



Der Arduino und die Entwicklungsumgebung

Das Herz und Gehirn des selbstfahrenden Autos ist der Arduino.

In dem Bereich ‚setup‘ müssen alle Befehle zur Initialisierung (also zur Einstellung der Ausgangssituation) vorgenommen werden. Im Bereich ‚loop‘ stehen die Befehle zur fortlaufenden Steuerung der Bewegung.

Bevor wir anfangen können, Programmcode zu schreiben, müssen wir erst verstehen, wie die Entstehung von Signalen und von Aktionen funktionieren.

Roboterbausätze haben Anschlüsse, die mit den Motoren bzw. den Sensoren verbunden sind. Diese Anschlüsse (Pins) sorgen dafür, dass durch ein Spannungssignal (logisch 1) der Motor dreht, bzw. dass durch Spannungen die Messwerte der Sensoren kodiert werden.

Das folgende Beispiel orientiert sich am Eleego-Bausatz eines Roboter-Autos. Andere Bausätze sind aber analog aufgebaut.

Die notwendigen Pins für die Motoren werden über Variablendefinitionen festgelegt. Hier:

```
//define L298n module IO Pin  
#define ENA 5  
#define ENB 6  
#define IN1 7  
#define IN2 8  
#define IN3 9  
#define IN4 11
```

Im vorliegenden Bausatz wird das Modul L298n zum Steuern der Motoren verwendet. Die Pins ENA und ENB aktivieren oder sperren die beiden Motoren für die Eingangssignale. Mit den Pins IN1-4 können die Motoren vorwärts bzw. rückwärts gedreht werden.

Dabei gelten für die Pins IN1-4 folgende Regeln:

- IN1 High-Pegel und IN2 Low-Pegel: Motor A (links) dreht vorwärts; Pegel invers: Motor dreht rückwärts
- IN3 Low-pegel und IN4 High-Pegel: Motor B (rechts) dreht vorwärts; Pegel invers: Motor dreht rückwärts
- Jeweils beide Pins gleicher Pegel: Motor steht still (stoppt).

Mit dem Befehl

```
digitalWrite(ENA,HIGH);
```

kann man einen Pegel auf dem Pin ausgeben. In dem Beispiel wird an ENA (Pin 5) eine logische 1 (High-Signal) ausgegeben.

Um die Pins in der gewünschten Richtung betreiben zu können, muss man festlegen, ob die Signale nur ausgegeben werden (OUTPUT), nur gelesen werden (INPUT) oder beides möglich sein soll (INOUT). Hier initialisieren wir die Pins als reine Ausgänge.



```
void setup() {  
  //öffnet die serielle Schnittstelle für Informationsausgaben  
  Serial.begin(9600);  
  //vor der Benutzung muss der Pin-Mode festgelegt werden  
  pinMode(IN1,OUTPUT);  
  pinMode(IN2,OUTPUT);  
  pinMode(IN3,OUTPUT);  
  pinMode(IN4,OUTPUT);  
  pinMode(ENA,OUTPUT);  
  pinMode(ENB,OUTPUT);  
}
```

Der grundlegende (schematische) Aufbau des Codes in der Arduino-Programmdatei sollte so aussehen:

```
//beispiel_1.ino  
<#include <somefile.h>>
```

Bei Bedarf können weitere Funktionen in andere Dateien ausgelagert werden und als Bibliotheksfunktionen wie in Standard-C importiert werden.

```
// Definitionen  
#define <Konstante> <Wert>
```

Alle notwendigen eigenen Definitionen:

- Konstanten
- eigene Typen
- Variablen

```
// lokale Typdefinitionen  
typedef unsigned char u8;
```

```
// lokale Variablen  
int duration = 0;
```

```
// lokale (Hilfs)Funktionen  
<Typ> funktion(<Parameter>){  
  ...  
}
```

Dann kommen lokale Hilfsfunktionen, z.B. zum Anhalten des Motors

```
void setup() {  
  ... // Initialisierungen  
}
```

Die setup()-Funktion wird ganz zu Anfang nur einmal ausgeführt

```
// Wiederholte Ausführungen  
void loop() {  
  ...  
}
```

Die loop()-Funktion wird endlos ausgeführt, sofern sie nicht extern beendet wird.

Hinweis: in der obigen Graphik wurden spitze Klammern für optionale Elemente verwendet. Der Code ist jedoch Standard-C++. Damit werden allerdings Standardbibliotheken ebenfalls mit den spitzen Klammern eingebunden (siehe somefile.h), während eigene Bibliotheken mit doppelten Anführungszeichen angegeben werden.