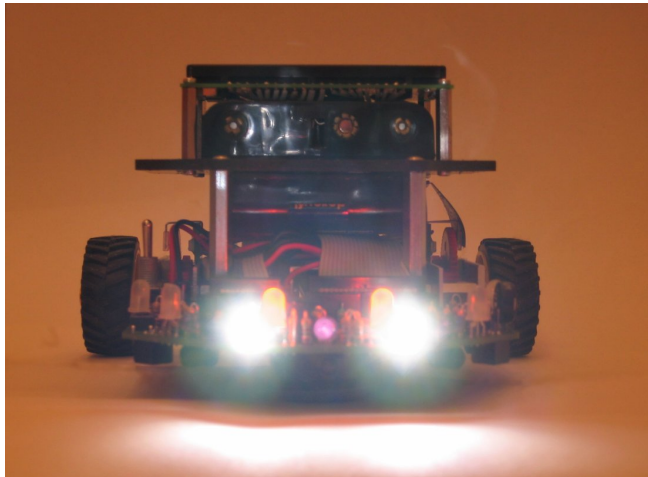


Roboterbausatz Nibo

Tutorial zur Programmierung



Inhaltsverzeichnis

1	Einleitung.....	3
2	Installation der Programmierumgebung.....	4
2.1	AVR Studio 4.....	4
2.2	WinAVR.....	4
2.3	Nibo Library.....	4
3	Ein erstes Projekt anlegen.....	5
4	Ein erstes Testprogramm.....	8
5	LEDs in Aktion.....	10
6	Inbetriebnahme des Displays.....	12
7	Darstellung der Sensorwerte.....	14
8	Bewegung – Ab die Post!.....	19
9	Links zu weiterführenden Internetseiten.....	22

1 Einleitung

Dieses Tutorial bietet eine Schritt für Schritt Anleitung für die erste Inbetriebnahme des Nibo. Mittels kleiner, ausführlich erklärter Beispiele soll der Leser mit der grundlegenden Funktionalität der Kernkomponenten des Roboters vertraut gemacht werden.

Sie lernen in den folgenden Abschnitten wie die Programmierumgebung installiert wird, wie die LEDs zum Blinken/Leuchten gebracht werden können, wie Sie Text und Sensorwerte auf dem Display sichtbar machen können und letztendlich auch, wie Sie den Nibo in Bewegung bringen!

Das Tutorial richtet sich an Robotik-/Programmieranfänger, um ihnen einen leichten Einstieg in die Gebiete der Programmierung, insbesondere der Mikrocontroller-Programmierung zu ermöglichen.

2 Installation der Programmierumgebung

2.1 AVR Studio 4

Zunächst muss das AVR Studio 4 von Atmel installiert werden. Hierzu laden Sie sich die Installationsdatei von folgender URL herunter, vorab verlangt Atmel von Ihnen eine Registrierung:

http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725

Doppelklicken Sie nun das heruntergeladene .exe-File und installieren Sie das AVR Studio 4 auf Ihrem Computer.

AVR Studio 4 ist eine von Atmel kostenlos zur Verfügung gestellte Entwicklungsumgebung (IDE) für AVR-Mikrocontroller mit der Sie Ihre Nibo-Projekte verwalten können.

2.2 WinAVR

Anschließend wird das WinAVR installiert. Hierzu laden Sie die aktuelle Version der Datei **WinAVR-xxx-install.exe** von der Seite <http://sourceforge.net/projects/winavr/> herunter. Doppelklicken Sie anschließend auf die Datei und folgen Sie dem Installationsmanager.

WinAVR ist eine Sammlung von vielen wichtigen Softwarepaketen für die AVR-Entwicklung unter Windows. Die Sammlung enthält unter anderem den C/C++ Compiler avr-gcc, die C-Standardbibliothek avr-libc, die „binutils“ und die Programmiersoftware AVRDUDE.

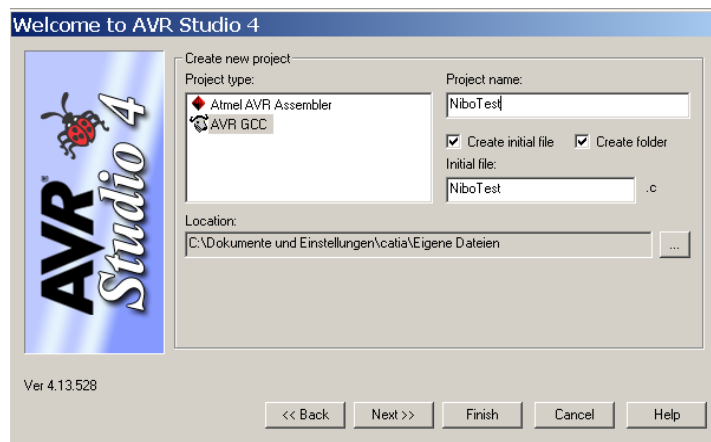
2.3 Nibo Library

Jetzt wird die Nibo Library installiert. Hierzu laden Sie sich den Installer **nibolib-xxx.msi** von der Seite <http://sourceforge.net/projects/nibo/> herunter und starten den Installer mit einem Doppelklick.

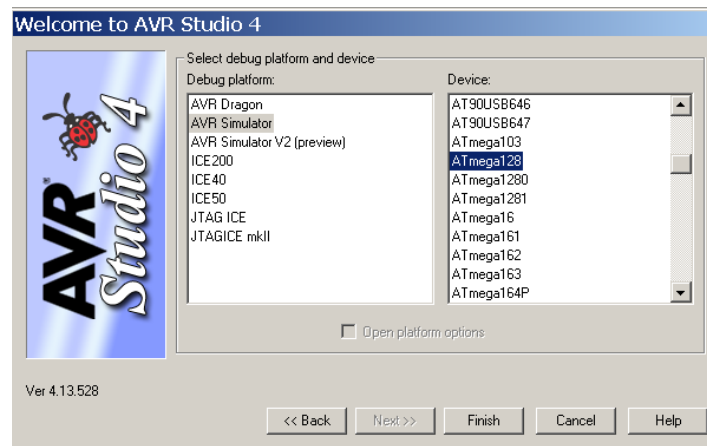
Die NiboLib enthält C und C++ Routinen zur einfachen Ansteuerung des Roboters.

3 Ein erstes Projekt anlegen

Nun soll im AVR Studio ein neues Projekt angelegt werden. Starten Sie dazu das AVR Studio 4. Klicken Sie im Willkommensbildschirm auf „*New Project*“. Im nun erscheinenden Fenster wählen Sie als *Project type* **AVR GCC** aus und tragen bei *Project name* **NiboTest** ein.

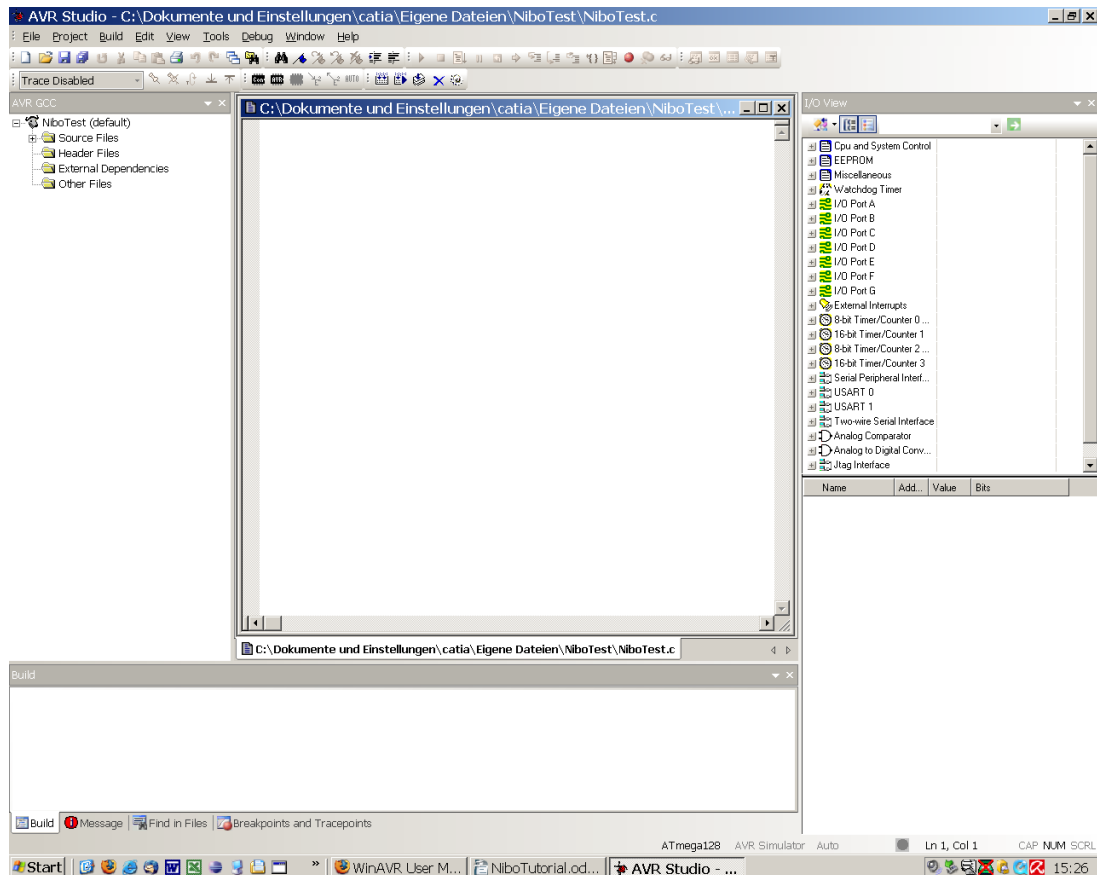


Klicken Sie auf *Next*. Im folgenden Bildschirm wählen Sie als *Debug platform* **AVR Simulator** und als *Device* **ATmega128** aus und klicken auf *Finish*.



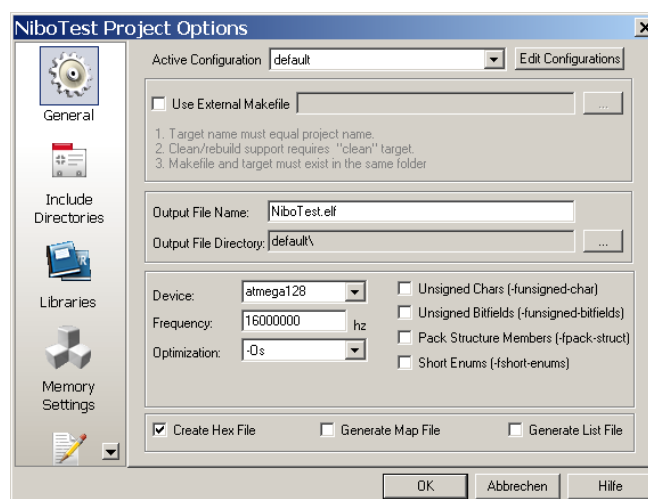
Unser neues Projekt „NiboTest“ ist nun angelegt!

Das AVR Studio sollte in etwa folgender Abbildung gleichen:



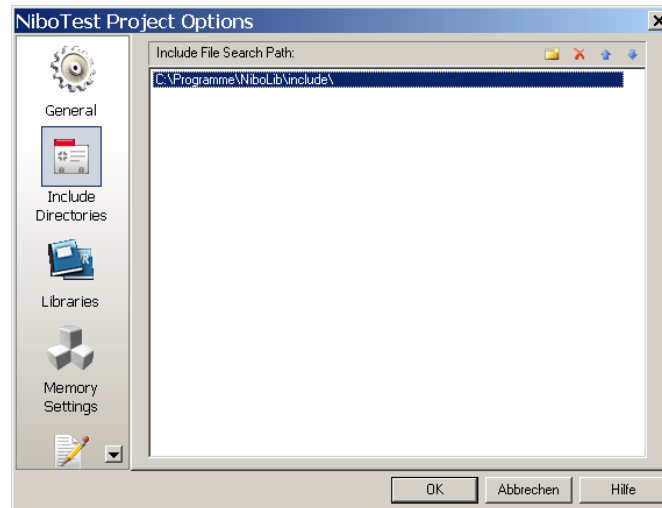
Nun müssen noch einige Voreinstellungen eingegeben werden.

Öffnen Sie dazu den *Project Options* Dialog: In der Menüleiste auf *Project -> Configuration Options* klicken.



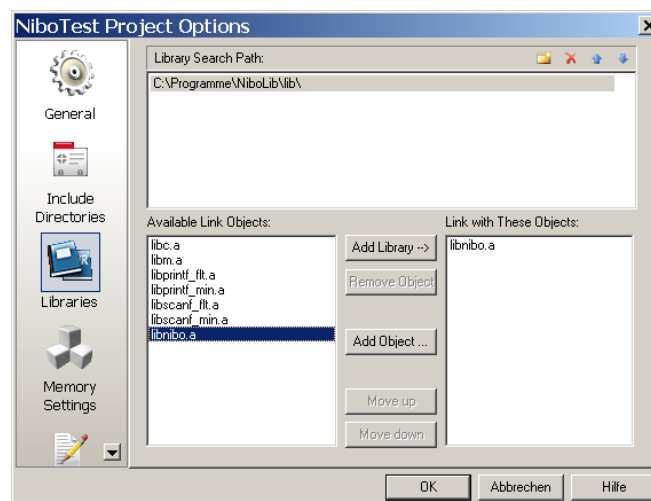
Tragen Sie im Bereich *General* im Feld *Frequency* den Wert **1600000** ein und wählen Sie im Feld *Optimization* den Wert **-Os** aus.

Wählen Sie nun im linken Bereich *Include Directories* aus. Klicken Sie auf das *Neuer Ordner* Symbol und tragen als *Include File Search Path* **C:\Programme\NiboLib\include** ein.



Im Bereich *Libraries* tragen Sie bei *Library Search Path* den Pfad **C:\Programme\NiboLib\lib** ein.

Dann wählen Sie aus den *Available Link Objects* **libnibo.a** aus und klicken auf den Button *Add Library->*. Die Bibliothek *libnibo.a* sollte nun im rechten Fenster erscheinen. Bestätigen Sie diese Einstellungen mit OK.

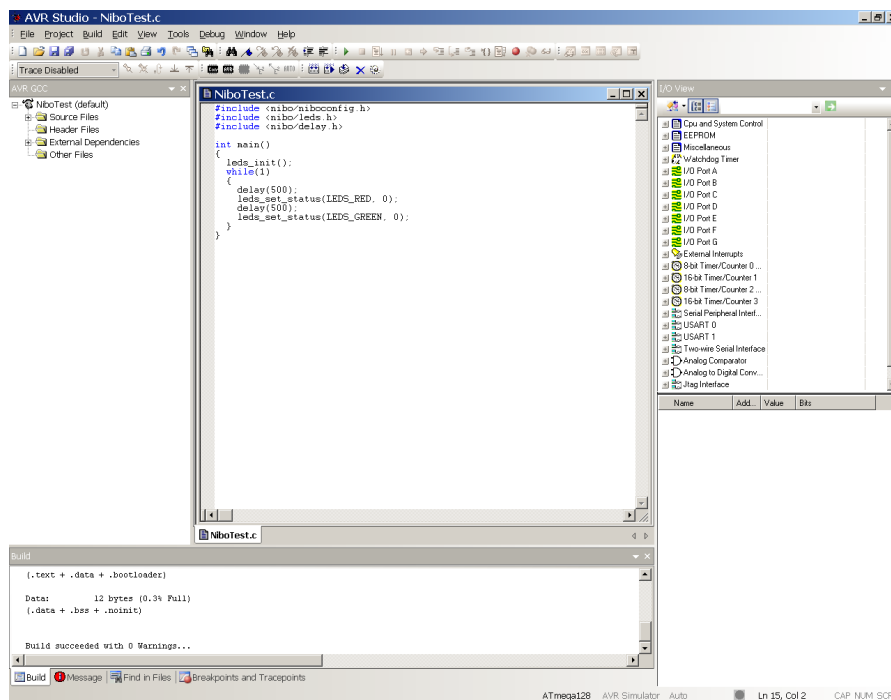


4 Ein erstes Testprogramm

Tippen Sie als erstes Testprogramm folgendes ein:

```
#include <nibo/niboconfig.h>
#include <nibo/leds.h>
#include <nibo/delay.h>

int main()
{
    leds_init();
    while(1==1)
    {
        delay(500);
        leds_set_status(LED_RED, 0);
        delay(500);
        leds_set_status(LED_GREEN, 0);
    }
    return 0;
}
```



Jetzt wird das Programm kompiliert, indem Sie den *Build*-Knopf drücken. Der Compiler und der Linker erzeugen dabei das sogenannte HEX-File, das Sie mit einem geeigneten Programmieradapter auf den Atmega128 des Nibos

übertragen müssen. Dafür bietet sich der speziell für den Nibo entwickelte Programmieradapter UCOM-IR an (siehe <http://ucom-ir.nicai-systems.de>). Alternativ kann auch das STK500 oder der AVR-ISP Programmieradapter von Atmel verwendet werden.

Schließen Sie den Programmieradapter an die entsprechende Schnittstelle Ihres Computers an. Sobald der Programmieradapter auf der anderen Seite mit dem Nibo verbunden ist, kann die Programmierung losgehen – wie das funktioniert sollte in der Bedienungsanleitung des Programmieradapters erklärt sein!

Wenn alles geklappt hat, sollte die LED2 (neben dem Poti) jetzt abwechseln rot und grün aufleuchten.

Erläuterungen zum Quellcode:

Die ersten drei Programmzeilen beginnen jeweils mit einer # und sind somit Präprozessoranweisungen. Der Präprozessor wird angewiesen, die in den spitzen Klammern angegebenen header-Dateien (.h-Dateien) einzubinden. Das Schlüsselwort hierfür heißt *include*.

Mit den Zeilen `int main()` und `{` beginnt das Hauptprogramm. Es endet mit `}`. Zwischen diesen geschweiften Klammern stehen alle Anweisungen, die abgearbeitet werden sollen.

Mit `leds_init()`; werden die IO-Ports für die LEDs initialisiert.

Der letzte Programmteil besteht aus einer *while*-Schleife `while(1==1){...}`. Solange die Bedingung in den runden Klammern, hier `1==1`, wahr ist, wird die *while*-Schleife ausgeführt. In unserem Beispiel handelt es sich also um eine Endlosschleife.

In den geschweiften Klammern steht der so genannte Anweisungsblock der von der *while*-Schleife immer wieder ausgeführt wird.

Die Anweisung `delay(500)`; weist den Controller an, 500 ms zu warten.

Die zweite Anweisung `leds_set_status(LED_RED, 0)`; setzt den Status der LED 0 auf *RED*. Damit leuchtet die LED Nummer 0 (neben dem Poti) rot.

Dann nochmals 500 ms warten.

Die vierte Anweisung `leds_set_status(LED_GREEN, 0)`; setzt den Status der LED 0 auf *GREEN*. Damit leuchtet die LED Nummer 0 jetzt grün.

Die letzte Anweisung `return 0`; wird nie ausgeführt, da die Endlosschleife nicht verlassen wird. Sie muss jedoch aus formalen Gründen vorhanden sein, da es sonst eine Compiler-Warnung gibt.

Das war's schon!

5 LEDs in Aktion

Hier wollen wir nun etwas kompliziertere Leuchtmuster gestalten.

Als erstes legen Sie ein neues Projekt mit dem Namen **Leuchtdioden1**, wie in Kapitel 3 beschrieben, an. Denken Sie dabei an die zusätzlichen Einstellungen bei den *Project-Options*.

Die Leuchtdioden sollen nacheinander reihum in einer Farbe eingeschaltet werden. Die Farbe soll bei jedem Umlauf zwischen *Aus*, *Rot*, *Grün* und *Orange* wechseln.

Tippen Sie folgenden Quellcode in das Editorfenster des AVR Studio ein:

```
#include <nibo/niboconfig.h>
#include <nibo/leds.h>
#include <nibo/delay.h>

int main()
{
    leds_init();
    while(1==1)
    {
        int farbe;
        for (farbe=0; farbe<4; farbe++)
        {
            int ledNr;
            for (ledNr=0; ledNr<6; ledNr++)
            {
                leds_set_status(farbe, ledNr);
                delay(150);
            }
        }
    }
    return 0;
}
```

Erläuterungen zum Quellcode:

Dieser Quellcode unterscheidet sich nur im Anweisungsblock der main-Funktion von dem Programm aus Kapitel 4.

Es beginnt wieder mit der Initialisierung der LEDs und einer while-Schleife, die „endlos“ läuft.

Mit der ersten Anweisung innerhalb der while-Schleife `int farbe;` wird eine Variable namens „farbe“ vom Typ `int` (Ganzzahl) deklariert.

Als nächstes werden zwei ineinander geschachtelte for-Schleifen verwendet.

Die äußere Schleife läuft über den Wert der Variablen `farbe`. Die Variable wird bei 0 initialisiert und nimmt bis zur Abbruchbedingung nacheinander die Werte 0, 1, 2 und 3 an.

Mit der Anweisung `int led;` wird eine Variable namens „ledNr“ vom Typ `int` deklariert.

Diese wird in der inneren Schleife verwendet. In der inneren for-Schleife stehen nun auch die eigentlichen Anweisungen:

Mit `leds_set_status(farbe, ledNr);` wird der LED mit der Nummer `ledNr` die Farbe `farbe` zugewiesen.

Anschließend wird 150 ms gewartet.

Aufgaben/Anregungen:

1. Verändern Sie Ihr Programm so, dass die Leuchtdioden langsamer hintereinander aufleuchten.
2. Verändern Sie Ihr Programm so, dass die Leuchtdioden schneller hintereinander aufleuchten.
3. Verändern Sie Ihr Programm so, dass sich die Laufrichtung umkehrt, sprich die Diode LED0 soll in jedem Durchlauf als letzte Diode aufleuchten.
4. Verändern Sie Ihr Programm so, dass jede Diode einzeln alle Farben durchläuft (Aus, Rot, Grün und Orange), bevor die nächste beginnt.

6 Inbetriebnahme des Displays

Als nächstes wollen wir uns die Ansteuerung des Grafikdisplays vornehmen: Dazu legen Sie wieder ein neues Projekt an, diesmal mit dem Namen **Display** (wie in Kapitel 3 beschrieben). Denken Sie auch diesmal wieder an die zusätzlichen Einstellungen bei den *Project-Options*.

Unser Programm soll den Text „hello world!“ auf dem Display ausgeben.

Tippen Sie folgenden Quellcode in das Editorfenster des AVR Studio ein:

```
#include <nibo/niboconfig.h>
#include <nibo/display.h>
#include <nibo/gfx.h>

int main()
{
    display_init();
    gfx_init();
    gfx_move(30, 20);
    gfx_set_proportional(0);
    gfx_print_text("hello world!");
    gfx_move(30, 30);
    gfx_set_proportional(1);
    gfx_print_text("hello world!");
    return 0;
}
```

Erläuterungen zum Quellcode:

Zunächst werden zusätzlich zur schon bekannten Header-Datei *niboconfig.h* zwei neue Header-Dateien *display.h* und *gfx.h* eingebunden. Dies geschieht wieder mit *#include*.

In der main-Funktion wird mit dem Funktionsaufruf `display_init()` *zunächst die Schnittstelle zum Display initialisiert* und anschließend mit der Anweisung `gfx_init()` das Grafikdisplay konfiguriert.

Danach wird mittels `gfx_move(30, 20)` der Stift auf eine Position 30 Pixel vom linken Rand und 20 Pixel vom Oberen Rand verschoben.

Die Anweisung `gfx_set_proportional(0)` sorgt dafür, dass die Texte nicht in Proportionalchrift ausgedruckt werden, das heisst alle ausgegebenen Zeichen haben die gleiche Breite.

Nun kann auch schon mit `gfx_print_text("hello world!")` der Text „hello world!“ ausgegeben werden.

Jetzt wandert der Cursor mit `gfx_move(30, 30)` eine Zeile tiefer. Der Text soll nun in Proportional-Schrift ausgegeben werden, dies geschieht durch den Aufruf der Funktion `gfx_set_proportional(1)`.

Die folgende Anweisung gibt wiederum den Text „hello world!“ auf dem Display aus, nur diesmal in der anderen Schriftart!

Aufgaben/Anregungen:

1. Geben Sie die Texte „links oben“, „rechts oben“, „links unten“ und „rechts unten“ an den jeweiligen Stellen auf dem Display aus.
2. Geben Sie mittig auf dem Display einen Countdown aus, der Countdown soll im Sekundentakt von 20 auf 0 Zählen. Benutzen Sie dazu die `delay()` Funktion, achten Sie auf eine saubere Ausgabe!

7 Darstellung der Sensorwerte

In diesem Beispiel wollen wir uns mit den Sensoren des Roboters beschäftigen. Das sind zum einen die beiden Boden- und die beiden Liniensensoren, zum anderen handelt es sich um die fünf Distanzsensoren, die von einem separaten Mikrocontroller (IRCO) angesteuert werden. Alle Sensoren arbeiten nach dem IR-Reflexionsverfahren. Dabei wird gemessen welcher Anteil vom ausgesendeten Licht zurück reflektiert wird. Alle Messwerte sollen auf dem Grafikdisplay ausgegeben werden.

Zusätzlich soll noch die aktuelle Versorgungsspannung des Roboters ausgegeben werden.

Legen Sie wie in den vorangegangenen Beispielen ein neues Projekt an. Zu den Einstellungen aus den vorherigen Beispielen müssen Sie dieses Mal in den Projektoptionen neben der Bibliothek **libnibo.a** zwei zusätzliche Bibliotheken zum Linken auswählen: **libm.a** und **libprintf_flt.a**. Weiterhin muss bei den Projektoptionen unter *Custom Options* -> *[Linker Options]* die Option „-Wl, -u, vfprintf“ hinzugefügt werden.

Tippen Sie anschließend folgenden Quellcode in das Editorfenster ein:

```
#include <nibo/niboconfig.h>
#include <nibo/display.h>
#include <nibo/gfx.h>
#include <nibo/irco.h>
#include <nibo/delay.h>
#include <nibo/iodefs.h>
#include <nibo/adc.h>
#include <nibo/bot.h>
#include <avr/interrupt.h>
#include <nibo/i2cmaster.h>
#include <nibo/floor.h>
#include <stdio.h>

uint16_t value_off[4];
uint16_t value_on[4];

void measure_ground()
{
    floor_disable_ir();
    delay(1);
    floor_update();
    value_off[0] = floor_r;
    value_off[1] = line_r;
    value_off[2] = line_l;
    value_off[3] = floor_r;

    floor_enable_ir();
}
```

```
    delay(1);
    floor_update();
    value_on[0] = floor_r;
    value_on[1] = line_r;
    value_on[2] = line_l;
    value_on[3] = floor_l;
    floor_disable_ir();
}

uint8_t diff_ground(int16_t val)
{
    if(val>255)
        val=255;
    if (val<0)
        val=0;
    return val;
}

int main()
{
    sei();
    bot_init();
    i2c_init();
    floor_init();
    display_init();
    gfx_init();

    gfx_move(62, 0);
    gfx_set_proportional(1);
    gfx_print_text("nibo test");
    gfx_set_proportional(0);
    gfx_move(5, 0);
    gfx_print_char('R');
    gfx_move(118, 0);
    gfx_print_char('L');

    delay(50);
    irco_startMeasure();

    while (1==1)
    {
        delay(10);
        char text[20]="--  --  --  --  --";

        // IR Controller
        if (irco_update())
        {
            sprintf(text, "%02x %02x %02x %02x %02x",
                    (int)irco_distance[0],
                    (int)irco_distance[1],
                    (int)irco_distance[2],
                    (int)irco_distance[3],
```

```
        (int) irco_distance[4]);
    }
    gfx_move(10, 55);
    gfx_print_text(text);

    // Bodensensoren
    measure_ground();

    sprintf(text, "%02x %02x %02x %02x",
            (int) (value_on[0]/4),
            (int) (value_on[1]/4),
            (int) (value_on[2]/4),
            (int) (value_on[3]/4));
    gfx_move(22, 30);
    gfx_print_text(text);

    sprintf(text, "%02x %02x %02x %02x",
            (int) (diff_ground(value_on[0]-value_off[0])),
            (int) (diff_ground(value_on[1]-value_off[1])),
            (int) (diff_ground(value_on[2]-value_off[2])),
            (int) (diff_ground(value_on[3]-value_off[3])));
    gfx_move(22, 40);
    gfx_print_text(text);

    // Spannung
    bot_update();
    float volt = 0.0166 * bot_supply - 1.19;
    sprintf(text, "%3.1fV", (double)volt);
    gfx_move(25, 0);
    gfx_print_text(text);
}

return 0;
}
```

Erläuterungen zum Quellcode:

Diesmal müssen wir am Anfang einen ganze Menge Header-Dateien einbinden, da in diesem Beispiel auf die meisten Teilsysteme des Roboters zugegriffen wird.

Anschließend reservieren wir uns mit den Anweisungen `uint16_t value_off[4]` und `uint16_t value_on[4]` Speicherplatz für zwei Felder mit jeweils vier Integer-Variablen. Darin werden wir später die Messwerte von den Boden- und Liniensensoren speichern. Ein Feld wird die Messwerte bei eingeschalteter IR-Beleuchtung aufnehmen und das andere die Werte bei ausgeschalteter Beleuchtung.

Im nächsten Abschnitt des Programms definieren wir uns die Funktion `measure_ground()`. Innerhalb dieser Funktion werden zunächst die IR-LEDs der Boden- und Liniensensoren abgeschaltet, anschließend wird kurz abgewartet und danach werden durch Aufruf der Funktion `floor_update()` die aktuellen Werte der Sensoren gemessen. Die aktualisierten Werte stehen jetzt in den globalen Variablen `floor_l`, `floor_r`, `line_l` und `line_r` und werden von dort an die richtigen Stellen in das Feld `value_off` kopiert. Anschließend wird Messung mit eingeschalteten IR-LEDs wiederholt. Zum Schluss der Funktion wird die IR-Beleuchtung wieder abgeschaltet.

Mit der Funktion `uint8_t diff_ground(int16_t val)` kann eine vorzeichenbehaftete Ganzzahl in eine Zahl zwischen 0 und 255 umgewandelt werden: Zahlen die größer als 255 sind werden zur Zahl 255 umgewandelt, negative Zahlen werden Null.

main-Methode:

Zunächst werden die Interrupts durch Aufruf der Funktion `sei()` aus der AVR Bibliothek `<avr/interrupt.h>` aktiviert. Dies ist notwendig, damit die Kommunikation über den I2C-Bus funktioniert. Zusätzlich müssen für die Kommunikation noch die beiden Funktionen `bot_init()` und `i2c_init()` aufgerufen werden. Die Funktion `floor_init()` initialisiert die Boden- und Liniensensoren, die Funktionen `display_init()` und `gfx_init()` initialisieren das Grafikdisplay.

Als nächstes werden noch ein paar Texte auf dem Display ausgegeben, kurz gewartet und durch Aufruf der Funktion `irco_startMeasure()` die Distanzmessung gestartet.

while-Schleife:

In der ersten Zeile warten wir erst einmal 10 ms.

In der zweiten Zeile definieren wir uns Speicherplatz für einen 20 Zeichen langen Text, den wir erst einmal mit dem Text „-- -- -- -- --“ initialisieren.

Als nächstes aktualisieren wir die Distanzwerte durch Aufruf der Funktion `irco_update()` und benutzen den Rückgabewert in der `if`-Anweisung, um den folgenden Block nur nach einem erfolgreichen Update auszuführen.

Bei erfolgreichem Update werden die Werte durch Aufruf der Funktion `sprintf()` aus der C-Bibliothek `<stdio.h>` in den Textpuffer geschrieben. Die Zeichenkette dient dabei als Schablone für den Text: „%02x“ bedeutet das 2 hexadezimale Ziffern, im Bedarfsfall mit führender Null, ausgegeben werden.

In jedem Fall gehen wir jetzt zur Position `x=10` und `y=55` und geben den Text aus.

Jetzt rufen wir die anfangs definierte Funktion `measure_ground()` auf, um die Werte der Boden- und Liniensensoren zu aktualisieren. Die absoluten Werte im eingeschalteten Zustand liegen im Intervall 0-1023. Wir teilen sie durch

vier damit sie im Intervall 0-255 liegen und mit 2 hexadezimalen Ziffern ausgegeben werden können, was wir an der Position $x=22$ und $y=30$ auch tun. Die relativen Werte bringen wir durch unsere Funktion `diff_ground()` auf das richtige Intervall und geben sie 10 Pixel unter der vorigen Zeile aus.

Als letztes wird durch Aufruf der Funktion `bot_update()` die Versorgungsspannung gemessen. Der gemessene Wert muss durch die Formel $0,0166 * bot_supply - 1,19$ in Volt umgerechnet werden. Die Umwandlung in einen Text geschieht wieder durch Aufruf der Funktion `sprintf()`. Der Text „%3.1f“ bedeutet, dass eine Fließkommazahl mit (mindestens) drei Zeichen ausgegeben werden soll. Dabei soll immer eine Nachkommastelle angezeigt werden. Der fertige Text wird an der Stelle $x=25$ und $y=0$ ausgegeben.

8 Bewegung – Ab die Post!

Im letzten Beispiel wollen wir etwas Bewegung ins Spiel bringen: Der Roboter soll ein Stück vorwärts fahren, kurz warten, dieselbe Strecke rückwärts fahren, wieder kurz warten und von vorne beginnen. Die Motoransteuerung wird von einem separaten Microcontroller (MOTCO) durchgeführt.

Dabei soll auch diesmal ständig die Batteriespannung auf dem Display ausgegeben werden.

Als kleiner Zusatz soll außerdem der zuletzt empfangene Code einer RC5-kompatiblen Fernbedienung ausgegeben werden.

Wie im vorigen Beispiel legen wir wieder ein neues Projekt an und wählen die drei Bibliotheken **libnibo.a**, **libm.a** und **libprintf_flt.a** aus und fügen bei den Projektoptionen die Option „-Wl,-u,vfprintf“ hinzu (siehe Kapitel 7).

Schreiben Sie anschließend folgenden Quellcode in das Editorfenster:

```
#include <nibo/niboconfig.h>
#include <nibo/display.h>
#include <nibo/gfx.h>
#include <nibo/irco.h>
#include <nibo/delay.h>
#include <nibo/iodefs.h>
#include <nibo/bot.h>
#include <nibo/motco.h>
#include <avr/interrupt.h>
#include <nibo/i2cmaster.h>
#include <stdio.h>

int main()
{
    sei();
    bot_init();
    i2c_init();
    display_init();
    gfx_init();

    gfx_move(62, 0);
    gfx_set_proportional(1);
    gfx_print_text("motion");
    gfx_set_proportional(0);

    gfx_move(5, 0);
    gfx_print_char('R');

    gfx_move(118, 0);
```

```
gfx_print_char('L');

delay(50);
irco_startMeasure();
motco_setSpeedParameters(5, 6, 7);

int counter=0;

while (1==1)
{
    delay(10);

    char text[20]="RC5: ----";
    if (irco_update())
    {
        sprintf(text, "RC5: %04x", (int)irco_rc5_cmd);
    }
    gfx_move(35, 45);
    gfx_print_text(text);

    bot_update();
    float volt = 0.0166 * bot_supply - 1.19;
    sprintf(text, "%3.1fV", (double)volt);
    gfx_move(25, 0);
    gfx_print_text(text);

    switch(++counter)
    {
        case 200:
            motco_setSpeed(20, 20);
            break;

        case 400:
            motco_stop();
            break;

        case 600:
            motco_setSpeed(-20, -20);
            break;

        case 800:
            motco_stop();
            counter=0;
            break;
    }
}
return 0;
}
```

Erläuterungen zum Quellcode:

Auch dieses Programm beginnt mit dem Einbinden der verschiedenen Header-Dateien.

Das Hauptprogramm aktiviert zuerst – wie im letzten Beispiel – die Interrupts und initialisiert die verschiedenen Subsysteme. Danach werden einige Texte ausgegeben. Nach einer kurzen Pause die Messung der Distanzwerte gestartet und die Parameter für die Motorregelung an den MOTCO übermittelt. Mit den Regelungsparametern 5, 6, 7 lassen sich gute Ergebnisse erzielen.

Für die `while`-Schleife gibt es diesmal eine Variable `counter`, die die Durchläufe der Endlosschleife zählt.

In der Endlosschleife aktualisieren wir zunächst die Werte vom IRCO, und geben den Empfangenen RC5-Code einer Fernbedienung auf dem Display aus.

Im nächsten Abschnitt geben wir, wie im vorangegangenen Beispiel, die aktuelle Versorgungsspannung aus.

Der anschließende `switch`-Block ist die zentrale Steuerung für die Bewegung: Die Variable `counter` wird hochgezählt; bei Erreichen der Werte 200, 400, 600 und 800 werden die entsprechenden Abschnitte ausgeführt.

Von 0-199 sollen sich die Räder nicht drehen.

Von 200-399 sollen sich beide Räder des Roboters mit einer Geschwindigkeit von 20 Ticks pro Sekunde vorwärts drehen.

Von 400-599 sollen sich die Räder nicht drehen.

Von 600-799 sollen sich beide Räder des Roboters mit einer Geschwindigkeit von 20 Ticks pro Sekunde rückwärts drehen.

Bei Erreichen des Wertes 800 wird der Zähler auf 0 zurückgesetzt.

Aufgaben/Anregungen:

1. Lassen Sie den Roboter eine kurze Zeit lang im Uhrzeigersinn drehen, und anschließend gegen den Uhrzeigersinn.
2. Benutzen Sie den empfangenen Code der RC5-Fernbedienung, um den Roboter fernzusteuern.

9 Links zu weiterführenden Internetseiten

In diesem Unterkapitel ist eine ausgewählte Linksammlung zu themenähnlichen Internetseiten aufgeführt.

Entwicklungsumgebungen:

- Atmel: <http://www.atmel.com> Webseite vom Hersteller der Mikrocontroller. Dort gibt es Datenblätter, Applikationsbeispiele und die Entwicklungsumgebung AVRStudio.
- WinAVR: <http://winavr.sourceforge.net/> AVR-GCC Compiler für Windows mit vielen Extras und „Add-on“ für das AVRStudio.
- AVRDude: <http://savannah.nongnu.org/projects/avrdude/> Freie Programmiersoftware (Downloader, für den Nibo geeignet!).

Weitere Informationen:

- Nibo Hauptseite: <http://nibo.nicai-systems.de> Die Homepage des Nibo Herstellers. Liefert technische Informationen, die Bauanleitung und weitere Links.
- Programmieradapter UCOM-IR: <http://ucom-ir.nicai-systems.de>
- Nibo Wiki: <http://www.nibo-roboter.de> Wiki des Nibo. Liefert alle Informationen rund um den Nibo.
- Nibo Shop: <http://shop.nicai-systems.de> Online-Shop für den Nibo und Erweiterungssets.
- Mikrocontroller: <http://www.mikrocontroller.net> Alles über Mikrocontroller und deren Programmierung.
- AVRFreaks: <http://www.avrfreaks.net> Informationen rund um den AVR.