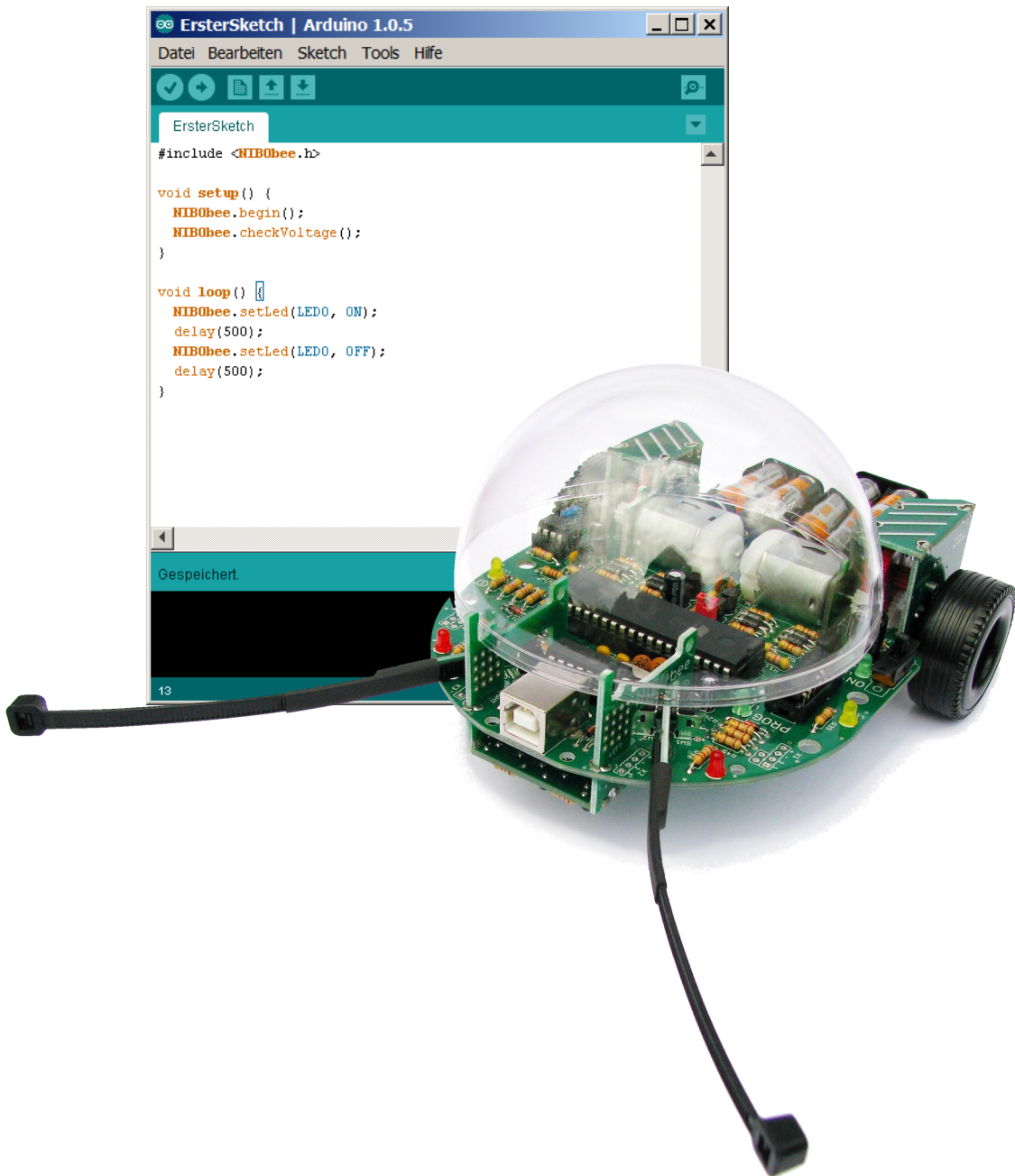


# Roboterbausatz NIBObee

## ARDUINO 1.05 Programmier-Tutorial



## Inhaltsverzeichnis

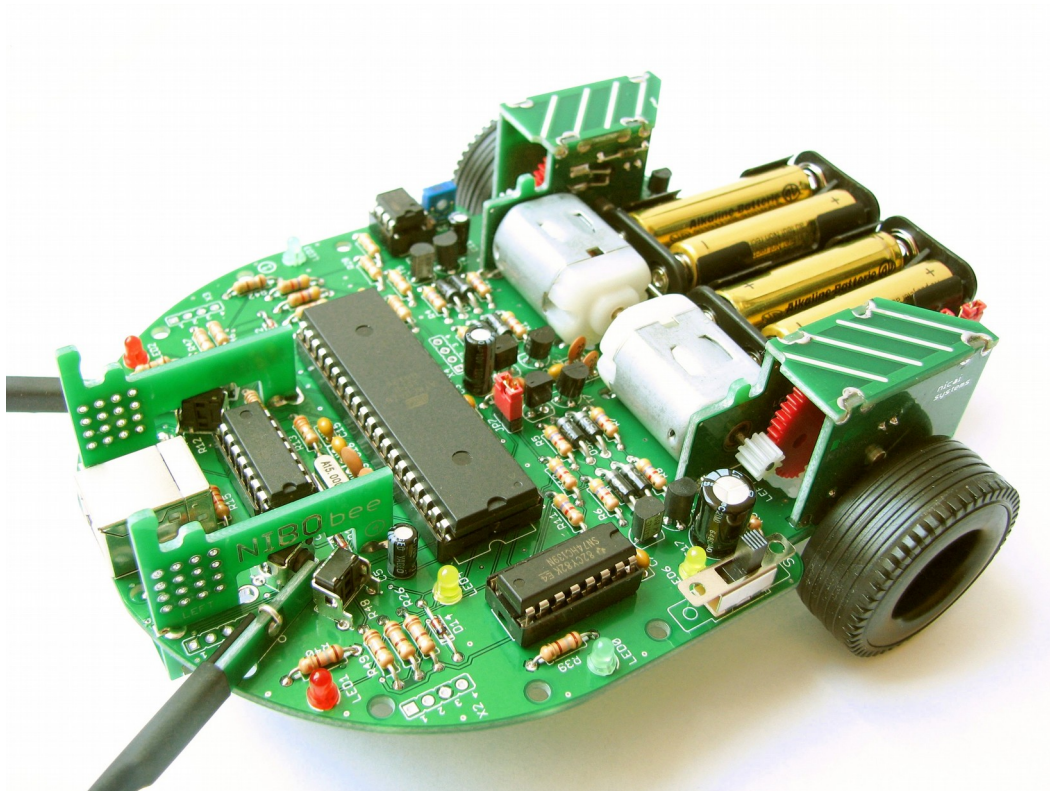
1	Einleitung.....	3
2	Installation der Programmierumgebung.....	4
2.1	ARDUINO.....	4
3	Installation der NiboRoboLib.....	6
4	Vorbereitungen.....	7
5	Der erste Sketch.....	10
6	LEDs.....	15
7	LEDs Action.....	17
8	Tastensensoren / Fühler.....	19
9	Ping Pong.....	22
10	Testen der Odometriesensoren.....	28
11	Ansteuerung der Motoren.....	30
12	Hindernisdetektion.....	32
13	Liniensensoren.....	38
13.1	Kalibrierung der Liniensensoren.....	38
13.2	Arbeiten mit den Liniensensoren.....	38
14	Dokumentation.....	41
15	Links zu weiterführenden Internetseiten.....	42

## 1 Einleitung

Dieses Tutorial bietet eine Schritt für Schritt Anleitung für die erste Inbetriebnahme des NIBObee mit **ARDUINO**. 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.



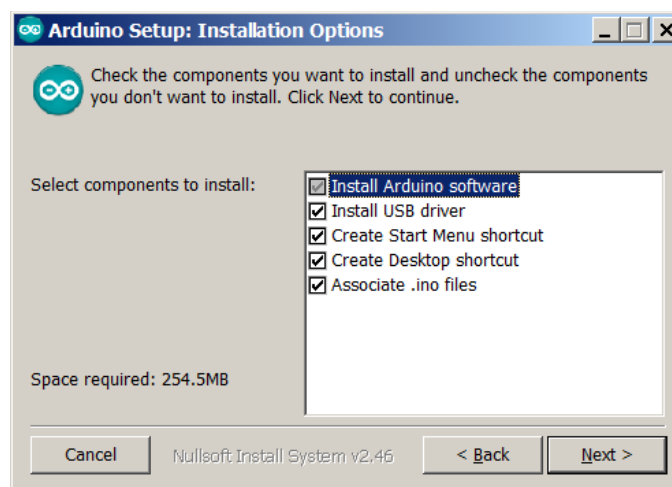
## 2 Installation der Programmierumgebung

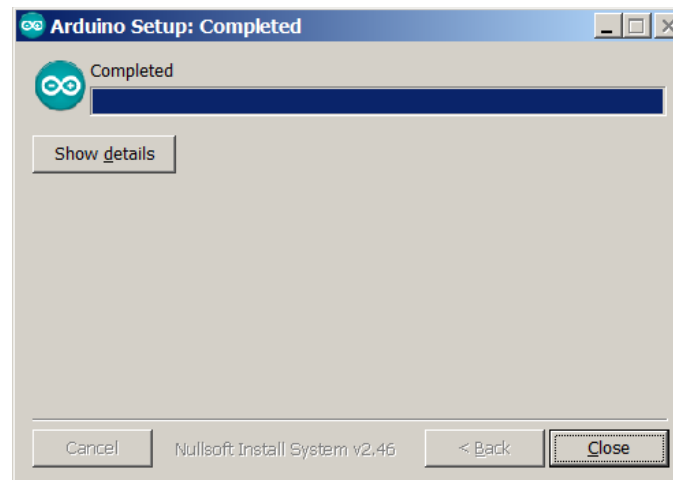
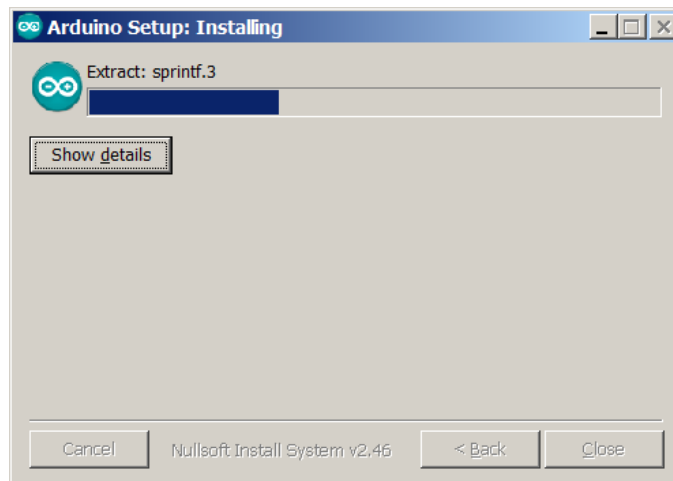
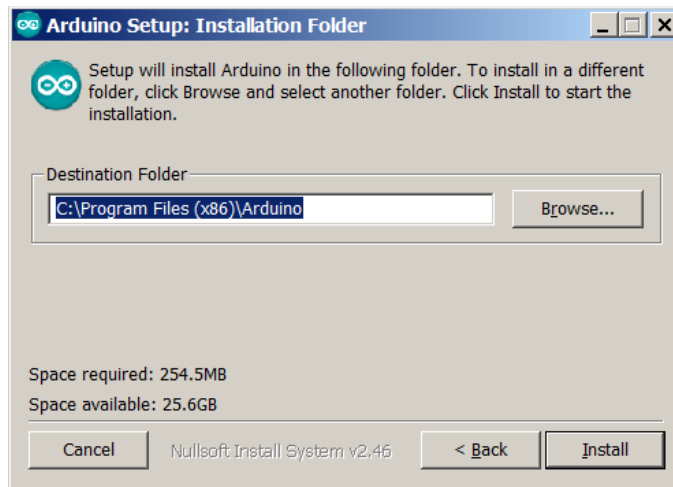
### 2.1 ARDUINO

Zunächst muss das ARDUINO installiert werden. Hierzu laden Sie sich die Installationsdatei *Arduino 1.0.5* von <http://arduino.cc> für Ihr entsprechendes Betriebssystem herunter, für Windows wählen Sie die Installer-Version aus.

Im Folgenden wird die Installation für Windows beschrieben:

Doppelklicken Sie nun die heruntergeladene Datei **arduino-1.0.5-windows.exe** und installieren Sie das Arduino wie vom Installer vorgeschlagen auf Ihrem Computer.





### 3 *Installation der NiboRoboLib*

Zunächst wird die NiboRoboLib installiert. Die **neueste** Version und eine **Installationsanleitung** als .pdf-Datei finden sich unter:



Alternativ befindet sich die Dateien auch auf der beiliegenden CD.

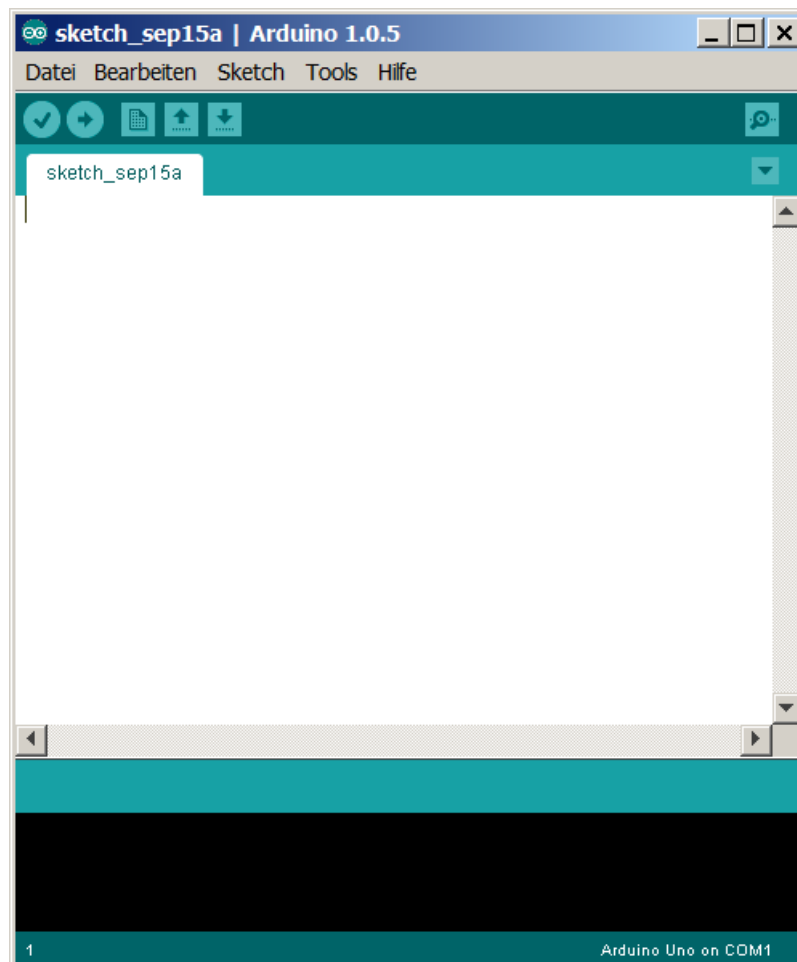
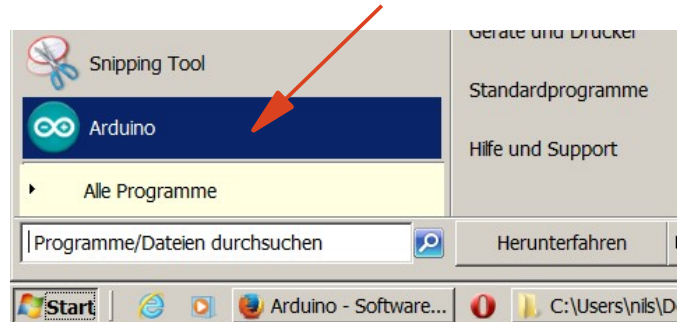
Die **NiboRoboLib** enthält:

- + Alle benötigten **Treiber** für den UCOM-IR2-X
- + Alle benötigten **Treiber** für den NIBObee
- + **RoboDude** (Übertragungsprogramm für .hex- und .xhex-Dateien)
- + **C-Bibliothek** und **Testprogramme** für den NIBO2
- + **C-Bibliothek** und **Testprogramme** für den NIBObee
- + **Kalibrierprogramme** für die Sensoren
- + **ARDUINO**-Bibliothek für den NIBO2
- + **ARDUINO**-Bibliothek für den NIBObee

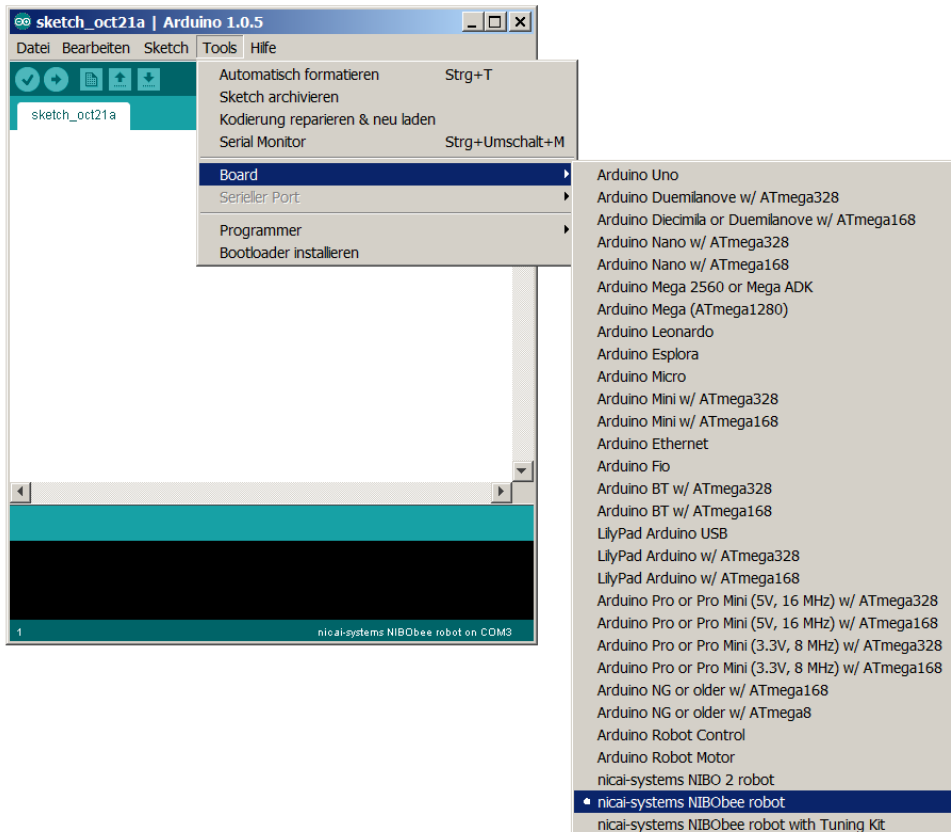
Während des Installationsvorgangs kann man wählen, ob man alle Pakete installieren möchte, oder nur eine Auswahl.

## 4 Vorbereitungen

Starten Sie die ARDUINO-Oberfläche:

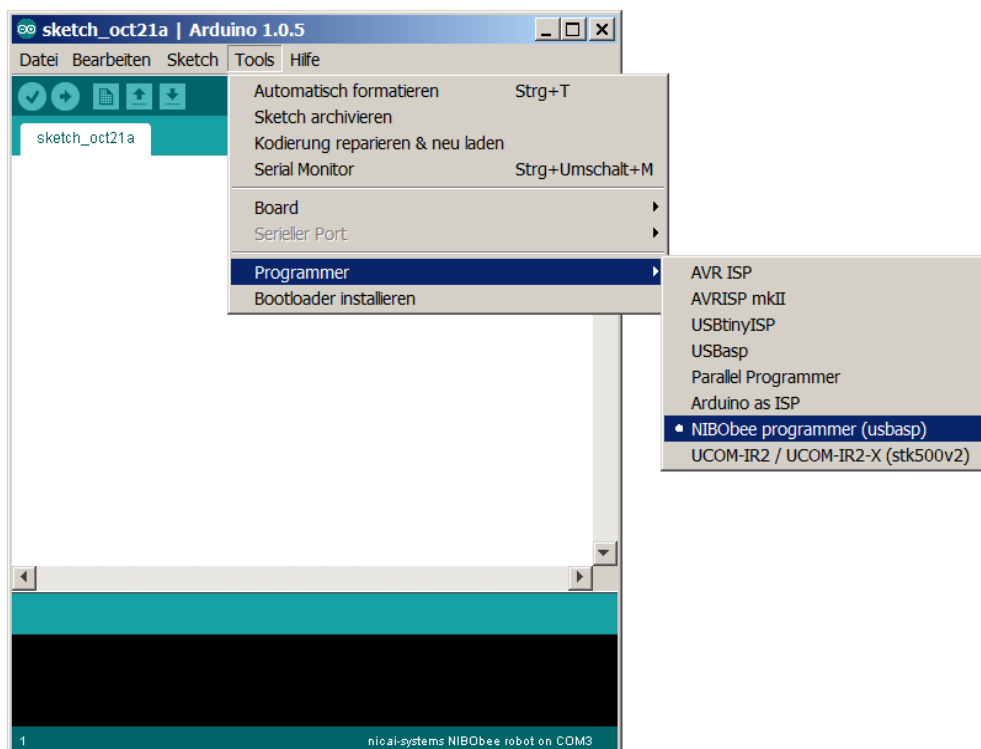


Über **Tools – Board – nicali-systems NIBObee robot** wird der NIBObee als Board ausgewählt:



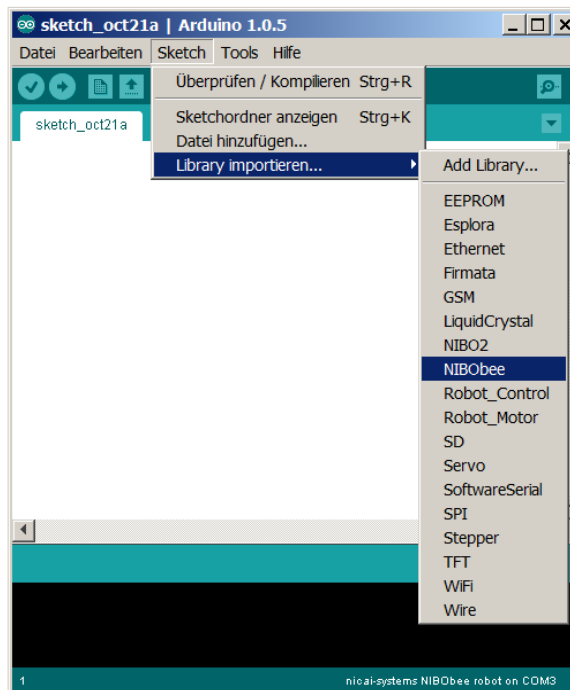


Über **Tools – Programmer – NIBObee programmer** wird der integrierte Programmer des NIBObee als Programmer ausgewählt:

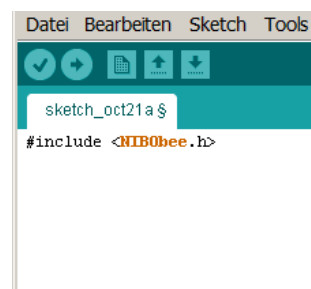


## 5 Der erste Sketch

Zunächst wird die NIBObee Library importiert:




Durch das Importieren wird automatisch der folgende Quelltext erzeugt:



Speichern Sie Ihren Sketch unter dem Namen *ErsterSketch*.

Ergänzen Sie den entstandenen Quellcode wie folgt:



```
#include <NIBObee.h>

void setup() {
  NIBObee.begin();
  NIBObee.checkVoltage();
}

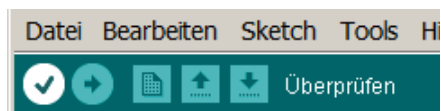
void loop() {
  NIBObee.setLed(LED0, ON);
  delay(500);
  NIBObee.setLed(LED0, OFF);
  delay(500);
}
```

Gespeichert.

13 nical-systems NIBObee robot on COM3

Speichern Sie Ihr geändertes Programm erneut ab!

Jetzt wird das Programm überprüft, indem Sie den **Überprüfen-Knopf** drücken:



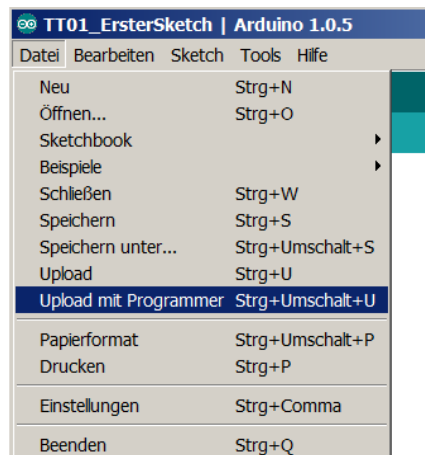
Der Sketch wird nun kompiliert und dabei überprüft. Eventuell auftretende

Fehlermeldungen erscheinen im schwarzen Bereich des Hauptfensters.

Wenn alles ohne Probleme geklappt hat, erscheint in dem türkisen Bereich die Meldung *Kompilierung abgeschlossen*.

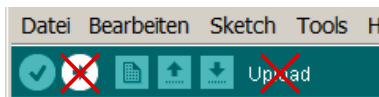
Schalten Sie jetzt den NIBObee ein und schließen ihn an Ihren Rechner an.

Jetzt können Sie den ersten Sketch auf den Roboter übertragen. Wählen Sie dazu im Menü „**Datei**“ -> „**Upload mit Programmierer**“ aus:



Wenn jetzt die linke gelbe LED (*LED0*) blinkt, hat alles bestens geklappt!

**Hinweis:** Es **muss** immer die Funktion „**Upload mit Programmierer**“ verwendet werden! Nur „**Upload**“ funktioniert nicht!



## Erläuterungen zum Quellcode:

Grundsätzlicher Aufbau eines ARDUINO-Sketches:

```
#include <Bibliothek.h>
// durch die #include-Anweisungen werden die gewählten Bibliotheken
// eingebunden. Dort sind Klassen, Variablen und Konstanten definiert.

void setup() {
// alle Anweisungen, die zwischen den geschweiften Klammern des
// setup-Blocks stehen, werden während des Programmablaufs
// genau einmal am Anfang ausgeführt.
}

void loop() {
// alle Anweisungen, die zwischen den geschweiften Klammern des
// loop-Blocks stehen, werden während des Programmablaufs
// immer wieder erneut ausgeführt.
}
```

Zurück zu unserem ersten Sketch:



```
ErsterSketch
1 #include <NIBObee.h>
2
3 void setup() {
4   NIBObee.begin();
5   NIBObee.checkVoltage();
6 }
7
8 void loop() {
9   NIBObee.setLed(LED0, ON);
10  delay(500);
11  NIBObee.setLed(LED0, OFF);
12  delay(500);
13 }
14
```

- 01 In der ersten Zeile wird durch die `#include` Anweisung die Header-Datei der NIBObee Bibliothek eingebunden.
- 03 - 06 Zwischen der Zeile `void setup() {` und der Zeile mit der schließenden geschweiften Klammer `}` befindet sich der Funktionsblock für die Programminitialisierung. Die Funktion wird genau einmal nach dem Einschalten des Roboters ausgeführt.  
Durch die Anweisung `NIBObee.begin();` wird der Roboter initialisiert.  
**Wichtig:** Diese Anweisung **muss** in jeder `setup`-Funktion als erstes ausgeführt werden!  
Die Anweisung `NIBObee.checkVoltage();` überprüft die Spannung der Akkus direkt nach dem Einschalten. Sollte die Spannung nicht OK sein (Akkus leer) blinkt der NIBObee wild mit den LEDs...
- 08 - 13 Zwischen der Zeile `void loop() {` und der Zeile mit der schließenden geschweiften Klammer `}` befindet sich die Hauptroutine des Programms. Die Funktion wird immer wieder, bis zum Ausschalten des Roboters ausgeführt.
- 09 Innerhalb des Funktionsblocks der Hauptroutine wird mit der Anweisung `NIBObee.setLed(LED0, ON);` die LED 0 (die gelbe LED auf der linken Seite) eingeschaltet.
- 10 Anschließend wird durch die Anweisung `delay(500);` eine Pause von einer halben Sekunde eingelegt, das entspricht 500 Millisekunden.
- 11 - 12 Nach der Pause wird die LED 0 wieder ausgeschaltet und mit der nächsten Zeile ist wieder eine Pause von 500 Millisekunden.
- 13 Nach der Pause endet die `loop`-Funktion und wird anschließend automatisch wieder von vorn abgearbeitet. **Das war's schon!**

**Tip:** Alle Beispiele aus diesem Tutorial sind auch unter „Datei“ -> „Beispiele“ -> „NIBObee“ -> „Tutorial“ zu finden.

## 6 LEDs

Hier wollen wir noch ein anderes Experiment mit den LEDs machen.

Erzeugen Sie einen neuen Sketch, indem Sie im Menü „Datei“ -> „Neu“ auswählen. Speichern Sie den neuen Sketch unter dem Namen „LEDs“.

Als Bibliothek importieren Sie wieder „NIBObee“ wie in Kapitel 5 beschrieben.

Bei diesem Sketch sollen die Leuchtdioden nacheinander reihum eingeschaltet und wieder ausgeschaltet werden.

Tippen Sie den folgenden Quellcode in das Editorfenster ein:



```
TT02_LEDs
1 #include <NIBObee.h>
2
3 void setup() {
4     NIBObee.begin();
5     NIBObee.checkVoltage();
6 }
7
8 void loop() {
9     for (int led=0; led<4; led++) {
10        NIBObee.setLed(led, ON);
11        delay(350);
12        NIBObee.setLed(led, OFF);
13        delay(150);
14    }
15 }
```

### Erläuterungen zum Quellcode:

01 - 06 Dieser Quellcode unterscheidet sich nur im Anweisungsblock der loop-Funktion von dem vorherigen Programm aus Kapitel 5.

08 - 15 Die erste Anweisung innerhalb der loop-Funktion „for (int led=0; led<4; led++)“ dient zur mehrfachen

Ausführung der Anweisungen im Block zwischen den geschweiften Klammern.

Bei dieser „for-Schleife“ wird der Block genau 4 mal ausgeführt, dabei hat die Variable „led“ jeweils einen anderen Wert: Beim ersten Durchlauf den Wert 0, und bei den folgenden Durchläufen die Werte 1, 2 und 3.

Die for-Schleife ist ein sehr mächtiges Universalwerkzeug - generell besteht sie aus den folgenden Komponenten:

```
for (Initialisierung; Bedingung; Fortsetzung) {  
  // dieser Block wird solange ausgeführt, wie die Bedingung wahr ist.  
}
```

09 - 14 Im Initialisierungsbereich wird die Ganzzahl-Variable „led“ angelegt und mit dem Wert 0 initialisiert. Die Bedingung für einen Durchlauf ist, dass der Wert der Variablen led kleiner als 4 sein muss. Im Fortsetzungsbereich wird der Wert der Variablen led für den nächsten Durchlauf um eins erhöht. Der Ausdruck „led++“ ist dabei eine Abkürzung für den Ausdruck „led = led + 1“. Damit läuft die for-Schleife über die Variable led, die nacheinander die Werte 0, 1, 2 und 3 annimmt.

10 - 13 Im Anweisungsblock der for-Schleife stehen die eigentlichen Anweisungen:

Mit der Anweisung „**NIBObee.setLed**(led, ON);“ wird die LED mit der Nummer **led** eingeschaltet.

Durch die Anweisung „**delay**(350);“ wird eine Zeitspanne von 350 ms gewartet, bevor die nächste Anweisung ausgeführt wird.

Mit der Anweisung „**NIBObee.setLed**(led, OFF);“ wird die LED mit der Nummer **led** ausgeschaltet.

Durch die letzte Anweisung im Block wird eine Zeitspanne von 150 ms gewartet, bevor der nächste Schleifendurchlauf beginnt.



## 7 LEDs Action

Hier wollen wir nun etwas kompliziertere Leuchtmuster gestalten.

Erzeugen Sie einen neuen Sketch, indem Sie im Menü „Datei“ -> „Neu“ auswählen. Speichern Sie den Sketch unter dem Namen „LEDsAction“.

Als Bibliothek importieren Sie wieder „NIBObee“ wie in Kapitel 5 beschrieben.

Bei diesem Sketch sollen die Leuchtdioden nacheinander reihum ein- und wieder ausgeschaltet werden. Nach jedem Durchlauf soll sich dabei die Geschwindigkeit verändern.

Tippen Sie den folgenden Quellcode in das Editorfenster ein:



```
TT03_LEDsAction
1 #include <NIBObee.h>
2
3 void setup() {
4     NIBObee.begin();
5     NIBObee.checkVoltage();
6 }
7
8 void loop() {
9     for (int time=50; time<800; time*=2) {
10        for (int led=0; led<4; led++) {
11            NIBObee.setLed(led, ON);
12            delay(time);
13            NIBObee.setLed(led, OFF);
14            delay(time);
15        }
16    }
17 }
```

### Erläuterungen zum Quellcode:

01 - 06 Dieser Quellcode unterscheidet sich nur im Anweisungsblock der loop-Funktion von dem vorherigen Programm aus Kapitel 5.

08 - 17 Die erste Anweisung innerhalb der loop-Funktion „for (int time=50; time<800; time\*=2)“ dient zur mehrfachen Ausführung der Anweisungen im Block zwischen den

08 - 17 geschweiften Klammern.

Im Initialisierungsbereich wird die Ganzzahl-Variable „time“ angelegt und mit dem Wert 50 initialisiert. Die Bedingung für einen Durchlauf ist, dass der Wert der Variablen `time` kleiner als 800 sein muss. Im Fortsetzungsbereich wird der Wert der Variablen `time` für den nächsten Durchlauf verdoppelt.

Der Ausdruck „`time*=2`“ ist dabei eine Abkürzung für den Ausdruck „`time = time * 2`“.

Damit läuft die äußere for-Schleife über die Variable `time`, die nacheinander die Werte 50, 100, 200 und 400 annimmt.

10 - 15 Die innere for-Schleife läuft über die Variable `led`. Die Variable nimmt dabei nacheinander die Werte 0 bis 3 an.

11 - 14 Im Anweisungsblock der inneren for-Schleife stehen die eigentlichen Anweisungen:

Mit der Anweisung „`NIBObee.setLed(led, ON);`“ wird die LED mit der Nummer `led` eingeschaltet.

Durch die Anweisung „`delay(time);`“ wird eine Zeitspanne entsprechend dem Wert der Variable `time` gewartet, bevor die nächste Anweisung ausgeführt wird.

Mit der Anweisung „`NIBObee.setLed(led, OFF);`“ wird die LED mit der Nummer `led` ausgeschaltet.

Durch die letzte Anweisung im Block wird eine Zeitspanne entsprechend dem Wert der Variable `time` gewartet, bevor der nächste Schleifendurchlauf beginnt.

#### **Ideen & Anregungen:**

1. Verändern Sie Ihr Programm so, dass sich die zeitliche Veränderung umkehrt, die Geschwindigkeit soll sich jetzt von Runde zu Runde erhöhen.
2. Verändern Sie Ihr Programm so, dass sich die Laufrichtung umkehrt, sprich die Diode LED0 soll in jedem Durchlauf als letzte LED aufleuchten.

## 8 Tastsensoren / Fühler

Als nächstes wollen wir die Tastsensoren ansteuern. Dazu legen wir wieder einen neuen Sketch an, den wir diesmal unter dem Namen „**Fuehler**“ speichern. Als Bibliothek importieren wir wieder „NIBObee“.

Tippen Sie den folgenden Quellcode in das Editorfenster ein:

```
TT04_Fuehler
1 #include <NIBObee.h>
2
3 void setup() {
4     NIBObee.begin();
5     NIBObee.checkVoltage();
6 }
7
8 void loop() {
9     int8_t left = FeelerL.get();
10    switch (left) {
11        case -1:
12            NIBObee.setLed(LED1, OFF);
13            NIBObee.setLed(LED0, ON);
14            break;
15        case +1:
16            NIBObee.setLed(LED1, ON);
17            NIBObee.setLed(LED0, OFF);
18            break;
19        default:
20            NIBObee.setLed(LED1, OFF);
21            NIBObee.setLed(LED0, OFF);
22            break;
23    }
24
25    int8_t right = FeelerR.get();
26    switch (right) {
27        case -1:
28            NIBObee.setLed(LED2, OFF);
29            NIBObee.setLed(LED3, ON);
30            break;
31        case +1:
32            NIBObee.setLed(LED2, ON);
33            NIBObee.setLed(LED3, OFF);
34            break;
35        default:
36            NIBObee.setLed(LED2, OFF);
37            NIBObee.setLed(LED3, OFF);
38            break;
39    }
40 }
```

Unser Programm soll folgendes machen:

Wird der **linke** Fühler nach **hinten** gedrückt, so leuchtet die **linke gelbe Led** LED0, wird er nach **vorne** gedrückt, so leuchtet die **linke rote Led** LED1.

Entsprechendes passiert auf der rechten Seite:

Wird der **rechte** Fühler nach **hinten** gedrückt, so leuchtet die **rechte gelbe Led** LED3, wird er nach **vorne** gedrückt, so leuchtet die **rechte rote Led** LED2.

### Erläuterungen zum Quellcode:

01 - 06 Dieser Quellcode unterscheidet sich nur im Anweisungsblock der loop-Funktion von dem vorherigen Programm.

#### loop-Funktion:

09 In dieser Zeile definieren wir uns die Variable „left“, in der wir uns den Rückgabewert der Funktion „FeelerL.get()“ speichern. Die Funktionen „FeelerL.get()“ und „FeelerR.get()“ liefern genau einen von drei Werten zurück:

- 1: Fühler wird nach hinten gedrückt
- 0: Fühler ist in Ruhestellung
- +1: Fühler wird nach vorne gedrückt

10 - 23 Die **switch**-Anweisung ab Zeile 10 ermöglicht eine Fallunterscheidung. Je nachdem welchen Wert die Variable „left“ hat wird ein bestimmter Fall abgearbeitet. Die einzelnen Fälle werden durch das Schlüsselwort **case** eingeleitet und durch eine **break**-Anweisung beendet. Die switch-Anweisung führt also die verschiedenen Anweisungs-Blöcke in Abhängigkeit vom Wert der Variablen „left“ aus:

11 - 14 Hat „left“ den Wert -1, dann wird die LED1 ausgeschaltet und die LED0 wird eingeschaltet.

15 - 18 Hat „left“ den Wert +1, dann wird die LED1 eingeschaltet und die LED0 wird ausgeschaltet.

- 19 - 22 In allen anderen Fällen werden beide LEDs ausgeschaltet.
- 25 In dieser Zeile definieren wir uns die Variable „right“, in der wir uns den Rückgabewert der Funktion „FeelerR.get()“ speichern.
- 26 - 39 Die **switch**-Anweisung ab Zeile 26 ermöglicht wieder eine Fallunterscheidung, diesmal jedoch nach dem Wert der Variablen „right“:
- 27 - 30 Hat „right“ den Wert -1, dann wird die LED2 ausgeschaltet und die LED3 wird eingeschaltet.
- 31 - 34 Hat „right“ den Wert +1, dann wird die LED2 eingeschaltet und die LED3 wird ausgeschaltet.
- 35 - 38 In allen anderen Fällen werden beide LEDs ausgeschaltet.

#### ***Ideen & Anregungen:***

1. Ändern Sie das Programm so ab, dass die roten LEDs durch Betätigung der Fühler nach vorne eingeschaltet werden und erst bei Betätigung in die Gegenrichtung wieder ausgeschaltet werden.

## 9 Ping Pong

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

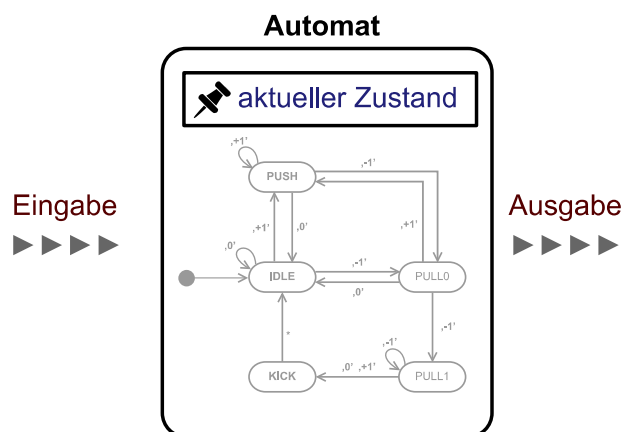
Beim Ping-Pong Spiel soll ein Ball hin und her gespielt werden. Als Ball dient uns dabei eine leuchtende LED, die Schläger sollen die beiden Fühler sein.

Der Ball kann auf zwei Arten gespielt werden:

- A:** Fühler wird nach vorne gedrückt, wenn die rote LED auf der entsprechenden Seite leuchtet.
- B:** Der Fühler wird nach hinten gezogen und dann losgelassen, wenn die gelbe LED leuchtet.

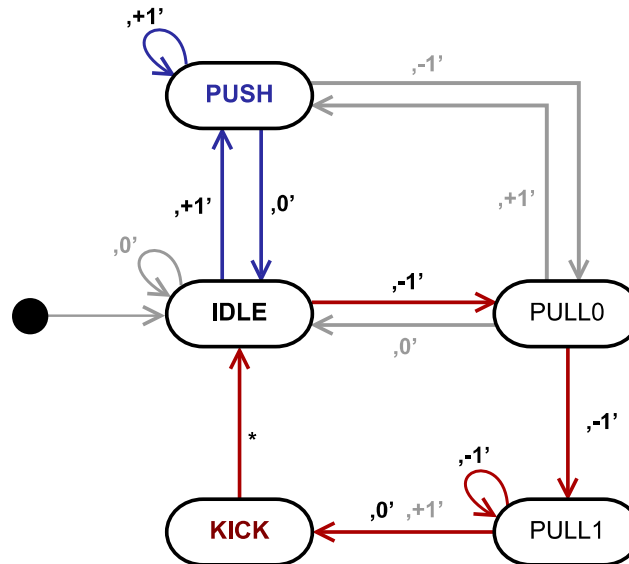
Für dieses Spiel werden wir für jeden Fühler einen Zustandsautomaten verwenden um die Eingaben auszuwerten. Ein Automat ist sehr praktisch zur Auswertung von solchen Eingaben, da man damit leicht sicher stellen kann, dass alle Eingabemöglichkeiten abgedeckt werden.

Der Automat arbeitet Schrittweise, in jedem Schritt verarbeitet er genau eine Eingabe und erzeugt eine Ausgabe. Der Automat wechselt dabei von Schritt zu Schritt seinen aktuellen Zustand.



Bei unserem Spiel werden genau 10 Schritte pro Sekunde abgearbeitet. Als Ausgabe verwenden wir einfach den aktuellen Zustand unseres Automaten.

Der Automat der die Signale der Fühler auswerten kann, sieht als UML Diagramm folgendermaßen aus:



Der blaue Weg ist für die Spielart A wichtig, der rote Weg für die Spielart B. Der schwarze Punkt mit dem Pfeil markiert den Startzustand. Die grauen Wege zeigen, was jeweils bei anderen Eingabe passieren soll...

Beim Einschalten wird der Zustand des Automaten auf "IDLE" gesetzt. Wenn der Fühler nicht gedrückt wird (Eingabe "0") bleibt der Automat im Zustand "IDLE".

#### A:

- Wird der Fühler jetzt nach vorne gedrückt (Eingabe "+1") wechselt der Automat in den Zustand "PUSH".
- In diesem Zustand bleibt der Automat solange der Fühler weiterhin nach vorne gedrückt wird.
- Wenn man den Fühler jetzt los lässt wechselt der Automat wieder in den Zustand "IDLE".

#### B:

- Wird der Fühler jetzt nach hinten gezogen (Eingabe "-1") wechselt der Automat in den Zustand "PULL0".
- Ist der Fühler auch beim nächsten Schritt (eine Zehntel Sekunde später) noch nach hinten gezogen, wechselt der Automat in den Zustand "PULL1", dort bleibt er dann so lange wie der Fühler nach

hinten weiterhin gezogen wird.

- Lässt man den Fühler jetzt los, wechselt der Automat in den Zustand "KICK".
- Direkt im nächsten Schritt wechselt der Automat unabhängig von einer Eingabe in den Zustand "IDLE".

Das Hauptprogramm interessiert sich abhängig von der Ballposition nur für zwei Zustände des jeweiligen Automaten: "PUSH" und "KICK":

- An Position 0 wird der Ball gespielt wenn links "gekicked" wird.
- An Position 1 wird der Ball gespielt wenn links "gepushed" wird.
- An Position 2 wird der Ball gespielt wenn rechts "gepushed" wird.
- An Position 3 wird der Ball gespielt wenn rechts "gekicked" wird.

Legen Sie wie in den vorangegangenen Beispielen einen neuen Sketch an, und speichern diesen unter dem Namen „**PingPong**“. Als Bibliothek importieren wir wieder „NIBObee“.

Ergänzen Sie den Quellcode im Editorfenster:

```
1 #include <NIBObee.h>
2
3 const unsigned int STATE_IDLE = 0;
4 const unsigned int STATE_PULL0 = 1;
5 const unsigned int STATE_PULL1 = 2;
6 const unsigned int STATE_KICK = 3;
7 const unsigned int STATE_PUSH = 4;
8
9 unsigned int ball_pos;
10 int direction;
11 unsigned int state_l;
12 unsigned int state_r;
13
14 unsigned int calculate_state(unsigned int state,
                             int sensor) {
15     switch (state) {
16         case STATE_PUSH:
17         case STATE_IDLE:
18             if (sensor== -1) {
19                 return STATE_PULL0;
20             } else if (sensor== +1) {
```



```
21     return STATE_PUSH;
22     }
23     return STATE_IDLE;
24 case STATE_PULL0:
25     if (sensor==-1) {
26         return STATE_PULL1;
27     } else if (sensor==+1) {
28         return STATE_PUSH;
29     }
30     return STATE_IDLE;
31 case STATE_PULL1:
32     if (sensor==-1) {
33         return STATE_PULL1;
34     }
35     return STATE_KICK;
36 case STATE_KICK:
37     return STATE_IDLE;
38 }
39 return state;
40 }
41
42 void setup() {
43     NIBObee.begin();
44     NIBObee.checkVoltage();
45     ball_pos = 3;
46     direction = -1;
47     state_l = STATE_IDLE;
48     state_r = STATE_IDLE;
49 }
50
51 void loop() {
52     delay(100);
53     state_l = calculate_state(state_l, FeelerL.get());
54     if (((state_l==STATE_PUSH) && (ball_pos==1)) ||
55         ((state_l==STATE_KICK) && (ball_pos==0)) ) {
56         direction = +1;
57     }
58     state_r = calculate_state(state_r, FeelerR.get());
59     if (((state_r==STATE_PUSH) && (ball_pos==2)) ||
60         ((state_r==STATE_KICK) && (ball_pos==3)) ) {
61         direction = -1;
62     }
63     if (direction==+1) {
64         if (ball_pos<3) {
65             ball_pos++;
66         } else {
67             direction = 0;
68         }
69     }
70     if (direction==-1) {
71         if (ball_pos>0) {
```

```
72     ball_pos--;
73   } else {
74     direction = 0;
75   }
76 }
77 NIBObee.setLed(LED0, ball_pos==0);
78 NIBObee.setLed(LED1, ball_pos==1);
79 NIBObee.setLed(LED2, ball_pos==2);
80 NIBObee.setLed(LED3, ball_pos==3);
81 }
```

### Erläuterungen zum Quellcode:

03 - 07 Das Programm beginnt mit der Definition der Konstanten für den **Zustandsautomaten**.

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 FeelerL.get() bzw. FeelerR.get() Funktionen.

14 - 40 Die Funktion calculate\_state berechnet die Reaktion des Automaten und bekommt den aktuellen Zustand und die Eingabe und liefert den zukünftigen Zustand zurück.

42 - 49 **setup-Funktion:**

43+44 Die Setup Funktion beginnt wie immer mit der Initialisierung und der Überprüfung der Spannung

- 45 Der „Ball“ wird an Position 3 gelegt...
- 46 ... und nach links auf die Reise geschickt
- 48+49 Beide Zustandsautomaten werden auf den Startzustand gesetzt.
- 51 - 81 **Loop-Funktion:**
- 52 Zuerst warten wir erst einmal 100 ms.
- 53 - 57 Nun 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.
- 58 - 62 Danach wird nach dem selben Schema die rechte Seite behandelt.
- 63 - 76 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.
- 77 - 80 Zum Abschluss wird die LED, die der aktuellen Position entspricht, eingeschaltet.

### ***Ideen & Anregungen:***

1. Ändern Sie das Programm so ab, dass der Ball mit PUSH nur zurückgespielt werden kann, wenn der Fühler nur an der 'gelben' Position gedrückt ist. Wenn man zu früh oder zu spät drückt, soll der Ball durchgelassen werden.
2. Zählen Sie die Punkte (blinken) wenn der Ball durchgelassen wurde.

## 10 Testen der Odometriesensoren

Als nächstes soll die Odometrie des NIBObee getestet werden.

Erzeugen Sie einen neuen Sketch „**Odometrie**“ und importieren Sie als Bibliothek wieder „NIBObee“.

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 den folgenden Quellcode in das Editorfenster ein:

```
TT06_Odometrie
1 #include <NIBObee.h>
2
3 void setup() {
4   NIBObee.begin();
5   NIBObee.checkVoltage();
6 }
7
8 void loop() {
9   if (FeelerL.get()) {
10    Engine.left.resetTicks();
11  }
12  if (FeelerR.get()) {
13    Engine.right.resetTicks();
14  }
15
16  NIBObee.setLed(LED0, Engine.left.getTicks()>10);
17  NIBObee.setLed(LED1, Engine.left.getTicks()>20);
18  NIBObee.setLed(LED2, Engine.right.getTicks()>20);
19  NIBObee.setLed(LED3, Engine.right.getTicks()>10);
20 }
```

### Erläuterungen zum Quellcode:

01 - 06

Das Programm beginnt mit dem Einbinden der NIBObee Bibliothek, der Initialisierung des Roboters und mit der Überprüfung der Akku-Spannung.

08 - 20

**loop-Funktion:**

09 - 14

Die beiden if-Anweisungen setzen jeweils den aktuellen Odometrieählerstand zurück wenn einer der beiden Fühler betätigt wird.

16 - 19

In den letzten vier Zeilen werden die LEDs in Abhängigkeit von den Zählerständen gesetzt.

Der Aufruf `Engine.left.getTicks()` liefert den aktuellen Zählerstand des linken Odometrieählers zurück ohne ihn (auf null) zurückzusetzen.

**Hinweis:**

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

**Ideen & Anregungen:**

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

## 11 Ansteuerung der Motoren

Als nächstes wollen wir die Motoren ansteuern.

Erzeugen Sie einen neuen Sketch „*Motor*“ und importieren Sie als Bibliothek wieder „NIBObee“.

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 den folgenden Quellcode in das Editorfenster ein:

```
TT07_Motor
1 #include <NIBObee.h>
2
3 void setup() {
4     NIBObee.begin();
5     NIBObee.checkVoltage();
6 }
7
8 void loop() {
9     int speed_l = 0;
10    int speed_r = 0;
11    switch (FeelerL.get()) {
12        case 1: speed_l = 800; break;
13        case 0: speed_l = 0; break;
14        case -1: speed_l = -800; break;
15    }
16    switch (FeelerR.get()) {
17        case 1: speed_r = 800; break;
18        case 0: speed_r = 0; break;
19        case -1: speed_r = -800; break;
20    }
21    Engine.setPWM(speed_l, speed_r);
22 }
```

### Erläuterungen zum Quellcode:

01-06 Das Programm beginnt wieder mit dem Einbinden der NIBObee Bibliothek, der Initialisierung des Roboters und mit der Überprüfung der Akku-Spannung.

**08 - 22 Loop-Funktion:**

**09 - 10** Es werden zwei lokale Variablen `speed_l` und `speed_r` deklariert, in denen der Sollwert für die Motoren gespeichert wird. Wir setzen die beiden Variablen zunächst auf 0.

**11 - 20** Die Sollwerte für die Motoren werden in den folgenden zwei switch-Anweisungen in Abhängigkeit von den Fühlern gesetzt.

**21** In der letzten Zeile der Loop-Funktion werden die beiden Sollwerte den Motoren zugewiesen.

**Hinweis:**

Die Angabe der Werte erfolgt in 1/1024 Schritten:

- +1024 bedeutet 100% vorwärts,
- + 512 bedeutet 50% vorwärts,
- 512 bedeutet 50% rückwärts.

Die Ansteuerung erfolgt mittels Pulsweitenmodulation (PWM).

**Ideen & Anregungen:**

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

## 12 Hindernisdetektion

Nun soll der NIBObee lernen, Hindernisse zu erkennen.

Erzeugen Sie einen neuen Sketch „*Hindernis*“ und importieren Sie als Bibliothek wieder „NIBObee“.

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

Um das zu erreichen verwenden wir wieder einen Zustandsautomaten, der aktuelle Zustand entspricht dabei dem „Verhalten“ des Roboters.

Tippen Sie den folgenden Quellcode in das Editorfenster ein:

```
1 #include <NIBObee.h>
2
3 const unsigned int MODE_STOP = 0;
4 const unsigned int MODE_DRIVE = 1;
5 const unsigned int MODE_BACK = 2;
6 const unsigned int MODE_STEER_R = 3;
7 const unsigned int MODE_STEER_L = 4;
8 const unsigned int MODE_AVOID_R = 5;
9 const unsigned int MODE_AVOID_L = 6;
10
11 unsigned int mode;
12 int speed_l;
13 int speed_r;
14 unsigned int counter_ms;
15
16 unsigned int perform_check(unsigned int mode);
17 unsigned int do_stop();
18 unsigned int do_drive();
19 unsigned int do_back();
20 unsigned int do_steer_r();
21 unsigned int do_steer_l();
22 unsigned int do_avoid_r();
23 unsigned int do_avoid_l();
24
25
26 void setup() {
27     NIBObee.begin();
28     NIBObee.checkVoltage();
29     mode = MODE_STOP;
30 }
31
32 void loop() {
33     delay(1);
```



```
34 mode = perform_check(mode);
35 switch (mode) {
36   case MODE_STOP:
37     mode = do_stop();
38     break;
39   case MODE_DRIVE:
40     mode = do_drive();
41     break;
42   case MODE_BACK:
43     mode = do_back();
44     break;
45   case MODE_STEER_R:
46     mode = do_steer_r();
47     break;
48   case MODE_STEER_L:
49     mode = do_steer_l();
50     break;
51   case MODE_AVOID_R:
52     mode = do_avoid_r();
53     break;
54   case MODE_AVOID_L:
55     mode = do_avoid_l();
56     break;
57 }
58 switch (mode) {
59   case MODE_STOP:
60     speed_l = 0;
61     speed_r = 0;
62     break;
63   case MODE_DRIVE:
64     speed_l = 500;
65     speed_r = 500;
66     break;
67   case MODE_BACK:
68     speed_l = -500;
69     speed_r = -500;
70     break;
71   case MODE_STEER_R:
72     speed_l = 600;
73     speed_r = 400;
74     break;
75   case MODE_STEER_L:
76     speed_l = 400;
77     speed_r = 600;
78     break;
79   case MODE_AVOID_R:
80     speed_l = -400;
81     speed_r = -600;
82     break;
83   case MODE_AVOID_L:
84     speed_l = -600;
```

```
85     speed_r = -400;
86     break;
87 }
88 Engine.setSpeed(speed_l, speed_r);
89 }
90
91 unsigned int perform_check(unsigned int mode) {
92     if (FeelerL.get() && FeelerR.get()) {
93         if ((FeelerL.get() == -1) && (FeelerR.get() == -1)) {
94             mode = MODE_BACK;
95         }
96         else {
97             mode = MODE_STOP;
98         }
99     }
100     return mode;
101 }
102
103 unsigned int do_stop() {
104     if ((FeelerL.get() == 0) && (FeelerR.get() == 0)) {
105         return MODE_DRIVE;
106     }
107     return MODE_STOP;
108 }
109
110 unsigned int do_back() {
111     if (FeelerL.get() == 0) {
112         return MODE_AVOID_L;
113     }
114     if (FeelerR.get() == 0) {
115         return MODE_AVOID_R;
116     }
117     return MODE_BACK;
118 }
119
120 unsigned int do_drive() {
121     if (FeelerR.get() == 1) {
122         return MODE_STEER_L;
123     }
124     if (FeelerL.get() == 1) {
125         return MODE_STEER_R;
126     }
127     if (FeelerR.get() == -1) {
128         return MODE_AVOID_L;
129     }
130     if (FeelerL.get() == -1) {
131         return MODE_AVOID_R;
132     }
133     return MODE_DRIVE;
134 }
135
```

```
136 unsigned int do_steer_r() {
137   if (FeelerL.get()==0) {
138     return MODE_DRIVE;
139   }
140   return MODE_STEER_R;
141 }
142
143 unsigned int do_steer_l() {
144   if (FeelerR.get()==0) {
145     return MODE_DRIVE;
146   }
147   return MODE_STEER_L;
148 }
149
150 unsigned int do_avoid_r() {
151   if (counter_ms==0) {
152     counter_ms=1000;
153   }
154   else {
155     counter_ms--;
156   }
157   if (counter_ms) {
158     return MODE_AVOID_R;
159   }
160   else {
161     return MODE_DRIVE;
162   }
163 }
164
165 unsigned int do_avoid_l() {
166   if (counter_ms==0) {
167     counter_ms=1000;
168   }
169   else {
170     counter_ms--;
171   }
172   if (counter_ms) {
173     return MODE_AVOID_L;
174   }
175   else {
176     return MODE_DRIVE;
177   }
178 }
```

## Erläuterungen zum Quellcode:

### 01 - 23 **Deklarationen**

03 - 09 Konstanten für die Zustände des Automaten

11 Zustandsvariable für den Automaten

12+13 Geschwindigkeit der Räder

14 Zähler für Millisekunden

### 26 - 30 **setup-Funktion:**

27+28 Die Setup Funktion beginnt wie immer mit der Initialisierung und der Überprüfung der Spannung

29 Der Automat wird auf seinen Startzustand gesetzt

### 32 - 178 **loop-Funktion:**

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

34 Die Funktion `perform_check()` reagiert (reflexartig) 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.

37 - 55

Die Funktionen `do_xxx()` reagieren abhängig vom aktuellen Zustand 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.

***Ideen & Anregungen:***

1. Zeichne ein Diagramm, das eine Übersicht über die Zustände zeigt.
2. Verbessere das Programm und erweitere es um zusätzliche Verhalten!

## 13 Liniensensoren

In diesem Beispiel wollen wir uns mit den Liniensensoren des NIBObee 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. Die Helligkeit des Bodens wird automatisch zweimal gemessen, einmal bei eingeschalteter und einmal mit ausgeschalteter IR-LED. Dadurch lassen sich die Einflüsse des Umgebungslichts minimieren.

Die gemessenen Werte werden von der Bibliothek normalisiert und stehen als Ganzzahlen im Bereich von **0 (schwarz)** bis **1024 (weiß)** zur Verfügung.

Vor der Verwendung müssen die Sensoren kalibriert werden. Die Parameter der Kalibrierung werden im EEPROM dauerhaft gespeichert. Bei einer Neuprogrammierung des ATmega16 bleiben die Kalibrierwerte erhalten!

### 13.1 Kalibrierung der Liniensensoren

Um die Liniensensoren des NIBObee zu kalibrieren, öffnen Sie den Sketch **Kalibrierung.ino**, den sie im Menü unter „Datei“->„Beispiele“->„NIBObee“->„Kalibrierung“ finden, und übertragen das Programm auf den Roboter.

Nach abgeschlossener Übertragung können Sie die Sensoren wie folgt kalibrieren:

Man platziert den NIBObee mit allen Bodensensoren auf einer **weißen Fläche** und drückt den **linken Fühler nach hinten**. Danach stellt man den NIBObee auf eine **schwarze Fläche** und drückt den **rechten Fühler nach hinten**. Beide Aktionen werden mit LED blinken quittiert.

Die so gemessenen Werte werden permanent im EEPROM **gespeichert**, wenn man danach **beide Fühler nach vorne** drückt.

### 13.2 Arbeiten mit den Liniensensoren

Nun können wir die Liniensensoren verwenden:

Das Testprogramm zeigt die Werte des linken und des rechten Liniensensors

mit Hilfe der LEDs an. 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 einen neuen Sketch an, und speichern diesen unter dem Namen „**Liniensensor**“. Importieren Sie als Bibliothek wieder „NIBObee“ und ergänzen Sie den Quellcode im Editorfenster:



```
1 #include <NIBObee.h>
2
3 void setup() {
4   NIBObee.begin();
5   NIBObee.checkVoltage();
6 }
7
8 void loop() {
9   NIBObee.setLed(LED0, LineSensor.getL()>160);
10  NIBObee.setLed(LED1, LineSensor.getL()>240);
11  NIBObee.setLed(LED3, LineSensor.getR()>160);
12  NIBObee.setLed(LED2, LineSensor.getR()>240);
13 }
```

### Erläuterungen zum Quellcode:

01 - 06 Das Programm beginnt wieder mit dem Einbinden der NIBObee Bibliothek, der Initialisierung des Roboters und mit der Überprüfung der Akku-Spannung.

#### 08 - 13 **Loop-Funktion:**

09 - 12 Nun wird mit den Funktionen „LineSensor.getR()“ und „LineSensor.getL()“ gearbeitet. Diese Funktionen bekommen 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

09 - 12 erkennen) und mit der Zahl 240 (um größere Abstände zu erkennen) verglichen. Ergibt dieser Vergleich „wahr“, so ist der zweite Parameter der Funktion „NIBObee . setLed“ 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.

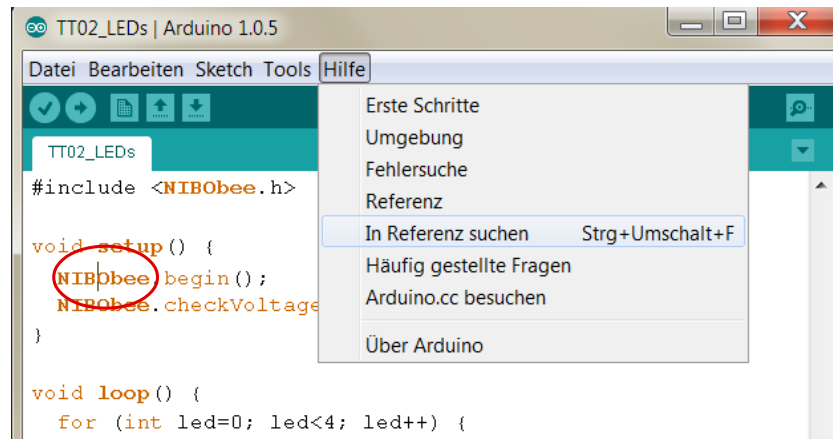
#### ***Ideen & Anregungen:***

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



## 14 Dokumentation

Eine Übersicht über die NIBObee ARDUINO Bibliothek kann folgendermaßen geöffnet werden: Klicken Sie im Editor in das Wort NIBObee (siehe Bild) und wählen Sie aus dem Menü „Hilfe“ -> „In Referenz suchen“ aus:



Nun öffnet sich in Ihrem Webbrowser folgendes Fenster:

### 📖 NIBObee robot class reference for ARDUINO

NIBObee · ARDUINO-Referenz

Einbindung durch: #include <NIBObee.h>

- class instance NIBObee

Funktion	Beschreibung
void begin ()	Initialisierung des NIBObee Roboters
unsigned int getVoltage ()	Versorgungsspannung messen. return-value: Spannung in Millivolt (4.8V ± 4800)
void checkVoltage ()	Versorgungsspannung überprüfen, im Fehlerfall anhalten und blinken
void setLed (int led, int value)	LED ein/ausschalten: led: LED0=0, LED1=1, LED2=2, LED3=3 value: OFF=0, ON=1
int getLed (int led)	LED abfragen: led: LED0=0, LED1=1, LED2=2, LED3=3 return-value: OFF=0, ON=1
unsigned int getRandomSeed ()	Zufallszahlen-Basis liefern
unsigned int getAnalog (unsigned char adc_channel, unsigned char active)	Rohwert eines analogen Kanals auslesen

- class instance Engine

Funktion	Beschreibung
void begin ()	Initialisierung
void setPWM (int left, int right)	PWM Werte (-1023 ... 0 ... +1023) für die beiden Motoren setzen
void setSpeed (int left, int right)	Geschwindigkeit für die beiden Motoren setzen. Die Werte werden in Ticks/Sekunde angegeben. 40 Ticks entsprechen einer Radumdrehung (116 mm Strecke)
void setTargetAbs (int left, int right, unsigned int speed)	Zielposition für Räder setzen. Die Werte werden in Ticks angegeben. 40 Ticks entsprechen einer Radumdrehung (116 mm Strecke)

## 15 Links zu weiterführenden Internetseiten

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

### Entwicklungsumgebungen:



**Arduino:** <http://www.arduino.cc>

Webseite der Arduino-Entwicklungsumgebung. Dort gibt es die Open Source Arduino Oberfläche für verschiedene Betriebssysteme zum Download.



**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

**AVRDude:** <http://savannah.nongnu.org/projects/avrdude/>

Freie Programmiersoftware (Downloader, für den Nibo geeignet!).



**Roboter.CC:** <http://www.roboter.cc>

Online Code Compiler speziell für Robotik-Projekte mit vielen Beispielen und Forum.

### Weitere Informationen:

- ➔ **Nibo Hauptseite:** <http://nibo.nicai-systems.de>  
Die Homepage des Nibo Herstellers. Liefert technische Informationen, die Bauanleitung und weitere Links.
- ➔ **Nibo Wiki:** <http://www.nibo-roboter.de>  
Wiki des Nibo. Liefert alle Informationen rund um den Nibo.
- ➔ **Mikrocontroller:** <http://www.mikrocontroller.net>  
Alles über Mikrocontroller und deren Programmierung.
- ➔ **AVRFreaks:** <http://www.avrfreaks.net>  
Informationen rund um den AVR.