Intelligente Programme für den NXT

Dieses Tutorial ist für alle die gedacht, die den Einstieg durch meine Programmiereinführung bereits hinter sich haben und nun intelligente Programme für den NXT schreiben wollen.

Teil 1: Displayausgaben

Fangen wir mit einem interessanten Thema an: Dem Display des NXT. Dieses ist groß genug um darauf Texte und Zahlen auszugeben und eignet sich daher hervorragend, um während das Programm läuft den Status der Sensoren auszulesen. Dies werden wir im ersten Beispiel auch tun und dabei ganz nebenbei noch den Licht- und den Ultraschallsensor kennen lernen:

```
task main()
{
 // Sensoren initialisieren
 SetSensorLowspeed(S4); // Der Ultraschallsensor
 SetSensorTouch(S1); // Berührungssensor
 SetSensorSound(S2); // Soundsensor
 SetSensorLight(S3); // Lichtsensor

 ClearScreen ();

 // Hauptschleife
 while (true)
 {
  TextOut (0, LCD_LINE1, "Beruehrung:");
  NumOut (1, LCD_LINE2, Sensor(S1));

  TextOut (0, LCD_LINE3, "Geraeusche:");
  NumOut (1, LCD_LINE4, Sensor(S2));

  TextOut (0, LCD_LINE5, "Licht:");
  NumOut (1, LCD_LINE6, Sensor(S3));

  TextOut (0, LCD_LINE7, "Ultraschall:");
  NumOut (1, LCD_LINE8, SensorUS(S4));

  Wait(250); // Bildschirnzeige wechselt sonst zu schnell

  ClearScreen (); // Den Bildschirm löschen
 }
}
```

Der Lichtsensor gibt einen Wert von 0 bis 100 aus, je höher, desto heller das einfallende Licht. Den Ultraschallsensor initialisiert man durch die Funktion „SetSensorLowspeed“, sie steht für einen besonderen Sensorentyp zu dem auch der Ultraschallsensor gehört. Seinen Wert, der für eine Entfernung steht, ließt man mit „SensorUS“ aus.

Richtig interessant sind natürlich die das Display betreffenden Funktionen. „TextOut“ und „NumOut“ ähneln sich dabei sehr:
Der erste Parameter gibt die x-Koordinate auf dem Display an, je höher, desto weiter rechts liegt auf dem Display der Anfangspunkt des Textes / der Zahl.
Der zweite Parameter ist die y-Koordinate und gibt die horizontale Position an. Anstatt von Zahlen verwendet man hier aber eher die „LCD_LINE“ Kostanten. Mit diesen kann man acht definierte Linien auf dem Display anwählen, in den Werten ist nämlich schon die Buchstabenhöhe miteinbezogen. „LCD_LINE1“ steht für die erste, „LCD_LINE2“ für die zweite Linie usw.

Der Dritte Parameter erwartet bei „NumOut“ einen long, also eine Ganzzahl, und bei „TextOut“ einen „string“, eine Zeichenkette. Letztere kann ein beliebiger Text sein, allerdings muss er in Anführungszeichen gesetzt werden.

Um die Werte der Sensoren auszugeben werden bei den „NumOut“-Befehelen einfach die „Sensor“-Funktionen benutzt. Sie geben die Sensorwerte aus.

Als letzte neue Funktion enthält das Beispielprogramm „ClearScreen“. Diese Funktion löscht einfach die derzeitige Bildschirmanzeige, ohne sie könnten sonst Darstellungsfehler auftreten.

Teil 2: Mehr über Funktionen

Bevor wir aber wirklich durchstarten können, kommt jetzt leider noch ein größerer Teil mit viel theoretischem Wissen. Die Aufteilung von Programmcode in verschiedene Funktionen und Tasks ist aber sehr wichtig, nur so kann man später echte Monsterprogramme verwalten. Also los geht’s!

Schreibt man für den NXT Programme, so ist es oft sinnvoll eine Reihe von Befehlen sozusagen „auszulagern“. Zum Beispiel könnte sich in einem Programm ein Teil um einen bestimmten Sensor kümmern und auf Veränderungen seiner Werte reagieren, und ein anderer Teil könnte sich um einen anderen Sensor kümmern. Nun wird der Code aber schnell unübersichtlich wenn man alles in den „main“-Task schreibt, und deshalb verteilt man alle Programmteile auf verschiedene Funktionen. Um das zu tun, müssen wir aber zunächst lernen, wie man seine eigenen Funktionen schreibt.

Von den Template-Funktionen wissen wir bereits, dass eine Funktion in der Programmierung:

1. einen Namen hat,
2. beliebig viele Parameter erwarten,
3. und auch einen Wert zurückgeben kann.

Möchte man eine eigene Funktion schreiben, so tut man das, indem man zuerst einen Datentyp notiert (dieser legt fest welche Werte eine Funktion zurückgeben kann), dann einen Namen für die Funktion nennt und zuletzt in runden Klammern die nötigen Parameter auflistet. Auch diese mit Datentyp und einem Namen. Letzterer ist nötig, damit später innerhalb der Funktion auf den übergebenen Wert zugegriffen werden kann.
Beispiel:

```
int MeineFunktion (int Parameter1, bool Parameter 2)
{
 // Hier steht, was die Funktion tut!
}
```

Wie aber gibt man in einer Funktion denn einen Wert zurück? Dies geschieht mit dem „return“-Befehl. Dieser beendet auch gleichzeitig die Funktion, selbst wenn danach noch Befehle stehen.

```
int Addition (int Summand1, int Summand2)
{
 return Summand1+Summand2;
}
```

Was hinter “return” steht, wird dann also zurückgegeben.
In einem Programm sähe das so aus:

```
int Addition (int Summand1, int Summand2)
{
 return Summand1+Summand2;
}

task main ()
{
 int ErsterSummand = 4;

 // Jetzt rufen wir unsere Funktion auf!
 int Ergebnis = Addition (ErsterSummand, 5);

 /* In der Variable "Ergebnis" wird jetzt "9" gespeichert sein.
 Man kann Parameter als Variable oder auch direkt als Wert übergeben. */
}
```

Zum Aufrufen wird also einfach der Funktionsname geschrieben.

Oft braucht eine Funktion gar keine Parameter (dann lässt man beim erstellen der Funktion einfach die runden Klammern leer), oder sie muss keinen Wert zurückgeben. Ist Letzteres der Fall, so schreibt man anstatt eines Datentyps einfach das Schlüsselwort „void“.

```
void FunktionMitLangemNamenDieNichtsZuruekgibt()
{
 // VieleTolleSachenMachen :-)
}
```

Schreibt man nun mehrere Funktionen und ruft sie nacheinander auf, so werden sie auch nacheinander ausgeführt, also zunächst die erste bis zum Ende, dann die zweite usw.

Es gibt aber auch eine Möglichkeit, mehrere Aufgaben gleichzeitig erledigen zu lassen – und hier kommt dann auch ein Punkt, an dem ich eine Aussage aus dem ersten Teil berichtigen muss: Ein „task“ ist tatsächlich keine Funktion, sondern ein sogenannter Thread. Der NXT unterstützt multi-threading und kann daher mehrere tasks (alias „threads“) gleichzeitig ausführen.

Tasks werden durch das Schlüsselwort „task“, einem Namen und zwei runden Klammern (allerdings keine Parameter, die können einem Task nicht übergeben werden) erstellt:

```
task MeinThread ()
{
 // Hier steht, was der Task tut!
}
```

Gestartet wird ein Task durch das „start“-Kommando (mit dem Tasknamen, aber ohne Klammern!):

```
start MeinThread;
```

Vorzeitig gestoppt werden, kann er durch „stop":

```
stop MeinThread;
```

Folgendes Programm zeigt neben der Verwendung von Tasks auch gleich noch eine neue Template-Funktion, mit der man den NXT Töne erzeugen lassen kann:

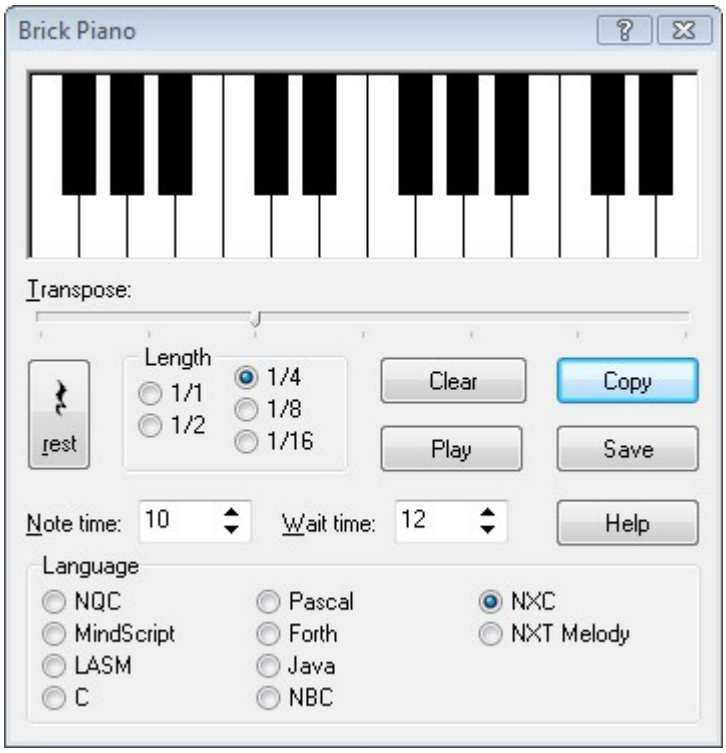
```
task Fahren ()
{
 OnFwd(OUT_BC, 50);
 Wait(1000);
 OnRev(OUT_C, 50);
 Wait(500);
 OnFwd(OUT_BC, 50);
 Wait(1000);
 OnRev(OUT_C, 50);
 Wait(500);
 OnFwd(OUT_BC, 50);
 Wait(1000);
 OnRev(OUT_C, 50);
 Wait(500);
 Off(OUT_BC);
}

task MusikSpielen ()
{
 PlayTone(262,800);
 Wait(960);
 PlayTone(247,400);
 Wait(480);
 PlayTone(233,800);
 Wait(960);
 PlayTone(247,400);
 Wait(480);
 PlayTone(262,800);
 Wait(960);
 PlayTone(330,400);
 Wait(480);
 PlayTone(262,800);
 Wait(960);
 PlayTone(247,200);
 Wait(240);
 PlayTone(262,800);
 Wait(960);
}

task main ()
{
 start Fahren;
 start MusikSpielen;
}
```

Führt man das Programm aus, so wird der NXT, während er Vorwärts fährt und dabei dreimal kurz abbiegt, außerdem eine Melodie spielen. Das Programm endet erst, wenn beide tasks abgeschlossen sind.

Neu in diesem Beispiel ist der „PlayTone“-Befehl. Der erste Parameter gibt die Frequenz (Die Tonhöhe) und der Zweite die Tondauer an. Nach jedem „PlayTone“ kommt ein „Wait“ damit die Töne nacheinander und nicht gleichzeitig gespielt werden.
Nun wäre es recht kompliziert eine Melodie auf diese Weise zu schreiben, und deshalb bietet Bricx das „Brick piano“. Dieses lässt sich öffnen, indem man auf Notensymbol in der Werkzeugleiste von Bricx drückt (aber nur, wenn ein NXT angeschlossen ist). Dann erscheint folgendes Fenster:



Um eine Melodie zu spielen, wählt man zunächst „NXC“ unter „Language“ aus, klickt dann „Clear“, dann die Länge der gewünschten Note (ganze Note, halbe Note, viertel Note usw.) und gibt dann den Notenwert über die „Klaviatur“ oben im Fenster ein. Den Notenwert kann man natürlich für jede Note neu festlegen. Anschließend klickt man auf „Copy“, schließt das Fenster, setzt den Cursor in den Code des eigenen Programms und drückt dann auf der Tastatur „Strg“ und „V“ gleichzeitig (einfügen). Dann wird die gespielte Melodie eingefügt.

Teil 3: Zeit messen

Nachdem uns nun auch bekannt ist, wie man den NXT Musik machen lässt, wird in diesem Teil erklärt, wie man mit dem NXT die Zeit messen kann.
Das folgende Programm reagiert auf Klatschen. Aber nicht nur das – es registriert ob mehrmals hintereinander geklatscht wurde und lässt den Roboter in diesem Fall anders reagieren: Bei einmaligem Klatschen fährt der Roboter nach links, bei mehrmaligem nach rechts.

```
#define CLAP 40 //Lautstaerke eines Handschlags

void ProcessSoundS ()
{
 TextOut (1,LCD_LINE5,"Soundsensor:");
 NumOut(78,LCD_LINE5,Sensor (S2));

 if (Sensor (S2) > CLAP)
 {
  /* Es wurde einmal geklatscht. Nun auf ein zweites Mal Klatschen warten
*/
  Off(OUT_BC);
  long WaitTime = 0;
  bool ClapAgain = false;

  Wait (300); // Vorheriges Klatschen muss ausklingen

  WaitTime = CurrentTick() + 300;
```

```
  /* Solange die Wartezeit noch nicht vorbei ist, soll überprüft werden, ob
ein zweites Mal geklatscht wurde */

  while (WaitTime > CurrentTick())
  {
   if (Sensor (S2) > CLAP)
   {
    ClapAgain = true;
   }
  }

  if (ClapAgain == true)
  {
   // Es wurde zweimal geklatscht

   PlayTone(330,400);
   Wait(480);
   PlayTone(330,400);
   Wait(480);

   // Rechts fahren
   OnRev(OUT_BC, 40); Wait(300);
   OnFwd(OUT_C, 75); Wait(300);
   OnFwdReg(OUT_BC,50,1);
  }
  else
  {
   // Es wurde nur einmal geklatscht

   PlayTone(330,400);
   Wait(480);

   // Links fahren
   OnRev(OUT_BC, 40); Wait(300);
   OnFwd(OUT_B, 75); Wait(300);
   OnFwdReg(OUT_BC,50,1);
  }
 }
}

// Hauptteil
task main()
{
 // Sensoren initialisieren
 SetSensorSound(S2);

 // Hauptschleife des Programms
 while (true)
 {
  OnFwd(OUT_BC,50);
  // Vorwärts!

  ClearScreen ();
  // Bildschirm säubern

  // Alle Sensoreingaben verarbeiten
  ProcessSoundS ();
 }
}
```

Wow. Im Gegensatz zu den vorherigen Beispielen ist dieses Programm ja ein echter Klotz. Bevor wir uns an Verstehen machen, schauen wir uns erstmal die Funktionen zum Zeitmessen an:

Der NXT misst natürlich nicht die echte Urzeit. Vielmehr hat er eine integrierte Stoppuhr, deren Stand man mit „CurrentTick()“ abfragen kann. Es wird in tausendstel Sekunden gemessen (1 Sekunde = 1000 tausendstel Sekunden). Dummerweise beginnt diese Stoppuhr aber nicht beim Programmstart von 0 an zu zählen, sondern zählt vom Einschalten des NXT an. Den Zeitpunkt, zu dem das Programm gestartet wurde, erhält man mit „FirstTick()“. Da die Uhr ja ununterbrochen zählt, kann sie extrem hohe Werte erreichen. **Daher sollten Zeitwerte immer in long-Variablen gespeichert werden!**

Nun zurück zum Beispielprogramm: Der Task „main“ ist eigentlich ganz einfach: Zuerst werden Soundsensor und Timer initialisiert. In der Hauptschleife folgen dann Befehle zum vorwärts fahren, den Bildschirm löschen und ein Aufruf der Funktion „ProcessSoundS“ (auf Deutsch: „Verarbeite Sound Sensor“).
Der Grund, warum hier Englisch statt Deutsch gewählt wurde ist einfach der, dass man in der Programmierung (bis auf seine Kommentare) immer in Englisch schreibt. Weil Programmierteams oft aus vielen Leuten bestehen, die eventuell auch andere Muttersprachen haben, wird eben Englisch gewählt. Da dies nun einmal Standard ist, werden wir ab jetzt immer Englisch verwenden.

Die Funktion „ProcessSoundS“ fängt erstmal mit zwei Displayausgaben an und überprüft anschließend, ob der vom Soundsensor gemessene Wert größer ist, als der Wert, der in der Konstante „CLAP“ gespeichert wurde. Wenn ja, dann wurde geklatscht. In diesem Falle beginnt auch der wichtigste Teil der Funktion:
Dort werden zunächst die Motoren gestoppt und zwei Variablen erstellt. Anschließend wird per „Wait“ ein (sehr) kurzer Moment gewartet. Das ist nötig, damit das erste Klatschgeräusch ausklingen kann, denn nun soll überprüft werden, ob ein zweites Mal geklatscht wurde.
Dazu definieren wir die Variable „WaitTime“ (= „Wartezeit“) mit der derzeitigen Zeitangabe, addieren aber gleichzeitig noch 300 tausendstel Sekunden dazu! Die Folgende Schleife wird dann so lange ausgeführt bis der Stand der Stoppuhr auch diesen Wert erreicht – mit dem Abschnitt:

```
while (WaitTime > CurrentTick())
```

wird also im Endeffekt 0,3 Sekunden gewartet. Der Trick ist nun, dass während dieser Wartezeit immer überprüft wird, ob der Soundsensor erneut einen Wert größer als „CLAP“ misst (in andern Worten: ob erneut geklatscht wurde). Wenn ja, so wird die Variable „ClapAgain“ auf „true“ gesetzt. Die dann folgende if-Abfrage überprüft diesen Wert, und lässt den Roboter entweder nach links oder nach rechts fahren (inklusive akustischer Rückmeldung).

Teil 4: Buttons

In diesem kurzen Teil werden wir lernen, den Zustand der am NXT angebrachten Buttons abzufragen:

```
task main ()
{
 while (true)
 {
```

```
  while (ButtonPressed(BTNCENTER, false) == true)
  {
   OnFwd(OUT_BC, 50);
  }

  while (ButtonPressed(BTNLEFT, false) == true)
  {
   OnRev(OUT_B, 50);
  }

  while (ButtonPressed(BTNRIGHT, false) == true)
  {
   OnRev(OUT_C, 50);
  }

  Off(OUT_BC);
 }
}
```

Dieses Programm zeigt eine neue Funktion namens „ButtonPressed“. Sie liefert „true“ zurück, falls der angegebene Button gedrückt wurde.
Für den ersten Parameter (der angibt, welcher Button überprüft werden soll), kommen in Frage:

BTNCENTER	Der große orangefarbene Button
BTNLEFT	Der nach Links zeigende graue Pfeil
BTNRIGHT	Der nach Rechts zeigende graue Pfeil
BTNEXIT	Der graue rechteckige Button (Da dieser jedoch gleichzeitig das Programm beendet, ist er praktisch nutzlos)

Der zweite Parameter gibt an, ob der Zähler des Buttons zurückgesetzt werden soll. Dieser Zähler gibt an, wie oft der Button gedrückt wurde. Sein Wert kann mit der Funktion „ButtonCount“ ausgelesen werden. Wird nun beim zweiten Parameter von „ButtonPressed“ oder „ButtonCount“ „true“ angegeben, so wird der Zähler auf 0 gesetzt.

Teil 5: Die Rotationssensoren der Motoren

Im letzten Teil werden wir noch zwei Befehle kennenlernen, mit denen man die Rotationssensoren der NXT-Motoren benutzen kann. Jeder NXT-Motor hat einen eigenen Rotationszähler, der misst, um wie viel Grad der Motor sich gedreht hat.
Den Stand des Zählers kann man sowohl abfragen, als auch auf 0 zurücksetzen:

```
task main()
{
 ResetRotationCount (OUT_B);
 ResetRotationCount (OUT_C);

 ClearScreen();

 while (MotorRotationCount(OUT_B) != 180)
 {
  OnFwd(OUT_B, 50);
```

```
  OnRev(OUT_C, 50);
 }

 Off (OUT_BC);

 NumOut(0, LCD_LINE1, MotorRotationCount(OUT_B));
 NumOut(0, LCD_LINE2, MotorRotationCount(OUT_C));

 Wait (20000);
}
```

Durch das Programm werden zunächst die Rotationszähler der Motoren B und C zurückgesetzt. Dies geschieht mit „ResetRotationCount“.
Anschließend wird noch das Display gelöscht und dann eine Schleife gestartet:
Die Motoren B und C laufen in entgegengesetzte Richtungen – der NXT dreht auf der Stelle. Erreicht nun Motor B einen Zählerstand von 180 (das ist nach einer Drehung von 90 Grad der Fall, wenn die Motoren entgegengesetzt laufen), so werden die Motoren gestoppt, und noch für 20 Sekunden die Zählerwerte angezeigt. Man benutzt dazu „MotorRotationCount“.

Abschluss:

Das Tutorial ist jetzt zu Ende. Mit dem erlangten Wissen lassen sich intelligente NXC-Programme schreiben. Und wenn man seinen Code dann noch in verschiedene Funktionen aufteilt, wird das ganze auch noch richtig übersichtlich.
Wer jetzt weiterlernen möchte, dem empfehle ich folgende Seiten:

- http://lukas.internet-freaks.net/nxt.php - Hier werden viele Funktionen erklärt, inklusive einiger Beispielprogramme zum Download
- http://www.debacher.de/wiki/NXC - Ausführliche Erklärungen und Beispiele

Zuletzt möchte ich mich fürs Lesen bedanken. Verbesserungsvorschläge sind immer willkommen und können mir über meine Website mitgeteilt werden:

viscil.bplaced.net

Dort gibt es unter anderem auch meine selbstgeschriebenen NXT Programme zum Download.

- Michael Krause