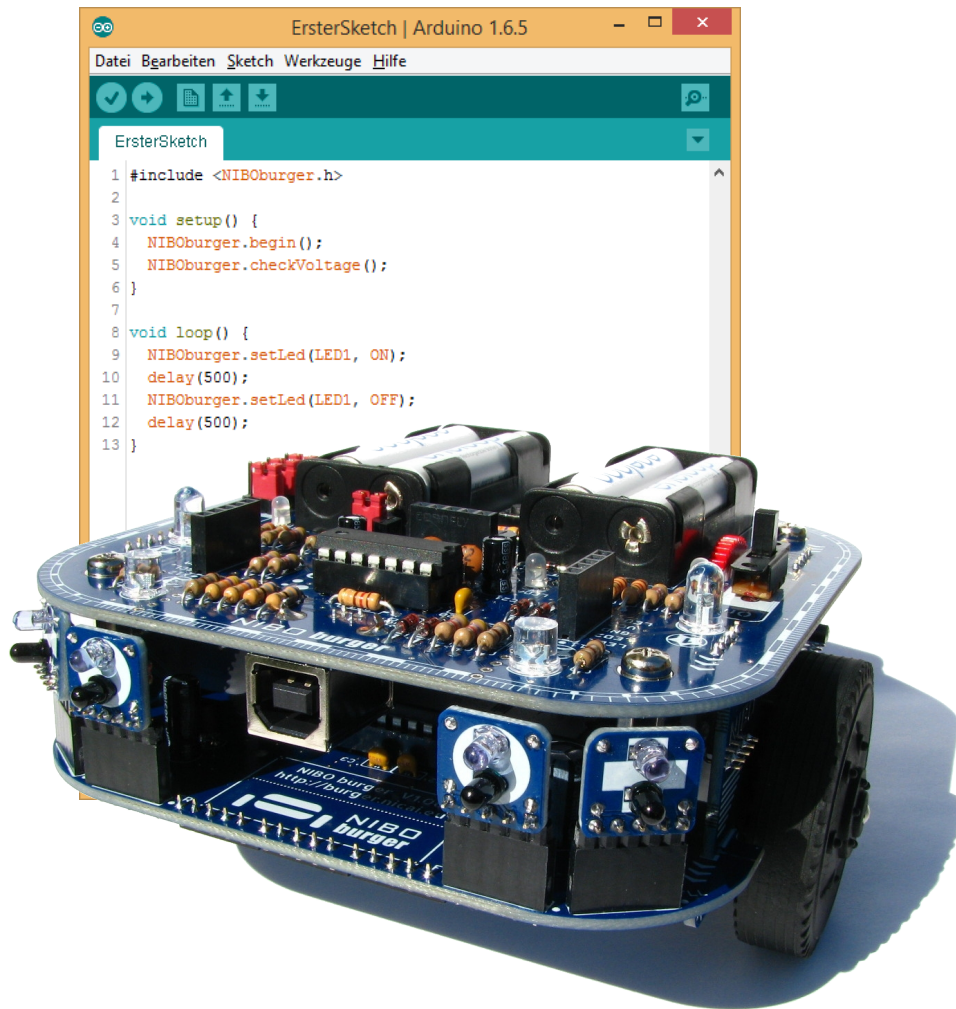


Robot Kit NIBO burger

ARDUINO 1.6.6 Programmier-Tutorial



Inhaltsverzeichnis

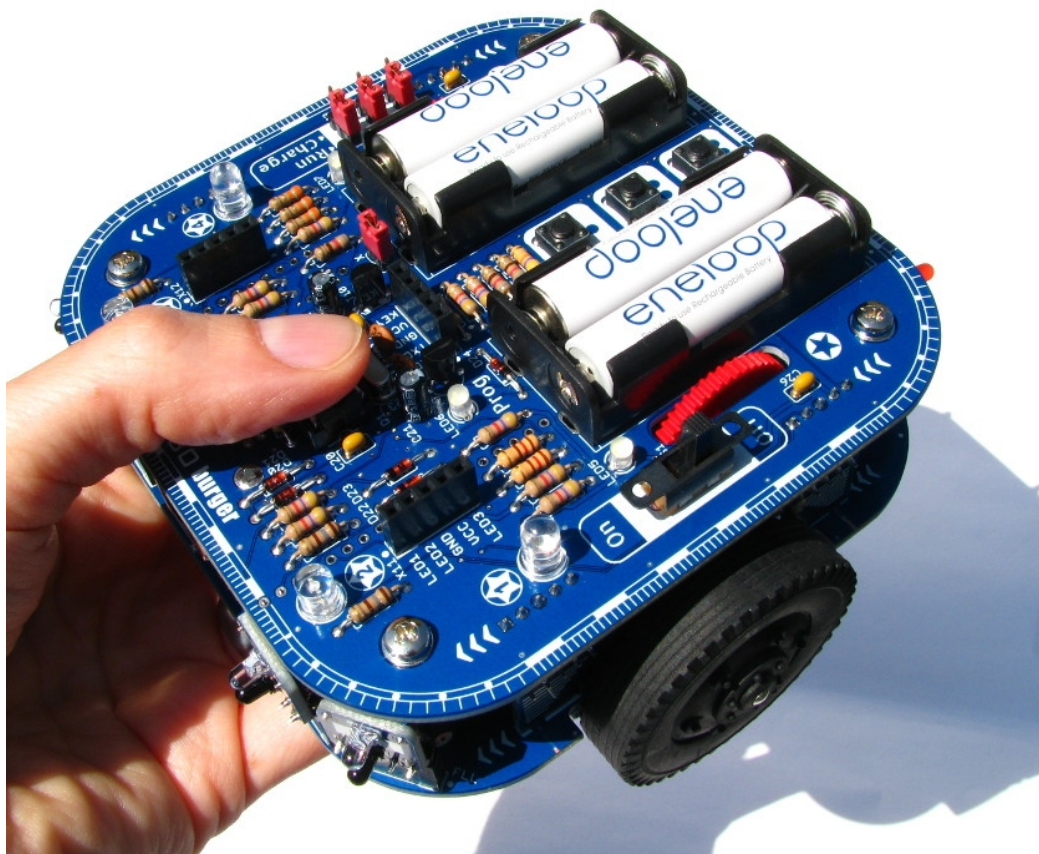
1 Einleitung.....	3
2 Installation der Programmierumgebung.....	4
2.1 ARDUINO.....	4
3 Installation der NiboRoboLib.....	6
4 Vorbereitungen.....	7
5 Der erste Sketch.....	10
6 LEDs.....	15
7 LEDs Action.....	17
8 LEDs Zufall.....	19
9 Taster.....	21
10 Reaktion.....	24
11 Testen der Odometriesensoren.....	28
12 Ansteuerung der Motoren.....	30
13 Abfrage der Sensoren.....	32
14 Hindernisdetektion.....	34
15 Kalibrierung des Farbsensors.....	37
16 Arbeiten mit dem Farbsensor.....	39
17 Dokumentation.....	41
18 Links zu weiterführenden Internetseiten.....	42

1 Einleitung

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

Man lernt in den folgenden Abschnitten wie die Programmierumgebung installiert wird, wie die LEDs zum Blinken/Leuchten gebracht werden können, wie die Farbsensoren, die IR-Sensoren und die Motoren angesteuert werden und letztendlich auch, wie Du den NIBO burger in Bewegung bringst!

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



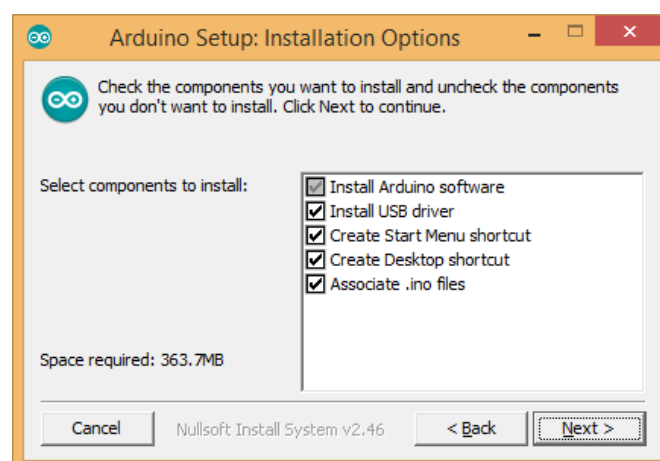
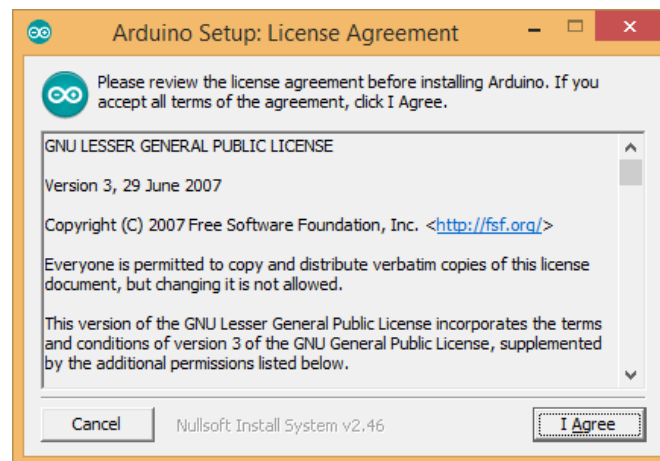
2 Installation der Programmierumgebung

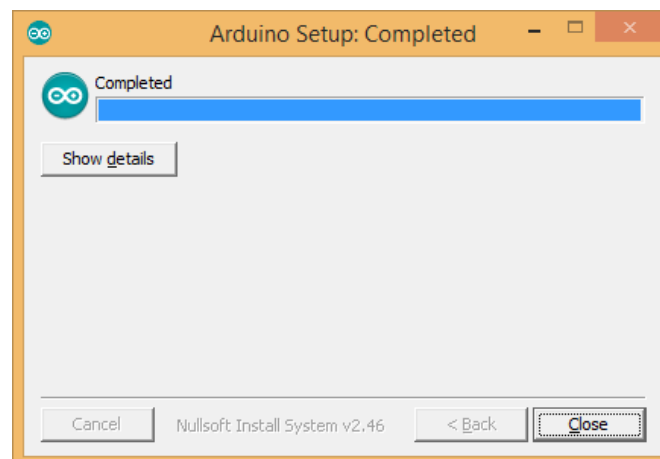
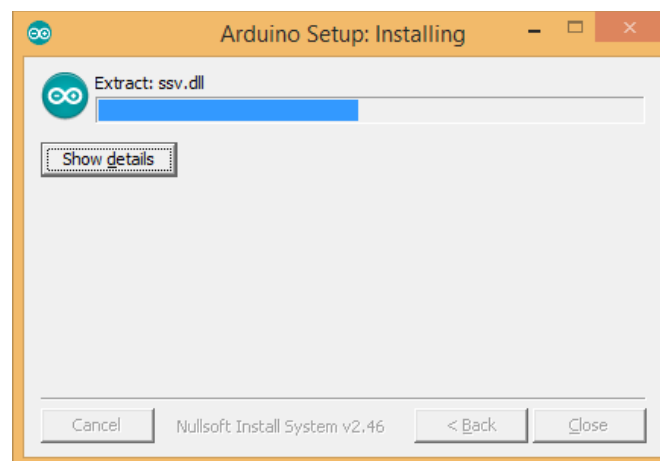
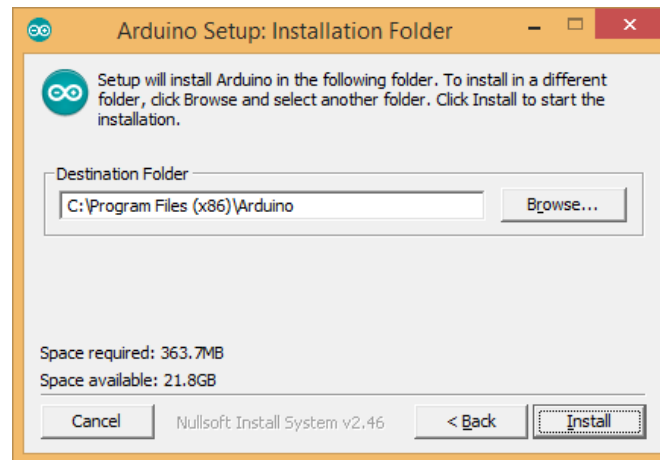
2.1 ARDUINO

Zunächst muss die ARDUINO Umgebung installiert werden. Lade hierzu die Installationsdatei **Arduino 1.6.6** von der Webseite <http://arduino.cc> für Dein entsprechendes Betriebssystem herunter, für Windows sollte man die „Installer-Version“ verwenden.

Im Folgenden wird die Installation für Windows beschrieben:

Doppelklicke nun die heruntergeladene Datei **arduino-1.6.6-windows.exe** und installiere Arduino wie vom Installer vorgeschlagen auf dem Computer.





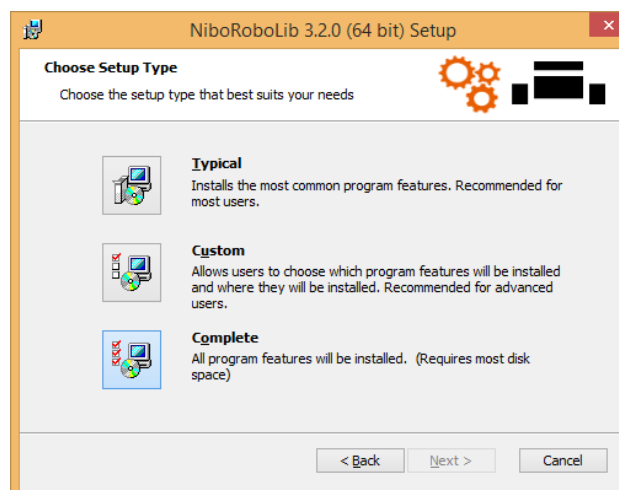
3 Installation der NiboRoboLib

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



Alternativ befindet sich die Dateien auch auf der CD zum Roboter.

Die einfachste Methode alle benötigten Bestandteile zu installieren ist die Komplettinstallation:

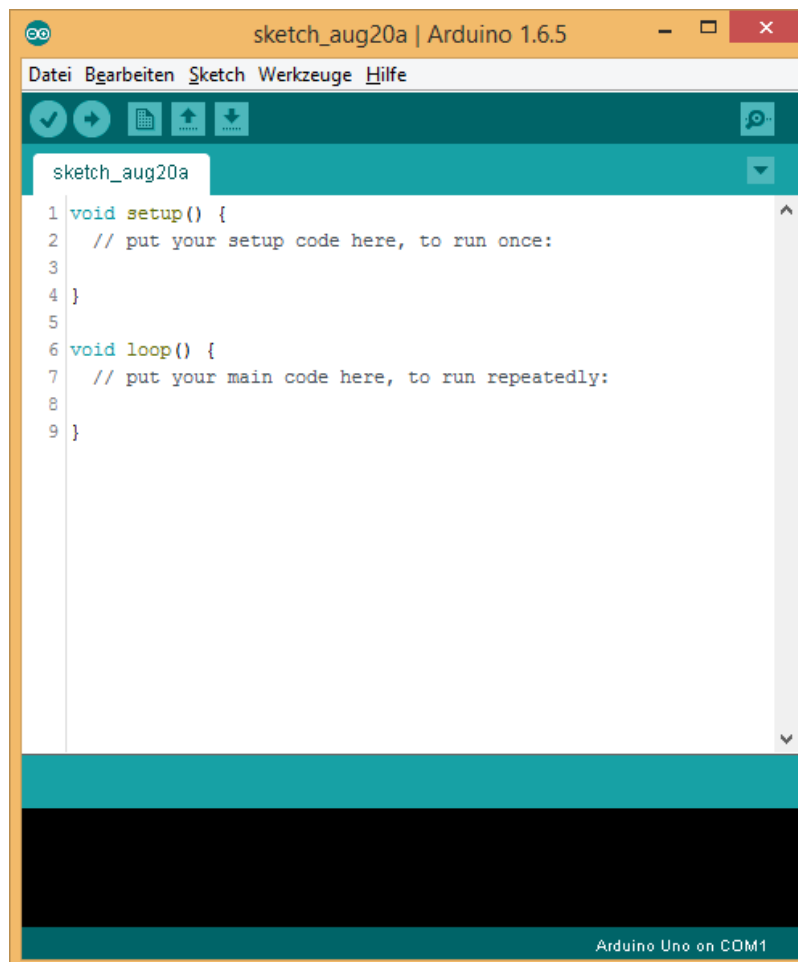
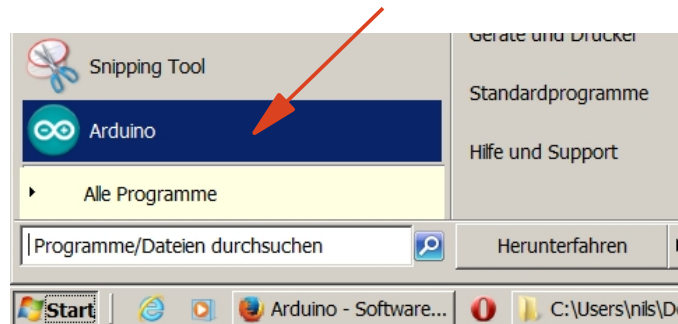


Die **NiboRoboLib** enthält:

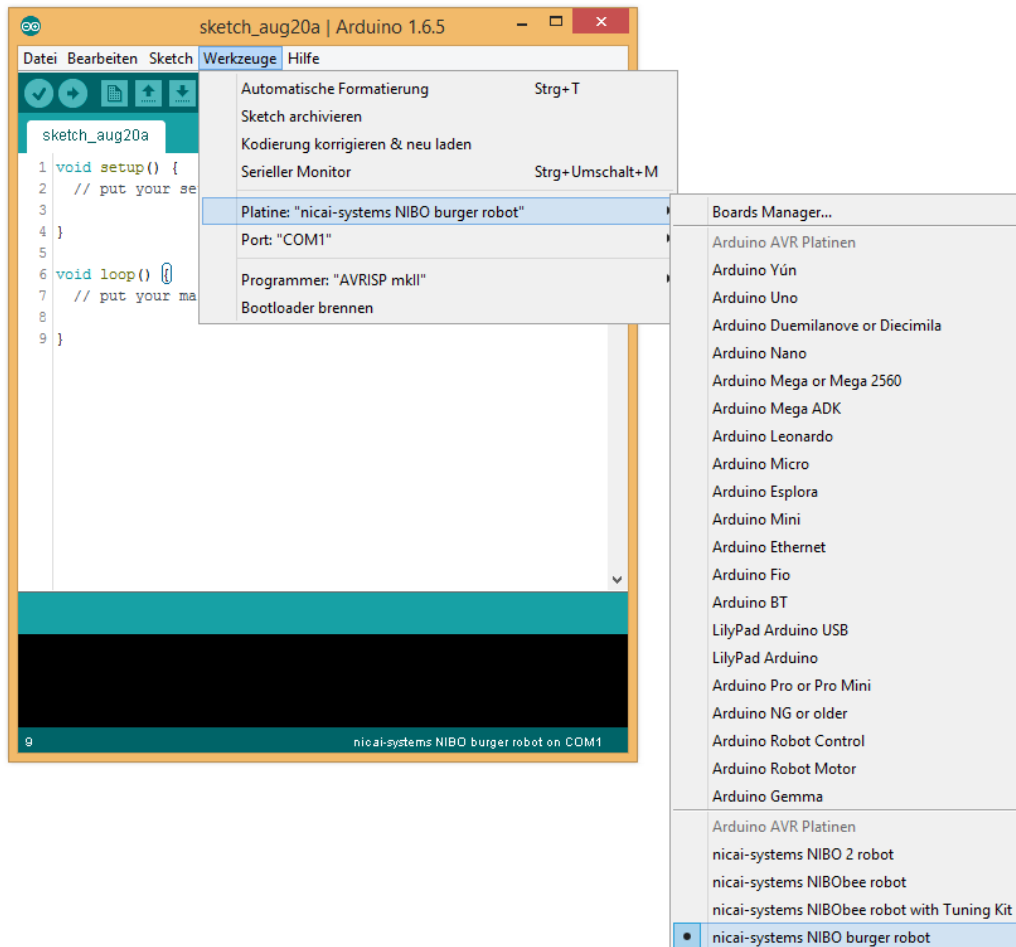
- + Alle benötigten **Treiber** für den UCOM-IR2-X, den NIBObee und den NIBO burger
- + **RoboDude** (Übertragungsprogramm für .hex- und .xhex-Dateien)
- + **C-Bibliotheken, Test- und Kalibrierprogramme** für NIBO 2, NIBO burger und NIBObee
- + **ARDUINO**-Bibliotheken für NIBO burger, NIBObee und NIBO 2

4 Vorbereitungen

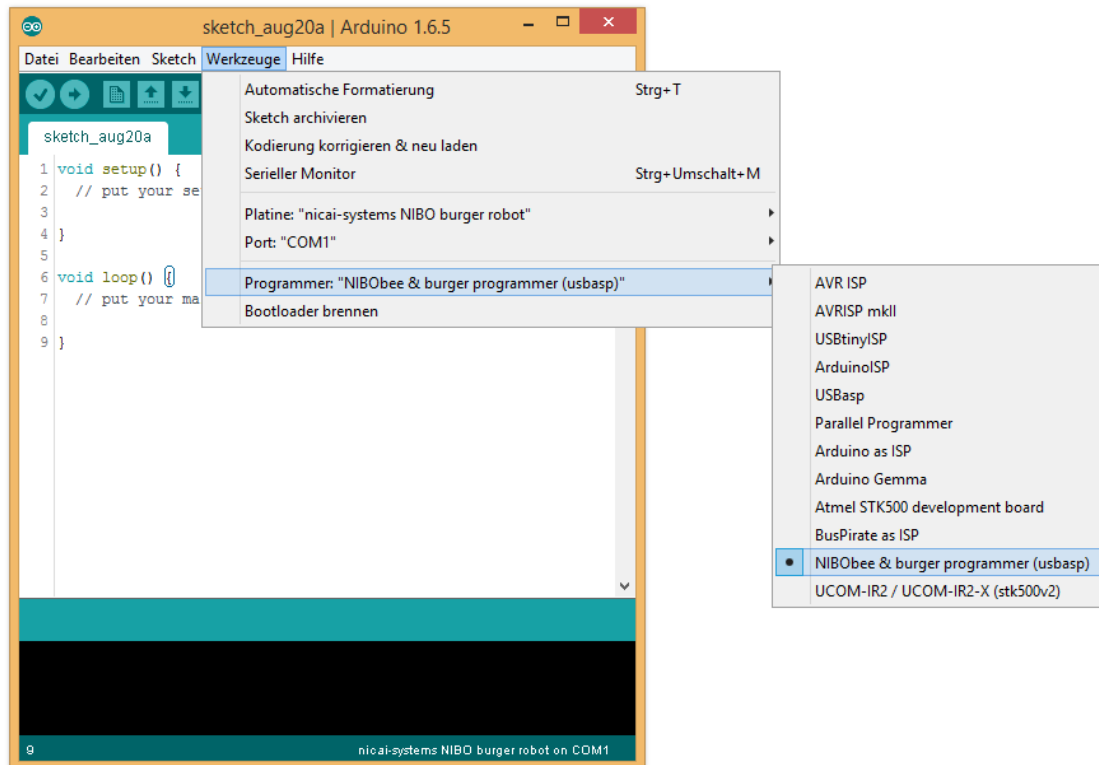
Starte die ARDUINO-Oberfläche:



Über **Werkzeuge – Platine – nicaï-systems NIBO burger robot** wird der NIBO burger als Platine ausgewählt:

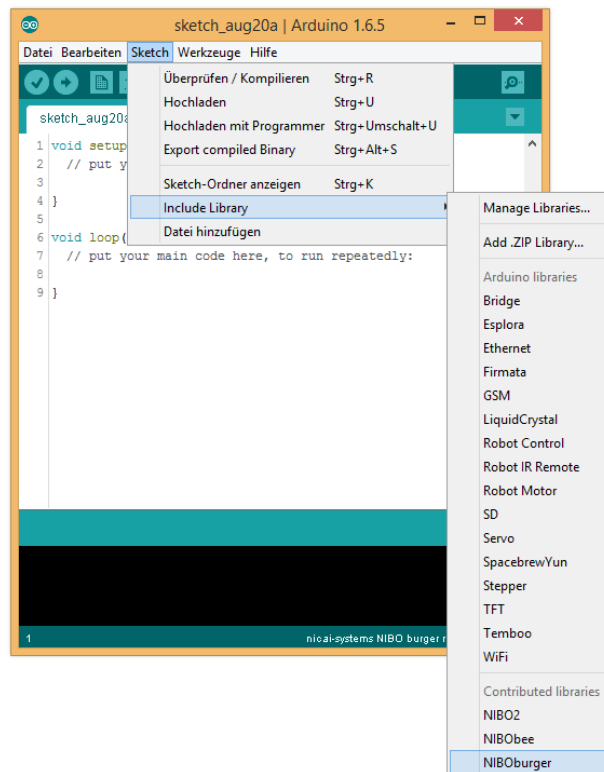


Über **Werkzeuge – Programmer – NIBObee & burger programmer** wird der integrierte Programmer des NIBO burger als Programmer ausgewählt:



5 Der erste Sketch

Zunächst wird die „NIBOburger“ Library importiert:



Durch das Importieren wird automatisch der folgende Quelltext erzeugt:

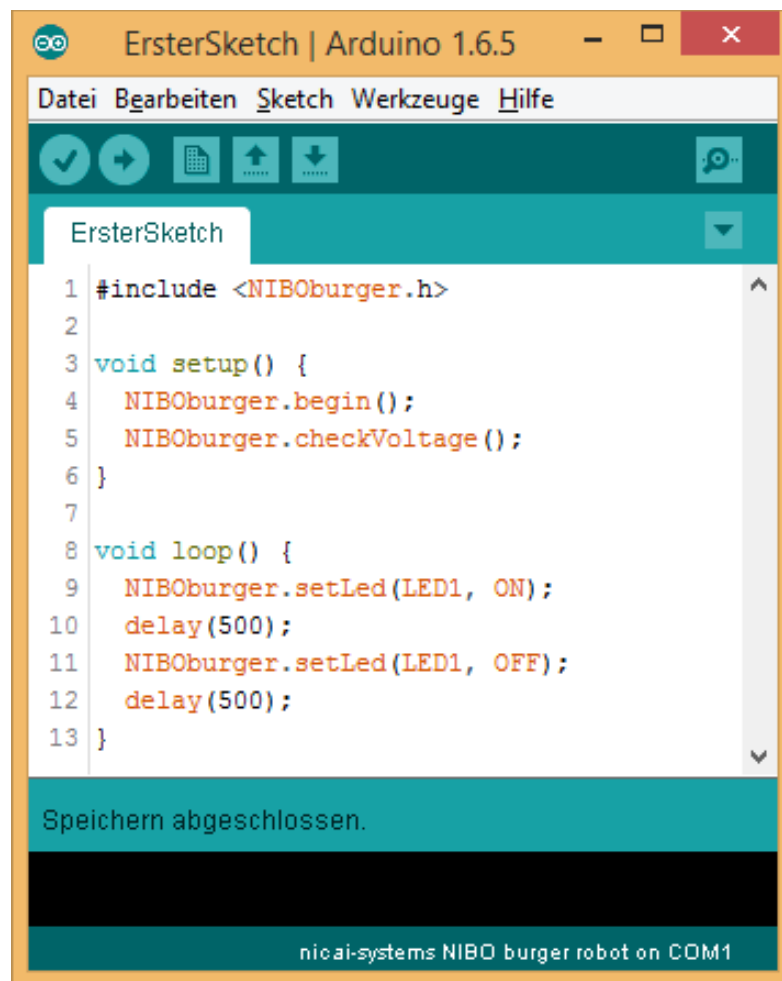
```

1 #include <NIBOburger.h>
2

```

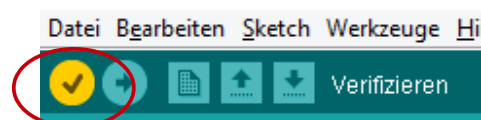
Speicher den Sketch unter dem Namen „*ErsterSketch*“.

Ergänze den entstandenen Quellcode wie folgt:



Speicher das geänderte Programm erneut ab!

Jetzt wird das Programm überprüft, indem Du den **Verifizieren-Knopf** drückst:



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*.

Schalte jetzt den NIBO burger ein und schließe ihn an den Rechner an.

Jetzt kannst Du den ersten Sketch auf den Roboter übertragen.
Drücke dazu den **Hochladen-Knopf**:



Wenn jetzt die linke rote LED (*LED1*) blinkt, hat alles bestens geklappt!

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 Programmlaufs
// genau einmal am Anfang ausgeführt.
}

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

Zurück zu unserem ersten Sketch:



```
ErsterSketch
1 #include <NIBOburger.h>
2
3 void setup() {
4   NIBOburger.begin();
5   NIBOburger.checkVoltage();
6 }
7
8 void loop() {
9   NIBOburger.setLed(LED1, ON);
10  delay(500);
11  NIBOburger.setLed(LED1, OFF);
12  delay(500);
13 }
```

01 In der ersten Zeile wird durch die `#include` Anweisung die Header-Datei der NIBO burger 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 `„NIBOburger.begin();“` wird der Roboter initialisiert.

Wichtig: Diese Anweisung **muss** in jeder setup-Funktion als erstes ausgeführt werden!

Die Anweisung `„NIBOburger.checkVoltage();“` überprüft die Spannung der Akkus direkt nach dem Einschalten. Sollte die Spannung nicht OK sein (Akkus leer) blinkt der NIBO burger 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 `„NIBOburger.setLed(LED1, ON);“` die LED 1 (die rote 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 1 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“ → „NIBO burger“ → „Tutorial“ zu finden.

6 LEDs


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

Erzeuge einen neuen Sketch, indem Du im Menü „Datei“ → „Neu“ auswählst.
Speicher den neuen Sketch unter dem Namen „LEDs“.

Als Bibliothek importierst Du wieder „NIBOburger“ wie in Kapitel 5 beschrieben.

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

Tippe den folgenden Quellcode in das Editorfenster ein:



```
1 #include <NIBOburger.h>
2
3 void setup() {
4   NIBOburger.begin();
5   NIBOburger.checkVoltage();
6 }
7
8 void loop() {
9   for (int led=1; led<5; led++) {
10    NIBOburger.setLed(led, ON);
11    delay(350);
12    NIBOburger.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=1; led<5; 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 1, und bei den folgenden Durchläufen die Werte 2, 3 und 4.

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 1 initialisiert. Die Bedingung für einen Durchlauf ist, dass der Wert der Variablen led kleiner als 5 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 1, 2, 3 und 4 annimmt.

10 - 13 Im Anweisungsblock der for-Schleife stehen die eigentlichen Anweisungen:
Mit der Anweisung „**NIBOburger.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 „**NIBOburger.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.

Erzeuge einen neuen Sketch, indem Du im Menü „**Datei**“ → „**Neu**“ auswählst. Speicher den Sketch unter dem Namen „**LEDsAction**“.

Als Bibliothek importierst Du wieder „NIBOburger“ 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.

Tippe den folgenden Quellcode in das Editorfenster ein:



```
1 #include <NIBOburger.h>
2
3 void setup() {
4   NIBOburger.begin();
5   NIBOburger.checkVoltage();
6 }
7
8 void loop() {
9   for (int time=50; time<800; time*=2) {
10     for (int led=1; led<5; led++) {
11       NIBOburger.setLed(led, ON);
12       delay(time);
13       NIBOburger.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 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 1 bis 4 an.

11 - 14

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

Mit der Anweisung „`NIBOburger.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 „`NIBOburger.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. Verändere Dein Programm so, dass sich die zeitliche Veränderung umkehrt, die Geschwindigkeit soll sich jetzt von Runde zu Runde erhöhen.
2. Verändere Dein Programm so, dass sich die Laufrichtung umkehrt, sprich die Diode LED1 soll in jedem Durchlauf als letzte LED aufleuchten.

8 LEDs Zufall

Jetzt werden wir ein zufälliges Leuchtmuster generieren...

Erzeuge dazu einen neuen Sketch, indem Du im Menü „Datei“ → „Neu“ auswählst. Speicher den Sketch unter dem Namen „LEDsZufall“.

Als Bibliothek importierst Du wieder „NIBOburger“ wie in Kapitel 5 beschrieben.

Für das zufällige Blinken wird bei jedem Durchlauf eine LED eingeschaltet und eine LED ausgeschaltet.

Tippe den folgenden Quellcode in das Editorfenster ein:

The image shows a screenshot of the Arduino IDE's code editor window. The window title is 'LEDsZufall'. The code is as follows:

```
1 #include <NIBOburger.h>
2
3 void setup() {
4   NIBOburger.begin();
5   NIBOburger.checkVoltage();
6   delay(100);
7   NIBOburger.randomize();
8 }
9
10 void loop() {
11   int led = NIBOburger.getRandomInt(1,4);
12   NIBOburger.setLed(led, ON);
13   delay(100);
14   led = NIBOburger.getRandomInt(1,4);
15   NIBOburger.setLed(led, OFF);
16   delay(100);
17 }
```

Erläuterungen zum Quellcode:

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

07 Mit der Anweisung „**NIBOburger.randomize()**;“ wird der Zufallszahlengenerator initialisiert. Dazu werden die zuletzt gemessenen Sensordaten (Sensor-Bricks, Versorgungsspannung und Spannung der Key-Tasten) als Startwert des Zufallszahlen-generators verwendet.

10 - 17 In jedem Durchlauf wird eine zufällige LED eingeschaltet und pausiert, und danach eine zufällige LED ausgeschaltet und pausiert.

11 Der Aufruf von „**NIBOburger.getRandomInt**(1, 4)“ liefert eine zufällige Zahl zwischen 1 und 4 zurück.

Ideen & Anregungen:

1. Verändere Dein Programm so, dass die Wartezeiten für die Pausen auch zufällig ausgewählt werden.
2. Verändere Dein Programm so, dass immer die LED ausgeschaltet wird, die vorher eingeschaltet wurde.

9 Taster

Als nächstes wollen wir die Taster abfragen. Dazu legen wir wieder einen neuen Sketch an, den wir diesmal unter dem Namen „**Taster**“ speichern. Als Bibliothek importieren wir wieder „NIBOburger“.

Tippe den folgenden Quellcode in das Editorfenster ein:



```
1 #include <NIBOburger.h>
2
3 void setup() {
4     NIBOburger.begin();
5     NIBOburger.checkVoltage();
6 }
7
8 void loop() {
9     char key = NIBOburger.getKeyChar();
10
11     switch (key) {
12         case 'A':
13             NIBOburger.setLed(LED1, ON);
14             break;
15         case 'B':
16             NIBOburger.setLed(LED2, ON);
17             break;
18         case 'C':
19             NIBOburger.setLed(LED3, ON);
20             break;
21         case 'a':
22             NIBOburger.setLed(LED1, OFF);
23             break;
24         case 'b':
25             NIBOburger.setLed(LED2, OFF);
26             break;
27         case 'c':
28             NIBOburger.setLed(LED3, OFF);
29             break;
30     }
31 }
```

Unser Programm soll folgendes machen:

Wird Taster 1 gedrückt soll die LED 1 leuchten.

Wird Taster 2 gedrückt soll die LED 2 leuchten.

Wird Taster 3 gedrückt soll die LED 3 leuchten.

Ein einzelner Tastendruck liefert zwei Events: Zuerst einen Großbuchstaben A-C für das Runterdrücken, und danach einen Kleinbuchstaben a-c für das Loslassen der Taste.

Erläuterungen zum Quellcode:

01 - 06 Dieser Quellcode unterscheidet sich nur im Anweisungsblock der loop-Funktion den vorherigen Programmen.

09 **loop-Funktion:**
In dieser Zeile definieren wir die Variable „key“. Dort speichern wir die letzte Keyboard-Aktion, die uns der Aufruf von „**NIBOburger.getKeyChar()**“ zurückliefert. Der Aufruf liefert uns einen der folgenden Werte zurück:
0: seit dem letzten Aufruf hat sich nichts geändert.
'A': Taste 1 wurde runtergedrückt.
'a': Taste 1 wurde losgelassen.
'B': Taste 2 wurde runtergedrückt.
'b': Taste 2 wurde losgelassen.
'C': Taste 3 wurde runtergedrückt.
'c': Taste 3 wurde losgelassen.

11 - 30 Die **switch**-Anweisung ab Zeile 10 ermöglicht eine Fallunterscheidung. Je nachdem welchen Wert die Variable „key“ 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 „key“ aus:

12 - 14 Hat „key“ den Wert 'A', dann wird die LED 1 eingeschaltet.

15 - 20	Entsprechend für ' B ' und ' C '...
21 - 23	Hat „key“ den Wert ' a ', dann wird die LED 1 ausgeschaltet.
24 - 29	Entsprechend für ' b ' und ' c '...













































Ideen & Anregungen:

1. Ändere Dein Programm so ab, dass die jeweilige LED durch ein erstes drücken der Taste eingeschaltet wird, und durch eine zweite Betätigung wieder ausgeschaltet wird.

10 Reaktion

In diesem Beispiel werden wir ein kleines „Spiel“ schreiben: Das Programm soll ein Reaktionstest sein und anzeigen wie schnell ein Mensch reagiert.

Zur Anzeige der verstrichenen Reaktionszeit werden die vier LEDs nach dem folgendem Schema verwendet:

0					
1					50 ms
2					100 ms
3					150 ms
4					200 ms
5					250 ms
6					300 ms
7					350 ms
8					400 ms
9					450 ms
10					

Gestartet wird durch einen Tastendruck auf Taster 1 – danach vergeht eine zufällige Zeit zwischen einer und vier Sekunden.

Sobald die LEDs anfangen zu leuchten, muss ein Taster gedrückt werden. Das Leuchtmuster steht für die Reaktionszeit.

Leuchten alle LEDs wurde entweder zu früh, zu spät oder gar nicht gedrückt!

Für dieses Spiel werden wir zunächst zwei Funktionen definieren:

Die Funktion `waitForKey()` wartet eine bestimmte Zeit lang auf einen Tastendruck. Wurde eine Taste innerhalb der angegebenen Zeit gedrückt liefert sie den Wert `true`. Wurde keine Taste gedrückt liefert sie nach Ablauf der Zeit den Wert `false` zurück. Der Datentyp für einen Wahrheitswert nennt sich `bool` und kann als Werte `true` oder `false` annehmen.

Die Funktion `displayState()` zeigt mit den LEDs den aktuellen Zustand (0-10) so an, wie er in der Tabelle oben zu sehen ist.

Lege wie in den vorangegangenen Beispielen einen neuen Sketch an, und speichere diesen unter dem Namen „**Reaktion**“. Als Bibliothek importieren wir wieder „NIBOburger“.

Ergänze den Quellcode im Editorfenster:

```
01 #include <NIBOburger.h>
02
03 bool waitForKey(unsigned int millisec) {
04     for (unsigned int t=0; t<millisec; t++) {
05         if (NIBOburger.getKeyChar()!=0) {
06             return true;
07         }
08         delay(1);
09     }
10     return false;
11 }
12
13 void displayState(unsigned int state) {
14     switch (state) {
15         case 0: NIBOburger.setLeds(OFF, OFF, OFF, OFF); break;
16         case 1: NIBOburger.setLeds( ON, OFF, OFF, OFF); break;
17         case 2: NIBOburger.setLeds(OFF,  ON, OFF, OFF); break;
18         case 3: NIBOburger.setLeds(OFF, OFF,  ON, OFF); break;
19         case 4: NIBOburger.setLeds(OFF, OFF, OFF,  ON); break;
20         case 5: NIBOburger.setLeds( ON, OFF, OFF,  ON); break;
21         case 6: NIBOburger.setLeds(OFF,  ON, OFF,  ON); break;
22         case 7: NIBOburger.setLeds(OFF, OFF,  ON,  ON); break;
23         case 8: NIBOburger.setLeds( ON, OFF,  ON,  ON); break;
24         case 9: NIBOburger.setLeds(OFF,  ON,  ON,  ON); break;
25         case 10: NIBOburger.setLeds( ON,  ON,  ON,  ON); break;
26     }
27 }
28
29 void setup() {
30     NIBOburger.begin();
31 }
32
33 void loop() {
34     NIBOburger.checkVoltage();
35     delay(100);
36     while (NIBOburger.getKeyChar()!='A') {
37         // auf KEY-DOWN von Taster 1 warten....
38     }
39
40     displayState(0);
41     while (NIBOburger.getKeyChar()!='a') {
42         // auf KEY-UP von Taster 1 warten....
43     }
44
45     NIBOburger.randomize();
46     unsigned int time = NIBOburger.getRandomInt(1000, 4000);
47
48     if (waitForKey(time)) {
49         displayState(10);
50         return;
51     }
52 }
```

```
53   for (unsigned int i=1; i<=10; i++) {  
54       displayState(i);  
55       if (waitForKey(50)) {  
56           return;  
57       }  
58   }  
59 }
```

Erläuterungen zum Quellcode:

03	Die Funktion waitForKey() bekommt als Parameter die Zeitdauer, die auf einen Tastendruck gewartet werden soll.
04	Die for Schleife wird pro Millisekunde genau einmal durchlaufen
05 - 07	Sollte eine Taste gedrückt worden sein, kehrt die Funktion mit dem Wert true zurück
10	Nach Ablauf der Zeit kehrt die Funktion mit dem Wert false zurück.
13	Die Funktion displayState() bekommt als Parameter den anzuzeigenden Zustand. Sie hat keinen Rückgabewert (void).
14 - 26	Je nach Zustand werden durch den jeweiligen Aufruf von NIBOburger.setLeds() alle vier LEDs auf einmal geschaltet.
29 - 31	setup-Funktion: Die Setup-Funktion beginnt wie immer mit der Initialisierung.
33 - 59	loop-Funktion
34 - 35	Zuerst wird die aktuelle Versorgungsspannung überprüft und kurz gewartet.
36 - 38	Die while Schleife läuft solange, bis der Taster 1 heruntergedrückt wird.
40 - 43	Wir schalten alle LEDs aus. Die folgende while Schleife läuft solange, bis der Taster 1 wieder losgelassen wird.
45 - 46	Der Zufallszahlengenerator wird mit dem zufälligen Sensordaten initialisiert und eine Zufallszahl zwischen 1000 und 4000 erzeugt.
48 - 51	Wir warten die zufällige Zeit lang auf einen Tastendruck. Sollte dieser erfolgt sein, dann war er zu früh: Status 10 wird angezeigt und der aktuelle Durchlauf verlassen.

53 Die Schleife wird genau zehn mal durchlaufen mit den folgenden Werten:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10

54 - 57 Durch Aufruf von **displayState()** wird der aktuelle Zustand angezeigt.
Sollte danach innerhalb von 50 Millisekunden eine Taste gedrückt worden
sein, sind wir fertig und verlassen die loop Funktion.

Ideen & Anregungen:

1. Ändere Dein Programm so ab, dass die Zeit in 500 ms Sekunden Schritten gezählt wird, dadurch können die einzelnen Zustände leicht beobachtet werden.
2. Ändere Dein Programm so ab, dass die Zeit in 20 ms Sekunden Schritten gezählt wird – dadurch verringert sich die maximale Reaktionszeit auf 180 ms...

11 Testen der Odometriesensoren

Als nächstes soll die Odometrie des NIBO burger getestet werden.

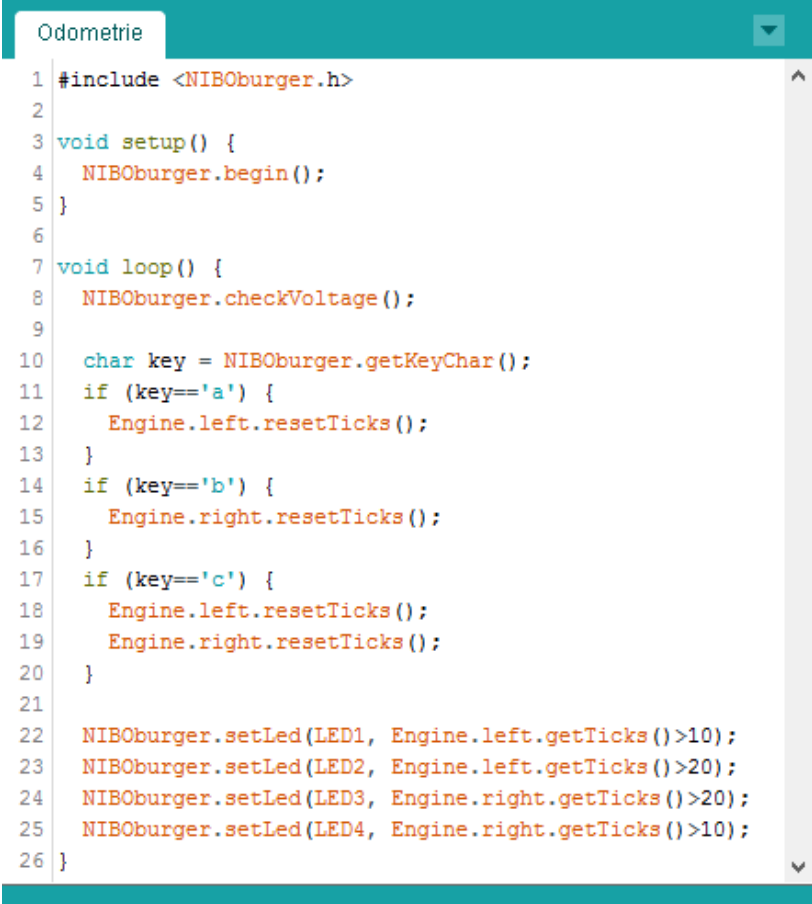
Erzeuge dazu einen neuen Sketch „**Odometrie**“ und importiere als Bibliothek wieder „NIBOburger“.

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

Durch Betätigung der Taster werden die Zähler zurückgesetzt:

Taster 1 setzt den linken Zähler zurück,
Taster 2 setzt den rechten Zähler zurück,
Taster 3 setzt beide Zähler zurück.

Tippe den folgenden Quellcode in das Editorfenster ein:



```
1 #include <NIBOburger.h>
2
3 void setup() {
4   NIBOburger.begin();
5 }
6
7 void loop() {
8   NIBOburger.checkVoltage();
9
10  char key = NIBOburger.getKeyChar();
11  if (key=='a') {
12    Engine.left.resetTicks();
13  }
14  if (key=='b') {
15    Engine.right.resetTicks();
16  }
17  if (key=='c') {
18    Engine.left.resetTicks();
19    Engine.right.resetTicks();
20  }
21
22  NIBOburger.setLed(LED1, Engine.left.getTicks()>10);
23  NIBOburger.setLed(LED2, Engine.left.getTicks()>20);
24  NIBOburger.setLed(LED3, Engine.right.getTicks()>20);
25  NIBOburger.setLed(LED4, Engine.right.getTicks()>10);
26 }
```

Erläuterungen zum Quellcode:

01 - 06	Das Programm beginnt mit dem Einbinden der NIBOburger Bibliothek, und der Initialisierung des Roboters
07 - 26	loop-Funktion:
08	In diesem Beispiel überprüfen wir regelmäßig die Akku-Spannung.
09	Die Variable key bekommt das aktuelle Tasten Event
11 - 20	Die drei if-Anweisungen setzen jeweils die aktuellen Odometrie-Zählerstände zurück wenn einer der Taster losgelassen wurde.
22 - 25	<p>In den letzten vier Zeilen werden die LEDs in Abhängigkeit von den Zählerständen gesetzt.</p> <p>Der Aufruf <code>Engine.left.getTicks()</code> liefert den aktuellen Zählerstand des linken Odometrie-Zählers zurück ohne ihn (auf null) zurückzusetzen.</p> <p>Hinweis:</p> <p>Die Odometrie-Zähler werden in diesem Beispiel unabhängig von der Drehrichtung immer hochgezählt. Im Motorbetrieb wird die Zählrichtung in Abhängigkeit von der Polarität automatisch umgeschaltet.</p>

Ideen & Anregungen:

Schreibe das Programm folgendermaßen um:

- Definiere die globale Variable `threshold` und initialisiere sie mit dem Wert 5
- Drücken von Taster 1 soll `threshold` um eins erhöhen.
- Drücken von Taster 3 soll `threshold` um eins verringern.
- Drücken von Taster 2 soll die Odometrie-Zähler zurücksetzen.
- Die blauen LEDs sollen leuchten, wenn der Odometrie-Zähler den Wert von `threshold` übertrifft, die roten wenn der Wert geringer ist...

12 Ansteuerung der Motoren

Als nächstes wollen wir die Motoren ansteuern.

Erzeuge einen neuen Sketch „**Motor**“ und importiere als Bibliothek wieder „NIBOburger“.

Das Programm steuert die Motoren mittels der Taster an. Eine Betätigung des Tasters 1 lässt die Motoren vorwärts drehen, eine Betätigung des Tasters 3 lässt die Motoren rückwärts drehen.

Der Taster 2 dient zum Stoppen – dabei soll schon auf das Runterdrücken der Taste reagiert werden (Großbuchstabe **B**). Die anderen beiden Tasten lösen die Aktion erst nach dem loslassen aus (Kleinbuchstaben **a** und **c**).

Tippe den folgenden Quellcode in das Editorfenster ein:



```
1 #include <NIBOburger.h>
2
3 void setup() {
4     NIBOburger.begin();
5 }
6
7 void loop() {
8     NIBOburger.checkVoltage();
9     int speed = 0;
10    char c = NIBOburger.getKeyChar();
11
12    if (c!=0) {
13        switch (c) {
14            case 'a': speed = 800; break;
15            case 'B': speed = 0; break;
16            case 'c': speed = -800; break;
17        }
18        Engine.setPWM(speed, speed);
19    }
20
21    delay(10);
22 }
```

Erläuterungen zum Quellcode:

01 - 06 Das Programm beginnt wieder mit dem Einbinden der NIBOburger Bibliothek, der Initialisierung des Roboters

08 - 22 **loop-Funktion:**

09 - 10 Es wird eine lokale Variable speed deklariert, in der der Sollwert für die Motoren gespeichert wird. Wir setzen die Variable zunächst auf den Wert 0.

11 - 20 Der Sollwert für die Motoren wird in der switch-Anweisung in Abhängigkeit von den Tasten gesetzt.

18 In dieser Zeile wird der Sollwert den beiden Motoren zugewiesen.

Hinweis:

Die Angabe der Werte erfolgt in 1/1024 Schritten:

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

Die Ansteuerung erfolgt mittels Pulsweitenmodulation (PWM).

Ideen & Anregungen:

1. Ändere das Programm so ab, dass die Vorwärtsdrehung schneller und die Rückwärtsdrehung langsamer erfolgt.
2. Ändere das Programm so ab, dass der Roboter bei einem Tastendruck auf 50% Leistung und bei einem zweiten Tastendruck auf 100% schaltet.

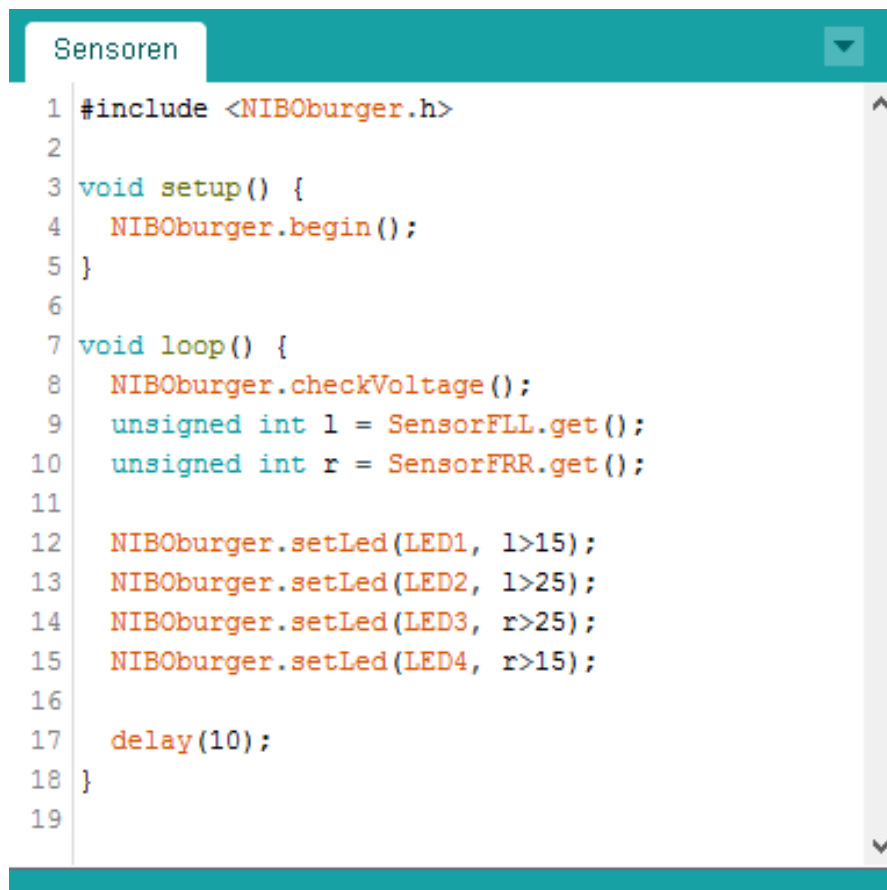
13 Abfrage der Sensoren

Als nächstes wollen wir die Messwerte der Sensoren auswerten.

Erzeuge einen neuen Sketch „**Sensoren**“ und importiere als Bibliothek wieder „NIBOburger“.

Das Programm zeigt die Messwerte der beiden Sensorslots **FLL** (vorne links außen) und **FRR** (vorne rechts außen) mit den LEDs an. Ist der Reflexionswert größer als 15, dann leuchtet die rote LED, ist er größer als 25, dann leuchtet zusätzlich die blaue LED.

Tippe den folgenden Quellcode in das Editorfenster ein:

A screenshot of the Arduino IDE interface. The top tab is labeled 'Sensoren'. The code editor shows the following C++ code:

```
1 #include <NIBOburger.h>
2
3 void setup() {
4   NIBOburger.begin();
5 }
6
7 void loop() {
8   NIBOburger.checkVoltage();
9   unsigned int l = SensorFLL.get();
10  unsigned int r = SensorFRR.get();
11
12  NIBOburger.setLed(LED1, l>15);
13  NIBOburger.setLed(LED2, l>25);
14  NIBOburger.setLed(LED3, r>25);
15  NIBOburger.setLed(LED4, r>15);
16
17  delay(10);
18 }
19
```


Erläuterungen zum Quellcode:

01 - 05 Das Programm beginnt wieder mit dem Einbinden der NIBOburger Bibliothek, der Initialisierung des Roboters

07 - 19 **loop-Funktion:**

08 Prüfen ob die Akkus noch geladen sind...

09 - 10 In den beiden lokalen Variablen **l** und **r** werden die Messwerte der Sensoren gespeichert.

12 - 15 Die LEDs werden in Abhängigkeit von den Werten eingeschaltet.

Ideen & Anregungen:

1. Ändere das Programm so ab, dass anstatt der äußeren die inneren (**FL** und **FR**) Sensoren verwendet werden.
2. Ändere das Programm so ab, dass die LEDs bei größeren bzw. kleineren Messwerten eingeschaltet werden. Überlege, welcher Bereich sinnvoll ist!

14 Hindernisdetektion

Nun wollen wir dem NIBO burger beibringen, Hindernissen auszuweichen.

Erzeuge einen neuen Sketch „*Hindernis*“ und importiere als Bibliothek wieder „NIBOburger“.

Mit diesem Programm kann der NIBO burger umherfahren. Sobald mit den Sensoren Hindernisse detektiert werden, soll er versuchen diesen auszuweichen.

Tippe den folgenden Quellcode in das Editorfenster ein:

```
01 #include <NIBOburger.h>
02
03 enum {
04     OBST_CLEAR = 0,
05     OBST_LEFT  = 1,
06     OBST_RIGHT = 2,
07     OBST_BOTH  = 3
08 };
09
10 unsigned int environment;
11
12 unsigned int calculateEnvironment() {
13     unsigned int left = max(SensorFLL.get(), SensorFL.get());
14     unsigned int right = max(SensorFRR.get(), SensorFR.get());
15
16     if ((left>15)&&(right>15)) {
17         return OBST_BOTH;
18     }
19     if (left>15) {
20         return OBST_LEFT;
21     }
22     if (right>15) {
23         return OBST_RIGHT;
24     }
25     return OBST_CLEAR;
26 }
27
28 void setup() {
29     NIBOburger.begin();
30     environment = OBST_CLEAR;
31 }
32
33 void loop() {
34     NIBOburger.waitAnalogUpdate();
35     NIBOburger.checkVoltage();
36     unsigned int new_env = calculateEnvironment();
37
38     if (new_env!=environment) {
```

```

39  environment = new_env;
40  switch (new_env) {
41      case OBST_CLEAR:
42          NIBOburger.setLed(LED2, 0);
43          NIBOburger.setLed(LED3, 0);
44          Engine.setSpeed(+25, +25);
45          break;
46
47      case OBST_LEFT:
48          NIBOburger.setLed(LED2, 1);
49          NIBOburger.setLed(LED3, 0);
50          Engine.setSpeed(+20, -20);
51          break;
52
53      case OBST_RIGHT:
54          NIBOburger.setLed(LED2, 0);
55          NIBOburger.setLed(LED3, 1);
56          Engine.setSpeed(-20, +20);
57          break;
58
59      case OBST_BOTH:
60          NIBOburger.setLed(LED2, 1);
61          NIBOburger.setLed(LED3, 1);
62          Engine.setSpeed(0, 0);
63          break;
64  }
65 }
66 }

```

Erläuterungen zum Quellcode:

01 - 23 Deklarationen

03 - 08 Hier werden Konstanten zur Beschreibung des „Zustands der Umgebung des Roboters“ definiert.

10 Globale Variable environment um uns den Zustand des letzten Durchlaufs der loop-Funktion zu merken.

12 - 26 Mit der Funktion calculateEnvironment wird der Zustand der Umgebung vor dem Roboter anhand der Sensorwerte berechnet.

13+14 Die Funktion max liefert den größeren Wert von zwei Werten zurück. Hier verwenden wir sie um den größten Sensorwert auf jeder Seite zu berechnen.

16 - 25 Wenn links ein nahes Hindernis (*engl. obstacle*) ist, liefert die Funktion OBST_LEFT zurück, wenn rechts ein nahes Hindernis ist den Wert OBST_RIGHT. Sollte sich auf beiden Seiten ein Hindernis befinden liefert

die Funktion den Wert OBST_BOTH zurück. Ist der Raum vor dem Roboter frei liefert die Funktion OBST_CLEAR zurück.

28 - 30 **setup-Funktion:**

Die Setup Funktion beginnt wie immer mit der Initialisierung des Roboters. Danach setzen wir die Variable environment auf OBST_CLEAR (kein Hindernis)

33 - 66 **loop-Funktion:**

34 - 36 Wir warten auf neue Messwerte, überprüfen die Versorgungsspannung und berechnen den aktuellen Zustand der Umgebung.

38 Den folgenden Block führen wir nur aus, wenn sich der Zustand der Umgebung geändert hat.

40 - 64 Abhängig vom neuen Zustand stellen wir den Strom für die Motoren ein und setzen die LEDs.

Ideen & Anregungen:

1. Erweitere das Programm um einen Zustand OBST_CLOSE, wenn ein sehr nahes Hindernis vorhanden ist. In diesem Fall sollte der Roboter langsam zurücksetzen.
2. Verbessere und erweitere das Programm!

15 Kalibrierung des Farbsensors

In diesem Beispiel werden wir uns mit dem Farbsensor des NIBO burger beschäftigen. Der Farbsensor ist aus drei farbigen LEDs (rot, grün und blau) und drei Phototransistoren aufgebaut. Zur Verwendung des Farbsensors muss dieser jedoch zunächst kalibriert werden.

Tippe dazu den folgenden Quellcode in das Editorfenster ein:



```
1 #include <NIBOburger.h>
2
3 void setup() {
4   NIBOburger.begin();
5 }
6
7 void loop() {
8   NIBOburger.waitAnalogUpdate();
9   NIBOburger.checkVoltage();
10
11   ColorRGB col = SurfaceSensor.getColorRGB();
12   int diff_black = col.diff(ColorRGB::CALIBRATE_BLACK);
13   int diff_white = col.diff(ColorRGB::CALIBRATE_WHITE);
14
15   NIBOburger.setLed(LED1, diff_black < 20);
16   NIBOburger.setLed(LED2, diff_white < 20);
17
18   char key = NIBOburger.getKeyChar();
19   if (key=='a') {
20     SurfaceSensor.calibrateBlack();
21     SurfaceSensor.storeCalibration();
22   } else if (key=='b') {
23     SurfaceSensor.calibrateWhite();
24     SurfaceSensor.storeCalibration();
25   }
26 }
```

Nach abgeschlossener Übertragung können die Sensoren wie folgt kalibriert werden:

- Man platziert den NIBO burger mit allen Bodensensoren auf einer **schwarzen Fläche** und drückt den **Taster 1**.
- Danach stellt man den NIBO burger auf eine **weiße Fläche** und drückt den **Taster 2**.
- Die Kalibrierdaten werden automatisch gespeichert.

Die Sensoren arbeiten nach dem IR-Reflexionsverfahren. Dabei wird gemessen welcher Anteil vom ausgesendeten Licht zurück reflektiert wird. Die Messung wird zweimal durchgeführt: Einmal mit abgeschalteter LED und einmal mit eingeschalteter LED. Durch einfache Subtraktion der beiden Ergebnisse können die Einflüsse des Umgebungslichts minimiert werden.

Die gemessenen Werte werden von der Bibliothek nach erfolgreicher Kalibrierung normalisiert und stehen mit als Ganzzahlen im Bereich von **0 (schwarz)** bis **1024 (weiss/farbig)** mit folgenden Ausdrücken zur Verfügung:

```
SensorBL.get()  
SensorBC.get()  
SensorBR.get()
```

Alternativ können auch direkt RGB bzw. HSV Farbwerte abgerufen werden:

```
SurfaceSensor.getRGB()  
SurfaceSensor.getHSV()
```

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

Falls anstatt des Farbsensors die IR-Sensoren zur Bodenanalyse verwendet werden sollen, ist eine erneute Kalibrierung notwendig!

16 Arbeiten mit dem Farbsensor

Nun können wir den Farbsensor verwenden:

Das Testprogramm soll mit den LEDs anzeigen auf welchem Feld der Farbkarte (blau, rot, grün oder gelb) der Roboter positioniert wird

Lege wie in den vorangegangenen Beispielen einen neuen Sketch an, und speichere diesen unter dem Namen „**Farbsensor**“. Importiere als Bibliothek wieder „NIBOburger“ und ergänze den Quellcode im Editorfenster:



```
1 #include <NIBOburger.h>
2
3 void setup() {
4   NIBOburger.begin();
5 }
6
7 void loop() {
8   NIBOburger.waitAnalogUpdate();
9   NIBOburger.checkVoltage();
10
11   ColorHSV col = SurfaceSensor.getColorHSV();
12   int diff_blue   = col.diff(ColorHSV::CALIBRATE_BLUE, 20);
13   int diff_red    = col.diff(ColorHSV::CALIBRATE_RED, 20);
14   int diff_green  = col.diff(ColorHSV::CALIBRATE_GREEN, 20);
15   int diff_yellow = col.diff(ColorHSV::CALIBRATE_YELLOW, 20);
16
17   int diff_min = min(min(diff_blue, diff_red), min(diff_green, diff_yellow));
18
19   if (diff_min < 600) {
20     if (diff_min == diff_blue) {
21       NIBOburger.setLeds(0, 1, 1, 0);
22     } else if (diff_min == diff_red) {
23       NIBOburger.setLeds(1, 0, 0, 1);
24     } else if (diff_min == diff_green) {
25       NIBOburger.setLeds(1, 1, 0, 0);
26     } else if (diff_min == diff_yellow) {
27       NIBOburger.setLeds(0, 0, 1, 1);
28     }
29   } else {
30     NIBOburger.setLeds(0, 0, 0, 0);
31   }
32 }
```

Erläuterungen zum Quellcode:

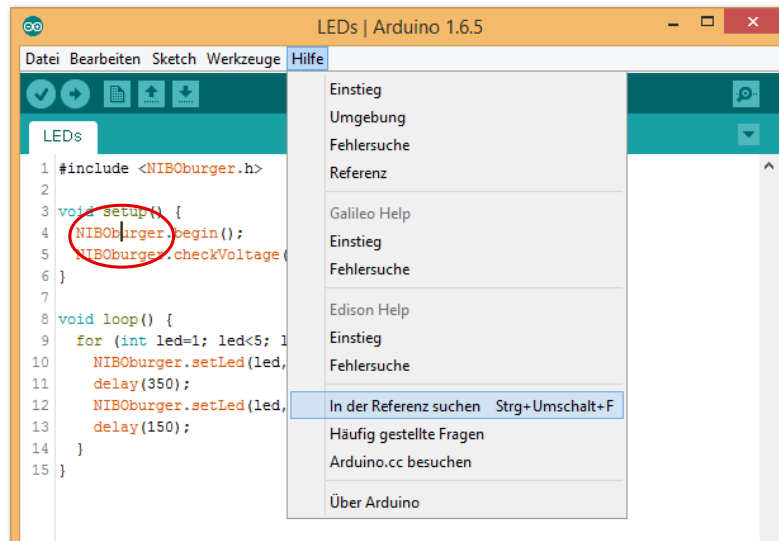
01 - 05	Das Programm beginnt wieder mit dem Einbinden der NIBO burger Bibliothek, der Initialisierung des Roboters.
08 - 13	loop-Funktion:
08 - 09	Zunächst warten wir auf neue Messwerte und überprüfen die Versorgungsspannung.
11	Die Zeile initialisiert die Variable col mit dem aktuellen Farbwert aus dem HSV-Farbraum.
12 - 15	Nun berechnen wir die Differenz der gemessenen Farbe zu den Werten der 4 Farbfelder.
17	Mit dieser Zeile ermitteln wir die geringste Differenz
19	Wenn die Differenz klein genug ist, haben wir eine Farbe detektiert.
20 - 28	Je nachdem welche die kleinste Diferenz war, werden die LEDs geschaltet.
29 - 31	Ansonsten wurde keine Farbe erkannt und die LEDs werden abgeschaltet.

Ideen & Anregungen:

1. Versuche mal das Programm vom HSV in den RGB Farbraum zu transformieren – klappt das genauso gut?
2. Informiere Dich in der Wikipedia über den RGB und den HSV Farbraum, welchen Vorteil bietet der HSV Farbraum für die Erkennung des Untergrundes?

17 Dokumentation

Eine Übersicht über die NIBOburger ARDUINO Bibliothek kann folgender-maßen geöffnet werden: Klicke im Editor in das Wort NIBOburger (siehe Bild) und wähle aus dem Menü „Hilfe“ → „In Referenz suchen“ aus:



Nun öffnet sich in Ihrem Webbrowser folgendes Fenster:

📖 NIBO burger robot class reference for ARDUINO

NIBO burger • ARDUINO-Referenz

Einbindung durch: `#include <NIBOburger.h>`

- class instance NIBOburger

Funktion	Beschreibung
<code>void begin ()</code>	Initialisierung des NIBOburger Roboters
<code>unsigned int getVoltage ()</code>	Versorgungsspannung messen. <i>return-value:</i> Spannung in Millivolt (4.8V ± 4800)
<code>void checkVoltage ()</code>	Versorgungsspannung überprüfen, im Fehlerfall anhalten und blinken
<code>void setLed (int led, int value)</code>	LED ein/ausschalten: <i>led:</i> LED1=1, LED2=2, LED3=3, LED4=4 <i>value:</i> OFF=0, ON=1
<code>int getLed (int led)</code>	LED abfragen: <i>led:</i> LED1=1, LED2=2, LED3=3, LED4=4 <i>return-value:</i> OFF=0, ON=1
<code>unsigned int getRandomSeed ()</code>	Zufallszahlen-Basis liefern
<code>unsigned int getAnalog (unsigned char adc_channel, unsigned char active)</code>	Rohwert eines analogen Kanals auslesen

+ class instance Engine

+ class instances Engine.left / Engine.right

+ class instance SurfaceSensor

+ class ColorRGB

+ class ColorHSV

+ class instances SensorFL, SensorFR, SensorFLL, SensorFRR, SensorBL, SensorBC, SensorBR

+ IO-Definitionen

🔗 [nicai-systems website](http://nibo.nicai-systems.de)

18 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 die NIBO Roboter 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>
Informationen rund um die NIBO Roboter.
- ➔ **Mikrocontroller:** <http://www.mikrocontroller.net>
Alles über Mikrocontroller und deren Programmierung.
- ➔ **AVRFreaks:** <http://www.avrfreaks.net>
Informationen rund um den AVR.