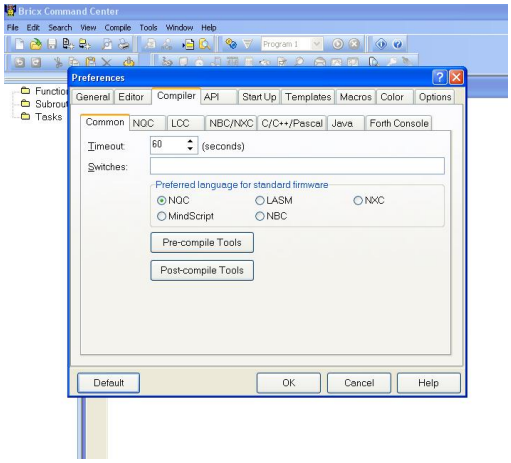
Inhaltsverzeichnis

1 Anfang

Bevor es losgeht, muss der NXT über USB mit dem Computer verbunden und an sein, da er sonst nicht vom Programm erkannt wird und es eine Fehlermeldung gibt.

Man startet das Programm über
Alle Programme - Informatik - BrixCC.
Verwende als Port für den Tower die Option *usb* oder *Automatic*. Wenn der NXT an ist, dann sollte es keine Fehlermeldung geben.

In seltenen Fällen, musst du noch eine Änderung in Bricx Command Center durchführen, da das Programm eventuell auf den gelben NQC eingestellt ist. Dazu musst du in der oberen Leiste unter *Edit-Preferences-Compiler* den Button von NQC auf NXC umstellen und auf Ok klicken.



Nach dem ganzen Aufwand kann es jetzt losgehen. Du kannst ein Programm aufrufen und es auf den NXT mit F6 oder dem Dreieck mit den zwei Strichen übertragen.
Ist die Übertragung beendet, gibt es ein Tonsignal.

2 Erste Schritte

Jetzt kannst du dein erstes Programm schreiben.

Der NXT führt alles in den geschwungenen Klammern nacheinander aus. Hier fährt der Roboter 1 Sekunde geradeaus und bleibt stehen.

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

Erklärung zum Programm:

task main() {	Jedes NXC-Programm beginnt mit *'task main()'* und *'{'* und hört mit *'}'* auf.
OnFwd(OUT_AB, 50);	Die Ausgänge A und B, an denen die Motoren angeschlossen sind, werden angeschaltet. Beachte, dass am Ende eines Befehls immer ein ';' steht. Die '50' gibt die Leistung zwischen 0 und 100 an, mit der der der Roboter fahren soll.
Wait(1000);	Der Roboter wartet 1 Sekunde. Die Zahl gibt die Zeit in tausendstel Sekunden an. So lange fährt er geradeaus.
Off(OUT_AB);	Beide Motoren werden wieder ausgeschaltet.
}	Die Klammer vom Anfang wird hier geschlossen.

Diese drei Befehlen kann man beliebig kombinieren, sodass der Roboter bestimmte Wege abfahren kann.

Um den Roboter zu drehen, kann man den einen Motor anmachen und den anderen mit *OnRev* anstatt *OnFwd* rückwärts fahren lassen.

3 Schleifen

3.1 repeat

Die einfachste Schleife ist die *repeat()*-Schleife. Es ist sehr aufwendig, wenn du zum Fahren eines Quadrates jedesmal die Drehung und das Vorwärtsfahren eingeben musst.

Man kann es vereinfachen, wenn du das Vorwärtsfahren und die 90°-Kurve viermal wiederholst.

```
1  task main()
2  {
3    repeat(4)
4    {
5      OnFwd(OUT_AB, 50);
6      Wait(1000);
7      OnRev(OUT_AB, 50);
8      Wait(1500);
9    }
10   Off(OUT_AB);
11 }
```

Mit *repeat(4)* werden die Befehle in den geschwungenen Klammern viermal wiederholt. Am Ende werden die Motoren ausgeschaltet, damit sich der Roboter nicht weiter im Kreis dreht.

Achte darauf, dass du nach einer '{' alle Befehle um zwei Zeichen einrückst, damit du nicht durcheinander kommst, wann du sie wieder zumachen musst.

3.2 while(true)

Manchmal ist es so, dass man einen Vorgang beliebig oft wiederholen möchte. Anstatt bei *repeat()* eine große Zahl einzugeben, kann man ihn mit *while(true)* durchführen. Alles was zwischen '{' und '}' steht wird beliebig wiederholt. Würde man im oberen Beispiel das *repeat(4)* durch das *while(true)* ersetzen, dann würde der Roboter bis zum Versagen des Akkus Quadrate abfahren.

3.3 Die if()-Schleife

Bisher haben wir nur Schleifen, die den Programmieraufwand verringern, kennengelernt. Mit der *if()*-Schleife werden Befehle durchgeführt, wenn die Bedingung in den Klammern erfüllt ist.

```
1  task main()
2  {
3    int a = 0;
4
5    if ( a == 0 )
6    {
7      OnFwd(OUT_AB, 50);
8      Wait(1000);
9    }
10   Off(OUT_AB);
11 }
```

Am Anfang wird mit *int* eine Variable *a* definiert, die den Wert 0 hat (mehr dazu später). In den Klammern von *if()* wird mit dem doppelten Gleichheitszeichen überprüft, ob die Bedingung erfüllt ist oder nicht. Da sie erfüllt ist (a ist null), werden die Befehle zwischen '{' und '}' ausgeführt und der Roboter fährt eine Sekunde vorwärts. Wäre die Bedingung nicht erfüllt, dann würde der Roboter nichts machen und einfach stehen bleiben.

Dieses Beispiel ist natürlich etwas merkwürdig, da die Bedingung *a==0* immer erfüllt ist. Wenn wir aber demnächst mit Sensoren anfangen, wird es sinnvollere Beispiele geben.

Ein Beispiel, in dem der Roboter Zick-Zack fährt:

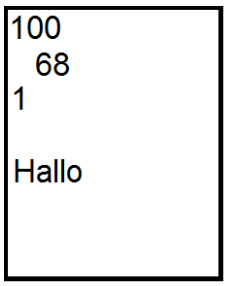
```
1  task main()
2  {
3    int a = 0;
4
5    while(true)
6    {
7      if ( a == 0 )
8      {
9        OnFwd(OUT_AB, 50);
10       Wait(1000);
11       OnRev(OUT_A);
12       Wait(500);
13       a = 1;
14     }
15     if ( a == 1 )
16     {
17       OnFwd(OUT_AB, 50);
18       Wait(1000);
19       OnRev(OUT_B);
20       Wait(500);
21       a = 0;
22     }
23   }
24 }
```

4 Das Display

Um sich Informationen des Roboters, während ein Programm läuft, anzeigen zu lassen, kann man das Display verwenden.

```
1  task main(){
2    int a = 1;
3    while(true)
4    {
5      NumOut(0, LCD_LINE1, 100);
6      NumOut(10, LCD_LINE2, SENSOR_2);
7      NumOut(0, LCD_LINE3, a);
8      TextOut(0, LCD_LINE5, "Hallo", true);
9    }
10 }
```

Programm



Displayanzeige

Mit dem Befehl *NumOut* gibt man Zahlen aus. Mit der ersten Zahl in der Klammer gibt man die x-Position, bei der die Zahl angezeigt wird, an. *LCD_Line* gibt die Zeile an, bei der die Zahl angezeigt werden soll. Die letzte Information ist die Zahl selber, die angezeigt werden soll. Es kann wie die *100* eine konkrete Zahl sein oder wie in Zeile 6 und 7 eine Platzhalter sein.

Für Texte gibt es den Befehl *TextOut* . Der Text muss in Anführungszeichen angegeben werden und zum Schluss kommt noch ein *true als Parameter dazu.*

5 Sensoren

5.1 Tastsensoren (Touch-Sensoren)

Bisher fährt der Roboter sein Programm ab, ohne das er beeinflusst werden kann. Mit der Hilfe von Sensoren kann der Roboter auf äußere Einflüsse reagieren, um z.B. ein Hindernis zu umfahren oder um sich an einer Mauer entlang zu tasten.

Der einfachste Sensor ist der Tastsensor. Wird der Taster gedrückt, dann liefert der Sensor den Wert 1, andernfalls den Wert 0. Hier ist ein einfaches Beispiel:

```
1  task main()
2  {
3    SetSensor(SENSOR_1, SENSOR_TOUCH);      // Der Sensor wird initialisiert.
4
5    while(true)                             // Hier beginnt die Endlosschleife.
6    {
7      OnFwd(OUT_AB, 50);                    // Fahre geradeaus.
8      if ( SENSOR_1 == 1 )                  // Wenn der Sensor gedrückt wird,
9      {                                     // dann fürht er das Folgende aus:
10       OnRev(OUT_B, 50);
11       Wait(1000);
12     }
13   }
14 }
```

Mit SetSensor(); wird beim NXT ein Sensor definiert. In den Klammern steht, um **welchen** Sensor es sich handelt (*SENSOR_1*) und um **was** für einen Sensor (*SENSOR_TOUCH*). Beides wird mit einem Komma getrennt.
In der *if*-Schleife wird überprüft, ob der Taster gedrückt ist. Sie fragt ab, ob *SENSOR_1* den Wert 1 hat. Wenn ja, dann führt der Roboter das in den geschwungenen Klammern stehende Programm aus (eine kurze Rechtsdrehung). Wenn nicht, dann überspringt sie es und fährt geradeaus weiter. Wenn der Taster nicht gedrückt ist, dann findet diese Überprüfung viele Male pro Sekunde statt.

Mit dem folgenden Programm kann man den Roboter im Raum fahren lassen. Wenn der rechte Taster gedrückt wird, dann fährt er rückwärts und dann nach links; wird der linke Taster gedrückt, dann fährt er auch rückwärts, aber später nach rechts.

```
task main()
{
  SetSensor(SENSOR_1, SENSOR_TOUCH);      // Der linke Sensor wird initialisiert.
  SetSensor(SENSOR_2, SENSOR_TOUCH);      // Der rechte Sensor wird initialisiert.

  while(true)                             // Hier beginnt die Endlosschleife.
  {
    OnFwd(OUT_AB, 50);                    // Fahre geradeaus.

    if ( SENSOR_1 == 1 )                  // Wenn der linke Sensor gedrückt wird,
    {                                     // dann fürht er das Folgende aus:
      OnRev(OUT_AB, 50);                  // Fahre Rückwärts.
      Wait(1000);
      OnFwd(OUT_A, 50);                   // Rechtsdrehung
      Wait(2000);

    if ( SENSOR_2 == 1 )                  // Wenn der rechte Sensor gedrückt wird,
    {                                     // dann fürht er das Folgende aus:
      OnRev(OUT_AB, 50);                  // Fahre Rückwärts.
      Wait(1000);
      OnFwd(OUT_B, 50);                   // Linksdrehung
      Wait(2000);
    }
  }
}
```

5.2 Lichtsensoren

Mit der Hilfe von Lichtsensoren lassen sich Helligkeiten - und im begrenzten Umfang auch Farben - auf dem Boden erkennen. Im Gegensatz zu den Tastsensoren, welche nur der Werte 0 und 1 liefern, geben Lichtsensoren Helligkeitswerte von 0 bis 100 zurück. Dabei gilt: Je heller der Untergrund, desto größer der Wert.

Damit die Lichtsensoren als solche erkannt werden, muss man sie wie die Tastsensoren am Anfang definieren. Mit

SetSensor(SENSOR_1, SENSOR_LIGHT);
oder
SetSensorLight(IN_1);

wird z.B. der erste Sensor als Lichtsensor definiert. Mit der oberen Zeile ist die Leuchtdiode aus und mit dem unteren an. *SENSOR_1* bzw. *IN_1* gibt den Eingang des NXTs an. Sollte man einen anderen Eingang verwenden möchten, muss man die 1 durch eine 2, 3 oder 4 ersetzen.

Um jetzt zwischen z.B. Schwarz und Weiß zu unterscheiden, muss man sich die Werte mit einem eigenen Programm über das Display anzeigen lassen.

Die Farbe Schwarz müsste im Bereich von 35-50 liegen und Weiß von 45-60 (je nach Sensor und deren Platzierung). Als Grenze, um zwischen Schwarz und Weiß zu unterscheiden, sollte man den Mittelwert nehmen, der sich wie folgt berechnet: $\frac{Schwarz+Weiß}{2}$. Beachte, dass die Sensoren 1-4 durchaus unterschiedliche Werte für Schwarz und Weiß anzeigen können. Du musst sie also alle einzeln überprüfen.

Ein Hinweis zur besseren Verwendung der Lichtsensoren:

Der Lichtsensor hat eine Leuchtdiode, die ein künstliches Licht erzeugt und eine Fotodiode, die den Lichteinfall misst. Nun ist es so, dass nicht nur die Leuchtdiode Licht erzeugt, sondern auch die Umgebung (Deckenlicht, Sonneneinstrahlung). Das kann zu unterschiedlichen Lichtwerten führen, wodurch der Roboter nicht optimal fährt.

Um den Lichteinfall der Leuchtdiode zu erhöhen und den Fremdlichteinfall zu minimieren, ist es sinnvoll, die Lichtsensoren so dicht wie möglich über dem Parcour zu plazieren. Natürlich sollte der Roboter noch über die kleinen Hindernisse fahren können und auch in der Lage sein, die Rampe zu meistern, aber ansonsten gilt die Regel: „So tief wie möglich.“

5.3 Umgang mit den Lichtwerten

Nun ist es so, dass man mit den Lichtwerten arbeiten muss, d.h. gewisse Bedingungen in der *if*-Schleife abfragt. Dabei kann man auch mehr als eine Bedingung verwenden. Wenn z.B die eine Bedingung *und* die andere erfüllt sein sollen, dann benutzt man das &&-Zeichen. Wenn man Befehle ausgeführen möchten, wenn die eine *oder* die andere Bedingung erfüllt ist, dann verwendet man das ||-Zeichen.

Hier sind einige Beispiele mit Erklärungen:

$if(SENSOR\_1 == 50)$	Wenn der Sensorwert 50 ist, dann ...
$if(SENSOR\_1 > 40)$	Wenn der Sensorwert größer als 40 ist, dann ...
$if(SENSOR\_1 < 60)$	Wenn der Sensorwert kleiner als 60 ist, dann ...
$if(SENSOR\_1 < 60 \;\&\&\; SENSOR\_1 > 40)$	Wenn der Sensorwert zwischen 40 und 60 ist, dann ...
$if(SENSOR\_1 > 60 \;\|\|\; SENSOR\_1 < 40)$	Wenn der Sensorwert größer als 60 oder kleiner als 40 ist, dann ...
$if(SENSOR\_1 == 100 \;\|\|\; SENSOR\_2 == 100)$	Wenn der erste Sensorwert 100 ist oder der zweite, dann ...

Die Anzahl der Bedingungen ist nicht auf zwei beschränkt, man kann sie beliebig erhöhen. Irgendwann wird es jedoch unübersichtlich.