20.3 Kommunikation per Kabel

Im Prinzip lassen sich zwei NXTs mit Bluetooth recht einfach steuern. Allerdings hat die Bluetooth-Verbindung auch einige Nachteile. Bisher haben wir nur den Fall betrachtet, dass zwei eigenständige Roboter miteinander kommunizieren, aber es kann auch durchaus passieren, dass Sie einen Roboter bauen wollen, der von zwei NXTs gesteuert werden soll, zum Beispiel weil Sie mehr als die drei verfügbaren Motorenausgänge oder mehr als vier Sensoreneingänge brauchen. Dann ist eine kabellose Verbindung nicht wichtig, dafür legen Sie aber Wert auf Geschwindigkeit. Und mit einer direkten Kabelverbindung ist die Reaktionszeit schneller als mit einer Bluetooth-Verbindung. Hinzu kommt, dass man bei einer Kabelverbindung die Verbindung nicht bei jedem Start des Roboters neu einstellen muss, wie das bei einer Bluetooth-Verbindung der Fall ist. Auch bei vielen offiziellen Wettbewerben wird meist eine Kabel-Verbindung benutzt, da hier drahtlose Verbindungen oft verboten sind, da der Roboter sonst von einem Computer aus ferngesteuert werden könnte. Aus all diesen Gründen werden wir uns im Folgenden anschauen, wie die NXTs über ein Kabel kommunizieren können, damit Sie später alle Möglichkeiten haben, je nach Projekt die für Sie passende Verbindung wählen zu können.

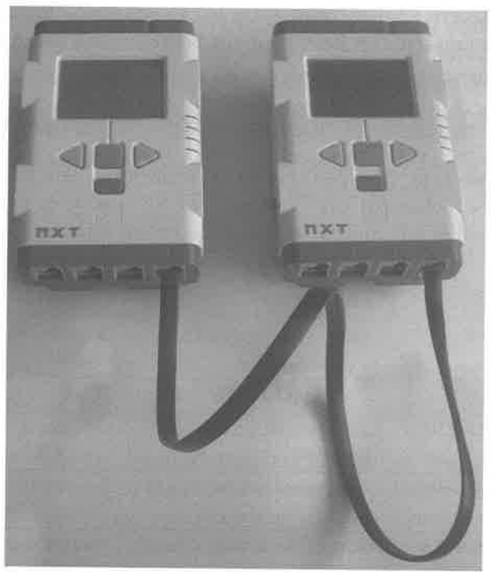

Abb. 20.9: Per Kabel verbundene NXTs

20.3.1 Verbindung herstellen

Das Herstellen einer Verbindung ist bei einer Kabelverbindung wesentlich einfacher als bei einer Bluetooth-Verbindung. Es reicht nämlich einfach, den vierten Sensoreingang der NXTs zu verbinden, dazu kann man ein normales Sensoren- beziehungsweise Motorenkabel verwenden. Dabei ist lediglich zu beachten, dass die beiden Ports mit der Nummer 4 verbunden werden müssen, da es mit den anderen Eingängen nicht funktioniert.

20.3.2 Programmierung

Da auch die direkte Kommunikation über Kabel auf dem Master/Slave-Prinzip basiert, brauchen wir auch hier wieder zwei Programme: eines für den Master-Roboter und eines für den Slave-Roboter. In jedem Programm, egal ob für den Master- oder den Slave-Roboter, muss erst festgelegt werden, dass der Roboter über ein Kabel kommunizieren soll, dazu dient der Befehl:

```
SetSensorType(IN_4, SENSOR_TYPE_HIGHSPEED);
```

Da die Kommunikation nur am vierten Port funktioniert, sieht der Befehl immer genau so aus. Außerdem werden noch zwei weitere Befehle, sowohl im Master- als auch im Slave-Programm gebraucht, die den Roboter sozusagen auf die Übertragung vorbereiten. Die beiden Befehle lauten:

```
SetHSState(HS_INITIALISE);
SetHSFlags(HS_UPDATE);
```

Da es recht kompliziert zu verstehen ist, was diese Befehle genau machen, wollen wir sie für uns einfach als gegeben ansehen, sie müssen einfach am Anfang jedes Programms stehen, das Daten über den vierten Port senden oder empfangen soll. Daher bietet es sich auch an, diese beiden Befehle zusammen mit dem `SetSensorType()`-Befehl in eine Header-Datei zu schreiben. Ob sich das für drei Zeilen Code schon lohnt, muss letztendlich jeder für sich selbst entscheiden.

An dieser Stelle müssen wir jetzt anfangen, zwischen dem Master- und dem Slave-Programm zu unterscheiden. Anfangen werden wir wieder mit dem etwas komplizierteren Programm für den Master:

```
//////////////////////
// Master-Programm  //
//////////////////////

#include "NXCDefs.h"

task main()
```

20.3 Kommunikation per Kabel

Im Prinzip lassen sich zwei NXTs mit Bluetooth recht einfach steuern. Allerdings hat die Bluetooth-Verbindung auch einige Nachteile. Bisher haben wir nur den Fall betrachtet, dass zwei eigenständige Roboter miteinander kommunizieren, aber es kann auch durchaus passieren, dass Sie einen Roboter bauen wollen, der von zwei NXTs gesteuert werden soll, zum Beispiel weil Sie mehr als die drei verfügbaren Motorenausgänge oder mehr als vier Sensoreneingänge brauchen. Dann ist eine kabellose Verbindung nicht wichtig, dafür legen Sie aber Wert auf Geschwindigkeit. Und mit einer direkten Kabelverbindung ist die Reaktionszeit schneller als mit einer Bluetooth-Verbindung. Hinzu kommt, dass man bei einer Kabelverbindung die Verbindung nicht bei jedem Start des Roboters neu einstellen muss, wie das bei einer Bluetooth-Verbindung der Fall ist. Auch bei vielen offiziellen Wettbewerben wird meist eine Kabel-Verbindung benutzt, da hier drahtlose Verbindungen oft verboten sind, da der Roboter sonst von einem Computer aus ferngesteuert werden könnte. Aus all diesen Gründen werden wir uns im Folgenden anschauen, wie die NXTs über ein Kabel kommunizieren können, damit Sie später alle Möglichkeiten haben, je nach Projekt die für Sie passende Verbindung wählen zu können.

Abb. 20.9: Per Kabel verbundene NXTs

20.3.1 Verbindung herstellen

Das Herstellen einer Verbindung ist bei einer Kabelverbindung wesentlich einfacher als bei einer Bluetooth-Verbindung. Es reicht nämlich einfach, den vierten Sensoreingang der NXTs zu verbinden, dazu kann man ein normales Sensoren- beziehungsweise Motorenkabel verwenden. Dabei ist lediglich zu beachten, dass die beiden Ports mit der Nummer 4 verbunden werden müssen, da es mit den anderen Eingängen nicht funktioniert.

20.3.2 Programmierung

Da auch die direkte Kommunikation über Kabel auf dem Master/Slave-Prinzip basiert, brauchen wir auch hier wieder zwei Programme: eines für den Master-Roboter und eines für den Slave-Roboter. In jedem Programm, egal ob für den Master- oder den Slave-Roboter, muss erst festgelegt werden, dass der Roboter über ein Kabel kommunizieren soll, dazu dient der Befehl:

```
SetSensorType(IN_4, SENSOR_TYPE_HIGHSPEED);
```

Da die Kommunikation nur am vierten Port funktioniert, sieht der Befehl immer genau so aus. Außerdem werden noch zwei weitere Befehle, sowohl im Master- als auch im Slave-Programm gebraucht, die den Roboter sozusagen auf die Übertragung vorbereiten. Die beiden Befehle lauten:

```
SetHSState(HS_INITIALISE);
SetHSFlags(HS_UPDATE);
```

Da es recht kompliziert zu verstehen ist, was diese Befehle genau machen, wollen wir sie für uns einfach als gegeben ansehen, sie müssen einfach am Anfang jedes Programms stehen, das Daten über den vierten Port senden oder empfangen soll. Daher bietet es sich auch an, diese beiden Befehle zusammen mit dem `SetSensorType()`-Befehl in eine Header-Datei zu schreiben. Ob sich das für drei Zeilen Code schon lohnt, muss letztendlich jeder für sich selbst entscheiden.

An dieser Stelle müssen wir jetzt anfangen, zwischen dem Master- und dem Slave-Programm zu unterscheiden. Anfangen werden wir wieder mit dem etwas komplizierteren Programm für den Master:

```
//////////////////////
// Master-Programm  //
//////////////////////

#include "NXCDefs.h"

task main()
```

```
{
 //Initialisierung
 SetSensorType(IN_4, SENSOR_TYPE_HIGHSPEED);
 SetHSState(HS_INITIALISE);
 SetHSFlags(HS_UPDATE);

 //Variablen
 string nachricht;
 byte laengenachricht;

 while(true)
 {
  nachricht = "Hallo Welt!"; //Nachricht festlegen
  laengenachricht = ArrayLen(nachricht); //Länge der Nachricht bestimmen
  SetHSOutputBuffer(0, laengenachricht, nachricht); //Nachricht vorbereiten
  SetHSOutputBufferOutPtr(0);
  SetHSOutputBufferInPtr(laengenachricht);
  SetHSState(HS_SEND_DATA); //Senden
  SetHSFlags(HS_UPDATE);
 }
}
```

Programm 20.12

Schauen wir uns das Programm einmal von Anfang an genau an. Die ersten drei Zeilen sind die Ihnen bereits bekannten Zeilen, die am Anfang jedes Programms stehen und den Roboter erst einmal auf das Übertragen vorbereiten. Darauf folgt die Deklaration der Variablen. Zum einen brauchen wir eine Variable vom Typ String, um die Nachricht, die wir übertragen wollen, zu speichern, zum anderen brauchen wir eine Variable vom Typ `byte`. Das Besondere an dieser Variablen ist, dass sie Daten von genau acht Bit aufnehmen kann (acht Bit entsprechen einem Byte, ein Megabyte entspricht 1.000.000 Byte, also 8.000.000 Bit). Anstatt des Typs `byte` könnten wir auch `char` verwenden, da auch dieser acht Bit speichern kann; da wir aber eigentlich keinen Buchstaben speichern wollen, bevorzugt man an dieser Stelle den Typ `byte`.

Danach legen wir fest, welche Nachricht gesendet werden soll (in unserem Fall das berühmte »Hallo Welt!«). Und weisen unserer Variablen vom Typ `byte` die Länge unserer Nachricht zu. Wundern Sie sich nicht, dass wir an dieser Stelle den Befehl `ArrayLen()` auf einen String und nicht auf ein Array anwenden. Für den NXT (und auch für den Computer) ist ein String ein Array von einzelnen Zeichen (also von Chars). Danach legen wir die Nachricht sozusagen in das Fach zum Absenden (an dieser Stelle wird die Nachricht noch nicht gesendet). Dazu dient folgender Befehl:

```
SetHSOutputBuffer(0, laengenachricht, nachricht);
```


Der erste Parameter bezieht sich auf die interne Adressierung des Speichers. Da das an dieser Stelle tiefe Einblicke in die Hardware erforderlich machen würde, begnügen wir uns damit, dass wir als ersten Parameter immer eine 0 übergeben, alles Weitere braucht uns nicht zu interessieren, wir wollen uns ja nicht mehr Umstände als nötig machen. Die beiden anderen Parameter sind dagegen für uns interessant, als Zweites übergeben wir nämlich unsere Längen-Variable und als Drittes unsere eigentliche Nachricht, also auch nicht weiter kompliziert.

Die beiden nächsten Befehle setzen sozusagen eine Fahne (in der Informatik spricht man von einem Pointer oder zu Deutsch auch von einem Zeiger), die den Anfang und das Ende unserer gesendeten Nachricht markiert, das ist nötig, damit der Slave-Roboter nachher weiß, wie viele Daten er zu erwarten hat. Auch hier gilt, dass die beiden Befehle unabhängig von der gesendeten Nachricht immer gleich sind.

Die beiden letzten Befehle senden unsere Nachricht dann endgültig ab. Auch diese Befehle wollen wir wieder als gegeben ansehen, ohne sie weiter zu hinterfragen, da die Betrachtung recht kompliziert ausfallen würde und uns nicht im Geringsten weiterbrächte. Es reicht einfach, wenn Sie sich merken, dass wir zum Senden folgende Befehle brauchen:

```
SetHSState(HS_SEND_DATA);
SetHSFlags(HS_UPDATE);
```

Schon ist unsere Nachricht gesendet. Im Gegensatz zur Datenübertragung per Bluetooth kennt die Übertragung per Kabel keinen separaten Befehl für das Übertragen von Zahlen. Hierzu wird einfach der `NumToStr()`- beziehungsweise der `StrToNum()`-Befehl verwendet. Übrigens bietet es sich auch sehr an, den gesamten Sendevorgang in eine Subroutine auszulagern, dann würde das Programm in etwa so aussehen:

```
 ////////////////////
 // Master-Programm //
////////////////////

#include "NXCDefs.h"

sub SendeNachricht(string nachricht)
{
 //Variable
 byte laengenachricht;

 laengenachricht = ArrayLen(nachricht); //Länge der Nachricht bestimmen
 SetHSOutputBuffer(0, laengenachricht, nachricht); //Nachricht vorbereiten
```

```
 SetHSOutputBufferOutPtr(0);
 SetHSOutputBufferInPtr(laengenachricht);
 SetHSState(HS_SEND_DATA); //Senden
 SetHSFlags(HS_UPDATE);
}

task main()
{
 //Initialisierung
 SetSensorType(IN_4, SENSOR_TYPE_HIGHSPEED);
 SetHSState(HS_INITIALISE);
 SetHSFlags(HS_UPDATE);

 //Variable
 string nachricht;

 while(true)
 {
  SendeNachricht("Hallo Welt!");
  Wait(10);
 }
}
```

Programm 20.13

Gerade falls man mehrmals innerhalb eines Programms eine Nachricht senden will, wird das Programm wesentlich übersichtlicher, wenn man den Sendevorgang in eine Subroutine auslagert.

Bisher fehlt uns noch das passende Slave-Programm, das die Nachricht empfängt, aber darum wollen wir uns jetzt kümmern:

```
 ///////////////////////
 //  Slave-Programm  //
///////////////////////

#include "NXCDefs.h"

task main()
{
 //Initialisierung
 SetSensorType(IN_4, SENSOR_TYPE_HIGHSPEED);
 SetHSState(HS_INITIALISE);
 SetHSFlags(HS_UPDATE);
```



```
//Variablen
 string nachricht;
 byte laengenachricht;

 while(true)
 {
  laengenachricht = HSInputBufferInPtr(); //Länge der Nachricht
  GetHSInputBuffer(0, laengenachricht, nachricht); //Nachricht empfangen

  TextOut(0, LCD_LINE1, nachricht); //Nachricht ausgeben

  SetHSInputBufferInPtr(0); //Länge zurücksetzen

  Wait(10);
  ClearScreen();
  nachricht =""; //Nachricht zurücksetzen
 }
}
```

Programm 20.14

Auch das Programm für den Slave-Roboter beginnt mit unseren drei Zeilen und benötigt dieselben Variablen wie das Programm für den Master-Roboter. Da wir die Länge der Nachricht nicht von Anfang an kennen, rufen wir sie mit dem Befehl `HSInputBufferInPtr()` ab. Danach reicht ein einfaches

```
GetHSInputBuffer(0, laengenachricht, nachricht);
```

um unsere Nachricht zu empfangen und in der Variablen `nachricht` zu speichern. Danach setzen wir noch mit `SetHSInputBufferInPtr(0)` das Eingangsfach auf leer, damit wir auch merken, falls keine Nachricht ankommt, aus demselben Grund setzen wir auch die Variable `nachricht` wieder auf leer zurück, denn sonst würde das Programm, falls keine neue Nachricht ankommt, einfach die alte Nachricht wieder verwenden. Es gibt natürlich auch Fälle, in denen das durchaus erwünscht sein kann, dann lässt man die letzte Zeile einfach weg.

Und damit haben Sie dann auch das komplizierteste und letzte Kapitel im Umgang mit NXT gemeistert. Wenn Sie das Buch von Anfang an aufmerksam gelesen haben, dann haben Sie jetzt den Punkt erreicht, an dem Sie alles wissen, um mit dem NXT unter der Verwendung von NXC selbst komplexeste Projekte zu realisieren. Die nachfolgenden Kapitel werden Ihnen helfen, Ihre erlangten Fähigkeiten zu intensivieren, und werden auch einen kleinen Ausblick und Anregungen darauf geben, was Sie mit Ihrem erlangten Wissen anstellen können; die Möglichkeiten sind nahezu unbegrenzt.