

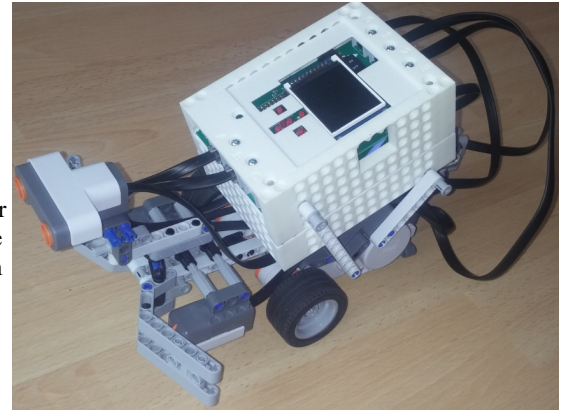
NNXT - Tutorials

[Startseite](#) [Tutorial #01](#) [Tutorial #02](#) [Tutorial #03](#) [Tutorial #04](#) [Tutorial #05](#) [Tutorial #06](#)

Tutorial #01 - Der Aufbau und das erste Programm

Grundsätzliches

Willkommen im ersten Tutorial zur NNXT-Programmierung im Fach RB! Der Roboter ist bereits aufgebaut und vorbereitet und wird in mehreren Gruppen geteilt, so dass Sie ihn nicht umbauen sollten, oder - falls doch ein Umbau mal notwendig sein sollte - ihn bitte am Ende der Übung in den ursprünglichen Zustand zurückbauen. Jeder Roboter hat eine Nummer aufgeklebt, die Sie sich bitte merken, so dass Sie in jeder Übung mit dem selben Roboter arbeiten können (die Motoren verhalten sich unterschiedlich; sollten Sie also eine Übungsaufgabe in einer Übungsstunde nicht fertig bekommen haben, ist es einfacher in der nächsten Stunde direkt weiter daran zu arbeiten).



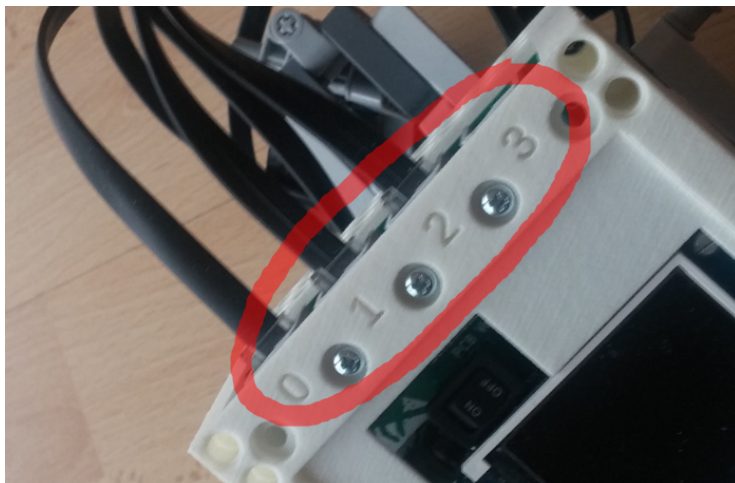
Aufbau des Roboters

Der Roboter hat in seinem Grundaufbau 2 Motoren angeschlossen, die Sie unabhängig voneinander ansteuern können. Achten Sie bitte darauf, an welchem Ports (Kennzeichnung "A", "B" oder "C") die Motoren angeschlossen sind.

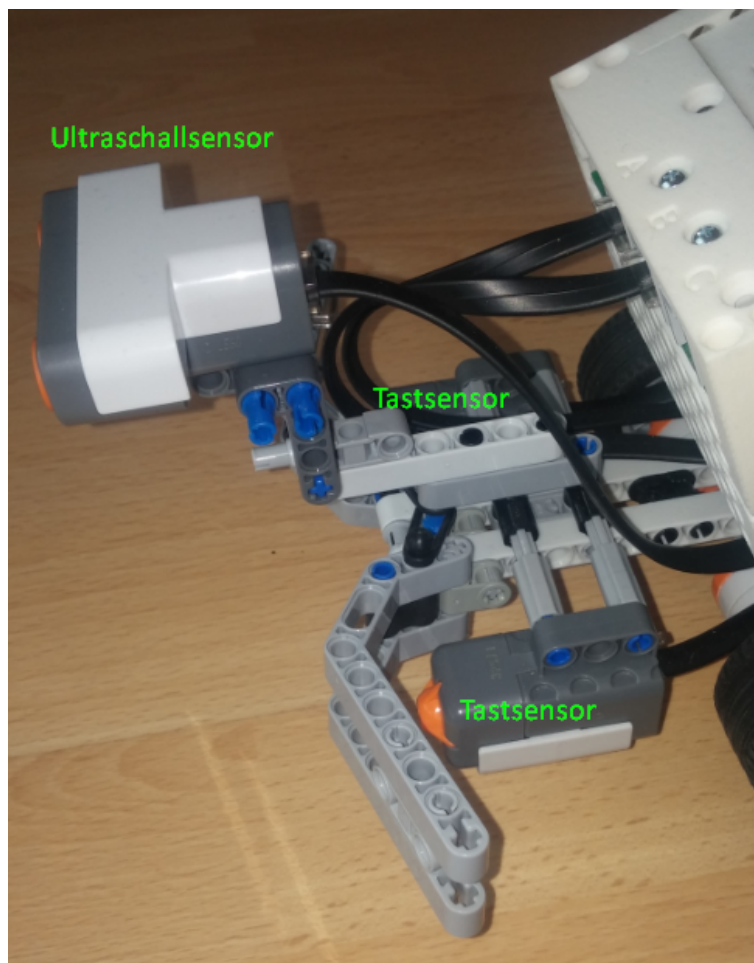


Dies kann sich von Übung zu Übung ändern, also bitte vorab immer mit der Annahme in Ihren Programmen vergleichen und ggf. Anschluss oder aber Programm ändern.

Sie können bis zu 4 Sensoren an den Roboter anschließen. Diese werden an die Ports "0" bis "3" angeschlossen. Achten Sie auch hier bei jeder Inbetriebnahme auf die Anschlüsse und vergleichen diese mit Ihren Annahmen in Ihren Programmen.



Schließen Sie bitte **niemals** Sensoren an die Motor-Ports "A" bis "C" an!
Standardmäßig sind 2 Tastsensoren an den Roboter angeschlossen, die am vorderen Ende des Roboters mit jeweils einem Hebelarm als Kontakt verbaut sind. Auf manchen Robotern ist bereits ein Ultraschallsensor als 3. Sensor verbaut:



Stromversorgung

Der Roboter verfügt über 2 Stromversorgungen: Batteriefach für 6 AA Batterien und ein USB-Anschluss

Batteriefach

Ein Batteriefach auf der Unterseite des Brick ist für Versorgung sowohl des Rechners als auch der Motoren und Sensoren zuständig. Die

Stromversorgung vom Batteriefach kann mittels des Ein-/Ausschalters oben auf dem Brick ein bzw. ausgeschaltet werden:



Wenn Sie den Roboter am Ende der Übungsstunde wieder in den Schrank an seinen Platz stellen, **stellen Sie bitte unbedingt sicher, dass der Schalter auf "OFF" steht!** Der Roboter verfügt nicht über eine Selbstabschaltung, so dass im Laufe der Zeit die Batterien verbraucht werden würden.

Sollten die Batterien einmal leer sein, so müssen Sie diese tauschen. Es kommen hierfür 6 AA-Batterien zum Einsatz, die Sie im Labor im Schrank unterstes Fach links finden. Dort steht auch ein Karton mit der Aufschrift "leere Batterien" und einem Einwurfschlitz. In diesen Einwurf bitte die 6 leeren Batterien einwerfen. **Leere Batterien bitte nur dort entsorgen, da es sich bei Batterien um Sondermüll handelt,** der nicht in die normalen Müllbehälter geworfen werden darf!

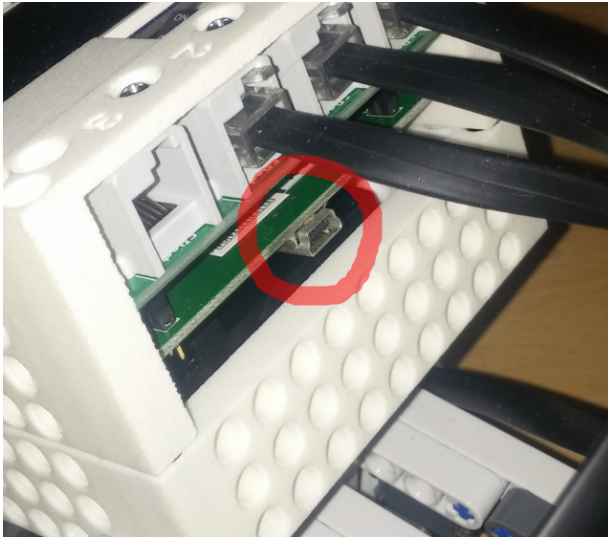
Für den Austausch der Batterien müssen Sie den Brick aus dem Roboter ausbauen. Der Aufbau ist so konzipiert, dass dies recht einfach möglich ist: Lösen Sie zunächst die seitlich angebrachten Steckverbinder auf beiden Seiten des Bricks, wie auf dem folgenden Bild dargestellt:



Lösen Sie anschließend die Kabel zu den Motoren und Sensoren und nehmen Sie den Brick aus dem Aufbau. Auf der Unterseite finden Sie dann ein handelsübliches Batteriefach, in das Sie die neuen Batterien einlegen können. **Achten Sie bitte unbedingt auf die richtige Polarität der Batterien!**

USB-Anschluss

Der USB-Anschluss dient dazu, Programme auf den Brick zu bekommen. Gleichzeitig ist er in der Lage, die Rechnerplatine des Roboters mit Strom zu versorgen, **aber nur diese!** Wenn die Batterien wie oben beschrieben abgeschaltet sind (oder keine Batterien vorhanden sind), dann sind sowohl Motoren als auch Sensoren **ohne Versorgung und können nicht funktionieren**. Für den Betrieb müssen Sie also **immer** die Stromversorgung über das Batteriefach einschalten. Neue Programme können Sie aber über USB auch auf den Roboter laden, wenn Sie den Roboter nur über ein USB-Kabel an einen Rechner angeschlossen haben (man sollte es aber nicht tun, da das Programm nach dem Herunterladen auf den Brick sofort startet und auf die Motor- und Sensor-Platine angewiesen ist). Den USB-Anschluss finden Sie im hinteren Teil des Brick unter den Anschlüssen für die Sensoren:

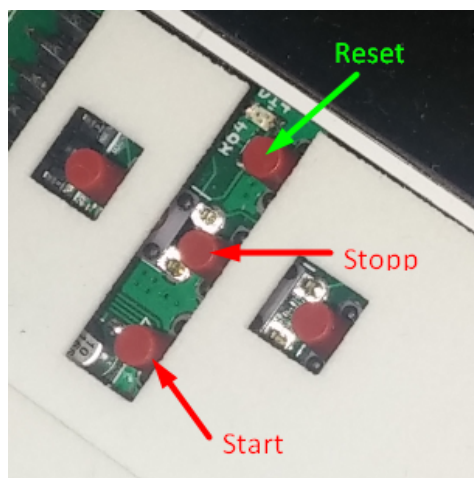


Damit haben wir alle zunächst wesentlichen Teile des Roboters kennengelernt und können nun mit dem ersten Programm beginnen!

Vorbemerkungen zum ersten Programm


Bevor das erste Programm geschrieben werden kann, zunächst ein paar grundsätzliche Informationen zur Rechnerplattform des NNXT: Auf dem Rechner läuft zunächst nichts, anders als bei denen von Ihnen gewohnten Systemen, wo ja immer ein Betriebssystem bereits vorinstalliert ist. Der NNXT wird im sogenannten "Bare Metal" Betrieb benutzt, d.h. Sie bekommen die reine HW ohne ein einziges Stück SW auf dem Gerät. Die SW des Gerätes werden Sie selbst schreiben. Natürlich bedarf es einiger Funktionen für den Betrieb und später in der Veranstaltung auch ein Betriebssystem. Diese Dinge sind jedoch immer Teil Ihrer SW. Damit Sie das nicht selbst schreiben müssen, haben wir das für Sie vorbereitet, so dass die Einstiegshürde niedrig ist. Bei der Projekterstellung wird dieser Teil der SW in Ihrem Projekt für Sie mit angelegt; also nicht wundern, wenn Sie bereits sehr viele SW-Funktionen im Projekt sehen, die Sie selbst gar nicht geschrieben haben. Wir werden hier zunächst ohne jegliches Betriebssystem beginnen und dieses erst in den späteren Übungen mit einbeziehen. Die dafür notwendigen Funktionen sind in der Dokumentation der API und den Tutorials erklärt. Da wir das Gerät "Bare Metal" betreiben, bedeutet dies auch, dass - sobald Sie das Gerät einschalten - das von irgendwem zuletzt auf den NNXT geladene Programm direkt startet. Gleichzeitig gibt es keine Verwaltung von Programmen auf dem NNXT, sondern es ist dort immer nur eines abgespeichert. Dies ist für Kleinrechner der technischen Informatik so üblich und Sie sollten sich gleich an diesen Umstand gewöhnen. Wenn Sie also mit Ihrer Übung starten und den NNXT einschalten, können Sie davon ausgehen, dass zunächst irgend ein fremdes Programm startet und keines, das Sie zuletzt geschrieben haben.

Da Programme normalerweise direkt starten, enthält die Plattform eine kleine Basis-SW, die Ihnen die Arbeit ein bisschen erleichtert. Wenn Sie Ihren Download im Debugging-Modus durchführen, wird die SW auf dem NNXT beim Anspringen Ihrer main-Funktion stoppen und sich mit dem Debugger auf dem Rechner synchronisieren, so dass Sie über die Debugging-Panels in der IDE den Ablauf steuern können. Sollten Sie den NNXT ohne Verbindung zur IDE starten, wird die SW zunächst gestartet, hängt aber direkt nach dem Start in einer Schleife fest (den sogenannten "Stopp-Modus"), die auf einen Tastendruck von Ihnen wartet. Dies ist so erstellt, damit das Gerät nicht gleich losfährt und z.B. vom Tisch fällt, da Sie ja nie wissen können, wer was für ein Programm auf den NNXT zuletzt aufgespielt hat. Sie können den Stopp-Modus durch Drücken der Start-Taste verlassen, und das Gerät wird dann das auf ihm geladene Programm durchführen. Durch Drücken der Stopp-Taste können Sie jederzeit das Gerät wieder in den Stopp-Modus bringen. Einen Überblick über die Tasten finden Sie in der folgenden Abbildung:



Sollten Sie während des Programmablaufes die Stopp-Taste gedrückt haben, so werden alle Motoren und der Sound abgestellt. Sie können dann zwar mit der Starttaste diesen Modus wieder verlassen, allerdings ist das nicht zu empfehlen, da die Einstellungen auf Motoren und Sound nicht wieder hergestellt werden. Hier ist es dann besser, mit der Reset-Taste das System neu zu starten. Auch diese Taste sehen Sie in der obigen Abbildung, und sie kann jederzeit während des Ablaufes eines Programmes gedrückt werden, wird dann einen Systemreset durchführen und endet im Stopp-Modus, aus dem Sie dann wieder per Start-Taste weitermachen können. Wenn dies so ist, warum dann überhaupt die Stopp-Taste? Der Grund dafür liegt in einer Neuprogrammierung des NNXT: wenn Sie ein neues Programm auf den NNXT spielen wollen, müssen Sie den Roboter wieder über USB an den Rechner anschließen. Dies gestaltet sich jedoch im allgemeinen als schwierig, wenn das Gerät Fahrbewegungen durchführt. Deshalb gibt es den Stopp-Knopf: durch Drücken werden alle Aktionen stillgelegt, und Sie können den Roboter bequem über USB verbinden und ein neues Programm herunterladen.

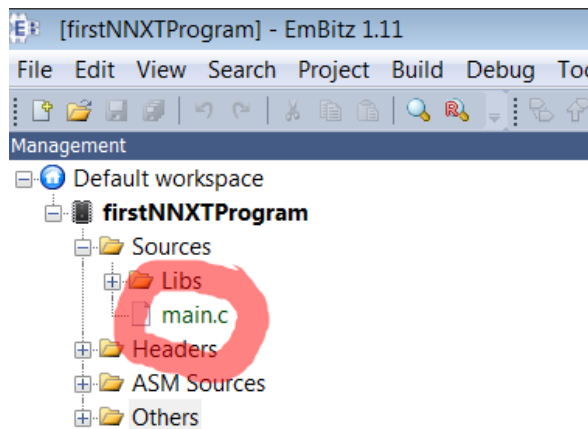
Das erste Programm

Nachdem nun also die Grundlagen geklärt sind, können Sie Ihr erstes Programm erstellen. Dafür müssen Sie zunächst ein Projekt eröffnen. Die Programmierung der NNXT-Programme erfolgt unter der EmBitz-IDE (wen es interessiert: [hier](#) die Webpage der IDE). EmBitz ist auf den Laborrechner installiert und sollte als Desktop-Icon  verfügbar sein. EmBitz erinnert an CodeBlocks und bietet die Möglichkeit,

Projekte zu erstellen, Debugging durchzuführen, Projekte auf das Target (hier der NNXT) herunter zu laden, verfügt über ein Syntaxhighlighting, Autovervollständigung und vieles mehr, was eine gute IDE zur Programmierung bieten sollte.

Um ein Projekt für den NNXT zu erstellen, müssen Sie beim Eröffnen eines neuen Projektes einige spezifische Dinge einstellen. Achten Sie insbesondere auf den Speicherort für Ihr Projekt: wie bei Laborrechnern üblich, werden Ihre Verzeichnisse ggf. bis zur nächsten Sitzung gelöscht, **Sie sollten also alle Projekte, die dauerhaft überleben sollen, auf Ihr G-Laufwerk speichern!** Wie Sie ein Projekt für den NNXT erstellen, ist [hier](#) beschrieben.

Erstellen Sie Ihr erstes Projekt wie in der [Dokumentation](#) beschrieben. In diesem Tutorial hat es den Namen "FirstNNXTProgram". Nach dem Erstellen des Projektes sind Sie in der Programmieransicht von EmBitz und sehen auf der linken Seite Ihre Projektstruktur. Ihr Programm legen Sie in der Datei main.c ab, die Sie im Source-Ordner Ihres Projektes finden:



Durch einen Doppelklick auf **main.c** können Sie es im Texteditor öffnen und sehen, was durch die Projekterstellung bereits für Sie eingefügt wurde. Das Programm wird folgendermaßen aussehen:

The image shows the text editor window for "main.c". The code is as follows:

```
1  #include "nnxt.h"
2
3
4  int main(){
5
6      return 0;
7  }
8
```

Das `#include "nnxt.h"` macht alle Funktionen, die Sie für die NNXT-Programmierung benötigen, bekannt. **Diese Zeile bitte in keinem Fall löschen!**

Ihr eigentliches Programm wird nun in der main-Funktion geschrieben, wie Sie es ja auch aus PG1 bereits kennen. Bitte verändern Sie grundsätzlich keine der anderen Source-Files, die Sie unter dem Libs-Ordner finden!

Da auf unserem Gerät zunächst nichts läuft, außer Ihrem Programm, macht es auch keinen Sinn, dass die main-Funktion sich beendet, sondern solange das Gerät angeschaltet ist, muss Ihr Programm etwas tun. Deswegen verändern wir als ersten Schritt die main-Funktion und unterteilen sie in zwei Bereiche:

- einen Start-Bereich, in dem Sie Initialeinstellungen vornehmen (Variablen festlegen und mit Werten versorgen, Sensoren definieren und initialisieren, Motoren definieren, etc.)
- einen Bereich, in dem dauernd das eigentliche Programm abläuft. Damit dieses dauernd abläuft, programmieren wir hier eine Endlos-Schleife, aus der man nie wieder heraus kommt, außer wenn man den Strom abstellt oder das System über einen Reset neu startet.

Damit sollte unser neues Programmgerüst nun folgendermaßen aussehen:

```
main.c x
1  #include "nnxt.h"
2
3
4  int main(){
5      // Initial-Bereich
6
7      // hier alle Initialisierungen festlegen
8
9
10     // Betriebsbereich
11     while(1) {
12
13         // hier das Programm aufbauen
14
15     }
16     return 0;
17 }
```

Alle Programme für den NNXT werden zunächst diese Grundstruktur haben, die wir nun mit Leben füllen können.

Als erstes Programm soll der NNXT die Motoren benutzen und ein Stück vorwärts fahren. Dazu müssen wir zunächst dem NNXT erklären, an welchen Ports jeweils ein Motor angeschlossen ist. Dieses muss nur initial geschehen und wird mittels der Funktion **MotorPortInit** eingestellt. Diese Funktion bekommt als Parameter den Port, an dem ein Motor angeschlossen ist. Die unterschiedlichen Ports sind über einen **enum**-Typ vordefiniert und können die Werte **Port_A**, **Port_B** und **Port_C** haben. Nehmen wir an, dass die Motoren an die Ports A und B angeschlossen sind (wie in der [Abbildung](#) weiter oben zu sehen). Damit ergibt sich der folgende neue Source-Code:

```
main.c x
1  #include "nnxt.h"
2
3
4  int main(){
5      // Initial-Bereich
6
7      // hier alle Initialisierungen festlegen
8      MotorPortInit(Port_A);
9      MotorPortInit(Port_B);
10
11     // Betriebsbereich
12     while(1) {
13
14         // hier das Programm aufbauen
15
16     }
17     return 0;
18 }
```

Nachdem die Motoren nun initialisiert sind, soll der Roboter das folgende Verhalten zeigen: Er soll erst für eine Sekunde vorwärts fahren, danach 1 Sekunde stehen, dann eine Sekunde rückwärts fahren und wieder eine Sekunde stehen. Dieses soll immer wieder geschehen. Dazu müssen wir die Motoren an- und ausschalten können. Grundsätzlich kann man die Motoren über eine Funktion namens **Motor_Drive** anschalten. Sobald die Motoren angeschaltet sind, werden sie sich drehen, unabhängig davon, was Ihr Programm in der Zeit macht, und sie werden erst wieder ausgeschaltet, wenn man entweder das Gerät ausschaltet oder mittels der Funktion **Motor_Stop** die Motoren wieder anhält. Schauen wir uns die Deklarationen der Funktionen an. Zunächst **Motor_Drive**:

```
sensor_error_t Motor_Drive(motorport_t port, motor_dir_t dir, uint8_t dutycycle);
```

- bei **port** muss wieder der Motorport (**Port_A**, ...) angegeben werden,
- **dir** kann einer der beiden Werte **Motor_dir_forward** oder **Motor_dir_backward** übergeben werden. Diese Werte bestimmt, in welche Richtung die Motoren drehen sollen
- der **dutycycle** schließlich gibt an, mit welcher Geschwindigkeit die Motoren drehen sollen. Hier sind Werte zwischen 0 (steht) und

100 (maximale Drehzahl des Motors) möglich.

Den Rückgabewert der Funktion können Sie ignorieren; die Rückgaben aller Funktionen der NNXT-SW-Library sind nach einem festen Standard aufgebaut und geben grundsätzlich eine Fehlerkennung zurück, die aber im Fall der Motorfunktion immer den Wert für "alles ok" zurückgibt.

Nun **Motor_Stop**:

```
sensor_error_t Motor_Stop(const motorport_t port, const motor_stop_t stop);
```

Hier muss wiederum der Port angegeben werden, an dem der Motor angeschlossen ist, der gestoppt werden soll. Als zweiter Parameter kann man noch festlegen, auf welche Weise der Motor anhalten soll: ausrollend (**Motor_stop_float**) oder abbremsend (**Motor_stop_break**). Bitte verwenden Sie, wann immer möglich, die ausrollende Variante, da das aktive Abbremsen Batteriestrom verbraucht und dieser Verbrauch auch nach dem Stoppen der Motoren anhält, um etwaige Drehungen des Motors zu verhindern.

Damit können wir nun die Motoren starten und stoppen. Jetzt müssen wir sie nur noch eine gewisse Zeitspanne fahren lassen. Hierfür benötigt man eine Wartefunktion, die von der NNXT-Library zur Verfügung gestellt wird: **Delay**.

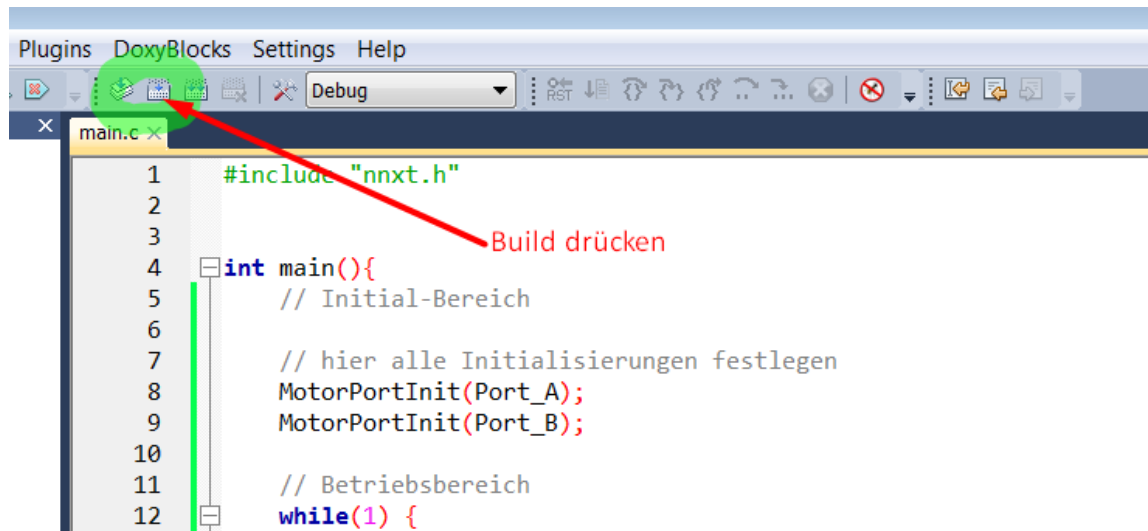
```
inline void Delay(const uint32_t ms);
```

Die **Delay**-Funktion enthält eine Warteschleife, die die als Parameter übergebene Anzahl an Millisekunden wartet und danach zurückkehrt. Möchte man also, wie oben gefordert, eine Sekunde warten, so muss man **Delay** mit einem Parameter von 1000 aufrufen.

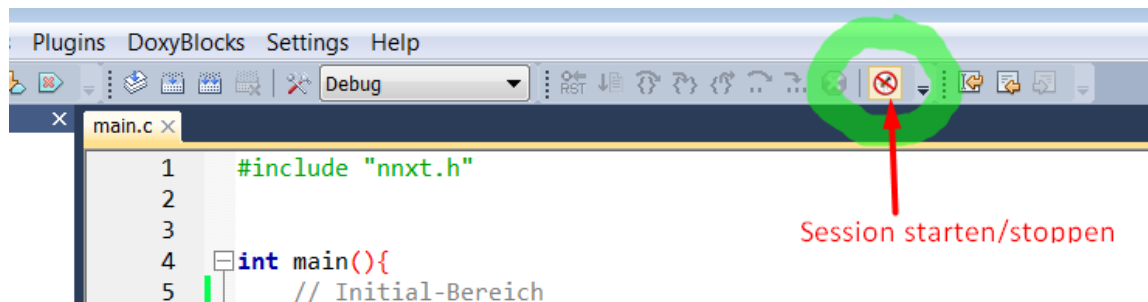
Nun haben wir alles zusammen und können das erste Programm vervollständigen:

```
main.c x
1  #include "nnxt.h"
2
3
4  int main(){
5      // Initial-Bereich
6
7      // hier alle Initialisierungen festlegen
8      MotorPortInit(Port_A);
9      MotorPortInit(Port_B);
10
11     // Betriebsbereich
12     while(1) {
13         // Beide Motoren mit 35% der maximalen Umdrehungszahl anschalten
14         Motor_Drive(Port_A, Motor_dir_forward, 35);
15         Motor_Drive(Port_B, Motor_dir_forward, 35);
16         // anschließend 1 Sekunde (=1000 Millisekunden) abwarten
17         Delay(1000);
18         // Motoren abschalten, dabei auslaufen lassen und nicht bremsen
19         Motor_Stop(Port_A, Motor_stop_float);
20         Motor_Stop(Port_B, Motor_stop_float);
21         // wieder eine Sekunde warten
22         Delay(1000);
23         // beide Motoren nun rückwärts anschalten
24         Motor_Drive(Port_A, Motor_dir_backward, 35);
25         Motor_Drive(Port_B, Motor_dir_backward, 35);
26         // anschließend 1 Sekunde (=1000 Millisekunden) abwarten
27         Delay(1000);
28         // Motoren abschalten, dabei auslaufen lassen und nicht bremsen
29         Motor_Stop(Port_A, Motor_stop_float);
30         Motor_Stop(Port_B, Motor_stop_float);
31         // wieder eine Sekunde warten
32         Delay(1000);
33         // und nun ist Schleife beendet und kann von neuem beginnen
34     }
35     return 0;
36 }
```

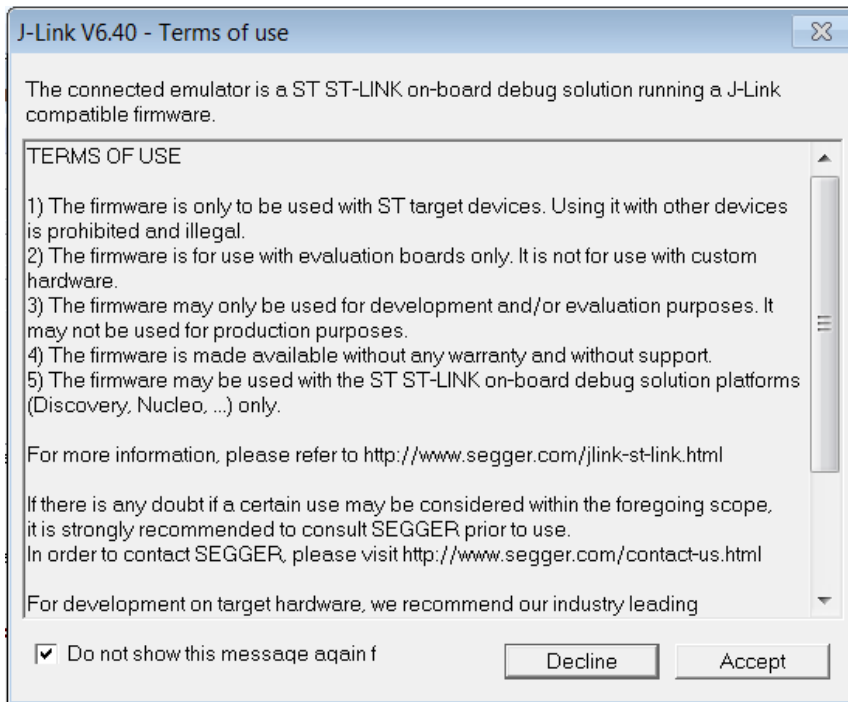

Dieses fertige Programm muss nun übersetzt werden. Sollte Ihre Build-Art auf "Release" stehen, so schalten Sie diese bitte auf "Debug" um, wie im unteren Bild zu sehen ist. Dann bitte den "Build"-Knopf drücken:



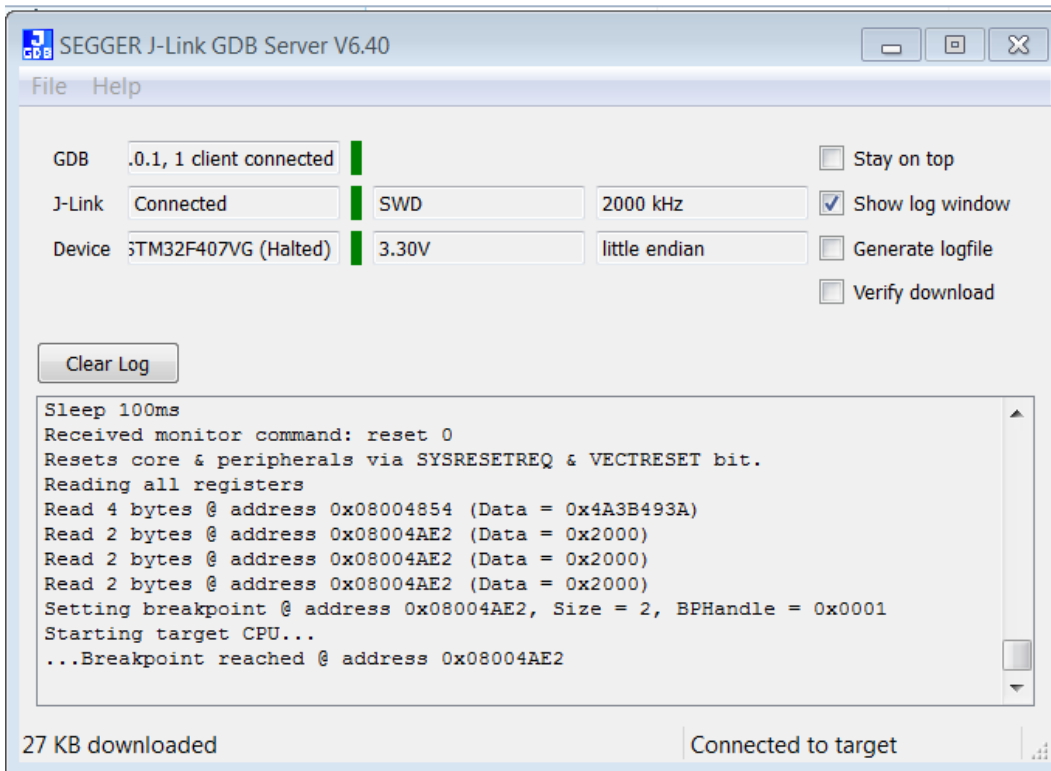
Anschließend im Fenster unten die Fehlermeldungen studieren und ggf. beheben und erneut den Build-Vorgang anstoßen. Wenn das Programm fehlerfrei übersetzt wurde, dann kann es auf den NNXT heruntergeladen werden. Hierzu müssen wir eine Session für den Debugger eröffnen, der die IDE über USB mit dem Roboter verbindet und den Roboter informiert, dass jetzt ein neues Programm kommt. Dafür gibt es in der Toolleiste oben einen Knopf namens "Session starten/stoppen", den Sie drücken müssen:



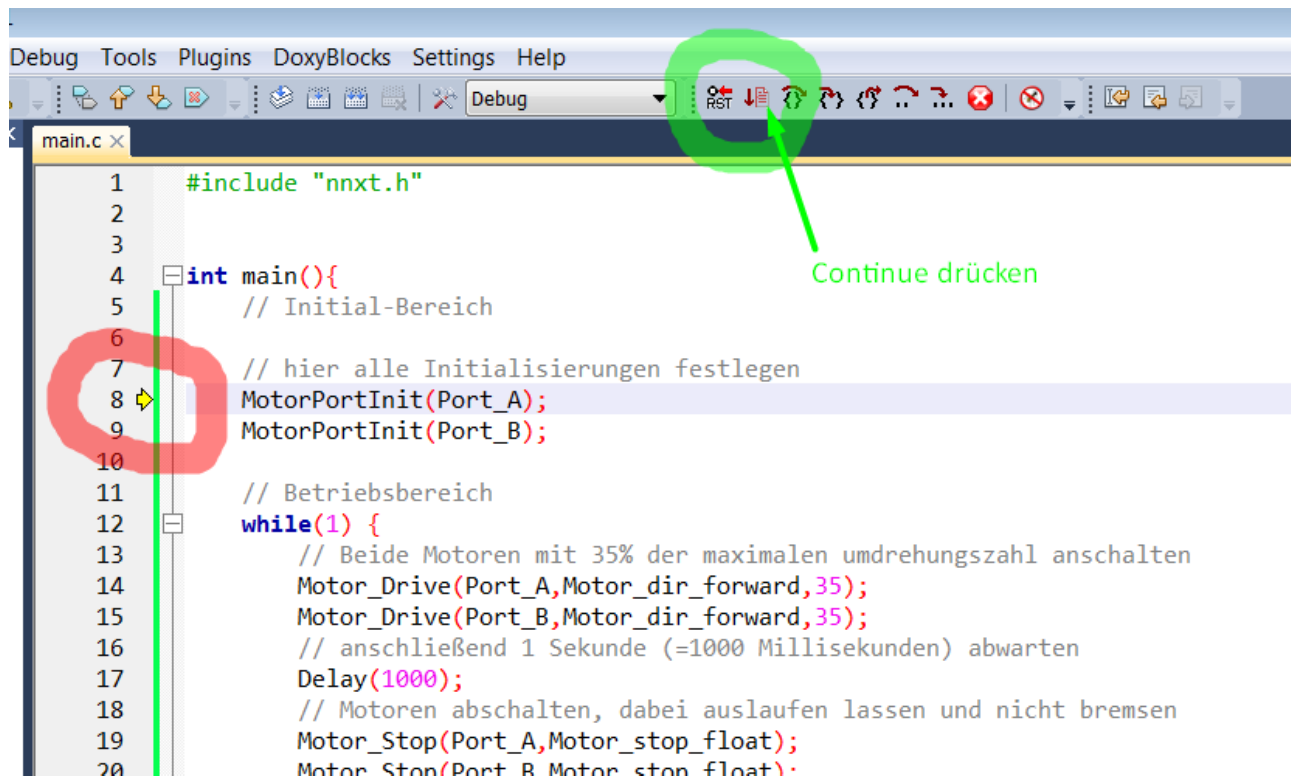
Hiermit wird eine Verbindung über den sogenannten JLink mit dem Roboter hergestellt. Da JLink ein zusätzliches Werkzeug ist, müssen Sie ggf. die Terms of Use annehmen, wie auf dem folgenden Screenshot zu sehen ist:



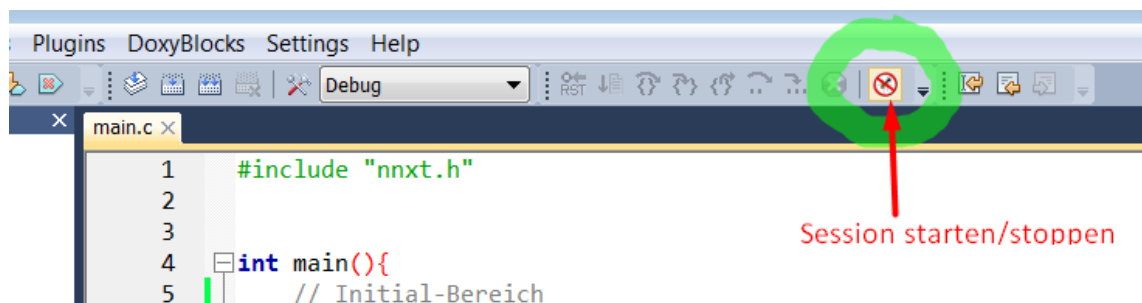
Hier am besten "Do not show ..." anhaken und "Accept" drücken. Anschließend wird das Log-Fenster der JLink-Verbindung aufgehen, und Sie müssen warten, bis Ihr Programm auf den Roboter aufgespielt wurde:



Sobald Sie obiges Fenster so sehen, ist das Programm aufgespielt und der Debugger gestartet. Ihr Programm läuft dann noch nicht, sondern erst, wenn Sie Ihr EmBitz-fenster wieder in den Vordergrund holen, sehen Sie, dass der Debugger gestartet ist und nun an der ersten Anweisung der main-Funktion angehalten hat. Durch einen Druck auf "Continue" in den Debugger-Tools können Sie nun Ihr Programm auf dem NNXT starten und die Bewegungen des Roboters bestaunen:



Wenn Sie nach dem Test das Programm verändern und neu übersetzen möchten, muss die Session zum JLink-Debugger übrigens wieder getrennt werden. Dies können Sie wieder über einen Klick auf das kleine rote Sessionwerkzeug in der Werkzeugleiste des EmBitz erreichen:



Damit sind nun alle Schritte zur Programmerstellung abgeschlossen. Probieren Sie nun ein wenig herum, verändern Sie die Motoransteuerung, die Zeiten, etc.

Wenn Sie mehr über das Debugging erfahren möchten, so schauen Sie in [dieser Dokumentation](#) nach.

Programmvereinfachung

In diesem Punkt soll das Programm noch etwas vereinfacht werden und Sie lernen dabei den Umgang mit den Datentypen der NNXT-Library und der Unterstützung durch die IDE EmBitz.

Zunächst wollen wir den Programmcode vereinfachen: die beiden Teile Vorwärtsfahren und Rückwärtsfahren unterscheiden sich ja letztlich nur durch den **dir**-Parameter der **Motor_Drive**-Funktion. Folglich können wir den einfach bei jedem Schritt umdrehen, müssen ihn also in eine Variable legen. Dazu wird die Endlosschleife zunächst umgeschrieben, indem wir eine neue Variable **motorDir** für die einzustellende Richtung der Motoren verwenden und den zweiten Teil der Motoransteuerung (das Rückwärtsfahren) entfernen:

```

main.c x
1  #include "nnxt.h"
2
3
4  int main(){
5      // Initial-Bereich
6
7      // hier alle Initialisierungen festlegen
8      MotorPortInit(Port_A);
9      MotorPortInit(Port_B);
10
11     // Betriebsbereich
12     while(1) {
13         // Beide Motoren mit 35% der maximalen umdrehungszahl anschalten
14         Motor_Drive(Port_A,motorDir,35);
15         Motor_Drive(Port_B,motorDir,35);
16         // anschließend 1 Sekunde (=1000 Millisekunden) abwarten
17         Delay(1000);
18         // Motoren abschalten, dabei auslaufen lassen und nicht bremsen
19         Motor_Stop(Port_A,Motor_stop_float);
20         Motor_Stop(Port_B,Motor_stop_float);
21         // wieder eine Sekunde warten
22         Delay(1000);
23         // und nun ist Schleife beendet und kann von neuem beginnen
24     }
25     return 0;
26 }

```

Nun muss diese Variable noch deklariert werden. Dazu müssen wir ihren Typ wissen. Dies kann man in der [API-Dokumentation](#) für die Motoren nachschauen, oder aber man nutzt die Möglichkeiten der IDE EmBitz aus, sich Deklarationen und Implementierungen von Typen und Funktionen anzeigen zu lassen. Letzteres hier als Beispiel:

Gehen Sie mit dem Cursor auf die Funktion **Motor_Drive** und drücken Sie die rechte Maustaste. Im dann aufgehenden Kontextmenü können Sie dann *Find declaration of: 'Motor_Drive'* auswählen:

```

while(1) {
    // Beide Motoren mit 35% der maximalen umdr
    Motor_Drive(Port_A,motorDir,35);
    Motor_Drive(Port_B,motorDir,35);
    // ansch
    Delay(1000);
    // Motor
    Motor_St
    Motor_St
    // wiede
    Delay(1000);
    // und n
}

```

- Find declaration of: 'Motor_Drive'
- Find implementation of: 'Motor_Drive'
- Find functions called by 'Motor_Drive'
- Find functions calling 'Motor_Drive'
- Find references of: 'Motor_Drive'
- Swap header/source
- Edit

Sobald Sie das angewählt haben, wird eine neues Textfenster aufgemacht, in dem Sie die Deklaration der ausgewählten Funktion sehen. Hier kann dann am Parameter **dir** auch der Typ **motor_dir_t** abgelesen werden:

```

nnxt_Ports.h x main.c
212 * @addtogroup Motorports_functions
213 * @{
214 */
215
216 /**
217 * @brief Initialize a motor port. This includes pins and ports configuration and resetting
218 * @param port The port to initialize.
219 * @warning This function must not be called in user code! It will be called during startup of
220 */
221 void MotorPortInit(motorport_t port);
222
223 /**
224 * @brief Starts the given Motor with direction and the duty cycle. Drive until #Motor.
225 * @param port The motorport, see #motor_ports_e
226 * @param dir The direction the motor shall drive, see #motor_dir_t
227 * @param duty cycle The duty cycle in percent. Values greater than 100 leads also in full :
228 * @return One of the errors described in #sensor_error_e
229 */
230 sensor_error_t Motor_Drive(motorport_t port, motor_dir_t dir, uint8_t duty cycle);
231
232 /**
233 * @brief Stops the given motor

```

Folglich müssen wir der Variable **motorDir** diesen Typ zuweisen. Gleichzeitig wollen wir noch die Initialeinstellung auf "Vorwärts" setzen. Dies geht am besten, wenn Sie die Autovervollständigungsfunktion von EmBitz nutzen:

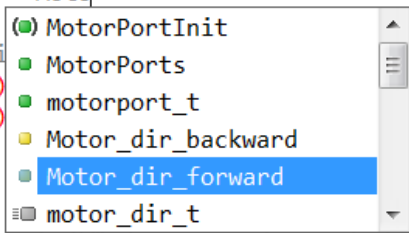
```

int main(){
    // Initial-Bereich
    motor_dir_t motorDir = MotorDir

    // hier alle Initiali
    MotorPortInit(Port_A)
    MotorPortInit(Port_B)

    // Betriebsbereich
    while(1) {
        // Beide Motoren mit 35% der maximalen Umdrehungs
        Motor_Drive(Port_A, motorDir, 35);
        Motor_Drive(Port_B, motorDir, 35);
    }
}

```



Anschließend muss die Richtung am Ende der Endlosschleife noch auf den jeweils anderen Wert gesetzt werden:


```

1  #include "nnxt.h"
2
3
4  int main(){
5      // Initial-Bereich
6      motor_dir_t motorDir = Motor_dir_forward;
7
8      // hier alle Initialisierungen festlegen
9      MotorPortInit(Port_A);
10     MotorPortInit(Port_B);
11
12     // Betriebsbereich
13     while(1) {
14         // Beide Motoren mit 35% der maximalen umdrehungszahl anschalten
15         Motor_Drive(Port_A,motorDir,35);
16         Motor_Drive(Port_B,motorDir,35);
17         // anschließend 1 Sekunde (=1000 Millisekunden) abwarten
18         Delay(1000);
19         // Motoren abschalten, dabei auslaufen lassen und nicht bremsen
20         Motor_Stop(Port_A,Motor_stop_float);
21         Motor_Stop(Port_B,Motor_stop_float);
22         // wieder eine Sekunde warten
23         Delay(1000);
24         // Fahrtrichtung umkehren
25         if (motorDir == Motor_dir_forward)
26             motorDir = Motor_dir_backward;
27         else
28             motorDir = Motor_dir_forward;
29         // und nun ist Schleife beendet und kann von neuem beginnen
30     }
31     return 0;
32 }

```

Damit haben wir wieder ein funktionstüchtiges Programm, welches das gleiche Verhalten hat wie die ursprüngliche Implementierung. Sie haben hier auch gesehen, wie Sie unter Nutzung der Möglichkeiten der EmBitz-IDE durch Typen und Funktionen der NNXT-Library navigieren können, um sich die Informationen zu beschaffen, die für das Programm notwendig sind. Im nächsten Schritt soll dann auch noch etwas auf dem Display angezeigt werden, damit das Projekt schöner wird ;-)

Nutzung des Display

Für das Debugging im printf-Style, wie Sie es aus PG1 kennen, bietet sich das Display des NNXT an. Das eingebaute Farbdisplay verfügt über 128x160 Pixel. Jedes Pixel kann in einer von 65536 Farben leuchten. Sie können vielfältige Dinge auf dem Display darstellen, in diesem Tutorial interessieren aber zunächst nur die einfachen Funktionen zum Drucken von Text auf das Display. Weitere Funktionen entnehmen Sie bitte der [Dokumentation der Display-API](#).

Zunächst soll unserer Roboter uns begrüßen und dazu einen simplen Begrüßungstext "Hallo Meister!" auf das Display bringen. Hier für benötigen wir die Funktion **NNXT_LCD_DisplayStringAtLine**, deren Signatur wir nachschlagen:

```
void NNXT_LCD_DisplayStringAtLine(uint16_t Line, char *ptr);
```

- Der erste Parameter Line bezeichnet die Zeile, in die der Text gedruckt werden soll. Das Display ist für die Textausgabe zeilenweise organisiert, wobei die erste Zeile die Ordnungsnummer 0 hat. Wieviele Zeilen Sie auf das Display benutzen können, hängt von der Font-Größe ab, die wir aber im Rahmen dieses Tutorials nicht weiter betrachten. Wer dort mehr wissen möchte, der sei auf die [Dokumentation der Display-API](#) für ein Selbststudium verwiesen.
- Der zweite Parameter ist ein Zeiger auf einen Speicherbereich, in dem der Text als '\0'-terminierte Zeichenkette steht.

Damit können wir nun unser Programm erweitern:

```
main.c x
1  #include "nnxt.h"
2
3
4  int main(){
5      // Initial-Bereich
6      motor_dir_t motorDir = Motor_dir_forward;
7
8      // Begrüßungsnachricht
9      NNXT_LCD_DisplayStringAtLine(0, "Hallo Meister!");
10
11     // hier alle Initialisierungen festlegen
12     MotorPortInit(Port_A);
13     MotorPortInit(Port_B);
14
15     // Betriebsbereich
16     while(1) {
17         // Beide Motoren mit 35% der maximalen Umdrehungszahl anschalten
18         Motor_Drive(Port_A, motorDir, 35);
19         Motor_Drive(Port_B, motorDir, 35);
20         // anschließend 1 Sekunde (=1000 Millisekunden) abwarten
21         Delay(1000);
22         // Motoren abschalten, dabei auslaufen lassen und nicht bremsen
23         Motor_Stop(Port_A, Motor_stop_float);
24         Motor_Stop(Port_B, Motor_stop_float);
25         // wieder eine Sekunde warten
26         Delay(1000);
27         // Fahrtrichtung umkehren
28         if (motorDir == Motor_dir_forward)
29             motorDir = Motor_dir_backward;
30         else
31             motorDir = Motor_dir_forward;
32         // und nun ist Schleife beendet und kann von neuem beginnen
33     }
34     return 0;
35 }
```

Testen Sie ein bisschen mit den Zeilennummern, den Breiten der Texte etc., um ein Gefühl für das Display zu bekommen.

Sie sehen bei den Parametern, dass anstelle der Standardtypen `int` oder `short int` hier Datentypen aus der `stdtypes.h` (die hier implizit eingebunden ist) genommen werden. Dies ist im Bereich der Programmierung technischer Kleinrechner Standard, da die eingebauten Datentypen, wie etwa `int`, in ihrer Bitbreite vom Prozessor und Compiler abhängig sind und sich ändern können, wenn Sie eines davon austauschen. Daher bietet die `stdtypes.h` eine Reihe von Datentypen an, die in ihrer Bitbreite genormt und von jedem Compiler entsprechend auf die eingebauten Datentypen angepasst sind. Die am häufigsten verwendeten integer Datentypen sind:

- `uint8_t` und `sint8_t` für 8-Bit Datentypen (u steht für unsigned, s für signed)
- `uint16_t` und `sint16_t` für 16-Bit Datentypen
- `uint32_t` und `sint32_t` für 32-Bit Datentypen

Sie sollten sich ebenfalls angewöhnen, diese Datentypen anstelle der allgemeinen und in der Bitbreite nicht normierten Datentypen, wie `int` oder `short int`, zu verwenden.

Aber nun zurück zum Display: Als nächstes möchten wir noch Zahlen auf dem Display darstellen. Dazu definieren wir uns einfach einen Zähler, der die Anzahl der Durchläufe durch die Schleife mitzählt, und diesen Wert möchten wir darstellen. Um die Zahl darstellen zu können, muss sie in einen Text umgewandelt werden. Dies geschieht am einfachsten mittels der Standardfunktion `sprintf` aus der `stdio.h` (wie in PG1 bereits kennengelernt). Daher können wir unser Programm auf einfache Weise erweitern:

```

main.c x
1  #include "nnxt.h"
2  #include <stdio.h>
3
4  int main(){
5      // Initial-Bereich
6      motor_dir_t motorDir = Motor_dir_forward;
7      // Zähler für die Anzahl der Schleifendurchläufe
8      uint16_t loopCounter = 0;
9      // char-Feld für die Ausgabe des Schleifenzählers
10     char loopCounterStr[20];
11     // Begrüßungsnachricht
12     NNXT_LCD_DisplayStringAtLine(0, "Hallo Meister!");
13     // hier alle Initialisierungen festlegen
14     MotorPortInit(Port_A);
15     MotorPortInit(Port_B);
16     // Betriebsbereich
17     while(1) {
18         // Schleifenzähler in Zeile 2 ausgeben
19         sprintf(loopCounterStr, "Schleife: %d", loopCounter);
20         NNXT_LCD_DisplayStringAtLine(2, loopCounterStr);
21         // Beide Motoren mit 35% der maximalen Umdrehungszahl anschalten
22         Motor_Drive(Port_A, motorDir, 35);
23         Motor_Drive(Port_B, motorDir, 35);
24         // anschließend 1 Sekunde (=1000 Millisekunden) abwarten
25         Delay(1000);
26         // Motoren abschalten, dabei auslaufen lassen und nicht bremsen
27         Motor_Stop(Port_A, Motor_stop_float);
28         Motor_Stop(Port_B, Motor_stop_float);
29         // wieder eine Sekunde warten
30         Delay(1000);
31         // Fahrtrichtung umkehren
32         if (motorDir == Motor_dir_forward)
33             motorDir = Motor_dir_backward;
34         else
35             motorDir = Motor_dir_forward;
36         // Schleifenzähler erhöhen
37         loopCounter++;
38         // und nun ist Schleife beendet und kann von neuem beginnen
39     }
40     return 0;
41 }

```

Beachten Sie bitte, dass die Funktion **NNXT_LCD_DisplayStringAtLine** nur über das bisher im Display stehende druckt, d.h. wenn Sie bei einem späteren Mal in die gleiche Zeile drucken, der gedruckte Text aber kürzer als der bisher dort stehende Text ist, dann wird der Text nur soweit verändert, wie der neue Text lang ist. Der Rest der Zeile wird nach wie vor dort stehen. Das ist vor allem bei Zahlen mit wechselnder Stellenanzahl manchmal sehr unschön, da man dieses Artefakt nicht auf Anhieb erkennen kann. In einem solchen Fall ist es besser, zunächst die Zeile mit lauter Leerzeichen zu überschreiben und anschließend den neuen Text zu drucken.

Damit ist dieses Tutorial beendet, und Sie können sich den weiteren Aufgaben von Übungszettel 1 widmen.

Viel Spaß dabei!