short, long, byte, bool und string.
Du hast auch gelernt, wie man Zufallszahlen erzeugt und verwendet, so dass sich der Roboter unvorhersehbar bewegt. Schließlich hast Du die Verwendung der while-Anweisung kennen gelernt um eine Endlosschleife zu programmieren.

4. Wenn-Dann-Bedingungen

In den vorhergehenden Kapiteln haben wir die Wiederholungsschleifen **repeat** und **while** kennen gelernt. Diese Anweisungen geben die Möglichkeit, das bestimmte Teile des Programms mehrmals ausgeführt werden. In diesem Kapitel lernst Du Wenn-Dann-Bedingungen kennen. Als erstes Kommt die **if**-Anweisung („Wenn“-Anweisung), dann die **while**(Bedingung)-Anweisung („während“-Anweisung bzw. „solange“-Anweisung).

4.1. Die if - Anweisung

Manchmal möchte man, dass ein bestimmter Teil eines Programms nur in bestimmten Situationen ausgeführt wird. In diesem Fall wird die **if**-Anweisung verwendet. Betrachten wir das an einem Beispiel: Wir werden wieder das bisherige Programm verwenden, und um die if-Anweisung erweitern. Wir wollen, dass der Roboter geradeaus fährt und dann entweder links oder rechts abbiegt. Dazu benötigen wir wieder Zufallszahlen. Wir wählen die Zufallszahlen über den Befehl Random(3) aus, d.h. die Zufallszahl ist entweder 1 oder 2. Wenn die Zahl 1 ist, soll der Roboter links herum fahren, andernfalls rechts herum. Hier nun das Beispielprogramm:

```
/* if-Anweisung */
#define FAHREN 300
#define DREHEN 150
task main()
{
	while(true)
	{
		OnFwd(OUT_BC, 75);
		Wait(FAHREN);
		if (Random(3) == 1)
		{
		OnRev(OUT_C, 75);
		Wait(DREHEN);
		}
		else
		{
		OnRev(OUT_B, 75);
		Wait(DREHEN);
		}
	}
}
```

Die if-Anweisung ähnelt auf dem ersten Blick der while-Anweisung. Wenn die Bedingung in der runden Klammer hinter der if eintritt, wird der Teil zwischen den geschweiften Klammern ausgeführt. Andernfalls wird der Teil zwischen den geschweiften Klammern nach dem Wort else ausgeführt. Werte und Zahlen können auf unterschiedlicher Weise miteinander verglichen werden. Hier die wichtigsten:

==	Vergleich ob zwei Werte exakt gleich sind
<	kleiner als
<=	kleiner oder gleich
>	größer als
>=	größer oder gleich
!=	ungleich

Du kannst die Bedingungen mit && untereinander kombinieren, welches „und” bedeutet, oder mit ||, welches „oder” bedeutet. (Den |-Strich erreichst du mit der Tastenkombination Alt Gr und ><).

Hier sind einige Beispiele für Bedingungen:

true	immer wahr
false	nie wahr
xyz != 3	wahr, wenn xyz ungleich 3 ist
(xyz >= 5) && (xyz <= 10)	wahr, wenn xyz zwischen 5 und 10 liegt
(thc == 10) || (mfg == 10)	wahr, wenn thc oder mfg (oder beide) genau 10 sind

Beachte, dass die if-Anweisung aus zwei Teilen besteht. Der erste Teil unmittelbar nach der Bedingung wird ausgeführt wenn die Bedingung wahr ist. Der zweite Teil (else-Teil) wird ausgeführt, wenn die Bedingung falsch ist. Das Schlüsselwort else und der sich daran anschließende Teil ist nicht immer nötig. Soll beispielsweise der Roboter keine Aktion durchführen, falls die if-Anweisung falsch ist, kann der else-Teil einfach entfallen.

4.2. Die while(Bedingung) – Anweisung

Es gibt noch andere Kontrollstrukturen, z.B. die while(Bedingung)-Anweisung. Diese weist folgende Form auf:

```
while (Bedingung)
{
Befehle;
}
```

Die Anweisungen zwischen den geschweiften Klammern wird solange ausgeführt, wie die Bedingung in den runden Klammern hinter dem while wahr ist. Hier ein Beispielprogramm: Der Roboter fährt 10 Sekunden (zufällig) umher. Sind die 10 Sekunden erreicht stoppt er.

```
/* while-Anweisung */
int FAHREN, DREHEN, ZEIT;

task main()
{
	ZEIT = 0;
	while (ZEIT < 10000)
	{
		FAHREN = Random(1000);
		DREHEN = Random(1000);
		OnFwd(OUT_BC, 75);
		Wait(FAHREN);
		OnRev(OUT_C, 75);
		Wait(DREHEN);
		ZEIT = ZEIT + FAHREN + DREHEN;
	}
	Off(OUT_BC);
}
```

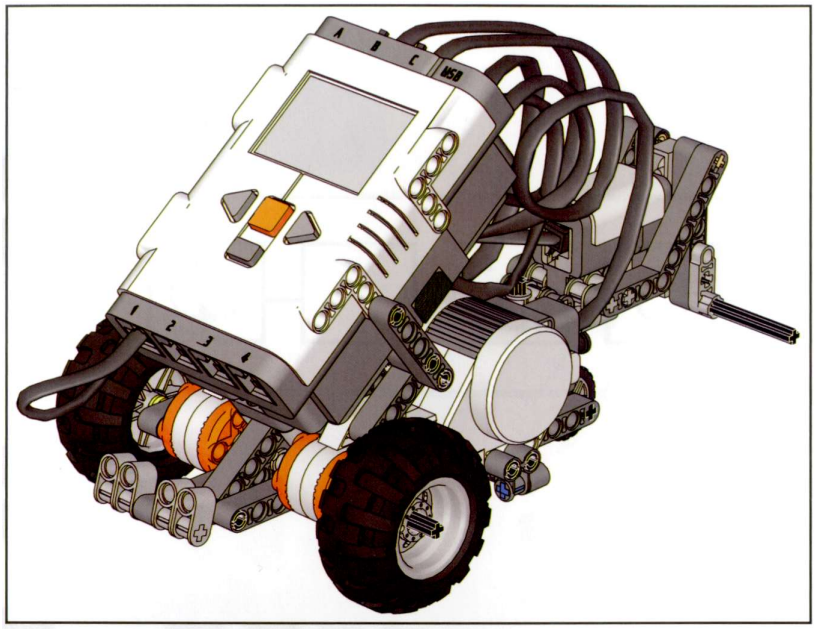
Das Programm arbeitet solange die Befehle, die in den geschweiften Klammern hinter dem while stehen ab, bis die Bedingung in der Zeile hinter dem while (in runden Klammern) nicht mehr erfüllt ist.

4.3. Zusammenfassung

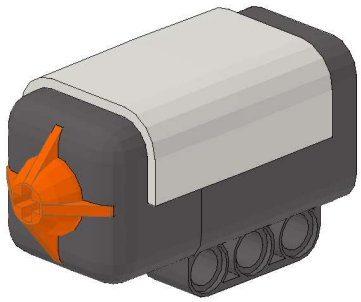
In diesem Kapitel haben wir zwei neue Strukturen kennen gelernt: die if-Anweisung und die while(Bedingung)-Anweisung. Zusammen mit der repeat-Anweisung und der while(true)-Anweisung, kennen wir somit mittlerweile vier Kontrollstrukturen.

5. Sensoren

Sensoren sind Fühler, wie für uns Menschen die Augen, die Ohren oder die Hände, mit denen der Roboter sehen, hören und tasten kann. Das tolle an unserem Lego-Roboter ist, dass wir mehrere Sensoren an ihm anschließen können, und dass wir den Roboter auf die Sensoren reagieren lassen können.
Bevor wir mit dem Einbinden von Sensoren in ein NXC-Programm beginnen können, müssen wir unseren Roboter etwas umbauen, indem wir einen Tastsensor hinzufügen. Baue zu diesem Zweck den Stoßfänger mit dem Tastsensor an deinen Roboter an, so wie es auf den Seiten 40 - 44 in dem Lego-Handbuch beschrieben ist. Dein Roboter sollte dann wie abgebildet aussehen.



Tast-Sensor:



Ändere jetzt den Aufbau deines Roboters: Baue den Tastsensor nicht an die Hinterseite des Roboters, sondern befestige ihn an der Vorderseite. Schließe den Tastsensor am Eingang 1 des NXT-Steins an.

5.1. Auf Sensor(-werte) warten

Beginnen wir mit einem sehr einfachen Programm, in dem der Roboter vorwärts fährt, bis er auf einen Gegenstand trifft.

```
/* Berührungssensor */
task main()
{
SetSensorTouch(IN_1);      // definiert Tastsensor an Eingang 1
while (Sensor(IN_1) == 0)  // während der Tastsensor noch nicht gedrückt ist
       {
       OnFwd(OUT_BC, 75);  // fährt der Roboter geradeaus
       }
Off(OUT_BC);               // wird der Tastsensor gedrückt, stoppen beide Motoren
}
```

SetSensorTouch(IN_1) sagt dem NXT, dass wir einen Tastsensor bzw. Berührungssensor am Roboter verwenden (engl. touch = berühren). IN_1 ist die Nummer des Eingangs, an den wir den Sensor angeschlossen haben. Die drei anderen Eingänge heißen IN_2, IN_3 und IN_4.

Für den Lichtsensor würden wir SetSensorLight verwenden, für den Mikrofonsensor SetSensorSound und für den Ultraschallsensor (Entfernungssensor) SetSensorLowspeed.

Der Tastsensor liefert nur zwei Zahlen. Wenn er gedrückt ist, liefert er die Zahl 1, wenn er nicht gedrückt liefert er die Zahl 0.

while (Sensor(IN_1) = = 0);
Diese Zeile bedeutet übersetzt „Während der Sensor am Eingang 1 (Tastsensor) noch nicht gedrückt ist“. Solange also der Tastsensor noch nicht gedrückt ist, führt der Roboter die Befehle in der geschweiften Klammer hinter der while(Bedingung) aus. In diesem Programm fährt er also vorwärts mit der Geschwindigkeit 75. Sobald der Tastsensor gedrückt wird, hat die Zahl Sensor(IN_1) den Wert 1 angenommen und das Programm verlässt sofort die while(Bedingung)-Schleife. Die Motoren werden also ausgeschaltet und das Programm ist am Ende.

5.2. Auf den Tastsensor reagieren

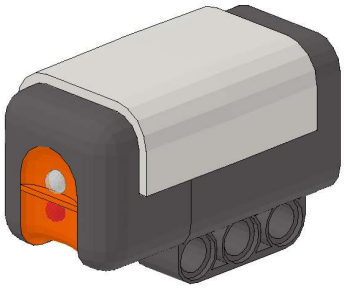
Versuchen wir nun, den Roboter Hindernissen ausweichen zu lassen. Wenn der Roboter gegen ein Objekt stößt, lassen wir ihn ein wenig rückwärts fahren und eine Drehung machen. Anschließend soll er wieder geradeaus fahren. Hier ist das Beispiel:

```
/* Berührungs-Sensor mit Ausweichen */
task main()
{
      SetSensorTouch(IN_1);      // definiert Tastsensor an Eingang 1
      while (true)               // Endlosschleife
      {
      OnFwd(OUT_BC, 75);         // fahre vorwärts
      if (Sensor(IN_1) == 1)     // wenn der Tastsensor gedrückt wird,
            {
            OnRev(OUT_BC, 75);   // dann fahre rückwärts
            Wait(500);           // für 0,5 Sekunden
            OnFwd(OUT_B, 75);    // und drehe
            Wait(500);           // für 0,5 Sekunden
            }
      }
}
```

Wie in dem vorherigen Beispiel haben wir zunächst die Art des Sensors festgelegt. Als nächstes folgt die Endlosschleife: Der Roboter fährt vorwärts. Wenn der Tastsensor gedrückt wird, dann fährt der Roboter für ½ Sekunde rückwärts und dreht ½ Sekunde. Anschließend fährt er wieder vorwärts, bis der Tastsensor erneut gedrückt wird. Da sich alle Fahrbefehle in der Endlosschleife (while(true)) befinden, muss die graue Taste auf dem NXT gedrückt werden, damit das Programm beendet wird.

5.3. Lichtsensor

Lichtsensor:



Neben zwei Tastsensoren enthält das LEGO Mindstorm NXT Set auch einen Lichtsensor. Der Lichtsensor kann in einen aktiven und einen passiven Modus versetzt werden. Im aktiven Modus leuchtet eine LED-Diode rot. Im passiven Modus ist die Leucht-Diode ausgeschaltet. Der aktive Modus kann bei der Messung des reflektierten Lichts nützlich sein, wenn der Roboter beispielsweise einer Linie auf dem Boden folgt. Dies werden wir im nächsten Beispielprogramm versuchen. Dazu müssen wir zunächst den Lichtsensor an unseren Roboter anbringen (siehe Lego Anleitung Seite 32 - 37). Entferne vorher noch den Tastsensor. Dann schließen wir den Lichtsensor an Eingang 1 an. Nun benötigen wir nur noch eine Bahn und natürlich das Programm. Das basiert auf folgenden Gedanken: Sobald der Roboter die Bahn verlässt, wird vom nun hellen Untergrund mehr Licht zurückgeworfen und der Roboter muss seine Bahn korrigieren. Es funktioniert aber nur, wenn der Roboter im Uhrzeigersinn, also rechts herum fährt.

```
/* Licht-Sensor */
task main()
{
      SetSensorLight(IN_1);      // definiert Lichtsensor an Eingang 1
      while (true)               // Endlosschleife
      {
      OnFwd(OUT_BC, 30);         // fahre vorwärts
      if (Sensor(IN_1) < 60)     // wenn der Lichtsensor dunkel sieht,
            {
            OnRev(OUT_B, 30);    // dann fahre mit dem rechten Motor rückwärts
            Wait(100);           // für 0,1 Sekunden
            }
      }
}
```

Zunächst wird dem Programm über SetSensorLight(IN_1); mitgeteilt, dass jetzt ein Lichtsensor am Eingang 1 angeschlossen ist. Dann kommt die bekannte Endlosschleife. Der Roboter fährt wieder geradeaus. Sobald der Roboter die dunkle Linie sieht, soll er mit dem rechten Motor für 0,1 Sekunden rückwärts fahren. Somit kann der Roboter der dunklen Linie folgen. In diesem Programm ist für die Grenze zwischen hellem und dunklem Untergrund der Wert 60 gewählt. Eventuell musst Du diese Zahl in Deinem Programm anpassen. Zum Messen der Lichtwerte gibt es zwei Möglichkeiten:

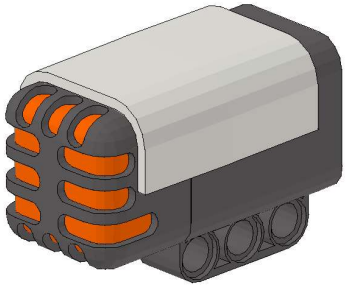
1) Schalte den NXT-Roboter ein und gehe mit der rechten Pfeiltaste drei Schritte nach rechts zu „VIEW“. Wähle „VIEW“ mit der roten Taste des NXT aus und drücke wieder die Pfeiltasten bis Du den richtigen Sensor (Reflected Light) gefunden hast. Wähle diesen durch Drücken der roten Taste aus. Wähle außerdem im Anschluss den Eingang, an dem der Lichtsensor angeschlossen ist, aus. Nun kannst du den Wert, den der Lichtsensor liefert am Display des NXT-Roboters ablesen. Der Wert liegt im Bereich zwischen 0 und 100. 0 ist sehr dunkel, 100 ist sehr hell.
2) Wähle im Programm Brixcc unter „Tools“ > „Watching the Brick“ aus. Mache hier die Einstellungen: „All“ und „Poll Regular“. Jetzt wird der Lichtsensorwert angezeigt.

Um das LED-Licht des Lichtsensors auszuschalten (passiver Modus), müssen unter der Zeile SetSensorLight(IN_1) im oberen Programm die folgenden drei Zeilen geschrieben werden.

```
SetSensorType(IN_1,IN_TYPE_LIGHT_INACTIVE);
SetSensorMode(IN_1,IN_MODE_PCTFULLSCALE);
ResetSensor(IN_1);
```

5.4. Mikrofonsensor

Mikrofonsensor:



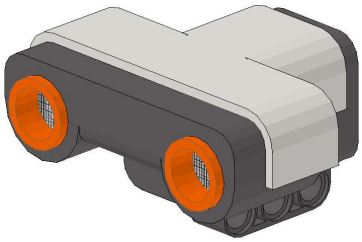
Mit dem Mikrofonsensor kannst du den NXT auf Geräusche reagieren lassen. Wir schreiben ein Programm, das auf ein lautes Geräusch wartet. Der Roboter bewegt sich anschließend für 1 Sekunde. Wenn es dann immer noch laut ist, dann fährt der Roboter wieder 1 Sekunde.
Baue den Mikrofonsensor nach der Lego Anleitung von Seite 24 - 27 auf und befestige ihn am Roboter. Vorher solltest Du noch den Lichtsensor abbauen. Schließe dann den Mikrofonsensor am Eingang 1 an und schreibe das folgende Programm:

```
/* Mikrofonsensor */
#define LAUT 50
task main()
{
      SetSensorSound(IN_1);                  //definiert Mikrofonsensor an Eingang 1
      while (true)                           // bildet eine Endlosschleife
      {
            if(Sensor(IN_1) > LAUT)          // Wenn ein Geräusch wahrgenommen wird
            {
            OnFwd(OUT_BC, 75);               // fährt der Roboter vorwärts
            Wait(1000);                      // für 1 Sekunde
            Off(OUT_BC);
            }
      }
}
```

In der allerersten Zeile definieren wir mit #define LAUT 50 einen Wert für die Lautstärke, ab der der Mikrofonsensor reagieren soll. Es kann sein, dass Du ihn, je nach den Lautstärkeverhältnissen im Raum, ändern musst. Dann teilen wir dem NXT mit, dass wir an Eingang 1 einen Mikrofonsensor angeschlossen haben und die uns schon bekannte Endlosschleife (while(true)) wird gestartet. Sobald der Roboter ein Geräusch über der Lautstärke 50 wahrnimmt, fährt er für 1 Sekunde geradeaus. Falls das Geräusch dann immer noch lauter als 50 ist, fährt der Roboter erneut 1 Sekunde. Wenn es wieder leiser geworden ist, bleibt der Roboter wieder stehen.
Der Mikrofonsensor liefert wie auch schon der Lichtsensor Werte zwischen 0 und 100.

5.5. Ultraschallsensor

Ultraschallsensor:



Schließlich ist im Lego-Mindstorms-Kasten noch ein Ultraschallsensor enthalten. Mit ihm kannst Du Entfernungen bestimmen. Der Sensor sendet Ultraschallwellen aus und misst die Zeit, die diese Wellen benötigen, bis sie wieder in den Sensor zurück reflektiert werden. Darüber lässt sich die Entfernung zu einem Gegenstand in Sensorrichtung ermitteln.
Mit diesem Sensor kann der Roboter „sehen“ und somit Hindernisse erkennen ehe er dagegen fährt, wie dies beispielsweise mit dem Tastsensor der Fall ist.

Wir schreiben ein Programm, das den Roboter solange geradeaus fahren lässt, bis er 25cm vor einem Hindernis ist. Dann soll der Roboter ausweichen. Baue den Ultraschallsensor nach der Lego Anleitung von Seite 28 - 31 auf und befestige ihn am Roboter. Baue zuvor alle anderen Sensoren ab. Schließe dann den Ultraschallsensor am Eingang 1 an und schreibe das folgende Programm:

```
/* Ultraschallsensor */
task main()
{
	SetSensorLowspeed(IN_1);		// definiert Ultraschallsensor an Eingang 1
	while (true)				// bildet eine Endlosschleife
	{
		OnFwd(OUT_BC, 75);		// Roboter fährt vorwärts
		if (SensorUS(IN_1) < 25)	// Wenn die Entfernung kleiner als 25cm
		{
		OnRev(OUT_BC,75);		// dann fährt der Roboter rückwärts
		Wait(500);			// für 0,5 Sekunden
		OnFwd(OUT_B,75);		// und dreht
		Wait(200);			// für 0,2 Sekunden
		}
	}
}
```

Zunächst teilen wir dem NXT-Roboter mit, dass wir an Eingang 1 einen Ultraschallsensor angeschlossen haben. Dann wird die Endlosschleife (**while** (**true**)) gestartet und der Roboter fährt in Vorwärtsrichtung. Nähert sich der Roboter einem Gegenstand näher als 25cm, so beginnt das Ausweichmanöver. Danach fährt der Roboter wieder geradeaus, bis der nächste Gegenstand in 25cm auftaucht. Beachte: Der Wert des Ultraschallsensors (US) wird mit SensorUS(IN_1) und nicht mit Sensor(IN_1) abgefragt.

5.6. Zusammenfassung

In diesem Kapitel haben wir gesehen, wie wir die im Lego NXT Baukasten enthaltenen Sensoren benutzen können. Ebenso haben wir den sinnvollen Einsatz von if- und while-Anweisungen im Zusammenhang mit Sensoren kennen gelernt. Ändere die Programme in diesem Kapitel ab, um das Abfragen der verschiedenen Sensoren richtig zu verstehen. Du hast nun alle nötigen Kenntnisse, um dem Roboter ein komplexes Verhalten zu verleihen.

6. Lampen und Zusatz-Sensoren

6.1. Lampen

Dem LegoMindstorms-Kasten liegen außerdem drei Lampen bei, die mit dem unten abgebildeten Kabel am NXT angeschlossen werden können.

Stecke zunächst eine Lampe auf den Legobaustein, der sich an dem einen Ende des Kabels befindet und schließe das andere Ende des Kabels am Ausgang A des NXT an. Die Lampen werden wie die Motoren gesteuert, zum Beispiel über den Befehl OnFwd(OUT_A, 100). Dabei kann die Helligkeit der Lampe über den Zahlenwert in dem Befehl geregelt werden, 100 bedeutet maximale Helligkeit, der Wert 50 bedeutet, dass die Lampe schwächer leuchtet. Hier ist ein Programm, mit dem Du eine Lampe zum Blinken bringen kannst. Schließe dazu eine Lampe wie oben beschrieben am Ausgang A des NXT-Steins an.

```
/* Blinklicht */
task main()
{
	while (true)
	{
		OnFwd(OUT_A,100);
		Wait(500);
		OnFwd(OUT_A,50);
		Wait(500);
		Off(OUT_A);
		Wait(500);
	}
}
```

6.2. Lego Farbsensor

Lego-Farbsensor:

Mit dem Lego Farbsensor ist es möglich, zwischen 6 verschiedenen Farben zu unterscheiden. Dabei wird jeder Farbe eine Zahl zugeordnet, die man der folgenden Tabelle entnehmen kann.

Farbe:	Farbnummer:
schwarz	1
blau	2
grün	3
gelb	4
rot	5
weiß	6

Das Auslesen der Farbwerte zeigt das folgende Programm:

```
/* Lego Farbsensor - Farbnummer siehe Tabelle */
task main()
{
	SetSensorColorFull(IN_1);
	while (true)
	{
	TextOut(15,LCD_LINE1,"Farbnummer", true);
	NumOut(15, LCD_LINE2, Sensor(IN_1));
	TextOut(0,LCD_LINE4,"z. B.: rot = 5", false);
	TextOut(0,LCD_LINE5,"z. B.: blau = 2", false);
	Wait(100);
	}
}
```

Zunächst wird der Farbsensor am Eingang 1 festgelegt. Der Farbsensor liefert je nach Farbe einen Zahlenwert zwischen 1 und 6 an. Teste dieses Programm mit den beiden Bällen aus dem NXT-Kasten oder mit den farbigen Bausteinen aus dem Erweiterungskasten.

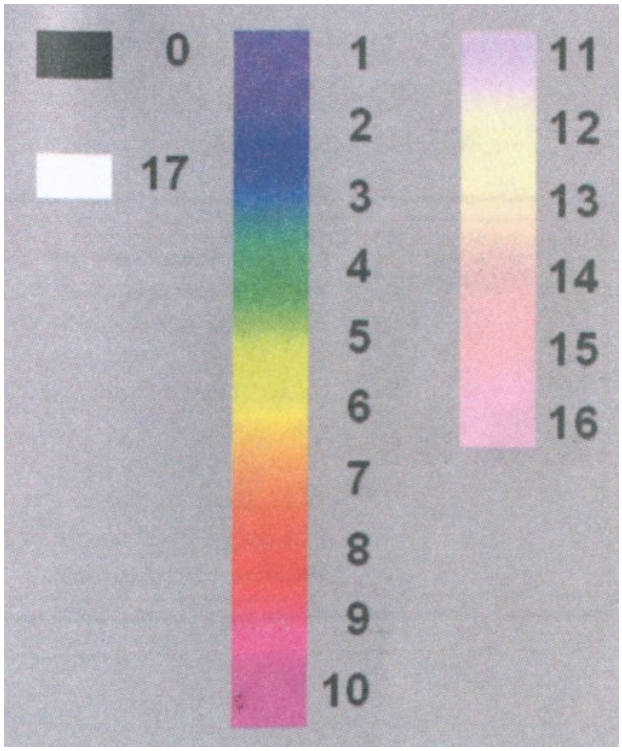
Zusätzlich besitzt der Lego Farbsensor die Fähigkeit, in den Farben rot, grün und blau zu leuchten. Das folgende Programm lässt den Lego Farbsensor nacheinander in den Farben blau, grün und rot für jeweils 0,1 Sekunden aufleuchten.

```
/* Lego Farbsensor - Blinklicht Blau - Gruen - Rot */
task main()
{
while(true)
	{
	SetSensorColorBlue(IN_1);
	Wait(100);
	SetSensorColorGreen(IN_1);
	Wait(100);
	SetSensorColorRed(IN_1);
	Wait(100);
	}
}
```

6.3. HiTechnic Farbsensor

Farbsensor:

Mit dem HiTechnic Farbsensor ist es möglich, zwischen 18 verschiedenen Farben zu unterscheiden. Dabei wird jeder Farbe eine Zahl zugeordnet, die man der folgenden Tabelle entnehmen kann.



Das Auslesen der Farbwerte zeigt das folgende Programm:

```
/* HiTechnic Farbsensor - Farbnummer siehe Tabelle */
int farbnummer, rot, gruen, blau;
task main()
{
       SetSensorLowspeed(IN_1);       // Definition des Farbsensors an Eingang 1
       while (true)
       {
       ReadSensorHTColor(IN_1, farbnummer, rot, gruen, blau);
       TextOut(15,LCD_LINE1,"Farbnummer", true);
       NumOut(15,LCD_LINE2, farbnummer, false);
       TextOut(0,LCD_LINE4,"z. B.: rot = 9", false);
       TextOut(0,LCD_LINE5,"z. B.: blau = 2", false);
       }
}
```

Zunächst werden in dem Programm mehrere Variablen definiert (farbnummer, rot, gruen und blau). Dann wird der Farbsensor am Eingang 1 festgelegt (identisch zum Ultraschallsensor!). Die Abfrage der Farbwerte geschieht über den Befehl:
ReadSensorHTColor(IN_1, farbnummer, rot, gruen, blau);
Dabei interessiert uns nur der Zahlenwert für die Variable „farbnummer“, die anderen Variablen sind uninteressant. Mit Hilfe der obigen Tabelle ist es nun möglich, die Farbnummer in die Farbe zu übersetzen. Lässt man das Programm auf dem NXT-Roboter laufen (HiTechnic-Farbsensor im Eingang 1), dann wird auf dem Display des NXT die Farbnummer angezeigt. Teste das Programm mit den beiden Bällen aus dem NXT-Kasten und den farbigen Bausteinen aus dem Erweiterungskasten.

6.4. HiTechnic Kreiselsensor (Gyrosensor)

Kreiselsensor:

Mit dem HiTechnic Kreiselsensor ist es möglich, Drehbewegungen festzustellen. Je nach Drehrichtung liefert der Sensor positive bzw. negative Werte. Das folgende Programm zeigt das Auslesen des Kreiselsensors.

```
/* HiTechnic Kreiselsensor (Gyrosensor) */
int offset = 400;
task main()
{
	while (true)
	{
		SetSensorHTGyro(IN_1); // Definition des Kreiselsensors an Eingang 1
		NumOut(0, LCD_LINE1, SensorHTGyro(IN_1, offset));
		Wait(100);
	}
}
```

Zunächst wird die Variable „offset“ mit dem Wert 400 festgelegt. Je nachdem, in welche Richtung der Kreiselsensor gedreht wird, liefert er Werte größer bzw. kleiner als 400. Der aktuelle Wert wird auf dem Display des NXT angezeigt. Bewege den NXT mit dem Kreiselsensor in verschiedene Richtungen und beobachte den Wert.

6.5. HiTechnic Beschleunigungssensor

Beschleungigungssensor:

Der HiTechnic Beschleunigungssensor (Acceleration-Sensor) misst Beschleunigungen in den drei Raumrichtungen. Die Beschleunigung wird in den drei Achsen x-, y- und z-Richtung gemessen. Mit Hilfe dieses Sensors kann die Beschleunigung des NXT Roboters in Bereich von -2g bis 2g gemessen werden mit einer Auflösung von 200 Einheiten pro g (g: Erdbeschleunigung).

```
/* HiTechnic Beschleungigungssensor */
task main()
{
  SetSensorLowspeed(IN_1);

  int x;
  int y;
  int z;

  while (true)
     {
          ReadSensorHTAccel(IN_1, x , y , z);
          NumOut(0,LCD_LINE1,x);
          NumOut(0,LCD_LINE2,y);
          NumOut(0,LCD_LINE3,z);
          Wait(300);
          ClearScreen();
     }
}
```

Zunächst wird in dem Programm der Beschleunigungssensor am Eingang 1 festgelegt (identisch zum Ultraschallsensor!). Dann werden die drei Variablen x, y und z für die drei Richtungen im Raum definiert. Die Abfrage des Beschleunigungssensors geschieht über den Befehl:
ReadSensorHTAccel(IN_1, x, y, z);
Dabei sind x, y und z die Beschleunigungen in den drei Raumrichtungen. Diese werden nach dem Auslesen über die Befehle NumOut auf dem Display des NXT-Roboters ausgegeben. Schließe den Beschleunigungssensor am Eingang 1 des NXTs an und lade das Programm auf den Roboter. Bewege nach dem Start des Programms den Sensor und beobachte die angezeigten Werte.

7. Töne und Musik

Der NXT besitzt einen eingebauten Lautsprecher. Dieser kann sowohl Töne, als auch Musikdateien wiedergeben. Dies ist vor allem nützlich um Deine Programme zu testen. Wenn Du z. B. wissen möchtest, wann der Roboter eine spezielle Stelle Deines Programms abarbeitet, kannst Du diese Stelle im Programm durch einen Ton „markieren“. Es kann aber auch sonst witzig sein, wenn Dein Roboter Musik macht oder spricht, während er gerade umher fährt.

7.1. Spielen von Tönen

Um einfache Töne abzuspielen kannst Du den Befehl PlayTone(*Tonhöhe, Dauer*) verwenden. Im Anschluss an diesen Befehl benötigst Du jeweils einen Wait-Befehl.

```
/* einfache Töne*/
task main ()
{
while (true)
        {
        PlayTone(262,400);
        Wait(500);
        PlayTone(294,400);
        Wait(500);
        PlayTone(330,400);
        Wait(500);
        PlayTone(294,400);
        Wait(500);
        }
}
```

Für die Tonhöhe (Frequenz) ist folgende Tabelle hilfreich.

Sound	3	4	5	6	7	8	9
H	247	494	988	1976	3951	7902	
A#	233	466	932	1865	3729	7458	
A	220	440	880	1760	3520	7040	14080
G#		415	831	1661	3322	6644	13288
G		392	784	1568	3136	6272	12544
F#		370	740	1480	2960	5920	11840
F		349	698	1397	2794	5588	11176
E		330	659	1319	2637	5274	10548
D#		311	622	1245	2489	4978	9956
D		294	587	1175	2349	4699	9398
C#		277	554	1109	2217	4435	8870
C		262	523	1047	2093	4186	8372

Um einen komplexeren Ton zu erzeugen, kannst Du folgenden Befehl verwenden:
PlayToneEx(*Tonhöhe, Dauer, Lautstärke, Ton ausklingen lassen?*);
Dieser Befehl hat vier Parameter in der Klammer. Die ersten beiden Werte sind wieder die Tonhöhe und die Dauer des Tons, der dritte ist die Lautstärke. Die Lautstärke kann im Bereich zwischen 0 (leise) und 4 (laut) gewählt werden. Die vierte Angabe in der Klammer bietet die Möglichkeit, den Ton ausklingen zu lassen (*true*) oder zu stoppen (*false*). Das folgende Programm zeigt eine Melodie mit dem PlayToneEx-Befehl:

```
/* komplexere Töne */
#define VOL 3
task main()
{
       PlayToneEx(262,400,VOL,FALSE);
       Wait(500);
       PlayToneEx(294,400,VOL,FALSE);
       Wait(500);
       PlayToneEx(330,400,VOL,FALSE);
       Wait(500);
       PlayToneEx(294,400,VOL,FALSE);
       Wait(500);
       PlayToneEx(262,1600,VOL,FALSE);
       Wait(2000);
}
```

Der NXT-Roboter wartet nicht darauf, bis ein Ton „fertig“ ist. Aus diesem Grund ist in dem Programm hinter jedem Ton ein Wait-Befehl.
Du kannst „Musikstücke“ leicht selbst komponieren, indem Sie das *Brick Piano*, welches ebenfalls in BricxCC enthalten ist nutzen (Tools > Brick Piano).

7.2. Spielen von „Musik“

BricxCC besitzt ein Konvertierungsprogamm, welches wav-Dateien in für den NXT verwendbare rso-Dateien konvertiert. Dies kann über das Menü Tools > Sound Conversion ausgeführt werden. Mit einem weiteren Programm (NXT Explorer) können die rso-Dateien auf den NXT übertragen und dort gespeichert werden. Dieses kann über das Menü Tools > NXT Explorer ausgeführt werden. Mit dem folgenden Befehl können die rso-Dateien aus einem Programm heraus abgespielt werden: PlayFileEx(*Dateiname, Lautstärke, Wiederholung*)

```
/* Musik */
#define VOL 3
task main()
{
       PlayFileEx("Woops.rso",4,false);
       Wait(600);
}
```

Beachte, dass der Dateinamen in Anführungsstrichen stehen muss. Die Lautstärke geht wieder von 0 bis 4. Soll die Sound-Datei ständig wiederholt werden, muss am Ende *true* stehen, andernfalls *false*. Der Wait-Befehl gibt an, wie lange die Datei wiedergegeben werden soll.

8. Das Display des NXT

Der NXT-Stein besitzt ein schwarz-weiß LCD-Display mit einer Auflösung von 100x64 Bildpunkten (Pixels). Das Display kann hilfreich sein, um z. B. die aktuellen Sensorwerte anzuzeigen.

8.1. Überblick über die Display-Befehle

Es gibt zahlreiche Befehle um Text, Zahlen, Punkte, Linien, Rechtecke, Kreise und sogar Bitmap Bilder (ric-Format) auf dem Display anzuzeigen. Das nachfolgende Beispiel versucht all diese Möglichkeiten abzudecken. Der Punkt (0,0) stellt den äußersten Bildpunkt links unten dar. Die Zeilen im Display werden mit LCD_Line1 – 8 bezeichnet.

```
/* Display */

#define X_MAX 99
#define Y_MAX 63
#define X_MID (X_MAX+1)/2
#define Y_MID (Y_MAX+1)/2

task main()
{
int i = 1234;
TextOut(15,LCD_LINE1,"Display", true);
NumOut(60,LCD_LINE1, 1, false);
PointOut(1,Y_MAX-1);
PointOut(X_MAX-1,Y_MAX-1);
PointOut(1,1);
PointOut(X_MAX-1,1);
Wait(200);
RectOut(5,5,90,50);
Wait(200);
LineOut(5,5,95,55);
Wait(200);
LineOut(5,55,95,5);
Wait(200);
CircleOut(X_MID,Y_MID-2,20);
Wait(800);
ClearScreen();
GraphicOut(30,10,"faceclosed.ric"); Wait(500);
ClearScreen();
GraphicOut(30,10,"faceopen.ric");
Wait(1000);
}
```

All diese Funktionen sind nahezu selbsterklärend, hier aber ein Beschreibung der dazugehörigen Parameter:

ClearScreen()
Löscht den Bildschirminhalt.

NumOut(*x-Koordinate, y-Koordinate, Zahl, Vorheriges löschen*)
Gibt eine Zahl an der Position (x,y) auf dem Display aus. Soll der Bildschirm an dieser Position vorher gelöscht werden muss bei *Vorheriges löschen* true eingetragen werden, andernfalls false.

TextOut(*x-Koordinate, y-Koordinate, Text, Vorheriges löschen*)
Wie der Befehl NumOut() nur für Text.

GraphicOut(*x-Koordinate, y-Koordinate, Datei, Vorheriges löschen*)
Gibt das Bild "Dateinamen.ric" bei den Koordinaten(x/y) auf dem Display aus.

CircleOut(*x-Koordinate, y-Koordinate, Radius, Vorheriges löschen*)
Zeichnet einen Kreis mit Mittelpunkt (x,y) und Radius.

LineOut(*x1, y1, x2, y2, Vorheriges löschen*)
Zeichnet eine Linie vom Punkt (x1,y1) zum Punkt (x2,y2).
PointOut(*x-Koordinate, y-Koordinate, Text, Vorheriges löschen*)
Zeichnet am Punkt (x/y) einen Punkt auf den Bildschirm.

RectOut(*x-Koordinate, y-Koordinate, Länge, Breite, Vorheriges löschen*)
Zeichnet ein Rechteck mit der linken unteren Ecke (x,y) und der angegebenen Länge und Breite.

ResetScreen()
Setzt den Bildschirm zurück.

8.2. Sensorwerte direkt auf das Display ausgeben

Die Ausgabe von Zahlenwerten kann zur Einstellung der Sensoren oder zur Fehlersuche hilfreich sein. Das folgende Programm dient dazu, den NXT als Entfernungsmesser zu verwenden. Schließe hierzu den Ultraschallsensor an den NXT und verbinde ihn mit Eingang 1.

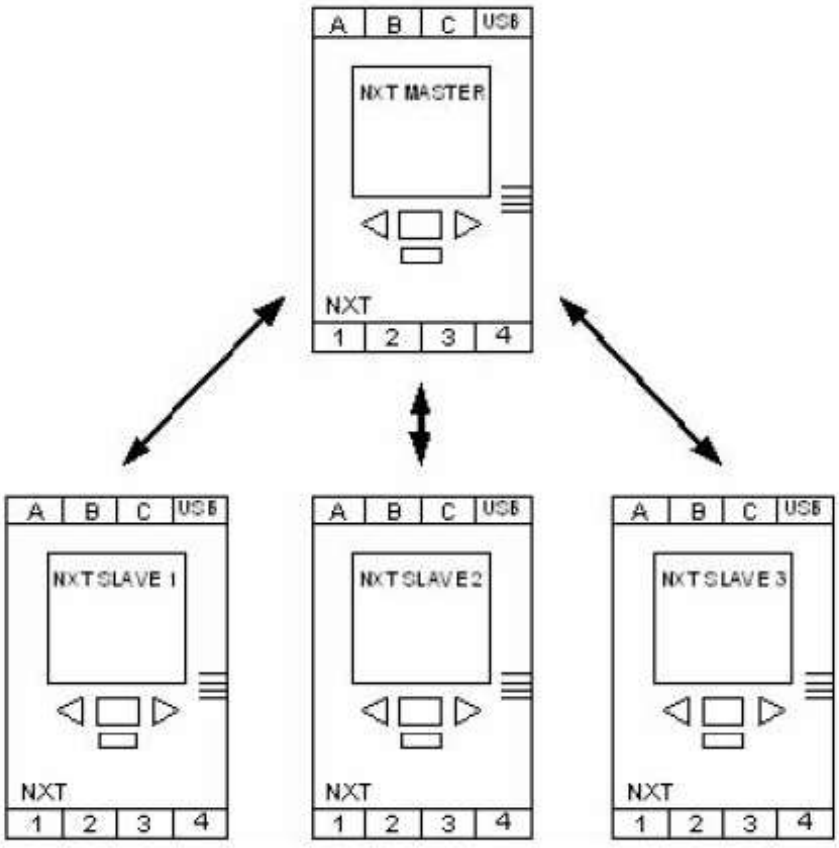
```
/* Entfernungsmesser */
task main()
{
SetSensorLowspeed(IN_1);                // Definiert Ultraschallsensor an Eingang 1
while (true)                            // bildet eine Endlosschleife
       {
       NumOut(80,LCD_LINE4, SensorUS(IN_1), true);   // Die Entfernung wird auf dem
       Wait(100);                                    // Display ausgegeben

       }
}
```

Verändere die Werte beim Befehl NumOut() (z.B.: x, y und true, false) und beobachte was sich dadurch auf dem Display ändert.
Durch den Anbau anderer Sensoren und umschreiben des Programms lassen sich so die Sensorwerte direkt auf der Anzeige des NXT ausgeben.

9. Kommunikation zwischen Robotern

Die NXT-Roboter können über Bluetooth (Radiowellen) Daten (z.B. Zahlen) austauschen. Außerdem können mit Hilfe der Bluetooth-Kommunikation mehrere Roboter zusammenarbeiten oder Du kannst einen sehr komplexen Roboter konstruieren, der zwei bis vier NXT-Steine nutzt. Bevor zwei NXT-Roboter miteinander kommunizieren können, müssen die beiden drahtlos miteinander verbunden werden. Dies erfolgt über das NXT-Menü. Gehe hierzu zunächst im Menü des NXT-Steins auf den Punkt Bluetooth und schalte das Bluetooth ein. Sobald das Bluetooth eingeschaltet ist befindet sich links oben im Display des NXT-Steins das Bluetooth-Zeichen. Jetzt kann der NXT über die Suchfunktion „Search“ andere NXTs über Bluetooth finden (bei diesen muss natürlich auch Bluetooth eingeschaltet sein). Damit zwei NXT-Roboter über Bluetooth miteinander kommunizieren können, müssen die beiden Roboter vorher über dieses Bluetooth-Menü verbunden werden. Der NXT, der die Verbindung zu dem anderen NXT aufbaut wird Master genannt. Der NXT, der diese Verbindungsaufforderung annimmt wird heißt 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 1 – 10) adressiert werden. Bei „Connections“ können im NXT-Menü die derzeit aktiven Bluetooth-Verbindungen angezeigt werden. Hier also die Struktur des NXT-Bluetooth-Netzwerks:



9.1. Senden und Empfangen von Zahlen

Der Master- und der Slave-Roboter können sich gegenseitig Zahlen zusenden. Dazu müssen die beiden Roboter über Bluetooth miteinander verbunden sein. Im folgenden Beispiel ist der Master über den Kanal 1 mit dem Slave verbunden. Der Slave ist mit dem Master dann automatisch über den Kanal 0 verbunden. Schalte also an beiden Robotern die Bluetooth-Funktion ein, suche mit dem Master-NXT den richtigen Slave und verbinde Dich mit ihm über den Kanal 1. Überspiele dann die beiden unten stehenden Programme auf den Master-NXT bzw. Slave-NXT. Starte dann zuerst das Programm auf dem Master-NXT, dann auf dem Slave-NXT.

```
/* Master - Verbunden über Kanal 1 mit dem Slave */
int zahl;
task main()
{
zahl = 1;
while(true)
{
zahl = Random(4);             // Generieren der Zufallszahl (0; 1; 2 oder 3)
SendRemoteNumber(1,5,zahl);  //Senden der Zufallszahl über Kanal 1 an den Briefkasten 5
TextOut(0,LCD_LINE1,"MASTER");    // Display des Masters
TextOut(0,LCD_LINE2,"Zahl");
NumOut(33,LCD_LINE2,zahl);
TextOut(45,LCD_LINE2,"versendet");
Wait(1000);
}
}
```

```
/* Slave - Verbunden über Kanal 0 mit dem Master */
int zahl;
task main()
{
zahl = 1;
while(true)
{
ReceiveRemoteNumber(5, true, zahl); //Empfangen der Zufallszahl über den Briefkasten 5
TextOut(0,LCD_LINE1,"SLAVE");       // Display des Masters
TextOut(0,LCD_LINE2,"Zahl");
NumOut(33,LCD_LINE2,zahl);
TextOut(45,LCD_LINE2,"empfangen");
Wait(1000);
}
}
```

Die beiden Befehle zum Senden bzw. Empfangen der Zahlen sind:
SendRemoteNumber(*Kanal, Briefkasten, zu versendende Zahl*)
ReceiveRemoteNumber(*Briefkasten, Leeren des Briefkastens ?, empfangene Zahl*)

10. Mehr über Motoren

Es gibt eine Reihe von zusätzlichen Motorbefehlen, die Du verwenden kannst, um die Motoren noch genauer zu steuern. Diese Steuerungsbefehle werden jetzt vorgestellt.

10.1. Sachtes Bremsen

Wenn Du den Off()-Befehl verwendest, stoppt der Motor sofort, der Motor bremst und hält die Position. Es ist aber auch möglich die Motoren sanft abzubremsen, quasi auslaufen zu lassen. Hierfür kannst Du die Befehle Float() oder Coast() verwenden. Hier ist ein einfaches Beispiel, dass die Verwendung des Float-Befehls veranschaulicht. Bei dem ersten Halt bremst der Roboter, beim zweiten Halt lässt er die Motoren auslaufen. Beachte den Unterschied, dieser ist je nach Art und Bauweise des Roboters beträchtlich.

```
/* Sofortiger Stopp, langsamer Stopp */
task main()
{
OnFwd(OUT_BC, 75);
Wait(500);
Off(OUT_BC);                //stoppt mit Bremse
Wait(1000);
OnFwd(OUT_BC, 75);
Wait(500);
Float(OUT_BC);              //stoppt, indem die Motoren langsam auslaufen
}
```

10.2. Weitere Motorbefehle

Der Befehl OnFwd() und OnRev() sind die einfachsten Möglichkeiten Motoren vorwärts bzw. rückwärts drehen zu lassen. Der NXT bietet aber auch die Möglichkeit, dass man zwei Motoren besser auf einander abstimmen kann. Man kann die Motoren **geregelt** zueinander betreiben. Dazu wird der Befehl OnFwdReg(*Ausgang, Geschwindigkeit, Betriebsmodus*) verwendet. Es gibt drei Betriebsmodi:

Betriebsmodus:	**Eigenschaft:**
OUT_REGMODE_IDLE	keine Regulation
OUT_REGMODE_SPEED	Regelt die Geschwindigkeit der Motoren: Die Motoren versuchen ihre Drehgeschwindigkeit zu halten.
OUT_REGMODE_SYNC	Synchronisiert die Drehung der Motoren: Die Drehgeschwindigkeit mehrer Motoren wird aufeinander angepasst.

Dies ist auch für die Rückwärtsfahrt möglich mit dem Befehl OnRevReg(*Ausgang, Geschwindigkeit, Betriebsmodus*). Um die verschiedenen Betriebsmodi kennen zu lernen dient das folgende Programm. Halte bei jedem Betriebsmodus kurz ein Rad des Roboters fest und lasse es dann wieder los. Beobachte dabei, wie sich das andere Rad verhält. Im Idle-Modus sollte das andere Rad unbeeinflusst weiterlaufen, im Speed-Modus steigt die Motorleistung, wenn du das Rad festhältst. Im Sync-Modus stoppt auch das andere Rad, die beiden Räder laufen jetzt synchron zueinander.

```
/*  Geregelte Betriebsmodi für Motoren   */
task main()
{
OnFwdReg(OUT_BC,50,OUT_REGMODE_IDLE);
Wait(2000);
Off(OUT_BC);
PlayTone(4000,50);
Wait(1000);
OnFwdReg(OUT_BC,50,OUT_REGMODE_SPEED);
Wait(2000);
Off(OUT_BC);
PlayTone(4000,50);
Wait(1000);
OnFwdReg(OUT_BC,50,OUT_REGMODE_SYNC);
Wait(2000);
Off(OUT_BC);
}
```

Außerdem kannst du die Motoren **synchronisieren**. Dies ist hilfreich, wenn der Roboter exakt geradeaus fahren soll. Der Befehl dazu lautet: OnFwdSync(*Ausgang, Geschwindigkeit, Versatz*). Dabei können für Versatz die Werte von -100 bis 100 eingetragen werden (von -100% bis 100%). Soll der Roboter also exakt geradeaus laufen, dann gibst du als Versatz 0 ein. Das folgende Programm zeigt, wie der synchronisierte Betriebsmodus der Motoren funktioniert:

```
/*  Synchronisierter Betriebsmodus für Motoren   */
task main()
{
OnFwdSync(OUT_BC,50,0);          // exakte Geradeausfahrt
Wait(2000);
Off(OUT_BC);
PlayTone(4000,50);
Wait(1000);
OnFwdSync(OUT_BC,50,10);         // Rechtskurve, ein Motor dreht sich
Wait(2000);                      // mit der Geschwindigkeit 10 langsamer
Off(OUT_BC);
PlayTone(4000,50);
Wait(1000);
OnFwdSync(OUT_BC,50,-10);        // Linkskurve, der andere Motor dreht
Wait(2000);                      // sich mit der Geschwindigkeit 10
Off(OUT_BC);                     // langsamer
PlayTone(4000,50);
Wait(1000);
}
```

Variiere die Werte für den Versatz im obigen Beispiel auch auf die Werte 50 (-50) und 100 (-100) um die Funktion des Befehls OnFwdSync besser zu verstehen. Für die synchronisierte Rückwärtsfahrt gibt es den entsprechenden Befehl OnRevSync.
Schließlich kannst du die Motoren um einen bestimmten **Winkel drehen bzw. rotieren** lassen: RotateMotor(*Ausgang, Geschwindigkeit, Drehwinkel*). Bei einem Drehwinkel von 360° dreht sich das Rad genau einmal ganz herum. Soll sich ein Rad genau einmal rückwärts drehen ist der Drehwinkel -360°. Überprüfe dies mit dem folgenden Programm. Merke dir dazu eine Stelle des Rades (nahe der Achse haben die Roboterfelgen eine Markierung).

```
/*  Drehung um einen bestimmten Winkel   */
task main()
{
RotateMotor(OUT_BC, 50,360);     // dreht die Motoren um eine Drehung
Wait(5000);                      // in Vorwärtsrichtung
RotateMotor(OUT_BC, 50,-360);    // dreht die Motoren um eine Drehung
Off(OUT_BC);                     // in Rückwärtsrichtung
}
```

Auch bei der Drehung um einen Winkel können Motoren zueinander synchronisiert werden. Dazu dient der Befehl RotateMotorEx(*Ausgang, Geschwindigkeit, Drehwinkel, Versatz, Synchronisation, Stopp*). Für Synchronisation und Stopp sind „true“ oder „false“ einzutragen. Für Synchronisation bedeutet „true“, dass die Motoren an den Ausgängen mit dem angegebenen Versatz synchronisiert werden sollen. Ist Stopp auf „true“ gesetzt, so werden die Motoren nach der Rotation abgebremst, bei „false“ werden sie auslaufen gelassen. Teste hierzu das Programm:

```
/*  Synchronisierte Drehung um einen bestimmten Winkel   */
task main()
{
RotateMotorEx(OUT_BC, 50, 360, 0, true, true);
RotateMotorEx(OUT_BC, 50, 360, 40, true, true);
RotateMotorEx(OUT_BC, 50, 360, -40, true, true);
RotateMotorEx(OUT_BC, 50, 360, 100, true, true);
}
```

10.3. Zusammenfassung

In diesem Kapitel hast Du weitere Motorbefehle kennen gelernt: Float(), der den Motor langsam stoppt, OnFwdReg(), und OnFwdSync(), die regulierte bzw. synchronisierte Betriebsmodi der Motoren erlauben. RotateMotor() and RotateMotorEx() sind Befehle, mit denen wir Motoren um einen bestimmten Winkel drehen lassen können.

11. Mehr über Sensoren

In Kapitel 5 haben wir die grundlegenden Aspekte der Verwendung von Sensoren kennen gelernt. Aber es gibt noch viel mehr Möglichkeiten, Sensoren einzusetzen. In diesem Kapitel diskutieren wir den Unterschied zwischen Sensor-Modus und Sensor-Typ.

11.1. Sensor-Typ und Sensor-Modus

Der SetSensor()-Befehl, den wir aus Kapitel 5 kennen, erledigt zwei Dinge: Er bestimmt den Typ des Sensors und er legt den Modus, in dem der Sensor arbeiten soll fest. Durch die separate Festelegung des Modus und des Typs kann das Verhalten des Sensors präziser gesteuert werden, was für bestimmte Anwendungen sehr nützlich sein kann.
Mit dem Befehl SetSensorType(*Eingang, Typ*) wird der **Sensortyp**, der am NXT verwendet werden soll, eingestellt. Es gibt insgesamt vier Eingänge IN_1, IN_2, IN_3 und IN_4.

Im Lego-Mindstorm Kasten sind vier verschiedene Sensortypen enthalten:

Berührungssensor:	SENSOR_TYPE_TOUCH
Lichtsensor:	SENSOR_TYPE_LIGHT (LED aus)
	SENSOR_TYPE_LIGHT_ACTIVE (LED an)
Mikrofonsensor:	SENSOR_TYPE_SOUND_DB
Ultraschallsensor:	SENSOR_TYPE_LOWSPEED

Das Einstellen des Sensor-Typs ist im Speziellen wichtig, um anzugeben, welche Leistung der Sensor benötigt (z.B. um die LED des Lichtsensors zu betreiben), oder um dem NXT-Stein mitzuteilen, dass es sich um einen digitalen Sensor handelt (z.B. Ultraschallsensor).

Der Sensor-Modus wird mit dem Befehl SetSensorMode(*Eingang, Modus*) bestimmt. Es gibt verschiedene Betriebsmodi.

1) SENSOR_MODE_RAW
Ein wichtiger Betriebsmodus ist SENSOR_MODE_RAW. In diesem Modus liegt der Wert beim Abfragen des Sensors zwischen 0 und 1023. Es ist sogenannte Rohwert des Sensors. Die Interpretation dieses Wertes ist abhängig vom jeweiligen Sensor. Zum Beispiel der Tastsensor: Ist der Tastsensor nicht gedrückt, liefert er einen Wert in der Nähe von 1023. Ist der Sensor stark gedrückt liefert er einen Wert um die 50. Wird der Sensor nur leicht gedrückt, gibt er einen Wert zwischen 50 und 1000 zurück. Wenn Du also den Berührungssensor auf den Rohmodus setzt, kannst Du herauszufinden, ob der Tastsensor nur leicht gedrückt wurde. Wird der Lichtsensor im Rohmodus betrieben, liefert er Werte zwischen 300 (sehr hell) bis 800 (sehr dunkel). Dadurch können wesentlich genauere Werte als mit dem SetSensor()-Befehl ermittelt werden.

2) SENSOR_MODE_BOOL
Der zweite Sensor-Modus ist SENSOR_MODE_BOOL. Dieser Modus liefert die Rückgabewerte 0 oder 1. Ist der gemessene Rohwert über 562, wird eine 0 zurückgegeben, ansonsten eine 1. SENSOR_MODE_BOOL ist die Standardeinstellung für den Tastsensor.

3) SENSOR_MODE_PERCENT
Der Befehl SENSOR_MODE_PERCENT wandelt den Rohwert in einen Prozentwert zwischen 0 und 100 um. SENSOR_MODE_PERCENT ist die Standardeinstellung für den Licht-Sensor.

4) und 5) SENSOR_MODE_EDGE und SENSOR_MODE_PULSE
SENSOR_MODE_EDGE und SENSOR_MODE_PULSE zählen Übergänge von einem niedrigen Rohwert zu einem höheren, bzw. von einem höheren zu einem niedrigeren Rohwert. Wird beispielsweise der Tastsensor gedrückt, hat dies einen Übergang von einem hohen Rohwert zu

einem niedrigeren Rohwert zur Folge. Wird der Tastsensor wieder losgelassen, erfolgt ein Übergang in die andere Richtung. Wird der Sensor in den Modus SENSOR_MODE_PULSE gesetzt, werden nur Übergänge von niedrig zu hoch gezählt. Damit würde das Drücken und Loslassen des Tastsensors als ein Übergang gezählt. Wir der Sensor dagegen im SENSOR_MODE_EDGE betrieben, werden beide Übergänge gezählt. Dieser Modus kann z.B. benutzt werden, um zu zählen wie oft der Tastsensor gedrückt wurde. Zusammen mit dem Lichtsensor kann dieser Modus dazu verwendet werden, um zu zählen wie oft eine Lampe ein- und ausgeschaltet wurde oder wie oft eine schwarze Linie überfahren wurde. Natürlich sollte man beim Zählen der Übergänge auch in der Lage sein, den „Zähler“ wieder auf 0 zu setzen. Hierfür kann der Befehl ClearSensor()genutzt werden.

Das folgende Programmbeispiel steuert einen Roboter über den Tastsensor. Schließe den Tastsensor mit einem langen Kabel an Eingang 1 des NXT an. Wenn der Tastsensor schnell zweimal hintereinander gedrückt wurde, fährt der Roboter vorwärts. Wird der Tastsensor einmal gedrückt, hält der Roboter an.

```
/*  Sensorart und Sensorbetriebsmodus beim Berührungssensor   */
task main()
{
      SetSensorType(IN_1, SENSOR_TYPE_TOUCH);     // Sensortyp wird festgelegt
      SetSensorMode(IN_1, SENSOR_MODE_PULSE);     // Betriebsmodus PULSE wird festgelegt
      while(true)
      {
            ClearSensor(IN_1);                      // Sensorzähler wird zurückgesetzt
            until (SENSOR_1 > 0);                   // Wenn der Tastsensor min. 1 mal
            Wait(500);                              // gedrückt wird, dann
            if (Sensor(IN_1) == 1) {Off(OUT_BC);}   // Einmal drücken: Ausschalten
            if (Sensor(IN_1) == 2) {OnFwd(OUT_BC, 75);} // Zweimal drücken: Fahren
      }
}
```

Beachte, dass wir zuerst den Typ des Sensors und anschließend den Modus definiert haben. Dies ist wichtig, da der Typ des Sensors auch den jeweiligen Modus beeinflussen kann.

11.2. Zusammenfassung

In diesem Kapitel hast Du zusätzliche Eigenschaften der Sensoren kennen gelernt. Du hast gesehen, wie unabhängig voneinander der Sensor-Typ und Sensor-Modus bestimmt werden können und wie diese die vom Sensor zurückgegebenen Daten beeinflussen.