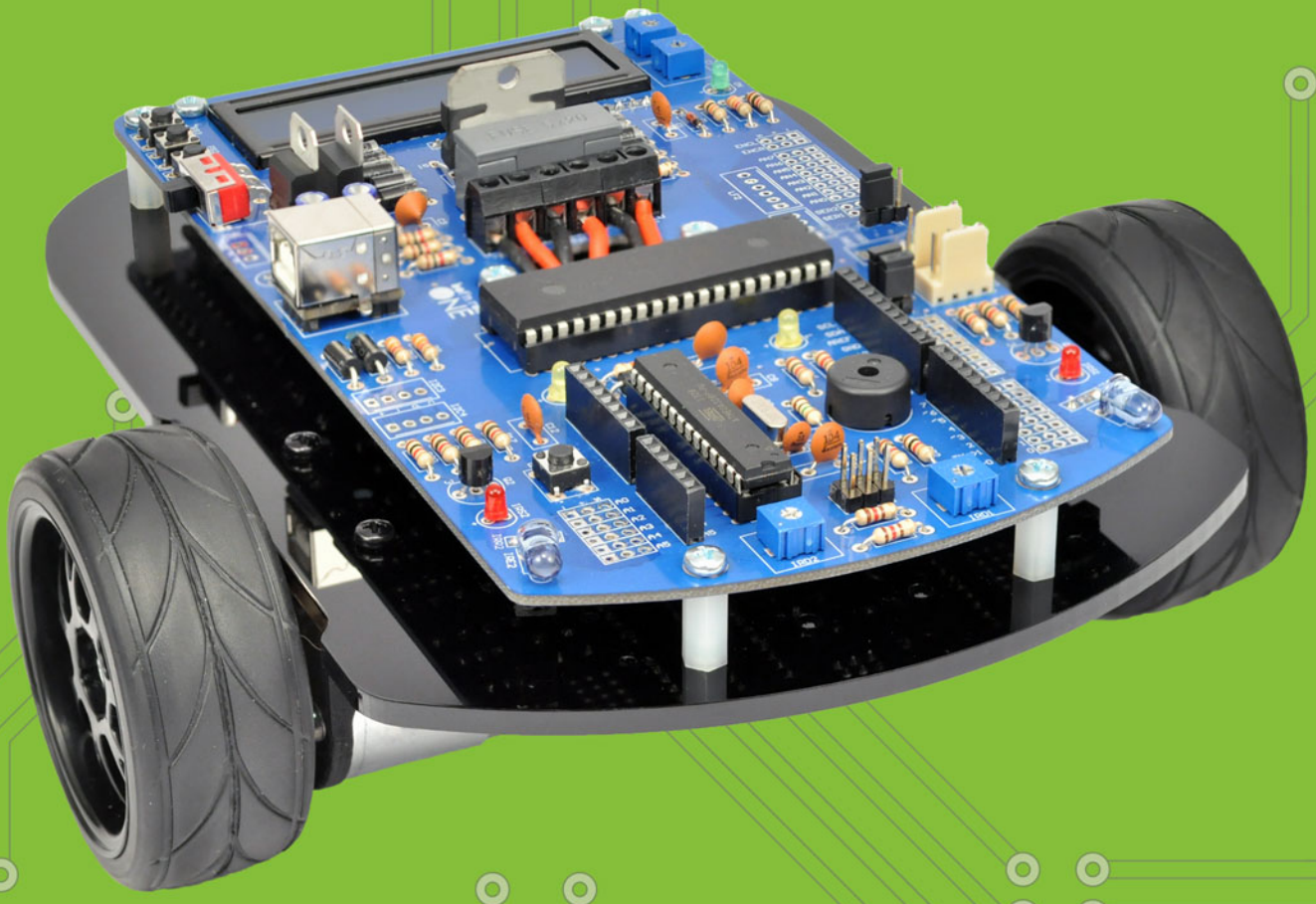
bot'n roll

ONE

build your own robot



Programmierhandbuch

www.botnroll.com

INHALT

1. Einleitung 3
1.1 Den Bot'n Roll ONE A programmieren 3
1.1.1 Arduino IDE 4
1.1.2 BnrOneA Library für Arduino 5
1.1.3 Die Programmiersprache C 6
2. Funktionen der BnrOneA Arduino Library 8
2.1.1 spiConnect(sspin) 9
2.1.2 minBat(batmin) 10
2.1.3 obstacleEmitters(state) 11
2.2 Lesefunktionen 12
2.2.1 obstacleSensors() 12
2.2.2 readIRSensors() 13
2.2.3 readAdc(byte) 14
2.2.4 readAdcX() 15
2.2.5 readButton() 16
2.2.6 readBattery() 16
2.2.7 readEncL() 17
2.2.8 readEncR() 17
2.2.9 readEncLInc() 18
2.2.10 readEncRInc() 18
2.3 Befehlsfunktionen 19
2.3.1 servo1(position) 19
2.3.2 servo2(position) 20
2.3.3 led(state) 21
2.3.4 move(speedL,speedR) 21
2.3.5 stop() 22
2.3.6 brake(torqueL,torqueR) 22
2.3.7 resetEncL() 23

2.3.8 resetEncR() 23

2.4 LCD-Schreibfunktionen 24

2.4.1 lcdX(string[]) 24

2.4.2 lcdX(number) 25

2.4.3 lcdX(string[],number) 26

2.4.4 lcdX(num1, num2) 27

2.4.5 lcdX(num1, num2, num3) 28

2.4.6 lcdX(num1 , num2 , num3 , num4) 29

Anhang A: VCP-Treiber für den USB-Seriell(RS232)-Wandler installieren 30

Anhang B: Die Arduino-Programmierumgebung 30

B.1 Arduino IDE installieren 30

B.2 Die **BnrOneA** Library auf Arduino installieren 30

B.3 Die Kommunikation mit dem Roboter konfigurieren 31

B.4 Ein Programm auf den Bot'n Roll ONE A laden 32

Stand: 15. Februar 2016

1. EINLEITUNG

Der Bot'n Roll ONE A wird in der Programmiersprache C in der Programmierumgebung Arduino IDE programmiert. Da der Mikrocontroller ATmega328, mit dem dieser Roboter arbeitet, mit dem Arduino Uno Bootloader ausgestattet ist, wird der Roboter so programmiert, als wäre er ein Arduino Uno.

Der Roboter hat einen zweiten Mikrocontroller: einen vorprogrammierten PIC18F45K22, der mit einer von botnroll.com entwickelten Software arbeitet. Beim Bot'n Roll ONE A dient er als **Slave**, der die Befehle des **Masters** ATmega328 ausführt.

Die beiden Mikrocontroller des Bot'n Roll ONE A kommunizieren über den SPI-Bus miteinander (SPI steht für „**Serial Peripheral Interface**", also „Serielle Peripherie-Schnittstelle"). Der Informationsaustausch zwischen den beiden Mikrocontrollern läuft koordiniert und nach klar definierten Vorgaben ab. Damit Master und Slave miteinander kommunizieren können, wurde ein geeignetes Datenübertragungsprotokoll entwickelt. Der Master verwendet eine Liste von Befehlen, die jeweils Steuerungsanweisungen entsprechen. Jeder Befehl generiert eine Antwort vom Slave. Die Befehlsliste und die Art und Weise, wie die Daten zwischen Master und Slave übermittelt werden, sind in der BnrOneA Library definiert.

Die BnrOneA Library für Arduino schafft die Grundlage dafür, dass der Nutzer den Roboter mühelos steuern kann, indem er einfach die Library-Befehle in Arduino IDE richtig verwendet. Diese Befehle werden in diesem Programmierhandbuch aufgeführt und erklärt.

Auch wenn beide Mikrocontroller in der Programmiersprache C programmiert werden können, sollte der ATmega328 mit dem Arduino Bootloader tageweise mit Hilfe der BnrOneA Library programmiert werden.

Der PIC18F45K22 kann mit Hilfe der Programmierumgebung MPLABX IDE von Microchip und XC8 Compiler oder einer anderen kompatiblen Software in der Programmiersprache C programmiert werden. Diese Möglichkeit sollten allerdings nur fortgeschrittene Anwender nutzen, denn um den PIC18F45K22 so zu programmieren, dass er neue Funktionen ausführen kann, muss man auch die BnrOneA Library aktualisieren, damit man die neue Funktion nutzen kann. Wenn Ihr eine neue Funktionen auf Eurem Bot'n Roll ONE A einrichten wollt, wendet Euch bitte an botnroll.com!

1.1 DEN BOT'N ROLL ONE A PROGRAMMIEREN

Damit Ihr den Bot'n Roll ONE A programmieren könnt, müsst Ihr Euren Computer mit allen erforderlichen Tools vorbereitet haben. Das heißt:

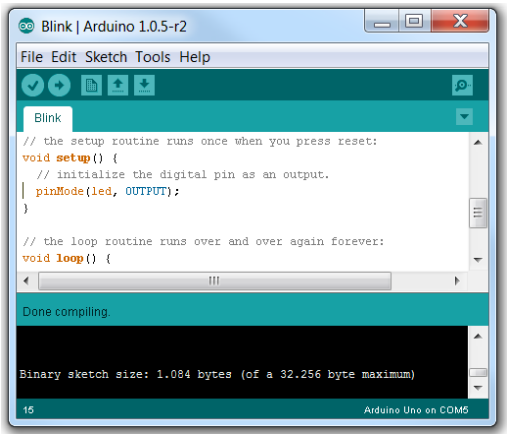
- Ihr habt den VCP-Treiber installiert (den USB-Port-Treiber für den Bot'n Roll ONE A; siehe ANHANG A);
- Ihr habt Arduino IDE installiert (siehe ANHANG B);
- Ihr habt in Arduino IDE die BnrOneA Library installiert (siehe ANHANG B).

Wie Ihr diese Tools installiert, wird ausführlich in den Anhängen A und B am Ende dieses Programmierhandbuchs beschrieben.

Auch die Programmiersprache C braucht Ihr, um den Bot'n Roll ONE A programmieren zu können. Für den Fall, dass Ihr Euch im Umgang mit der Programmiersprache C noch nicht so sicher fühlt, werdet Ihr feststellen, dass Euch die Beispiele aus der Library eine gute Orientierung für den Einstieg ins Programmieren bieten. Auch die Präsentationen zum Programmieren mit C, die für die RoboParty zusammengestellt wurden, sind ein gutes Hilfsmittel – und darüber hinaus gibt es natürliche Tausende von Internetseiten, auf denen die Programmiersprache C erklärt wird!

1.1.1 Arduino IDE

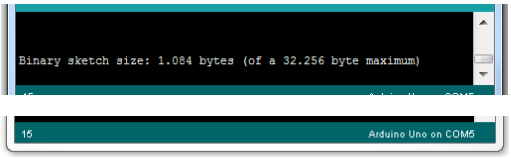
Die Arduino Programmierumgebung umfasst einen Texteditor zum Schreiben von Codes, ein Meldefenster, eine Textkonsole, eine Symbolleiste mit den wichtigsten Funktionen und eine Reihe von Menüs. Sie ermöglicht die Verbindung zur Arduino-Hardware des Bot'n Roll ONE A und sorgt somit dafür, dass Codes zum Roboter übertragen werden können und dass mit dem Roboter kommuniziert werden kann.



Ein Arduino-Programm wird als „***Sketch***" (deutsch „Skizze") bezeichnet. Es wird mit dem Texteditor geschrieben und mit der Dateierweiterung „.io" auf Eurem Computer gespeichert.

Im Meldefenster werden Mitteilungen angezeigt, dass ein Programm gespeichert oder exportiert wird, oder aufgetretene Fehler gemeldet.

In der Textkonsole erscheinen Textmeldungen mit ausführlichen Informationen zu aufgetretenen Fehlern und andere Informationen.



Rechts unten im Bildschirmfenster wird angezeigt, welche Leiterplatte programmiert und welcher serielle Port dafür verwendet werden soll.

Abbildung 1: Arduino IDE Module

Dies sind die wichtigsten Schaltflächen in der Symbolleiste mitsamt ihrer jeweiligen Funktion:

Verify: Überprüft den Code auf Fehler.

Upload: Kompiliert den Code und sendet ihn an Arduino.

New: Erstellt einen neuen Sketch.

Open: Öffnet einen auf dem Computer gespeicherten Sketch.

Save: Speichert den **Sketch**.

Serial Monitor: Öffnet die Kommunikation über die seriellen Ports, damit Ihr diese überwachen könnt.

Mit dem Serial Monitor könnt Ihr die Daten einsehen, die vom Arduino an den Computer gesendet werden. Außerdem bietet er die Möglichkeit, Daten vom Computer an den Arduino zu senden. Er ist sehr hilfreich beim Programmieren, weil Ihr hier Text und Variablenwerte ausdrucken könnt. Somit dient der Serial Monitor als ein „**Debugging**"-Instrument für Euer Programm. Wenn Ihr den Serial Monitor öffnet, startet das Arduino-Programm neu.

1.1.2 BnrOneA Library für Arduino

Eine Library ist eine „vorgefertigte" Reihe von Routinen, die Ihr in Euren Code einbinden und in Eurem Programm verwenden könnt. Damit Ihr die **BnrOneA** Library nutzen könnt, braucht Ihr sie einfach nur in Euren Code aufnehmen:

#include <BnrOneA.h>

und anschließend eine Instanz für die Klasse BnrOneA erstellen

BnrOneA one;

Damit könnt Ihr ab sofort auf alle Library-Funktionen zugreifen, denen die Instanz vorangestellt ist, die Ihr definiert habt, also: **one.***library_function()*;

Normalerweise wird eine Library angelegt, um Daten oder Hardware zu manipulieren, und enthält mindestens zwei Dateien, aber bei Arduino gibt es zusätzlich noch eine Dritte Datei mit der Endung „**.txt**".

- Eine Datei mit der Endung „**.h**" („***header***"), die die Liste aller verfügbaren Funktionen, Befehle und Library-Definitionen enthält;
- Eine Datei mit der Endung „**.cpp**" („***c++ source***") mit dem aktuellen Code für die in der Header-Datei enthaltenen Funktionen.
- Eine Datei ***keywords.txt***, mit der Arduino IDE die Library-Funktionen identifizieren kann; die Library-Funktionen erscheinen in einer anderen Farbe als der restliche Code.

Die BnrOneA Library wurde angelegt, um die mit dem PIC18F45K22 verknüpfte Hardware zu manipulieren, und sorgt dafür, dass der Arduino über den SPI-Kommunikationsbus mit ihr interagieren kann. Der Arduino hat Zugriff auf die gesamte Hardware und alle Funktionen, die in der Library und in der PIC18F45K22-Software definiert wurden. Die BnrOneA Library und die PIC18F45K22-Software wurden „füreinander geschaffen". Wenn eine von beiden geändert wird, muss die andere entsprechend angepasst werden.

Alle BnrOneA Library-Funktionen werden in Teil 2 dieses Programmierhandbuchs aufgeführt und erläutert.

1.1.3 Die Programmiersprache C

Die Programmiersprache C wurde 1972 von Dennis Ritchie in den Bell Labs in New Jersey entwickelt. Sie wurde als leistungsfähige und schnelle Sprache konzipiert, die im Betriebssystem UNIX verwendet werden kann. Im Laufe der Zeit wurde sie immer weiter verbessert und aktualisiert und erwies sich dabei als ausgesprochen robust und zuverlässig. Darum wurde sie nach und nach auch von anderen Betriebssystemen wie Windows, MacOS und Linux genutzt. Seit der Veröffentlichung der ersten Version „**K & R C**" ist sie kontinuierlich weiterentwickelt worden. 1989 folgte die erste Spezifikation als Standard des American National Standards Institute unter der Bezeichnung „**ANSI C**". 1990 brachte die Internationale Organisation für Normung (ISO) den Standard „ISO C" heraus. 1999 wurde der Standard „**C99**" verabschiedet. Die bisher letzte Überarbeitung erfolgt im Dezember 2011 mit ISO/**IEC 9899:2011**, besser bekannt unter der Bezeichnung „C11".

All diese Updates und Überarbeitungen zielten auf die Verwendung der Programmiersprache C für die Entwicklung von Programmen für Personal Computer, bei denen es keine Einschränkungen durch die Speicher- und Verarbeitungsressourcen gibt. Für Mikrocontroller wird eine „Light"-Version der Programmiersprache C verwendet, weil hier die Speicher- und Verarbeitungskapazitäten begrenzt sind. Wenn Ihr Euren Bot'n Roll ONE A also erfolgreich programmieren wollt, müsst Ihr einige Grundregeln der Sprache für Arduino kennen und wissen, wie der eine oder andere Befehl funktioniert.

Alle Programme für Arduino haben zwei obligatorische Routinen oder Funktionen. Die Konfigurationsroutine „**setup()**" wird nur einmal beim Start Eures Programms ausgeführt. Hier müsst Ihr den gesamten Code für die Initialisierung der Variablen schreiben, Ein- und Ausgangspin festlegen, die SPI-Kommunikation, Serial, I2C festlegen – kurzum: alle erforderlichen Konfigurationen vornehmen.

Nach der Konfiguration wechselt Euer Programm in die Routine **loop()** und bleibt dort auf unbestimmte Zeit. „**Loop**" bedeutet „Schleife" – und in diesem Fall handelt es sich um eine Endlosschleife: Wenn das Programm am Ende der Schleife ankommt, geht es zurück zum Anfang und beginnt wieder von vorn! Hier schreibt Ihr das Programm und macht Euren Roboter zu einem intelligenten Wesen!

Das Programmieren mit C wird in diesem Programmierhandbuch nicht erläutert. Bitte haltet Euch an die Beispiele aus der BnrOneA Library und die allgemeinen Arduino-Beispiele. Alle Codes werden angemessen kommentiert – und Ihr habt die Aufgabe, durch Ausprobieren und Testen selbst dahinterzukommen, wie es geht. Wir geben Euch aber ein paar Tipps mit auf den Weg, die wir im Laufe der Zeit im Rahmen von **botnroll.com** gesammelt haben:

- Geht beim Erstellen neuer Programme von den Basisbeispielen aus. Versucht, ausgehend von den Basisbeispielen 3 oder 4 Funktionen des Roboters im gleichen Programm zusammenzuführen!
    - Es kommt selten vor, dass ein Programm gleich beim ersten Mal funktioniert! Lasst Euch nicht unterkriegen, sondern analysiert das Problem und löst es!
- Geht beim Code-Eingeben systematisch vor und testet dabei die Codes immer wieder einmal, um zu kontrollieren, ob alles so funktioniert wie erwartet.
- Wer ein Programm erarbeiten will, das funktioniert, wendet mehr Zeit für das Testen auf als für das eigentliche Schreiben des Programms!
    - Nutzt Tools wie die Debug-LED, den Serial Monitor oder das LCD-Display, um den Wert von Variablen auszudrucken und zu kontrolliere, ob das Programm über einen bestimmten Punkt des Codes hinausgelangt.
- Programmieren ist wie das Trainieren einer neuen Sportart. Am Anfang ist es schmerzhaft, weil man noch nicht die nötige körperliche Kondition hat, die Regeln nicht kennt und eine Weile braucht, um sie zu verstehen. Durch Trainieren und Üben wird man allerdings in allen Punkten immer besser. Wer Einsatz zeigt, schafft es zu guter Letzt in die A-Mannschaft!

Der Bot'n Roll ONE A bietet die Möglichkeit, mit sehr unterschiedlichen Hardware-Elementen zu interagieren. Es gibt mit dem Bot'n Roll ONE A kompatible Extras, die üblicherweise als „Arduino Shields“ (Aufsteckplatinen für Arduino) bezeichnet werden und mit denen Ihr fast alles machen könnt, was Euch in den Sinn kommt! Alle Shields verfügen über Libraries, die Euch bei der Nutzung und Integration helfen – lasst Eurer Fantasie also freien Lauf!

2. FUNKTIONEN DER BNRONEA ARDUINO LIBRARY

Die Funktionen und Routinen der BnrOneA Library sorgen dafür, dass der Arduino (ATmega328) auf alle Peripheriegeräte zugreifen kann, die vom PIC18F45K22 gesteuert werden. Diese Routinen werden in drei Gruppen unterteilt: Konfiguration, Lesen und Schreiben. Viele dieser Funktionen verwenden für den Informationsaustausch – also für das Senden und/oder Empfangen von Variablenwerten – **Parameter/Argumente**.

Ein Argument ist jeder Ausdruck in einer Funktion, der in Klammern steht – zum Beispiel:

- **one.spiConnect(sspin);**

Das **Argument** ist **sspin** und wird bei der Ausführung der Funktion **spiConnect** als Parameter übergeben.

Eine Funktion kann, wenn sie ausgeführt wird, als Ergebnis auch einen Rückgabewert liefern:

- **float battery = one.readBattery();**

Die Akku-Lesefunktion liefert, wenn sie ausgeführt wird, als Ergebnis den Wert der Batterie. Der Akku-Wert wird in der Variablen **battery** gespeichert.

In der folgenden Liste seht Ihr alle aus der Datei **BnrOneA.h** extrahierten Funktionen der BnrOneA Library.

Liste der Funktionen der BnrOneA Library

Setup-Routinen	**Leseroutinen**
void spiConnect(byte sspin);	byte obstacleSensors();
void minBat(float batmin);	byte readIRSensors();
void obstacleEmitters(boolean state);	IntreadAdc(byte);
	int readAdc0();
	int readAdc1();
	int readAdc2();
Schreibroutinen	int readAdc3();
void servo1(byte position);	int readAdc4();
void servo2(byte position);	int readAdc5();
void led(boolean state);	int readAdc6();
void move(intspeedL,intspeedR);	int readAdc7();
void stop();	intreadButton();
void brake(byte torqueL,bytetorqueR);	float readBattery();
void resetEncL();	intreadEncL();
void resetEncR();	intreadEncR();
	intreadEncLInc();
	intreadEncRInc();
Schreibroutinen für LCD-Zeile 1	**Schreibroutinen für LCD-Zeile 2**
void lcd1(byte string[]);	void lcd2(byte string[]);
void lcd1(const char string[]);	void lcd2(const char string[]);
void lcd1(int number);	void lcd2(int number);
void lcd1(unsigned int number);	void lcd2(unsigned int number);
void lcd1(long int number);	void lcd2(long int number);
void lcd1(double number);	void lcd2(double number);
void lcd1(const char string[],int number);	void lcd2(const char string[],int number);
void lcd1(const char string[],unsigned int number);	void lcd2(const char string[],unsigned int number);
void lcd1(const char string[],long int number);	void lcd2(const char string[],long int number);
void lcd1(const char string[],double number);	void lcd2(const char string[],double number);
void lcd1(int num1, int num2);	void lcd2(int num1, int num2);
void lcd1(unsigned int num1, unsigned int num2);	void lcd2(unsigned int num1, unsigned int num2);
void lcd1(int num1, int num2, int num3);	void lcd2(int num1, int num2, int num3);
void lcd1(int num1, int num2, int num3, int num4);	void lcd2(int num1, int num2, int num3, int num4);
void lcd1(unsigned int num1, unsigned int num2, unsigned int num3);	void lcd2(unsigned int num1, unsigned int num2, unsigned int num3);
void lcd1(unsigned int num1, unsigned int num2, unsigned int num3, unsigned int num4);	void lcd2(unsigned int num1, unsigned int num2, unsigned int num3, unsigned int num4);

2.1.1 spiConnect(sspin)

Beschreibung:

Initialisiert den SPI-Kommunikationsbus, indem die Pins SCK, MOSI und SS als Ausgänge festgelegt werden. Setzt die Pins SCK und MOSI auf eine niedrige Spannung (0 V) und SS auf eine hohe Spannung (5 V). Beim Bot'n Roll ONE A entspricht der Pin SS („Slave Select") für die SPI-Kommunikation zwischen dem ATmega328 und PIC18F45K22 standardmäßig dem **digitalen Ausgang 2**, aber er kann verändert werden, indem man den SSP-Jumper entfernt und die gewünschte Verbindung herstellt. Der standardmäßige SS-Pin des ATmega328 kann für die SPI-Kommunikation mit jedem Shield genutzt werden, das diese Verbindung verwendet.

Parameter:

Sspin: der digitale Ausgang, der für die SPI-Kommunikation zwischen dem ATmega328 und dem PIC18F45K22 als „Slave Select" verwendet werden soll (*byte*)

Rückgabewerte:

Keine

Beispiel:

```
#include <BnrOneA.h>          // Bot'n Roll ONE A Library
#include <SPI.h>              // von BnrOne.cpp benötigte SPI-Kommunikations-Library
BnrOneA one;                  // Deklaration der Objektvariablen für die Steuerung des Bot'n Roll ONE A
#define SSPIN 2               // Slave Select (SS)-Pin für die SPI-Kommunikation
void setup()
{
    one.spiConnect(SSPIN);    // SPI-Kommunikationsmodul starten
}
```

2.1.2 minBat(batmin)

Beschreibung:

Legt den Mindestwert der Akkuspannung fest, bei der der Roboter die Bewegung einstellt, und schreibt eine Warnmeldung in die zweite Zeile des LCD-Displays. Dieser Einstellwert wird ins EEPROM des PIC18F45K22 geschrieben und geht nicht verloren, wenn Ihr den Bot'n Roll ONE A ausschaltet. Während der Einstellwert ins EEPROM geschrieben wird, ist der PIC18F45K22 nicht erreichbar. Während dieses Vorgangs, der ungefähr 5 ms dauert, dürfen also keine Befehle gesendet werden. Sobald die Akkuspannung unter das eingestellte Minimum sinkt, reagiert der Roboter nicht auf Bewegungsbefehle und meldet in Zeile 2 des LCD-Displays eine zu niedrige Akkuspannung. Diese Funktion trägt dazu bei, die Lebensdauer des Akkus zu erhalten, und ist sehr hilfreich, wenn Lithium-Akkus verwendet werden, denn diese Akkus gehen kaputt, wenn die Spannung unter das als Sicherheitswert festgelegte Minimum sinkt.

Parameter:

batmin: Die für den Akku zulässige Mindestspannung (Float-Parameter)

Rückgabewerte:

Keine

Beispiel:

```
#include <BnrOneA.h>          // Bot'n Roll ONE A Library
#include <SPI.h>              // von BnrOne.cpp benötigte SPI-Kommunikations-Library
BnrOneA one;                  // Deklaration der Objektvariablen für die Steuerung des Bot'n Roll ONE A
#define SSPIN 2               // Slave Select (SS)-Pin für die SPI-Kommunikation
void setup()
{
    one.spiConnect(SSPIN);    // SPI-Kommunikationsmodul starten
    one.minBat(8.5);          // Mindest-Akkuspannung festlegen
    delay(5);                         // 5 ms warten
}
```

2.1.3 obstacleEmitters(state)

Beschreibung:

Steuert den Zustand der LED-Infrarotsender. Die Infrarotsender können, wenn nötig, ausgeschaltet werden; die normale Hinderniserkennung wird deaktiviert. Beim Wert 0 werden die Sender ausgeschaltet; beim Wert 1 werden die Sender aktiviert.

Parameter:

state: der Zustand, in den die LEDs versetzt werden sollen (*Boolescher Parameter*)

Rückgabewerte:

Keine

Beispiel:

```
#include <BnrOneA.h>            // Bot'n Roll ONE A Library
#include <SPI.h>                // von BnrOne.cpp benötigte SPI-Kommunikations-Library
BnrOneA one;                    // Deklaration der Objektvariablen für die Steuerung des Bot'n Roll ONE A
#define SSPIN 2                 // Slave Select (SS)-Pin für die SPI-Kommunikation
void setup()
{
    one.spiConnect(SSPIN);      // SPI-Kommunikationsmodul starten
    one.obstacleEmitters(ON);   // LED-Infrarotsender aktivieren
}
```

2.2 LESEFUNKTIONEN

2.2.1 obstacleSensors()

Beschreibung:

Liefert als Rückgabewert das Ergebnis der Hinderniserkennung; dabei gibt es vier Möglichkeiten:

- **0**: Keine Hindernisse
- **1**: am linken Sensor Hindernisse erkannt
- **2**: am rechten Sensor Hindernisse erkannt
- **3**: an beiden Sensoren Hindernisse erkannt

Die Hinderniserkennung wird alle 20 ms vom PIC18F45K22 durchgeführt, und diese Funktion liefert das Ergebnis des letzten Erkennungsvorgangs. Damit eine Hinderniserkennung durchgeführt werden kann, müssen die Infrarotsender aktiviert sein!

Parameter:

Keine

Rückgabewerte:

Der bei der Hinderniserkennung ermittelte Wert *(Byte-Parameter*)

Beispiel:

```
...
void loop()
{
    byte obstacle=one.obstacleSensors(); // Hindernissensoren auslesen
...
}
```

2.2.2 readIRSensors()

Beschreibung:

Liefert als Rückgabewert den aktuellen Zustand der SHARP GP1UX511QS Infrarotsensoren; dabei gibt es vier Möglichkeiten:

- **0**: beide Sensoren auf „*High*“;
- **1**: linker Sensor „*High*“ und rechter Sensor „Low“
- **2**: linker Sensor „Low“ und rechter Sensor „*High*“
- **3**: beide Sensoren „*Low*“

Diese Funktion zwingt den PIC18F45K22, sofort den Zustand der Infrarotsensoren auszulesen. Bitte beachtet, dass die SHARP GP1UX511QS Sensoren ein invertiertes Signal an den Infrarotsender übermitteln.

Parameter:

Keine

Rückgabewerte:

Der aktuelle Zustand der Infrarotsensoren (*Byte-Parameter)*

Beispiel:

```
...
void loop()
{
    byte obstacle=one.readIRSensors(); // aktuellen Zustand der IR-Sensoren lesen
...
}
```


2.2.3 readAdc(byte)

Beschreibung:

Liefert als Rückgabewert den Wert der ADC-Wandlung („Analog-zu-Digital-Wandlung") – bezogen auf den PIC18F45K22 – für die im Parameterkanal angegebene Funktion. Der Parameter legt fest, über welchen der analogen Kanäle AN0 bis AN7 des Bot'n Roll ONE A die Wandlung erfolgen soll, und nimmt die folgenden Werte an:

- **0** (entspricht Kanal AN0)
- **1** (entspricht Kanal AN1)
- **2** (entspricht Kanal AN2)
- **3** (entspricht Kanal AN3)
- **4** (entspricht Kanal AN4)
- **5** (entspricht Kanal AN5)
- **6** (entspricht Kanal AN6)
- **7** (entspricht Kanal AN7)

Der Wert, der sich aus der Wandlung ergibt, variiert beim PIC18F45K22 zwischen 0 and 1023, da dieser mit einem 10-Bit-Analog-Digital-Wandler ausgestattet ist.

Parameter:

Der ADC-Kanal (*Byte-Parameter*)

Rückgabewerte:

Der Wert der ADC-Wandlung (int)

Beispiel:

```
...
void loop()
{
    for(i=0;i<8;i++)
    {
    adc[i]=one.readAdc(i);        // den ADC-Wert auslesen
    ...
    }
}
```


2.2.4 readAdcX()

Beschreibung:

Liefert als Rückgabewert den Wert der ADC-Wandlung („Analog-zu-Digital-Wandlung") – bezogen auf den PIC18F45K22 – vom ANX-Kanal des Bot'n Roll ONE A. Der Wert, der sich aus der Wandlung ergibt, variiert zwischen 0 und 1023, da der PIC18F45K22 mit einem 10-Bit-Analog-Digital-Wandler ausgestattet ist.

Der Buchstabe „X" entspricht dem ADC-Auslesekanal, der ausgelesen werden soll, und sollte durch eine Zahl von 0 bis 7 ersetzt werden.

Parameter:

Keine

Rückgabewerte:

Der Wert der ADC-Wandlung (int)

Beispiel:

```
...
void loop()
{
      int adc1 =one.readAdc1();    // den ADC-Wert auslesen
      int adc6 =one.readAdc6();    // den ADC-Wert auslesen
...
}
```


2.2.5 readButton()

Beschreibung:

Gibt an, welcher der Drucktaster PB1, PB2 oder PB3 gerade gedrückt wird. Als Ergebnis liefert die Funktion einen der möglichen Werte:

- **0**: kein Drucktaster ist gedrückt
- **1**: PB1 gedrückt
- **2**: PB2 gedrückt
- **3**: PB3 gedrückt

Wenn mehrere Drucktaster gleichzeitig gedrückt werden, wird der niedrigste Wert ausgegeben.

Parameter:

Keine

Rückgabewerte:

Nummer des gedrückten Drucktasters (*int*)

Beispiel:

```
...
void loop()
{
    int pbutton=one.readButton();     // den Drucktaster-Wert auslesen
...
}
```

2.2.6 readBattery()

Beschreibung:

Liest den Wert der Akkuspannung in Volt.

Parameter:

Keine

Rückgabewerte:

Die Spannung des Akkus (float)

Beispiel:

```
...
void loop()
{
    float battery=one.readBattery();          // Akkuspannung lesen
...
}
```


2.2.7 readEncL()

Beschreibung:

Liefert als Rückgabewert den Zählwert des linken Encoders; der Encoder führt einen Reset durch und löscht den Zählwert. Diese Funktion ist hilfreich, wenn Ihr die Drehzahl eines Rades messen und regeln wollt. Wenn die Drehzahl des Rades konstant ist, solltet Ihr in genau festgelegten Zeitintervallen die gleichen Zählwerte erhalten.

Der Encoder entspricht einem 16-Bit-Zähler, und der Zählwert variiert zwischen -32768 und 32767.

Parameter:

Keine

Rückgabewerte:

Der Zählwert des linken Encoders (*int*)

Beispiel:

```
...
void loop()
{
    int encL=one.readEncL();
...
}
```

2.2.8 readEncR()

Beschreibung:

Liefert als Rückgabewert den Zählwert des rechten Encoders; der Encoder führt einen Reset durch und löscht den Zählwert. Diese Funktion ist hilfreich, wenn Ihr die Drehzahl eines Rades messen und regeln wollt. Wenn die Drehzahl des Rades konstant ist, solltet Ihr in genau festgelegten Zeitintervallen die gleichen Zählwerte erhalten.

Der Encoder entspricht einem 16-Bit-Zähler, und der Zählwert variiert zwischen -32768 und 32767.

Parameter:

Keine

Rückgabewerte:

Der Zählwert des rechten Encoders (*int*)

Beispiel:

```
...
void loop()
{
    int encR=one.readEncR();
...
}
```


2.2.9 readEncLInc()

Beschreibung:

Liefert als Rückgabewert den gelesenen inkrementellen Zählwert für den linken Encoder. Diese Funktion ist hilfreich, wenn die von einem Rad zurückgelegte Strecke kontrolliert werden soll. Wenn man weiß, dass eine Erhöhung des Encoder-Wertes um eine Einheit einer bestimmten Strecke entspricht, kann man die von einem Rad zurückgelegte Strecke kontrollieren. Der Encoder-Wert wird nicht zurückgesetzt. Der Encoder entspricht einem 16-Bit-Zähler, und der Zählwert liegt zwischen -32768 und 32767. Der Nutzer sollte den Encoder-Überlauf überwachen.

Parameter:

Keine

Rückgabewerte:

Inkrementeller Zählwert des linken Encoders (*int*)

Beispiel:

```
…
void loop()
{
    int encL=one.readEncLInc();
…
}
```

2.2.10 readEncRInc()

Beschreibung:

Liefert als Rückgabewert den gelesenen inkrementellen Zählwert für den rechten Encoder. Diese Funktion ist hilfreich, wenn die von einem Rad zurückgelegte Strecke kontrolliert werden soll. Wenn man weiß, dass eine Erhöhung des Encoder-Wertes um eine Einheit einer bestimmten Strecke entspricht, kann man die von einem Rad zurückgelegte Strecke kontrollieren. Der Encoder-Wert wird nicht zurückgesetzt. Der Encoder entspricht einem 16-Bit-Zähler, und der Zählwert liegt zwischen -32768 und 32767. Der Nutzer sollte die Situation des Encoder-Überlaufs überwachen.

Parameter:

Keine

Rückgabewerte:

Inkrementeller Zählwert des rechten Encoders (*int*)

Beispiel:

```
…
void loop()
{
    int encR=one.readEncRInc();
…
}
```


2.3 BEFEHLSFUNKTIONEN

2.3.1 servo1(position)

Beschreibung:

Bewegt den an SER1 angeschlossenen Servo. Der Funktionsparameter legt die Winkelposition fest, die von 0 bis 180 variiert.

Ein Standard-Servo positioniert sich selbst in einem festgelegten Winkel, der über einen Funktionsparameter übermittelt wird und zwischen 0 und 180° variiert.

Ein Dauerrotations-Servo verändert seine Drehgeschwindigkeit entsprechend dem Wert, der über den Funktionsparameter übermittelt wird. „0" entspricht der maximalen Drehung in einer bestimmten Richtung, „180" entspricht dem Maximum in die entgegengesetzte Richtung, und „90" entspricht dem Anhalten des Servos.

Parameter:

position: Die Winkelposition des Servos (*Byte-Parameter*)

Rückgabewerte:

Keine

Beispiel:

```
...
void loop()
{
    one.servo1(70);
...
}
```

2.3.2 servo2(position)

Beschreibung:

Bewegt den an SER2 angeschlossenen Servo. Der Funktionsparameter legt die Winkelposition fest, die von 0 bis 180 variiert.

Ein Standard-Servo positioniert sich selbst in einem festgelegten Winkel, der über einen Funktionsparameter übermittelt wird und zwischen 0 und 180º variiert.

Ein Dauerrotations-Servo verändert seine Drehgeschwindigkeit entsprechend dem Wert, der über den Funktionsparameter übermittelt wird. „0" entspricht der maximalen Drehung in einer bestimmten Richtung, „180" entspricht dem Maximum in die entgegengesetzte Richtung, und „90" entspricht dem Anhalten des Servos.

Parameter:

position: Die Winkelposition des Servos (*Byte-Parameter*)

Rückgabewerte:

Keine

Beispiel:

```
...
void loop()
{
    one.servo2(130);
...
}
```


2.3.3 led(state)

Beschreibung:

Schaltet die LED des Bot'n Roll ONE A ein oder aus. Der LED-Status wird durch den Parameter definiert, der an die Funktion übermittelt wird, und hat zwei mögliche Zustände:

- **0**: LED Aus
- **1**: LED An

Parameter:

state: der Zustand des LED (*Boolescher Parameter*)

Rückgabewerte:

Keine

Beispiel:

```
...
void loop()
{
    one.led(HIGH);          // LED einschalten
...
}
```

2.3.4 move(speedL,speedR)

Beschreibung:

Bewegt die Motoren des Bot'n Roll ONE A. Der Parameter legen die Drehzahl für jeden Motor (links und rechts) fest, die von -100 bis 100 reicht. Der Wert -100 entspricht der maximalen Drehzahl bei der Rückwärtsfahrt; 100 entspricht der maximalen Drehzahl bei der Vorwärtsfahrt, und bei 0 hält der Motor an.

Parameter:

speedL: Drehzahl des linken Motors (*int*)

speedR: Drehzahl des rechten Motors (int)

Rückgabewerte:

Keine

Beispiel:

```
...
void loop()
{
    one.move (70,70);          // Vorwärtsfahrt mit Drehzahl 70.
...
}
```


2.3.5 stop()

Beschreibung:

Hält beide Motoren des Bot'n Roll ONE A an. Unterbricht die Energiezufuhr zu den Motoren, die sich daraufhin frei drehen, bis sie zum Stillstand kommen; es wird kein Bremsmoment ausgeübt.

Parameter:

Keine

Rückgabewerte:

Keine

Beispiel:

```
...
void loop()
{
    one.stop ();          // Hält Motoren ohne Bremsmoment an
...
}
```

2.3.6 brake(torqueL,torqueR)

Beschreibung:

Hält die Motoren des Bot'n Roll ONE A an und übt dabei ein Bremsmoment aus. Der Bremskraft jedes Motors wird durch die Funktionsparameter festgelegt und variiert zwischen 0 und 100. Der Wert 0 entspricht dem Stoppen ohne Bremsmoment. Der Wert 100 entspricht dem Stoppen mit maximalem Bremsmoment und blockiert den Motor sofort!

Parameter:

torqueL: Bremsmoment des linken Motors (*Byte-Parameter*)

torqueR: Bremsmoment des rechten Motors (*Byte-Parameter*)

Rückgabewerte:

Keine

Beispiel:

```
...
void loop()
{
    one.brake (60,60);          // Motoren mit 60 % Bremskraft bremsen.
...
}
```


2.3.7 resetEncL()

Beschreibung:

Löscht den Zählwert des linken Encoders und setzt dessen Wert zwangsweise auf 0.

Parameter:

Keine

Rückgabewerte:

Keine

Beispiel:

```
...
void loop()
{
    one.resetEncL();        // linken Encoder zurücksetzen
...
}
```

2.3.8 resetEncR()

Beschreibung:

Löscht den Zählwert des rechten Encoders und setzt dessen Wert zwangsweise auf 0.

Parameter:

Keine

Returns:

Keine

Beispiel:

```
...
void loop()
{
    one.resetEncR();        // rechten Encoder zurücksetzen
...
}
```

2.4 LCD-SCHREIBFUNKTIONEN

2.4.1 lcdX(string[])

Beschreibung:

Zeigt in der Zeile X des LCD-Displays eine Zeichenkette (String) oder einen als Parameter versendeten Text in Anführungszeichen („zu sendender Text"). Als Parameter werden Variablen vom Typ (char) und vom Typ (constchar) akzeptiert. Ein String oder ein in Anführungszeichen gesetzter Text kann höchstens 17 Zeichen umfassen – das entspricht 16 auf dem LCD-Display angezeigten Zeichen plus dem abschließenden Zeichen „**\0**".

Der Buchstabe „X" steht für die Zeile des LCD-Displays, in die der Nutzer schreiben will, und muss entweder den Wert 1 oder 2 haben.

Parameter:

string[]: Zeichenkette mit den Zeichen, die auf dem LCD-Display angezeigt werden sollen; (*char*) oder (*constchar*).

Rückgabewerte:

Keine

Beispiel:

```
...
char string[]=" String Test ";       // den String deklarieren und initialisieren
void loop()
{
    one.lcd1(string);                //String in LCD-Zeile 1 ausgeben
    one.lcd2(" Text an LCD! ");          //Text in LCD-Zeile 2 ausgeben
...
}
```

2.4.2 lcdX(number)

Beschreibung:

Gibt in der Zeile X des LCD-Displays eine Zahl oder Variable aus, die als Parameter gesendet wird. Die Zahl kann eine Variable vom Typ (int) (unsigned int) (long int) (double) oder (float) sein.

Der Buchstabe „X" steht für die Zeile des LCD-Displays, in die der Nutzer schreiben will, und muss entweder den Wert 1 oder 2 haben.

Parameter:

number: Zahl oder Variable, die ausgegeben werden soll; (*int*), (*unsigned int*), (*long int*), (*double*) oder (*float*).

Rückgabewerte:

Keine

Beispiel:

```
...
intvar1 =-32768;                // die Variable deklarieren und initialisieren
unsigned int var2 = 0;          // die Variable deklarieren und initialisieren
long int var3 = - 2147483648;   // die Variable deklarieren und initialisieren
float var4=123.12;              // die Variable deklarieren und initialisieren
double var5=123.12;             // die Variable deklarieren und initialisieren

void loop()
{
    one.lcd1(var1);             //Variablenwert am LCD-Display ausgeben
    ...
    one.lcd2(var2);             //Variablenwert am LCD-Display ausgeben
    ...
    one.lcd1(var3);             //Variablenwert am LCD-Display ausgeben
    ...
    one.lcd2(var4);             //Variablenwert am LCD-Display ausgeben
    ...
    one.lcd1(var5);             //Variablenwert am LCD-Display ausgeben
    ...
    one.lcd2(32767);            //Variablenwert am LCD-Display ausgeben
    ...
    one.lcd1(2147483647);       //große Zahl am LCD-Display ausgeben
    ...
    one.lcd2(321.01);           //Zahl mit einfacher Genauigkeit am LCD-Display ausgeben
...
}
```


2.4.3 lcdX(string[],number)

Beschreibung:

Gibt in der Zeile X des LCD-Displays den Text und die Zahl aus, die als Parameter gesendet werden. Als Text akzeptiert werden Variable vom Typ (constchar) oder Text in Anführungszeichen. Die Anzahl der Zeichen, die ausgegeben werden sollen, sollte höchstens 16 betragen. Die Zahl kann eine Variable vom Typ (int) (unsigned int) (long int) (double) oder (float) sein.

Der Buchstabe „X" steht für die Zeile des LCD-Displays, in die der Nutzer schreiben will, und muss entweder den Wert 1 oder 2 haben.

Parameter:

string[]: Zeichen, die am LCD-Display ausgegeben werden sollen; (*constchar*).

number: Zahl oder Variable, die ausgegeben werden soll; (*int*), (*unsigned int*), (*long int*), (*double*) oder (*float*).

Rückgabewerte:

Keine

Beispiel:

```
...
intvar1 =-32768;                // die Variable deklarieren und initialisieren
unsigned int var2 = 0;          // die Variable deklarieren und initialisieren
long int var3 = - 2147483648;   // die Variable deklarieren und initialisieren
float var4=123.12;              // die Variable deklarieren und initialisieren
double var5=123.12;             // die Variable deklarieren und initialisieren

void loop()
{
    one.lcd1("Text: ", var1);              //Text und einen Variablenwert am LCD-Display ausgeben
    ...
    one.lcd2("Text: ", var2);              //Text und einen Variablenwert am LCD-Display ausgeben
    ...
    one.lcd1("Text: ",var3);               //Text und einen Variablenwert am LCD-Display ausgeben
    ...
    one.lcd2("Text: ",var4);               //Text und einen Variablenwert am LCD-Display ausgeben
    ...
    one.lcd1("Text: ",var5);               //Text und einen Variablenwert am LCD-Display ausgeben
    ...
    one.lcd2("Text: ",32767);              //Text und einen Variablenwert am LCD-Display ausgeben
    ...
    one.lcd1("Text: ",2147483647);         //Text und eine große Zahl am LCD-Display ausgeben
    ...
    one.lcd2("Text: ",321.01);             //Text und eine Zahl mit einfacher Genauigkeit  am LCD-Display ausgeben
...
}
```


2.4.4 lcdX(num1, num2)

Beschreibung:

Gibt in der Zeile X des LCD-Displays Zahlen oder Variable aus, die als Parameter gesendet werden. Zahlen können Variable des Typs (int) oder (unsigned int) sein.

Der Buchstabe „X" steht für die Zeile des LCD-Displays, in die der Nutzer schreiben will, und muss entweder den Wert 1 oder 2 haben.

Parameter:

num1: Zahl oder Variable, die ausgegeben werden soll; (*int*) oder (*unsigned int*).

num2: Zahl oder Variable, die ausgegeben werden soll; (*int*) oder (*unsigned int*).

Rückgabewerte:

Keine

Beispiel:

```
...
intvar1 =-32768;                // die Variable deklarieren und initialisieren
unsigned int var2 = 0;          // die Variable deklarieren und initialisieren

void loop()
{
    one.lcd1(var1 , 32767);             //Variable und Zahl am LCD-Display ausgeben
    ...
    one.lcd2(var2 , 65535);             //Variable und Zahl am LCD-Display ausgeben
...
}
```

2.4.5 lcdX(num1, num2, num3)

Beschreibung:

Gibt in der Zeile X des LCD-Displays Zahlen oder Variable aus, die als Parameter gesendet werden. Zahlen können Variable des Typs (int) oder (unsigned int) sein.

Der Buchstabe „X" steht für die Zeile des LCD-Displays, in die der Nutzer schreiben will, und muss entweder den Wert 1 oder 2 haben.

Parameter:

num1: Zahl oder Variable, die ausgegeben werden soll; (*int*) oder (*unsigned int*).

num2: Zahl oder Variable, die ausgegeben werden soll; (*int*) oder (*unsigned int*).

num3: Zahl oder Variable, die ausgegeben werden soll; (*int*) oder (*unsigned int*).

Rückgabewerte:

Keine

Beispiel:

```
...
intvar1 =-32768;                    // die Variable deklarieren und initialisieren
unsigned int var2 = 0;              // die Variable deklarieren und initialisieren

void loop()
{
        one.lcd1(var1 ,32767, var2);          //Variablenwert und Zahl am LCD-Display ausgeben
...
}
```

2.4.6 lcdX(num1 , num2 , num3 , num4)

Beschreibung:

Gibt in der Zeile X des LCD-Displays Zahlen oder Variable aus, die als Parameter gesendet werden. Zahlen können Variable des Typs (int) oder (unsigned int) sein.

Der Buchstabe „X" steht für die Zeile des LCD-Displays, in die der Nutzer schreiben will, und muss entweder den Wert 1 oder 2 haben.

Parameter:

num1: Zahl oder Variable, die ausgegeben werden soll; (*int*) oder (*unsigned int*).

num2: Zahl oder Variable, die ausgegeben werden soll; (*int*) oder (*unsigned int*).

num3: Zahl oder Variable, die ausgegeben werden soll; (*int*) oder (*unsigned int*).

num4: Zahl oder Variable, die ausgegeben werden soll; (*int*) oder (*unsigned int*).

Rückgabewerte:

Keine

Beispiel:

```
...
intvar1 =-32768;                // die Variable deklarieren und initialisieren
unsigned int var2 = 0;          // die Variable deklarieren und initialisieren

void loop()
{
    one.lcd2(var1 ,32767, var2 , 65535);    //Variablenwert und Zahl am LCD-Display ausgeben
...
}
```

ANHANG A: VCP-TREIBER FÜR DEN USB-SERIELL(RS232)-WANDLER INSTALLIEREN

Dieser Treiber sorgt dafür, dass das Betriebssystem Eures Computers mit dem Bot'n Roll ONE A kommunizieren kann.

Um den Treiber zu installieren, geht auf die Support-Website für den Bot'n Roll ONE A auf http://botnroll.com/onea/ und ladet Euch den „**VCP Driver - Windows**" oder den „**VCP Driver - Mac OS X**" herunter, indem Ihr den Link anklickt, der zu Eurem Betriebssystem passt. Nach dem Download entpackt Ihr die „*.zip*"-Datei und startet die Anwendung.

Jedes Mal, wenn Ihr Euren Roboter mit dem USB-Kabel an Euren Computer anschließt, wird ein virtueller COM-Port (VCP) erzeugt, über den die Kommunikation zwischen Eurem Bot'n Roll ONE A und Eurem PC abgewickelt wird. Die Umgebung für die Anwendungsprogrammierung nutzt den Port für die Kommunikation mit dem Bot'n Roll ONE A und überträgt zu diesem Zweck Eure Programme auf Euren Roboter.

Bei dem USB-Seriell-Wandler, den der Bot'n Roll ONE A verwendet, handelt es sich um einen **PoUSB12** von *PoLabs*, der mit der ***Bridge* CP2102** von *Silicon Labs* arbeitet.

ANHANG B: DIE ARDUINO-PROGRAMMIERUMGEBUNG

Programmiert wird der Roboter mit der Software Arduino IDE. Diese Anwendung wird benötigt, damit Ihr die Programme in der Programmiersprache C bearbeiten könnt. Die Software wird auch dazu verwendet, Eure Programme auf den Bot'n Roll ONE A zu überspielen.

B.1 ARDUINO IDE INSTALLIEREN

Um die Software Arduino IDE zu installieren, geht bitte auf die Support-Website für den Bot'n Roll ONE A auf http://botnroll.com/onea/ und klickt dort auf „**Arduino IDE - Windows**" oder „**Arduino IDE - Mac OS X**", um die für Euer Betriebssystem passende Software herunterzuladen.

Nach dem Download entpackt die .zip-Datei und kopiert sie in einen Ordner Eurer Wahl auf Eurem Computer.

Dieser Ordner enthält mehrere Unterordner und Dateien – darunter auch die Anwendung „**arduino**". Diese ausführbare Datei startet die Software Arduino IDE. Sehr wichtig ist auch der Unterordner „***libraries***": Er enthält alle Arduino-Libraries. Die Libraries sind Euer Handwerkszeug für das Programmieren.

B.2 DIE **BNRONEA** LIBRARY AUF ARDUINO INSTALLIEREN

Die **BnrOneA** Library wurde von **botnroll.com** für Arduino IDE entwickelt und enthält alle nötigen Befehle für die Steuerung des Roboters. Diese Library muss auf Arduino IDE installiert werden.

Ladet Euch von der Support-Website für den Bot'n Roll ONE A auf http://botnroll.com/onea/ die Datei **BnrOneA.zip** herunter, indem Ihr auf „**Arduino Library**" klickt.

Entpackt die Datei und speichert den extrahierten Ordner „**BnrOneA**" im Verzeichnis „***libraries***", das im vorhergehenden Abschnitt beschrieben wurde. Jetzt habt Ihr alles, was Ihr braucht, um Euren **Bot'n Roll ONE A** zu programmieren!

B.3 DIE KOMMUNIKATION MIT DEM ROBOTER KONFIGURIEREN

Bevor Ihr diesen Schritt in Angriff nehmt, vergewissert Euch bitte, dass Ihr den VCP-Treiber korrekt installiert habt. Schließt den Bot'n Roll ONE A mit dem mitgelieferten USB-Kabel an Euren Computer an. Jetzt wird automatisch ein COM-Port für die Kommunikation mit dem Roboter zugewiesen.

Öffnet jetzt Arduino IDE, klickt auf die Option „***Tools -> Board***" und wählt unter „**Board**" (Leiterplatte) „**Arduino Uno**" aus. Der **Bot´n Roll ONE A** wird nun so programmiert, als wäre er ein Arduino Uno.

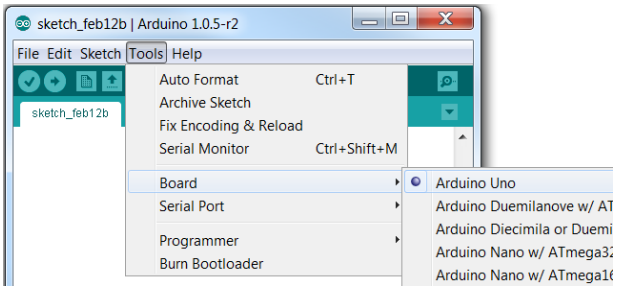


Abbildung 2: Die Leiterplatte auswählen, die programmiert werden soll

Wählt unter „***Tools -> Serial Port***" den richtigen COM-Port aus, der dem **Bot'n Roll ONE A** zugewiesen wurde.

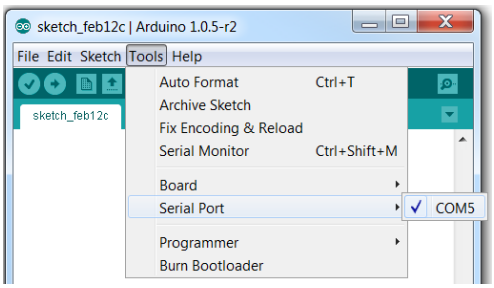


Abbildung 3: Den seriellen Port auswählen

Falls hier kein COM-Port angeboten wird, habt Ihr vermutlich den VCP-Treiber für den USB-Seriell-Wandler nicht korrekt installiert.

Öffnet den Windows Gerätemanager und sucht nach dem Eintrag „Anschlüsse (COM and LPT)". Wenn Ihr diesen Eintrag anklickt, werden alle verfügbaren Kommunikationsanschlüsse (COM-Anschlüsse) angezeigt.

„***Silicon Labs CP210x USB to UART Bridge***" ist die Bezeichnung für den Kommunikationsanschluss, der mit dem Bot'n Roll ONE A „sprechen" kann (in dem abgebildeten Beispiel wurde der Port **COM5** zugewiesen).

Wenn der Eintrag „***Silicon Labs CP210x USB to UART Bridge***" nicht in der Liste erscheint, müsst Ihr den VCP-Treiber korrekt installieren.

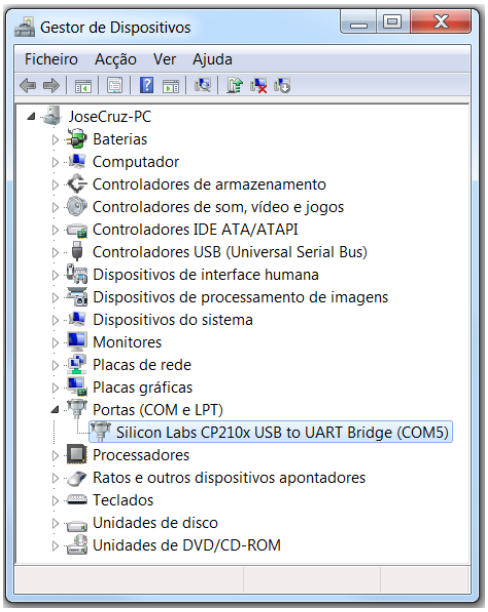


Abbildung 4: Kommunikationsanschlüsse (COM-Ports) im Windows Gerätemanager

B.4 EIN PROGRAMM AUF DEN BOT'N ROLL ONE A LADEN

In der Anwendung Arduino IDE findet Ihr verschiedene Beispielprogramme, die Ihr auf Euren Roboter laden könnt.

Klickt auf „***File -> Examples -> 01.Basics -> Blink***“. Es öffnet sich ein neues Fenster mit dem Code für das gewählte Beispiel.

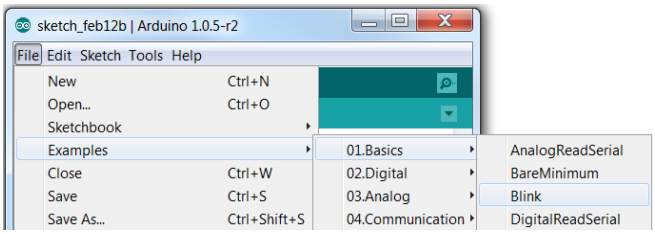


Abbildung 5: Ein Beispielprogramm laden

Klickt auf „***File -> Upload***“ oder auf das Pfeil-nach-rechts-Symbol, um das Programm an den Roboter zu senden. Sobald der Upload abgeschlossen ist, muss die gelbe LED im Sekundentakt blinken!

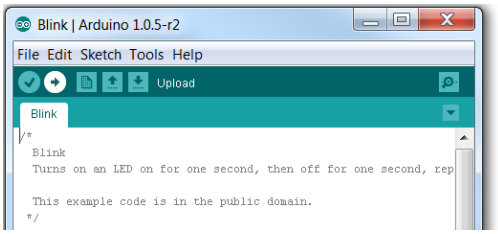


Abbildung 6: Ein Programm an den Roboter senden

Wenn Ihr auf „***File -> Examples -> BnrOneA-> ...***“ klickt, findet Ihr alle Programme, die botnroll.com speziell für den Bot'n Roll ONE A entwickelt hat.

Unter „***File -> Examples -> BnrOneA-> Basic -> ...***“ findet Ihr die einfachsten Programme, die vor allem dazu dienen, um die komplette Hardware Eures Roboters zu testen. Mit diesen kleinen Programmen müsst Ihr Euch gründlich vertraut machen!

Unter „***File -> Examples -> BnrOneA-> Advanced -> ...***“ gibt es einige fortgeschrittenere Programme, die für Euch womöglich erst verständlich sind, wenn Ihr mit den einfachen Programmen vertraut seid.

Unter „***File -> Examples -> BnrOneA-> Extra -> ...***“ findet Ihr Programme, die mit den Extras des Bot'n Roll ONE A zusammenhängen, die den **Bot´n Roll ONE A** noch leistungsfähiger machen.

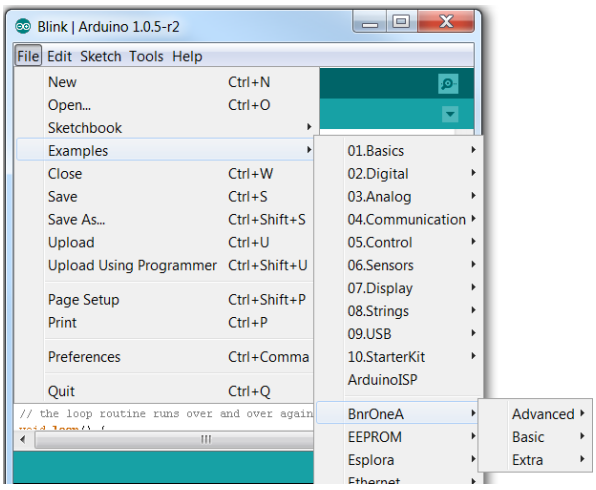


Abbildung 7: Programme aus der BnrOneA Library

