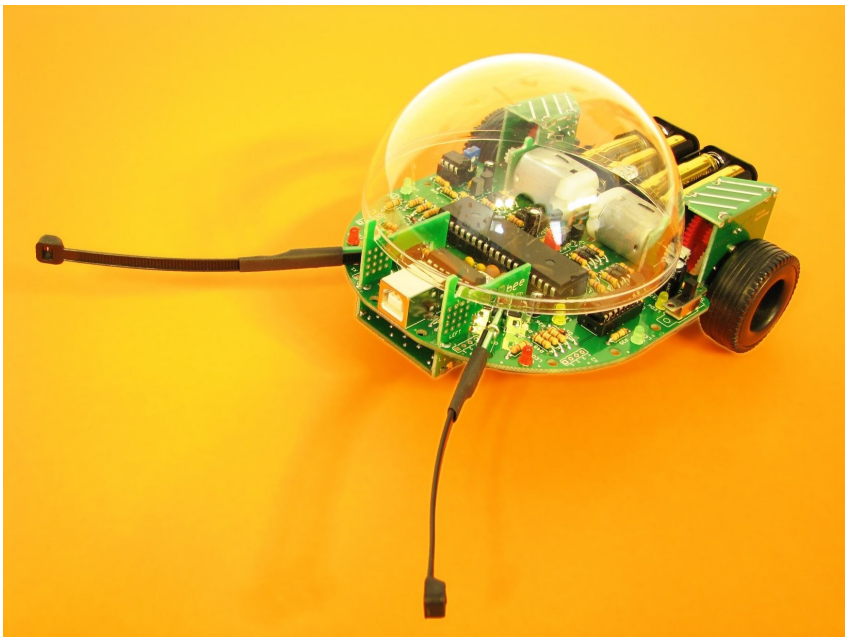
Roboterbausatz NIBObee

Tutorial zur Programmierung



nicai
systems

nibobee.nicai-systems.de

Inhaltsverzeichnis

1 Einleitung

Dieses Tutorial bietet eine Schritt für Schritt Anleitung für die erste Inbetriebnahme des NIBObee. 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, wie die Fühler, die Bodensensoren und die Motoren angesteuert werden und letztendlich auch, wie Sie den NIBObee 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.

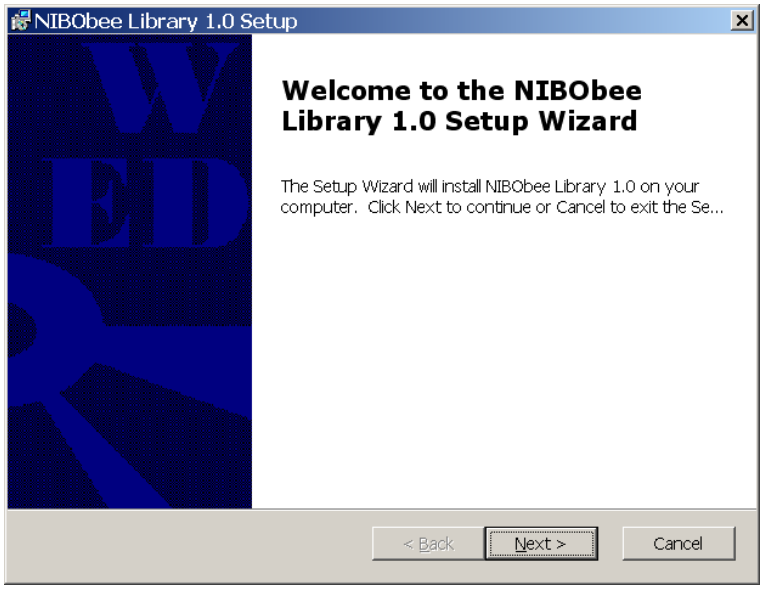
2 Installation der NIBObee Library

Als erster Schritt wird auf dem Rechner die aktuellste Version der NIBObee Bibliothek installiert. Um dafür den automatischen Installer zu verwenden, muss die Datei **NIBObeeLib-xxx.msi** auf dem Computer gespeichert werden. Sie finden diese Datei unter: http://sourceforge.net/projects/nibobeelib/

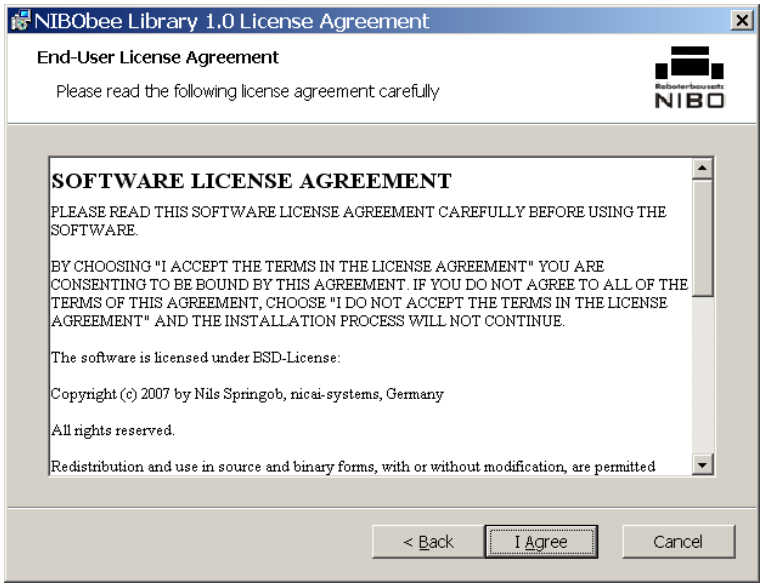
In der NIBObee Library ist Folgendes enthalten:

- C und C++ Routinen zur einfachen Ansteuerung des Roboters
- benötigte Treiber
- Kalibrierprogramm für die Sensoren
- Programmer zur Übertragung von .hex-Dateien auf den Roboter
- Beispielprogramme aus dem Tutorial

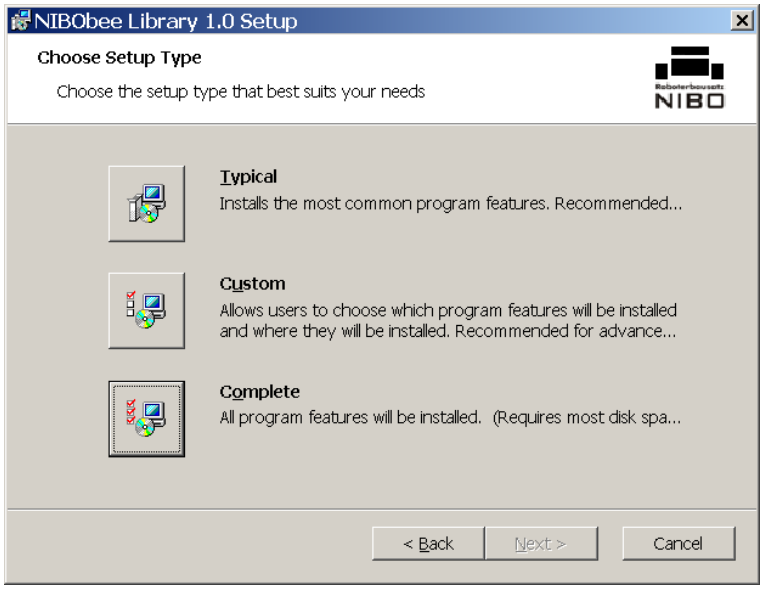
Speichern Sie die Datei z.B. auf dem Desktop ab und starten die die Installation der NIBObee Library per Doppelklick auf das Symbol.

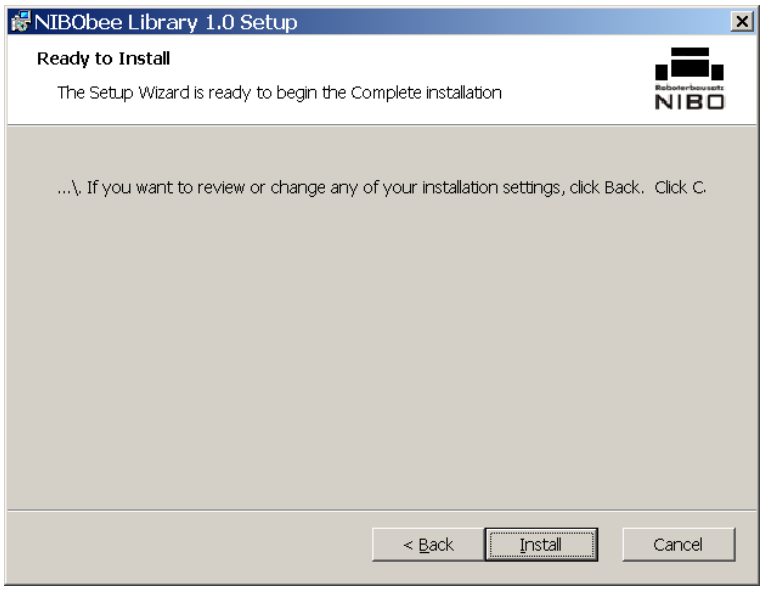


Das erste Informationsfenster bestätigen Sie mit „***Next***“ und klicken im folgenden Fenster auf „***I Agree***“:

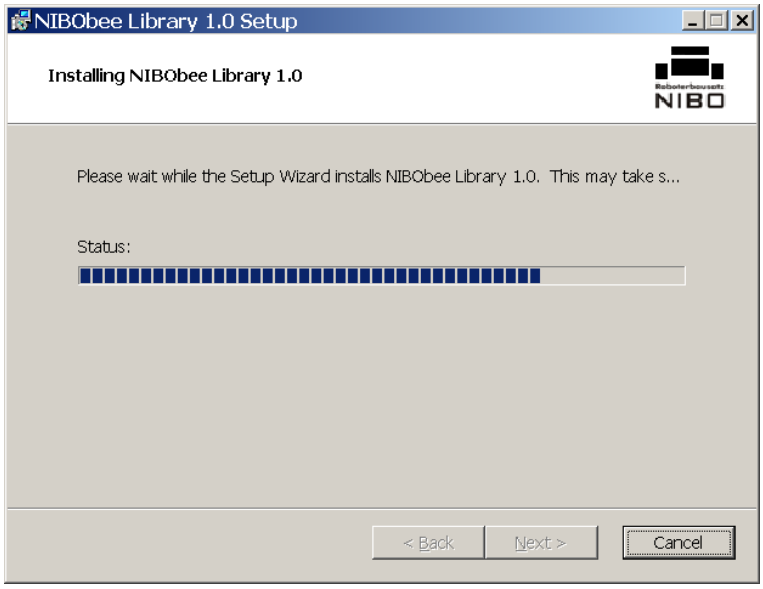


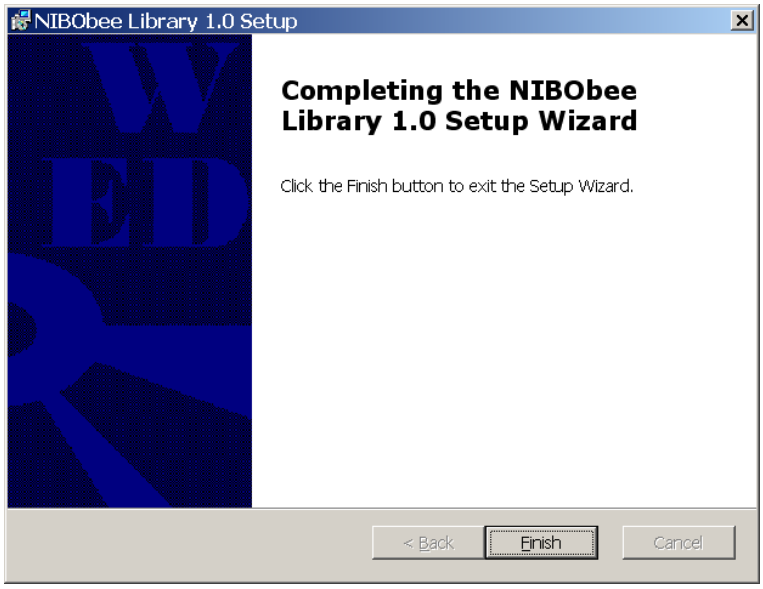
Im anschließenden Fenster wird die Art der Installation ausgewählt. Wählen Sie hier die Option „***Complete***“ aus und bestätigen Sie mit „***Next***“.





Per Klick auf den Knopf „***Install***“ wird die Installation gestartet. Anschließend wird der Dialog mit der Schaltfläche „***Next***“ verlassen.





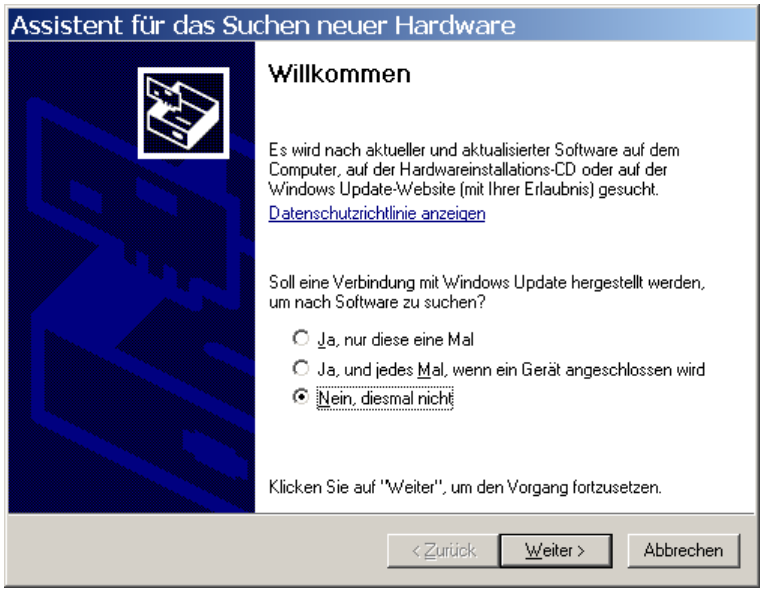
Die gesamte Installation wird mit „***Finish***“ abgeschlossen.

Auf Ihrem Rechner sollten nun unter C:\Programme\NIBObeeLib\ alle benötigten Dateien / Programme vorhanden sein.

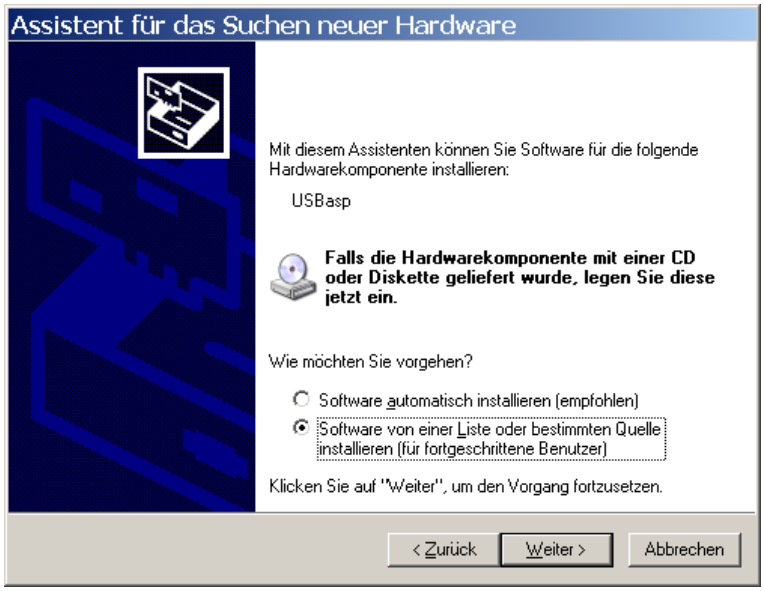
3 Erste Verbindung mit dem PC

Vor dem Anschluss des USB-Kabels muss der NIBObee **eingeschaltet** werden. Dann kann der eingeschaltete Roboter mittels des beiliegenden USB-Kabels mit einem Rechner verbunden werden.

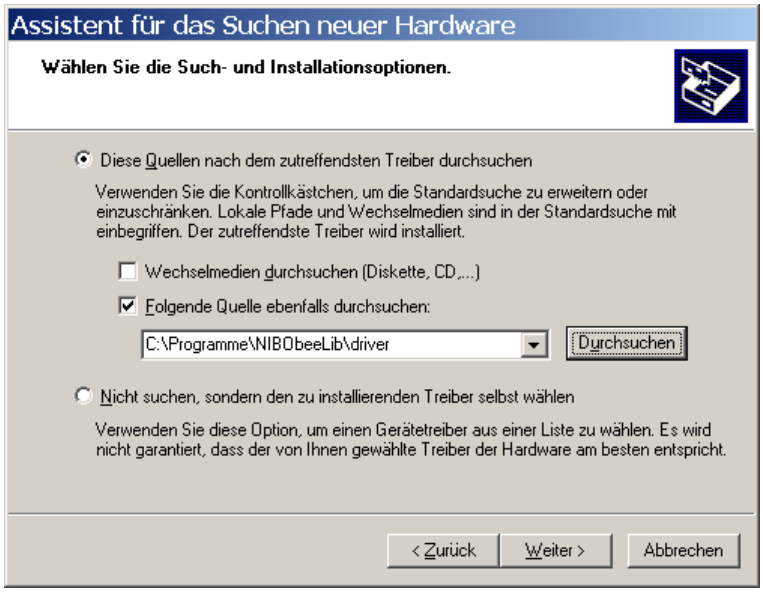
Unter Windows wird Sie der Hardware-Assistent bei der Installation des benötigten Treibers unterstützen:



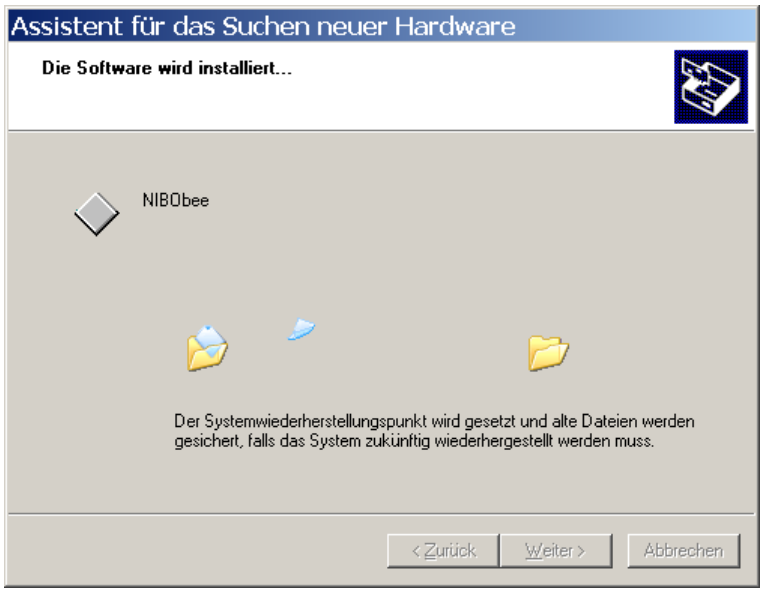
Wählen Sie hier „***Nein, diesmal nicht***“ aus und bestätigen mit „***Weiter***“.

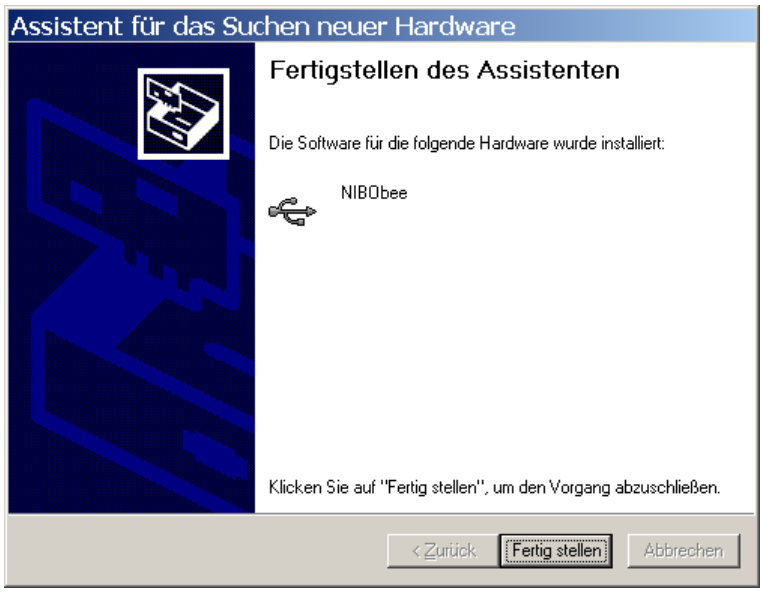


Im anschließenden Dialog wird die Option „***Software von einer Liste oder bestimmten Quelle installieren (für fortgeschrittene Benutzer)***“ ausgewählt und mit „***Weiter***“ bestätigt. Im folgenden Dialog wird der Punkt „***Diese Quellen nach dem zutreffendsten Treiber durchsuchen***“ ausgewählt:



Es wird die Option „***Folgende Quellen ebenfalls durchsuchen***“ angehakt. Per Klick auf „***Durchsuchen***“ erhält man einen Dateimanager, in dem man das Verzeichnis „***C:\Programme\NIBObeeLib\driver***“ auswählt.
Jetzt kann mit der Schaltfläche „***Weiter***“ die Installation des Treibers gestartet werden:





Nach einem Klick auf „***Fertig stellen***“ wird der Installationsvorgang abgeschlossen und der NIBObee ist nun programmierbereit.

Hinweis: Jedes Mal wenn Sie einen anderen USB-Anschluss an Ihrem Rechner verwenden, verlangt Windows diese Installation erneut.

4 Installation der Programmierumgebung

4.1 AVR Studio 4

Nun wird 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.

4.2 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.

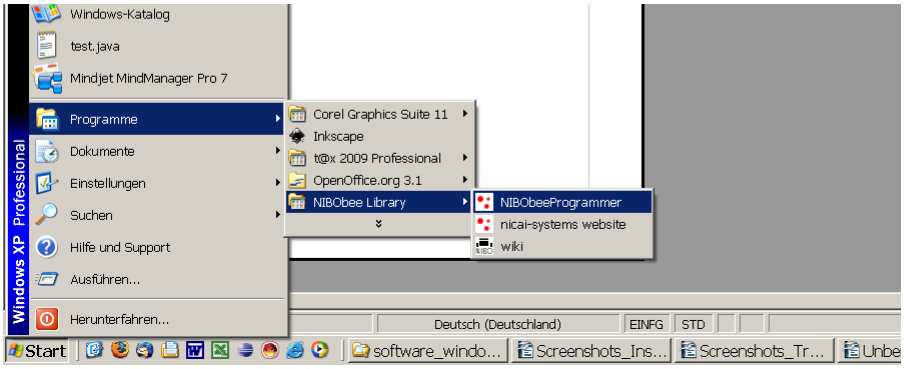
4.3 NIBObeeProgrammer

Um die vom Compiler erzeugten .hex-Dateien auf den NIBObee zu übertragen, verwenden Sie den auf der CD enthaltenen Programmer „*NIBObeeProgrammer*“.

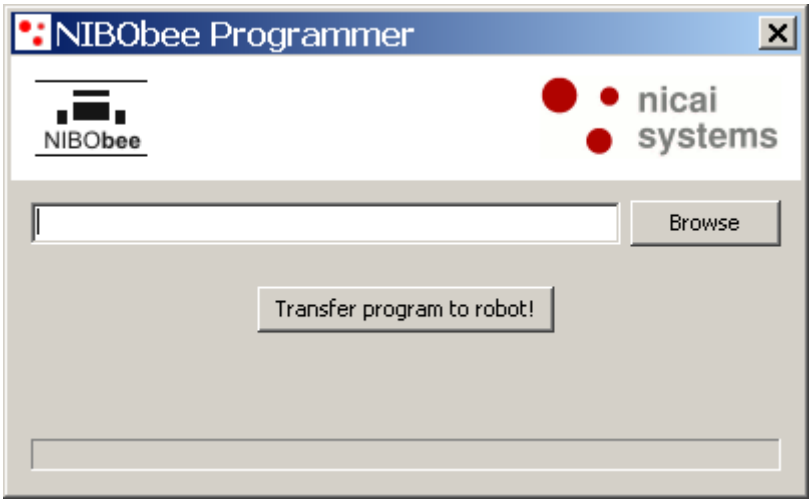
Der Programmer wurde bei der Installation der NIBObee Library automatisch mit installiert und kann nun über

Start -> ***Programme*** -> ***NIBObee Library*** -> ***NIBObeeProgrammer***

gestartet werden:



Über die Schaltfläche „***Browse***“ wird die gewünschte .hex-Datei ausgewählt, und dann mit Klick auf „***Transfer program to robot!***“ auf den NIBObee übertragen.



Erscheint über dem Fortschrittsbalken der Text „***Programming finished***“, dann ist die Übertragung beendet.

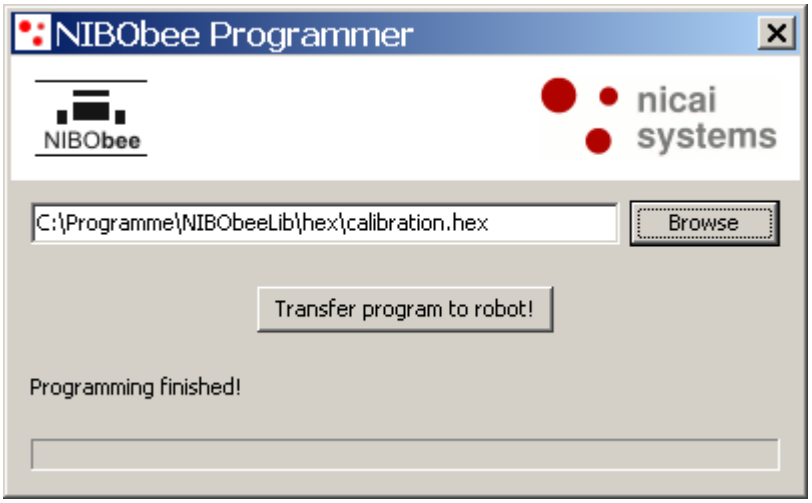
5 Kalibrierung der Sensoren

Zunächst wollen wir die Boden- und die Liniensensoren des NIBObee kalibrieren, um die Eigenschaften der optoelektronischen Bauteile aufeinander abzustimmen. So können bei anschließender Programmierung bestmögliche Ergebnisse erzielt werden.

Für diese Kalibrierung wird nun das mitgelieferte Programm „***calibration.hex***“ auf den Roboter übertragen. Quelle: „C:\Programme\NIBObeeLib\hex\“

Um .hex-Dateien auf den NIBObee zu übertragen, verwenden Sie den durch die Installation der Library bereits mit installierten Programmer „NIBObeeProgrammer“:

Start -> ***Programme*** -> ***NIBObee Library*** -> ***NIBObeeProgrammer***

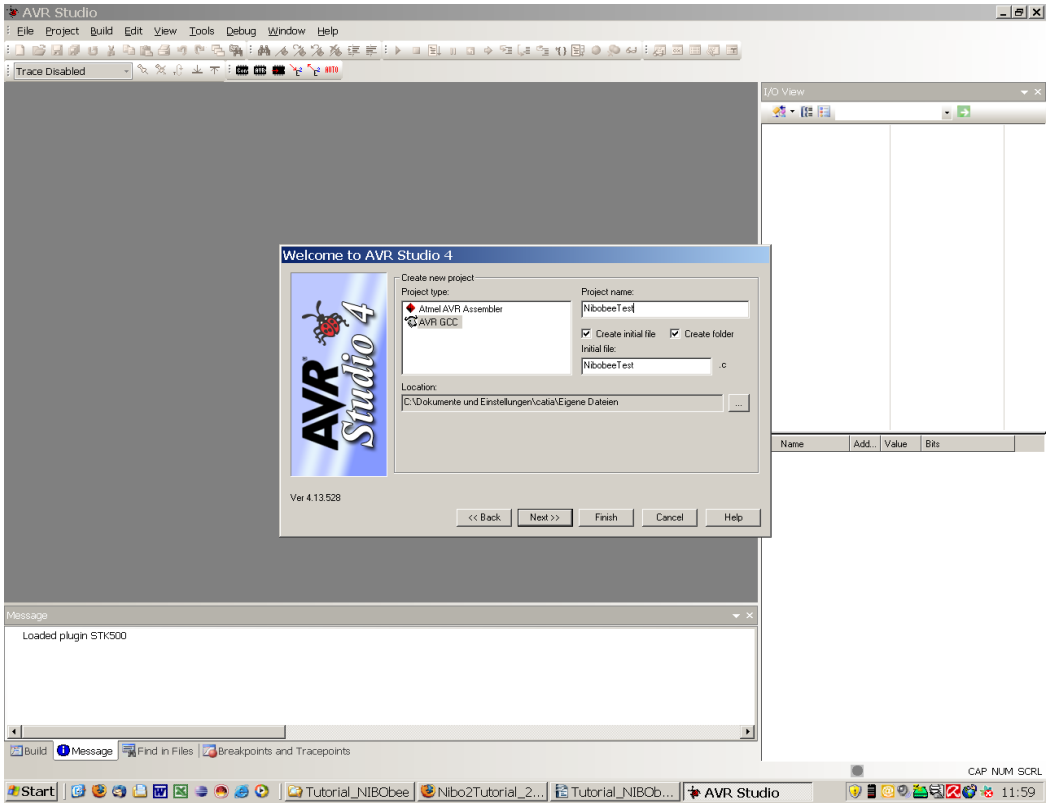


Über die Schaltfläche „***Browse***“ wird die gewünschte .hex-Datei ausgewählt, und dann mit Klick auf „***Transfer program to robot!***“ auf den NIBObee übertragen.

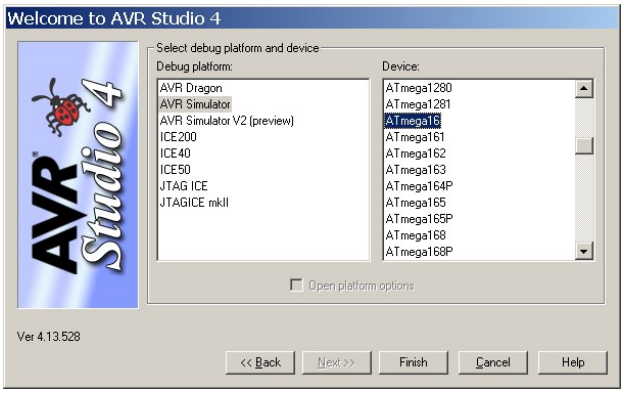
Nach der Übertragung der Datei „***calibration.hex***“ sind die physikalischen Eigenschaften der optischen Bauelemente, also der einzelnen optischen Sensoren, perfekt aufeinander abgestimmt.

6 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* **NibobeeTest** ein.

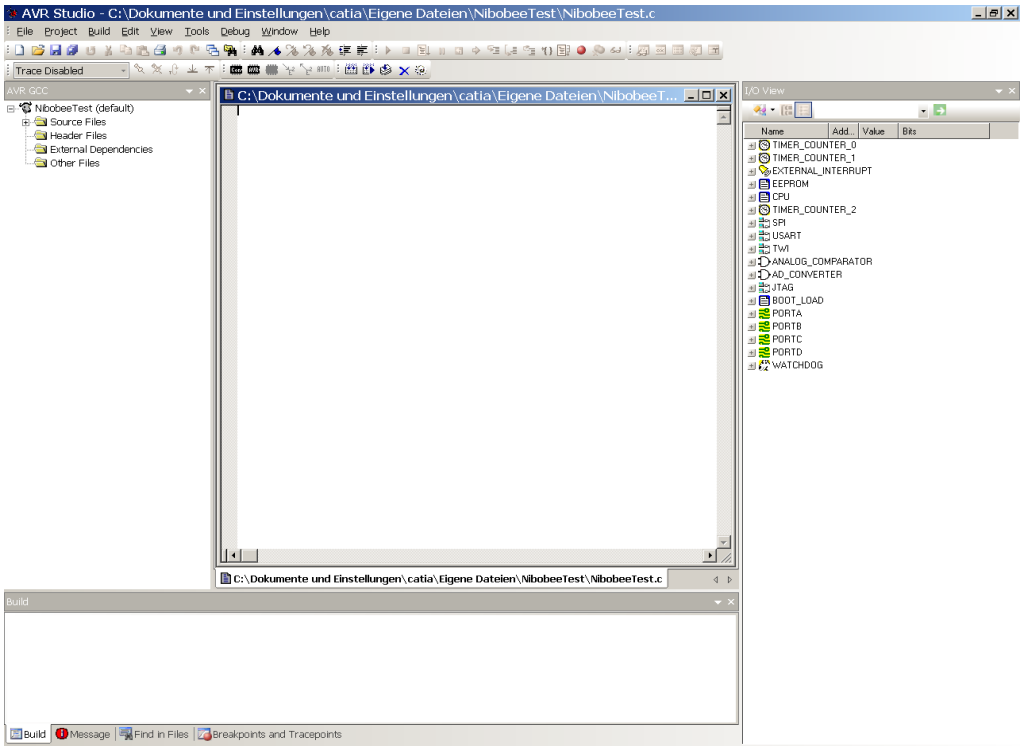


Klicken Sie auf *Next*. Im folgenden Bildschirm wählen Sie als *Debug platform* **AVR Simulator** und als *Device* **ATmega16** aus und klicken auf *Finish*.



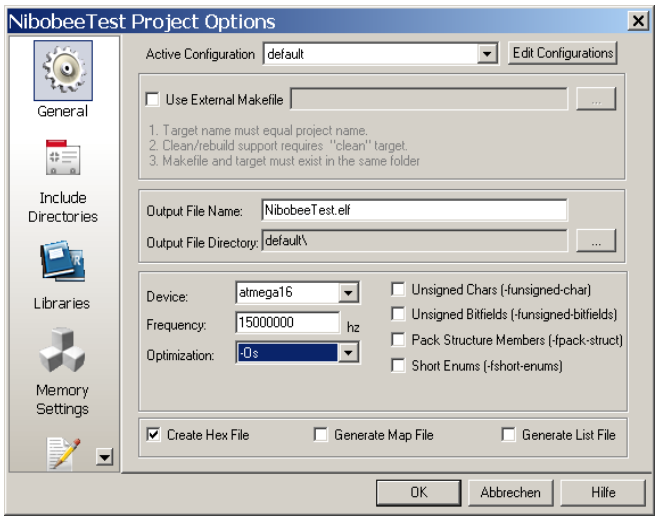
Das neue Projekt „*NibobeeTest*“ ist nun angelegt.

Das AVR Studio sollte in etwa folgender Abbildung gleichen:



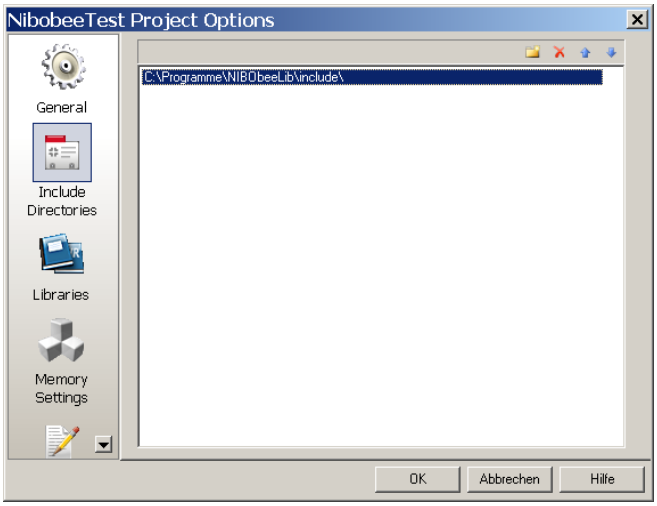
Nun müssen noch einige **Voreinstellungen** eingegeben werden.

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



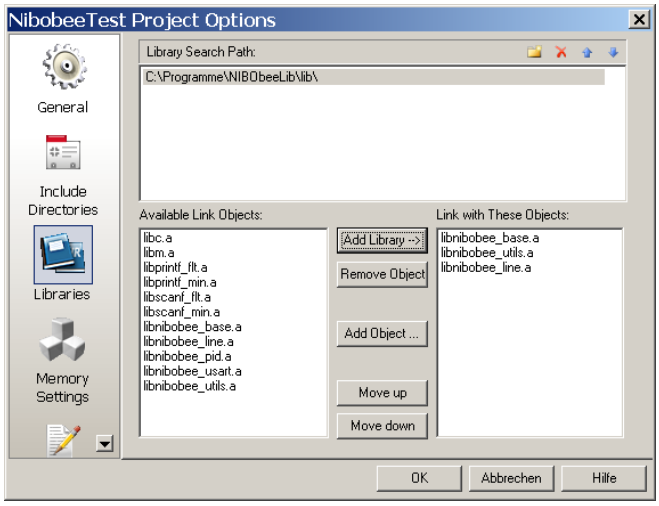
Tragen Sie im Bereich *General* im Feld *Frequency* den Wert **15000000** ein und wählen Sie im Feld ***O**ptimization* den Wert **-Os** aus.

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\NIBObeeLib\include** ein.

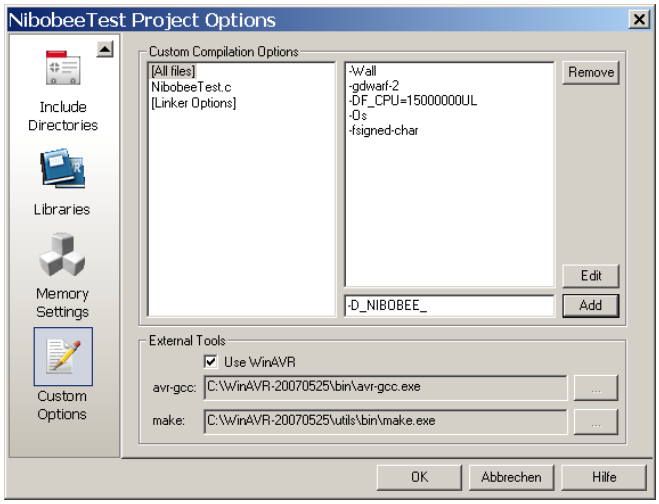


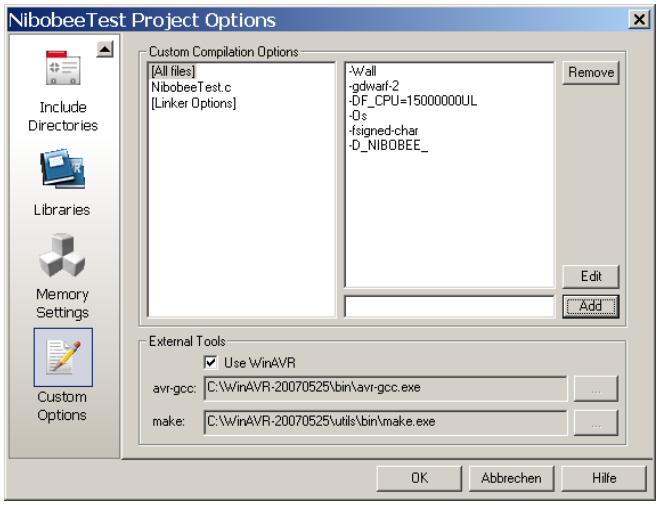
Nach Auswahl des Bereichs *Libraries* tragen Sie bei *Library Search Path* den Pfad **C:\Programme\NIBObeeLib\lib** ein.

Nun wählen Sie aus den *Available Link Objects* **libnibobee_base.a** aus und klicken auf den Button *Add Library->*. Die Bibliothek **libnibobee_base.a** sollte nun im rechten Fenster erscheinen. Wählen nun nach demselben Prinzip zusätzlich aus der Liste **libnibobee_utils.a** und **libnibobee_line.a** aus:



Im Bereich *Custom Options* (unter *Memory Settings*) wird jetzt noch die Option **-D_NIBOBEE_** durch Eintragung in das Textfeld und anschließendes Klicken auf *Add* hinzugefügt.





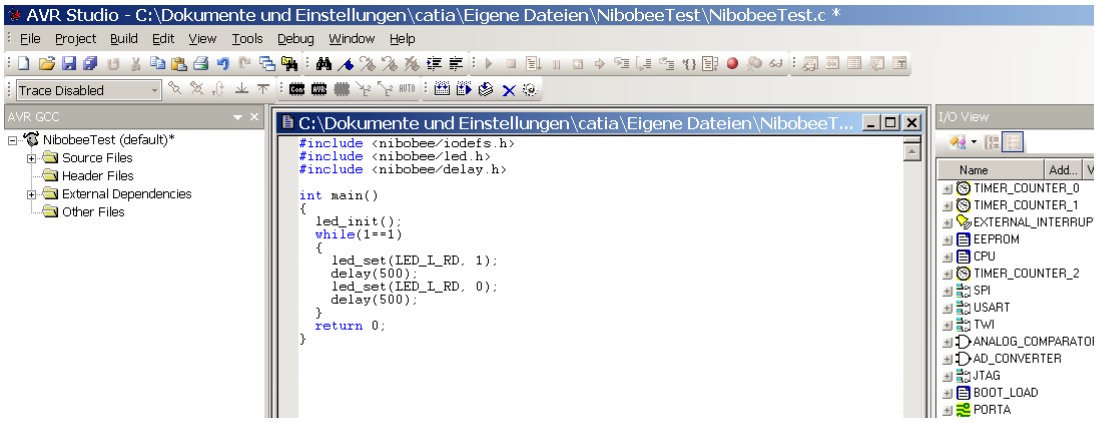
Bestätigen Sie die Einstellungen mit OK.

7 Ein erstes Testprogramm

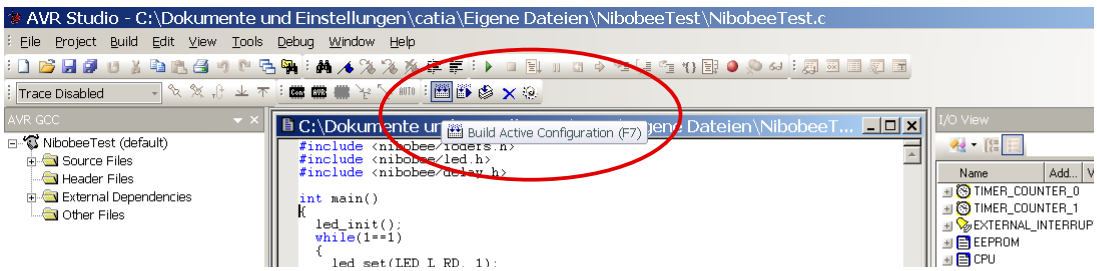
Tippen Sie als erstes Testprogramm folgendes ein:

```
#include <nibobee/iodefs.h>
#include <nibobee/led.h>
#include <nibobee/delay.h>

int main()
{
  led_init();
  while(1==1)
  {
    led_set(LED_L_RD, 1);
    delay(500);
    led_set(LED_L_RD, 0);
    delay(500);
  }
  return 0;
}
```

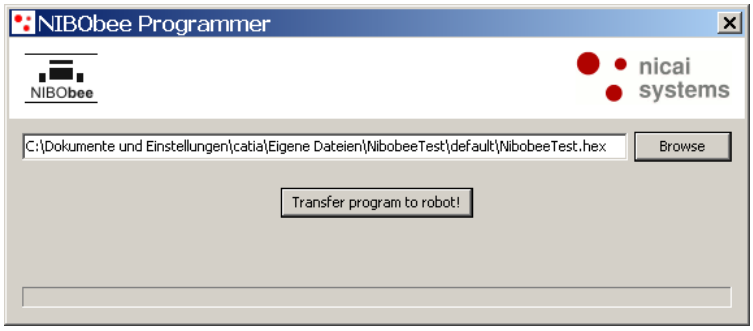


Nun wird das Programm compiliert, indem Sie den *Build Active Configuration* Knopf *(F7)* drücken:



Falls es beim Compilieren keinen Fehler gegeben hat, dann hat der Compiler die Datei *Nibobee.hex* erzeugt. Diese Datei muss nun mit dem NIBObeeProgrammer auf den NIBObee übertragen werden:

!! Wichtig !! Der NIBObee muss unbedingt **eingeschaltet** sein, bevor er mit der **USB-Schnittstelle** verbunden wird!



Die Programmübertragung wird mittels Verlaufsbalken am unteren Fensterrand visualisiert. Eine ausführliche Erklärung zur Bedienung des NIBObeeProgrammers ist in Kapitel 4 – Kalibrierung der Sensoren – zu finden.

Die Datei *Nibobee.hex* befindet sich in dem von Ihnen zu Beginn angelegten Projekt-Verzeichnis *NibobeeTest* im Unterverzeichnis *default*. Den zugehörigen Pfad kann man bei Bedarf in der Fensterleiste vom AVR-Studio nachschauen.

Hinweis: Die Programmierung des NIBObee erfolgt also in zwei Stufen mit zwei verschiedenen Programmen:

1. Die eigenen Programme / Beispielprogramme werden im Editor des **AVR-Studio** eingetippt und dort compiliert. Der Compiler erzeut eine .hex-Datei und speichert diese in den jeweiligen Projektordner.
2. Mit dem **NIBObeeProgrammer** wird die .hex-Datei anschließend per USB-Schnittstelle auf den Roboter übertragen.

Wenn die Übertragung des Testprogramms geklappt hat, sollte am NIBObee die linke vordere LED (LED1) jetzt rot blinken.

Erläuterungen zum Quellcode:

Die ersten drei Programmzeilen beginnen jeweils mit einer # und sind somit Präprozessoranweisungen. Der Präpozessor 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 Hauptprgramm. Es endet mit `}`. Zwischen diesen geschweiften Klammern stehen alle Anweisungen, die abgearbeitet werden sollen.

Mit `led_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 `led_set(LED_L_RD, 1);` schaltet die linke (**L**) rote (**RD**) LED ein (**1**).

Die Anweisung `delay(500);` weist den Controller an, 500 ms (0,5 Sekunden) zu warten.

Die dritte Anweisung `led_set(LED_L_RD, 0);` schaltet die linke rote LED wieder aus (0).

Dann nochmals 500 ms warten.

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!

8 Lauflicht

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 (LED0, LED1, LED2, LED3) ein- und wieder ausgeschaltet werden.

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

```
#include <nibobee/iodefs.h>
#include <nibobee/led.h>
#include <nibobee/delay.h>

int main()
{
  led_init();
  while(1==1)
  {
    int ledNr;
    for (ledNr=0; ledNr<4; ledNr++)
    {
      led_set(ledNr, 1);
      delay(350);
      led_set(ledNr, 0);
      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 ledNr;` wird eine Variable namens „`ledNr`“ vom Typ *int* (Ganzzahl) deklariert.

Als nächstes wird eine for-Schleife verwendet. Die Schleife läuft über den

Wert der Variablen `ledNr`. Die Variable wird mit dem Wert 0 initialisiert und nimmt bis zur Abbruchbedingung nacheinander die Werte 0, 1, 2 und 3 an.

Innerhalb der for-Schleife stehen die eigentlichen Anweisungen:

Mit `led_set(ledNr, 1);` wird die LED mit der Nummer `ledNr` eingeschaltet.

Anschließend wird 350 ms (0,35 Sekunden) gewartet.

Mit der Anweisung `led_set(ledNr, 0);` wird die LED mit der Nummer `ledNr` wieder ausgeschaltet.

Dann wird 150 ms gewartet.

Nun wird die Variable `ledNr` um eins erhöht und der Schleifenkörper wird erneut durchlaufen bis die Bedingung `ledNr<4` nicht mehr erfüllt ist.

Da die for-Schleife innerhalb der endlos-while-Schleife liegt, wird die for-Schleife anschließend wieder erneut aufgerufen, so dass das Lauflicht automatisch immer wieder eine neue Runde beginnt.

Aufgaben/Anregungen:

- Schreiben Sie ein Programm, bei dem die leuchtende LED hin und her wandert.
- Schreiben Sie ein Programm, bei dem die LEDs nacheinander eingeschaltet und danach nacheinander wieder ausgeschaltet werden.

9 LEDs in Aktion

Hier wollen wir nun etwas kompliziertere Leuchtmuster gestalten.

Als erstes legen Sie ein neues Projekt mit dem Namen **Leuchtdioden2**, wie in Kapitel 3 beschrieben, an. Denken Sie dabei an die zusätzlichen Einstellungen bei den *Project-Options*.

Die Leuchtdioden sollen wieder nacheinander reihum (LED0, LED1, LED2, LED3) ein- und wieder ausgeschaltet werden, allerdings soll sich diesmal nach jedem Durchlauf die Geschwindigkeit verändern!

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

```
#include <nibobee/iodefs.h>
#include <nibobee/led.h>
#include <nibobee/delay.h>

int main()
{
  led_init();
  while(1==1)
  {
    int time;
    for (time=50; time<800; time*=2)
    {
      int ledNr;
      for (ledNr=0; ledNr<4; ledNr++)
      {
        led_set(ledNr, 1);
        delay(time);
        led_set(ledNr, 0);
        delay(time);
      }
    }
  }
  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 time;` wird eine Integer-Variable namens „time“ deklariert.

Als nächstes werden zwei ineinander geschachtelte for-Schleifen verwendet.

Die äußere Schleife läuft über den Wert der Variablen `time`. Die Variable wird mit dem Wert 50 initialisiert und nimmt bis zur Abbruchbedingung nacheinander die Werte 50, 100, 200 und 400 an.

Mit der Anweisung `int ledNr;` wird eine Integer-Variable namens „ledNr“ deklariert.

Diese wird in der inneren for-Schleife verwendet, die über die Variable `ledNr` läuft. In dieser inneren Schleife stehen nun auch die eigentlichen Anweisungen:

Mit `led_set(ledNr, 1);` wird die LED mit der Nummer *ledNr* eingeschaltet.

Anschließend wird je nach aktuellem Wert der Variable `time` gewartet.

Die Anweisung `led_set(ledNr, 0);` schaltet die LED mit der Nummer *ledNr* wieder aus. Dann wird noch einmal je nach Wert der Variable `time` gewartet.

Aufgaben/Anregungen:

- Kehren Sie den zeitlichen Ablauf um, die Geschwindigkeit soll sich von Runde zu Runde erhöhen.
- Tauschen Sie die beiden for-Schleifen aus...

10 Inbetriebnahme der Tastsensoren / Fühler

Als nächstes wollen wir die Tastsensoren, also die Fühler des NIBObee in Betrieb nehmen: Dazu legen Sie wieder ein neues Projekt an, diesmal mit dem Namen **Fühler** (wie in Kapitel 3 beschrieben). Denken Sie auch diesmal wieder an die zusätzlichen Einstellungen bei den *Project-Options*.

Jeder Fühler steuert zwei Taster an. Unser Programm soll folgendes bewirken: Wird der linke Fühler nach vorne gedrückt, so leuchtet die linke rote LED, wird er nach hinten gedrückt, so leuchtet die linke gelbe LED (analog für die rechte Seite).

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

```
#include <nibobee/iodefs.h>
#include <nibobee/led.h>
#include <nibobee/sens.h>

int main()
{
  led_init();
  sens_init();

  while(1==1) {
    int8_t status_L = sens_getLeft();

    switch (status_L) {
      case -1: led_set(LED_L_RD, 0); led_set(LED_L_YE, 1);
      break;
      case +1: led_set(LED_L_RD, 1); led_set(LED_L_YE, 0);
      break;
      default: led_set(LED_L_RD, 0); led_set(LED_L_YE, 0);
      break;
    }

    int8_t status_R = sens_getRight();

    switch (status_R) {
      case -1: led_set(LED_R_RD, 0); led_set(LED_R_YE, 1);
      break;
      case +1: led_set(LED_R_RD, 1); led_set(LED_R_YE, 0);
      break;
      default: led_set(LED_R_RD, 0); led_set(LED_R_YE, 0);
      break;
    }
  }
  return 0;
}
```

Erläuterungen zum Quellcode:

Zunächst werden zusätzlich zur schon bekannten Header-Datei iodefs.*h* und *led.h* eine neue Header-Datei *sens.h* eingebunden. Dies geschieht wieder mit *#include*.

In der main-Funktion wird mit dem Funktionsaufruf `led_init()` zunächst die IO-Ports der LEDs als Ausgänge konfiguriert und anschließend mit der Anweisung `sens_init()` die Fühlersensoren initialisiert.

In der darauf folgenden endlos-while-Schleife werden zwei switch-case-Anweisungen verwendet:

Innerhalb der runden Klammern einer switch-case-Anweisung steht die sogenannte Entscheidungsvariable. In unserem Fall ist das einmal die Variable `status_L`, der zuvor der Rückgabewert der Funktion `sens_getLeft()` zugewiesen wird, so dass die Variable `status_L` den aktuellen Status des linken Fühlers enthält. Die Variable `status_L` ist vom Typ `int8_t`, also ein Integer Wert mit 8 byte Speicherplatz.

Der Compiler wertet nun am Anfang der switch-case-Anweisung die Variable `status_L` aus und führt anschließend je nach Wert der Variable die Zeile mit dem passenden case aus. Es können drei Fälle auftreten:

Die Variable `status_L` hat den Wert 0: keine Betätigung des Fühlers

Die Variable `status_L` hat den Wert 1: Betätigung des Fühlers nach vorne

Die Variable `status_L` hat den Wert -1: Betätigung des Fühlers nach hinten

Hat also beispielsweise die Variable `status_L` den Wert 1, wurde also der linke Fühler nach vorne gedrückt, dann führt der Compiler die Anweisungen des case +1 aus: `led_set(LED_L_RD, 1); led_set(LED_L_YE, 0);`

Die linke rote LED wird also eingeschaltet und die linke gelbe LED wird ausgeschaltet. Mit der Anweisung **break**; wird die switch-case-Anweisung wieder verlassen. Diese Anweisung sollte am Ende jedes case stehen! Der **default**-case ist für den Fall, dass alle anderen cases nicht gepasst haben.

Ganz analog funktioniert die switch-case-Anweisung für den rechten Fühler.

Aufgaben/Anregungen:

- Ändern Sie das Programm so ab, dass die LEDs durch Betätigung der Fühler eingeschaltet bleiben und bei Betätigung in die Gegenrichtung ausgeschaltet werden.

11 Ping Pong

In diesem Beispiel wollen wir uns intensiver mit den Tastsensoren des Roboters beschäftigen, und dabei ein paar Programmiertechniken lernen.

Legen Sie wie in den vorangegangenen Beispielen ein neues Projekt an.

Tippen Sie anschließend folgenden Quellcode in das Editorfenster ein:

```
#include <nibobee/iodefs.h>
#include <nibobee/led.h>
#include <nibobee/sens.h>
#include <nibobee/delay.h>


enum {
  STATE_IDLE  = 0,
  STATE_PULL0 = 1,
  STATE_PULL1 = 2,
  STATE_KICK  = 3,
  STATE_PUSH  = 4
};

uint8_t calculate_state(uint8_t state, int8_t sensor) {
  switch (state) {

    case STATE_PUSH:
    case STATE_IDLE:
      if (sensor==-1) {
        return STATE_PULL0;
      } else if (sensor==+1) {
        return STATE_PUSH;
      }
      return STATE_IDLE;

    case STATE_PULL0:
      if (sensor==-1) {
        return STATE_PULL1;
      } else if (sensor==+1) {
        return STATE_PUSH;
      }
      return STATE_IDLE;

    case STATE_PULL1:
      if (sensor==-1) {
        return STATE_PULL1;
      }
      return STATE_KICK;

    case STATE_KICK:
      return STATE_IDLE;
```

```
  }
  return state;
}

int main() {
  led_init();
  sens_init();

  uint8_t ball_pos = 3;
  int8_t direction = -1;

  uint8_t state_l = STATE_IDLE;
  uint8_t state_r = STATE_IDLE;

  while(1==1) {
    delay(100);

    state_l = calculate_state(state_l, sens_getLeft());
    if( ((state_l==STATE_PUSH) && (ball_pos==1)) ||
         ((state_l==STATE_KICK) && (ball_pos==0)) ) {
      direction = +1;
    }

    state_r = calculate_state(state_r, sens_getRight());
    if( ((state_r==STATE_PUSH) && (ball_pos==2)) ||
         ((state_r==STATE_KICK) && (ball_pos==3)) ) {
      direction = -1;
    }

    if(direction==+1) {
      if(ball_pos<3) {
        ball_pos++;
      } else {
        direction=0;
      }
    }

    if(direction==-1) {
      if(ball_pos>0) {
        ball_pos--;
      } else {
        direction=0;
      }
    }

    led_set(LED_L_YE, ball_pos==0);
    led_set(LED_L_RD, ball_pos==1);
    led_set(LED_R_RD, ball_pos==2);
    led_set(LED_R_YE, ball_pos==3);
  }
  return 0;
}
```

Erläuterungen zum Quellcode:

Das Programm beginnt mit der Definition von fünf Konstanten für einen Zustandsautomaten. Ein Zustandsautomat ist recht nützlich, da man mit ihm gut auf Eingaben in verschiedenen Situationen (Zuständen) reagieren kann. In diesem Fall beschreibt der Automat den Zustand eines Fühlers:

- `STATE_IDLE`: Der Fühler wurde nicht bewegt
- `STATE_PUSH`: Der Fühler wurde nach vorne gedrückt
- `STATE_PULL0` : Der Fühler wurde nach hinten gezogen
- `STATE_PULL1` : Der Fühler wurde nach hinten gezogen und entprellt
- `STATE_KICK`: Der Fühler wurde nach hinten gezogen und ist gerade losgelassen worden

Als Eingaben dient dem Automaten der Rückgabewert der `sens_getLeft()` bzw. `sens_getRight()` Funktion.

Die Funktion `calculate_state` berechnet die Reaktion des Automaten und bekommt den aktuellen Zustand und die Eingabe und liefert den zukünftigen Zustand zurück.

Im Hauptprogramm werden einige Initialisierungen durchgeführt. Danach beginnt die Hauptschleife. Dort wird zunächst 100 ms lang gewartet. Danach wird der neue Zustand für den linken Fühler berechnet. Falls sich der Ball an der Position 1 befindet und der Fühler gedrückt wurde, wird die Bewegungsrichtung (`direction`) nach rechts gesetzt. Dies geschieht auch wenn sich der Ball an der Position 0 befunden hat und die KICK-Funktion ausgelöst wurde.

Danach wird nach dem selben Schema die rechte Seite behandelt.

Im Anschluss daran wird die Bewegung behandelt: Wenn die Richtung positiv, also nach rechts ist soll der Ball sich nur solange bewegen, bis er die Position 3 erreicht hat. Nach dem Erreichen wird `direction` auf 0 gesetzt. Sinngemäß gilt das Gleiche für eine Bewegung nach links.

Zum Abschluss wird die LED, die der aktuellen Position entspricht, eingeschaltet.

Aufgaben/Anregungen:

- Zeichnen Sie ein Diagramm in dem Sie die Zustände und die Eingabemöglichkeiten übersichtlich darstellen.

12 Testen der Odometriesensoren

Als nächstes soll die Odometrie getestet werden: Dazu legen Sie wieder ein neues Projekt an, diesmal mit dem Namen **Odometrie** (wie in Kapitel 3 beschrieben). Denken Sie auch diesmal wieder an die zusätzlichen Einstellungen bei den *Project-Options*.

Dieses Testprogramm zeigt die Stände der Odometriezähler mit den LEDs an: Überschreitet der Zähler den Wert 10 so leuchtet die gelbe LED, überschreitet der Zähler den Wert 20 so leuchtet die rote LED.

Durch Betätigung eines Fühlers werden die Zähler zurückgesetzt.

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

```
#include <nibobee/iodefs.h>
#include <nibobee/led.h>
#include <nibobee/sens.h>
#include <nibobee/odometry.h>

int main() {
  led_init();
  odometry_init();
  sens_init();

  while(1==1) {
    enable_interrupts();
    if (sens_getLeft() || sens_getRight()) {
      odometry_reset();
    }

    led_set(LED_L_YE, odometry_getLeft(0)>10);
    led_set(LED_L_RD, odometry_getLeft(0)>20);
    led_set(LED_R_RD, odometry_getRight(0)>20);
    led_set(LED_R_YE, odometry_getRight(0)>10);
  }
  return 0;
}
```

Erläuterungen zum Quellcode:

Zunächst werden die Header-Datei iodefs.*h*, *led.h*, *sens.h* und zusätzlich die Header-Datei *odometry.h* durch die *#include* Direktive eingebunden.

In der main-Funktion werden zunächst durch `led_init()` die IO-Ports der LEDs als Ausgänge konfiguriert, mit `odometry_init()` die Odometrie-berechnung initialisiert und anschließend mit der Anweisung `sens_init()` die Fühlersensoren initialisiert.

In der darauf folgenden endlos-while-Schleife wird zu Anfang sichergestellt, das die Interrupts aktiviert sind. Dies ist notwendig, da die Odometrie-berechnung interruptgesteuert im Hintergrund durchgeführt wird.

Die folgende if-Anweisung setzt den aktuellen Odometriezählerstand zurück wenn einer der beiden Fühler betätigt wird.

Der Aufruf `odometry_getLeft(0)` liefert den aktuellen Zählerstand des linken Odometriezählers zurück ohne ihn (auf null) zurückzusetzen.

In den letzen vier Zeilen des while-Blocks werden die LEDs in Abhängigkeit von den Zählerständen gesetzt.

Hinweis:

Die Odometriezähler werden in diesem Beispiel unabhängig von der Drehrichtung immer hochgezählt. Im Motorbetrieb wird die Zählrichtung in Abhängigkeit von der Polarität automatisch umgeschaltet.

Aufgaben/Anregungen:

- Ändern Sie das Programm so ab, dass durch Betätigen des linken Fühlers der linke Odometriezähler zurück gesetzt wird und durch Betätigen des rechten Fühlers der rechte Zähler zurück gesetzt wird.

13 Ansteuerung der Motoren

Als nächstes sollen die Motoren angesteuert werden: Dazu legen wir ein Projekt **Motoren** an. Denken Sie auch diesmal wieder an die zusätzlichen Einstellungen bei den *Project-Options*.

Das Programm steuert die Motoren mittels der Fühler an. Eine Betätigung des Fühlers nach vorne lässt den jeweiligen Motor vorwärts drehen, eine Betätigung nach hinten lässt den Motor rückwärts drehen.

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

```
#include <nibobee/iodefs.h>
#include <nibobee/sens.h>
#include <nibobee/motpwm.h>

int main() {
  motpwm_init();
  sens_init();

  while(1==1) {
    enable_interrupts();
    int16_t speed_l=0;
    int16_t speed_r=0;

    switch (sens_getLeft()) {
      case  1: speed_l =  500; break;
      case  0: speed_l =    0; break;
      case -1: speed_l = -500; break;
    }

    switch (sens_getRight()) {
      case  1: speed_r =  500; break;
      case  0: speed_r =    0; break;
      case -1: speed_r = -500; break;
    }

    motpwm_setLeft(speed_l);
    motpwm_setRight(speed_r);

  }
  return 0;
}
```

Erläuterungen zum Quellcode:

Zunächst werden die Header-Dateien *iodefs.h*, *sens.h* und *motpwm.h* durch die *#include* Direktive eingebunden.

In der main-Funktion wird zunächst durch Aufruf von `motpwm_init()` die Motoransteuerung initialisiert. Mit der Anweisung `sens_init()` werden wie in den vorangegangenen Kapiteln die Fühlersensoren initialisiert.

In der darauf folgenden endlos-while-Schleife wird wieder zu Anfang sichergestellt, das die Interrupts aktiviert sind. Dies ist notwendig, da auch die Motoransteuerung interruptgesteuert im Hintergrund durchgeführt wird.

Anschließend werden zwei lokale Variablen deklariert in denen der Sollwert für die Motoren gespeichert wird. Die Sollwerte werden in den folgenden zwei switch/case Blöcken in Abhängigkeit von den Fühlern gesetzt. In den letzten beiden Zeilen der while-Schleife werden die Sollwerte den Motoren zugewiesen.

Hinweis:

Die Angabe der Werte erfolgt in 1/1024 Schritten: +1024 bedeutet 100% vorwärts, -512 bedeutet 50% rückwärts. Die Ansteuerung erfolgt mittels Pulsweitenmodulation (PWM).

Aufgaben/Anregungen:

- Ändern Sie das Programm so ab, dass die Vorwärtsdrehung schneller und die Rückwärtsdrehung langsamer erfolgt.
- Ändern Sie das Programm so ab, dass der Roboter möglichst sinnvoll über Kreuz angesteuert wird. (linker Fühler - rechtes Rad, rechter Fühler - linkes Rad)

14 Hindernisdetektion

Nun soll der NIBObee lernen, Hindernisse zu erkennen: Dazu legen wir ein Projekt **Hindernis** an. Denken Sie auch diesmal wieder an die zusätzlichen Einstellungen bei den *Project-Options*.

Mit diesem Programm kann der NIBObee umher fahren. Sobald mit den Fühlern Hindernisse detektiert werden, wird versucht diesen auszuweichen.

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

```
#include <nibobee/iodefs.h>
#include <nibobee/motpwm.h>
#include <nibobee/delay.h>
#include <nibobee/sens.h>

enum {
  MODE_STOP,
  MODE_DRIVE,
  MODE_BACK,
  MODE_STEER_R,
  MODE_STEER_L,
  MODE_AVOID_R,
  MODE_AVOID_L
};

int16_t speed_l;
int16_t speed_r;
uint16_t counter_ms;

uint8_t perform_check(uint8_t mode);

uint8_t do_stop();
uint8_t do_drive();
uint8_t do_back();
uint8_t do_steer_r();
uint8_t do_steer_l();
uint8_t do_avoid_r();
uint8_t do_avoid_l();


int main() {
  motpwm_init();
  sens_init();
  uint8_t mode;

  while(1==1) {
    enable_interrupts();
    delay(1);
```

```
    mode = perform_check(mode);

    switch (mode) {
      case MODE_STOP:    mode = do_stop(); break;
      case MODE_DRIVE:   mode = do_drive(); break;
      case MODE_BACK:    mode = do_back(); break;
      case MODE_STEER_R: mode = do_steer_r(); break;
      case MODE_STEER_L: mode = do_steer_l(); break;
      case MODE_AVOID_R: mode = do_avoid_r(); break;
      case MODE_AVOID_L: mode = do_avoid_l(); break;
    }

    switch (mode) {
      case MODE_STOP:    speed_l =    0; speed_r =    0;
      break;
      case MODE_DRIVE:   speed_l =  500; speed_r =  500;
      break;
      case MODE_BACK:    speed_l = -500; speed_r = -500;
      break;
      case MODE_STEER_R: speed_l =  600; speed_r =  400;
      break;
      case MODE_STEER_L: speed_l =  400; speed_r =  600;
      break;
      case MODE_AVOID_R: speed_l = -400; speed_r = -600;
      break;
      case MODE_AVOID_L: speed_l = -600; speed_r = -400;
      break;
    }

    motpwm_setLeft(speed_l);
    motpwm_setRight(speed_r);
  }
  return 0;
}

uint8_t perform_check(uint8_t mode) {
  if (sens_getLeft() && sens_getRight()) {
    if ((sens_getLeft()==-1) && (sens_getRight()==-1)) {
      mode = MODE_BACK;
    } else {
      mode = MODE_STOP;
    }
  }
  return mode;
}


uint8_t do_stop() {
  if ((sens_getLeft()==0) && (sens_getRight()==0)) {
    return MODE_DRIVE;
  }
```

```
  return MODE_STOP;
}

uint8_t do_back() {
  if (sens_getLeft()==0) {
    return MODE_AVOID_L;
  }
  if (sens_getRight()==0) {
    return MODE_AVOID_R;
  }
  return MODE_BACK;
}

uint8_t do_drive() {
  if (sens_getRight()==1) {
    return MODE_STEER_L;
  }
  if (sens_getLeft()==1) {
    return MODE_STEER_R;
  }
  if (sens_getRight()==-1) {
    return MODE_AVOID_L;
  }
  if (sens_getLeft()==-1) {
    return MODE_AVOID_R;
  }
  return MODE_DRIVE;
}

uint8_t do_steer_r() {
  if (sens_getLeft()==0) {
    return MODE_DRIVE;
  }
  return MODE_STEER_R;
}

uint8_t do_steer_l() {
  if (sens_getRight()==0) {
    return MODE_DRIVE;
  }
  return MODE_STEER_L;
}

uint8_t do_avoid_r() {
  if (counter_ms==0) {
    counter_ms=1000;
  } else {
    counter_ms--;
  }
  if (counter_ms) {
    return MODE_AVOID_R;
  } else {
```

```
    return MODE_DRIVE;
  }
}

uint8_t do_avoid_l() {
  if (counter_ms==0) {
    counter_ms=1000;
  } else {
    counter_ms--;
  }
  if (counter_ms) {
    return MODE_AVOID_L;
  } else {
    return MODE_DRIVE;
  }
}
```

Erläuterungen zum Quellcode:

Das Programm Hindernisdetektion basiert auf einem Zustandsautomaten mit folgenden Zuständen:

- `MODE_STOP`: Anhalten
- `MODE_DRIVE`: Geradeaus fahren
- `MODE_BACK`: Rückwärts fahren
- `MODE_STEER_R`: Ausweichen, Kurve nach rechts fahren
- `MODE_STEER_L`: Ausweichen, Kurve nach links fahren
- `MODE_AVOID_R`: Zurückweichen, dabei nach rechts drehen
- `MODE_AVOID_L`: Zurückweichen, dabei nach links drehen

Die Funktion `perform_check()` reagiert in den Situationen in denen beide Fühler aktiviert wurden – unabhängig vom aktuellen Zustand: Wurden beide Fühler nach hinten gedrückt, so wechselt das Programm in den Zustand `MODE_BACK`. Ansonsten wird in den Zustand `MODE_STOP` gewechselt.

Die Funktionen `do_xxx()` reagieren auf die Eingaben durch die Fühler, indem sie den Nachfolgezustand zurück liefern.

Die beiden Funktionen `do_avoid_r()` und `do_avoid_l()` sind zeitgesteuert, sie kehren nach 1000 Aufrufen (ca. 1 Sekunde) in den Zustand `MODE_DRIVE` zurück.

Aufgaben/Anregungen:

- Erweitern und verbessern Sie das Programm!

15 Arbeiten mit den Liniensensoren

In diesem Beispiel wollen wir uns mit den Liniensensoren des Roboters beschäftigen. Es handelt sich dabei um zwei IR-LEDs und drei Phototransistoren. Die Sensoren arbeiten nach dem IR-Reflexionsverfahren. Dabei wird gemessen welcher Anteil vom ausgesendeten Licht zurück reflektiert wird.

Das Testprogramm zeigt die Werte des linken und des rechten Liniensensors mit Hilfe der LEDs an. Die Sensoren müssen dazu vorher kalibriert werden. Bei fehlender Reflexion sind alle LEDs aus, bei geringer Reflexion leuchten die gelben LEDs, bei höherer Reflexion leuchten sowohl die gelben als auch die roten LEDs.

Legen Sie wie in den vorangegangenen Beispielen ein neues Projekt an.

Tippen Sie anschließend folgenden Quellcode in das Editorfenster ein:

```
#include <nibobee/iodefs.h>
#include <nibobee/led.h>
#include <nibobee/line.h>

int main() {
  led_init();
  line_init();

  while(1==1) {
    enable_interrupts();
    led_set(LED_L_YE, line_get(LINE_L)>160);
    led_set(LED_L_RD, line_get(LINE_L)>240);
    led_set(LED_R_YE, line_get(LINE_R)>160);
    led_set(LED_R_RD, line_get(LINE_R)>240);
  }
  return 0;
}
```

Erläuterungen zum Quellcode:

Zusätzlich zu den schon bekannten Header-Dateien wird in diesem Beispiel noch die Datei *line.h* eingebunden.

In der main-Methode werden mit der Methode `led_init()` wieder die LEDs initialisiert und danach werden mittels der Methode `line_init()` alle Methoden rund um die Linienfolge bereitgestellt.

Die weiteren Anweisungen laufen innerhalb einer endlos-while-Schleife ab:

Zunächst werden mit dem Aufruf der Methode `enable_interrupts()` die Interrupt-Routinen aktiviert.

Nun wird mit der Funktion „line_get“ gearbeitet. Diese Funktion bekommt als Parameter den Wert eines Liniensensors (rechts: LINE_R, links: LINE_L) übergeben. Nach einer kurzen Verrechnung wird der errechnete Wert mit der Zahl 160 (um geringe Abstände zu erkennen) und mit der Zahl 240 (um größere Abstände zu erkennen) verglichen. Ergibt dieser Vergleich „wahr“, so ist der zweite Parameter der Funktion „led_set“ eine 1, die jeweilige LED wird also eingeschaltet, ansonsten 0, womit die betreffende LED ausgeschaltet wird.

Insgesamt wird so erreicht, dass bei fehlender Reflexion alle LEDs aus sind, bei geringer Reflexion die gelben LEDs leuchten und bei höherer Reflexion sowohl die gelben als auch die roten LEDs leuchten.

Aufgaben/Anregungen:

- Schreiben Sie ein Programm (mit Ihrem Wissen aus den vorangegangenen Kapiteln), mit dem der Roboter einer schwarzen Linie auf weißem Untergrund folgen kann.

16 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 (für den NIBObee geeignet).

Weitere Informationen:

- NIBObee Hauptseite: http://nibobee.nicai-systems.de Die Homepage des NIBObee Herstellers. Liefert technische Informationen, die Bauanleitung und weitere Links.
- NIBObee und Nibo2 Wiki: http://www.nibo-roboter.de Liefert alle Informationen rund um den NIBObee und den Nibo2.
- Shop: http://shop.nicai-systems.de Online-Shop für die Nibo Roboter und Erweiterungssets.
- Mikrocontroller: http://www.mikrocontroller.net Alles über Mikrocontroller und deren Programmierung.
- AVRFreaks: http://www.avrfreaks.net Informationen rund um den AVR.