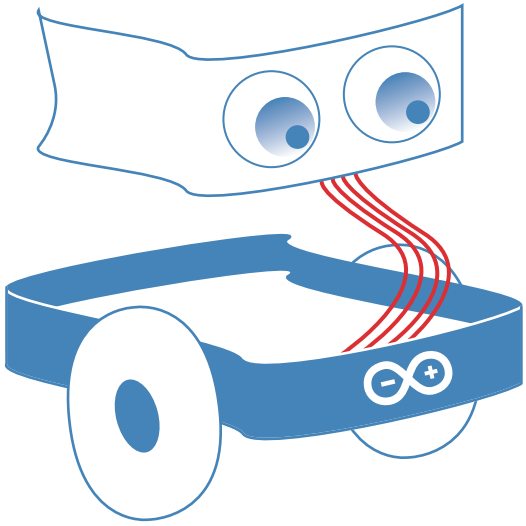
ROBOTIK MIT ARDUINO



ROBINO

Bau- und Programmieranleitungen nebst vielen technischen Details zum ultraschallgesteuerten Fahrzeug

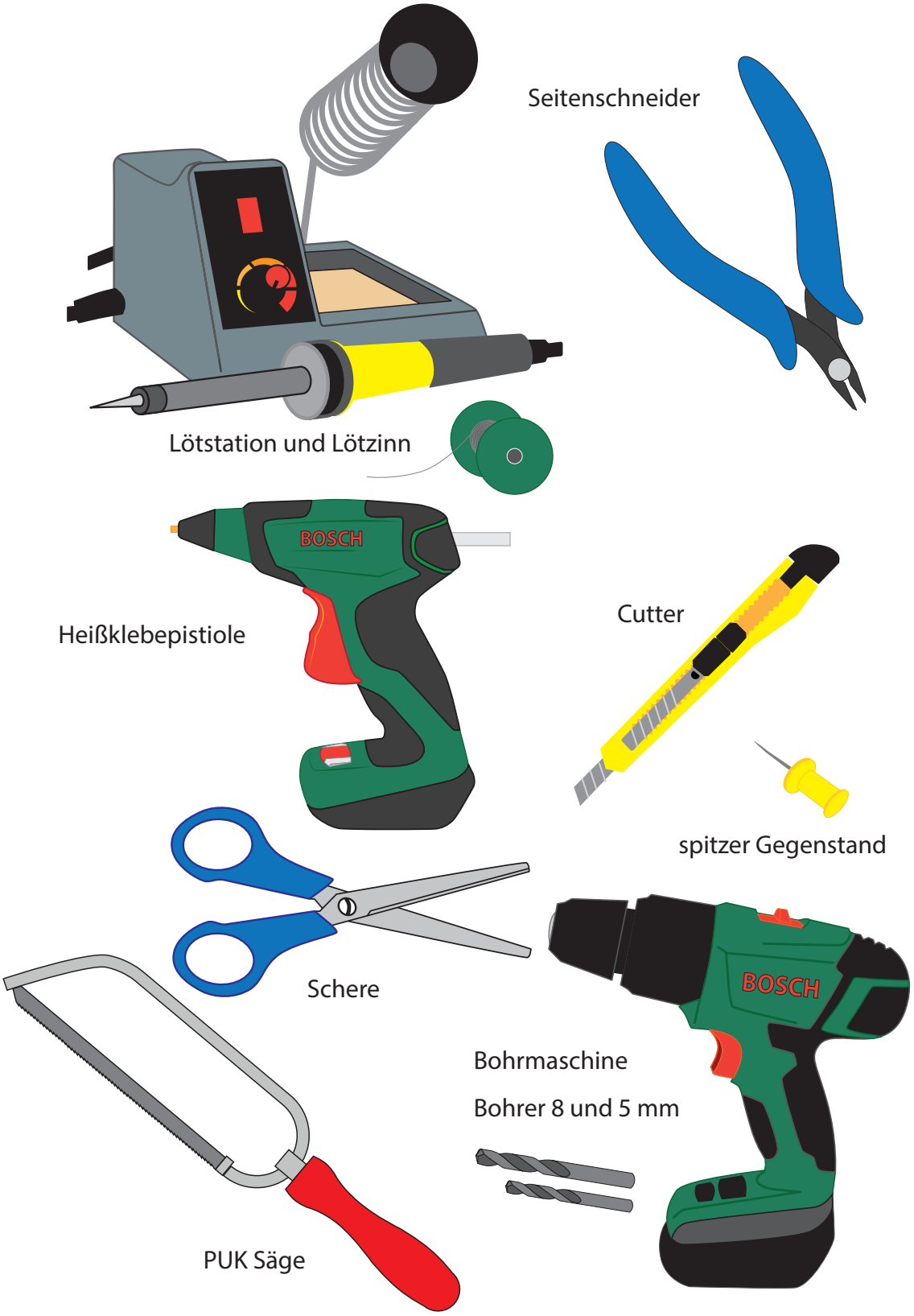
mint-unt.de

Inhalt Seiten

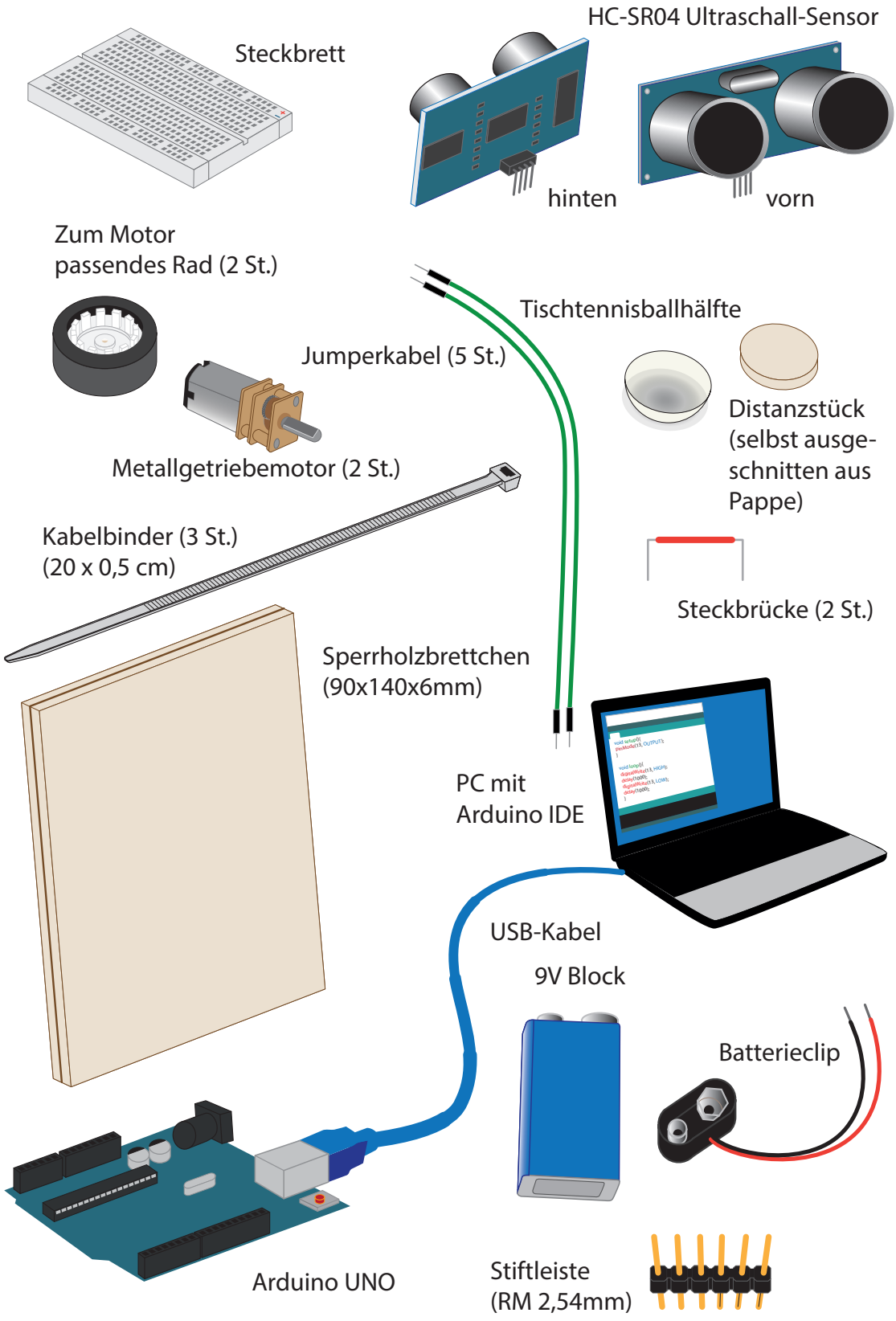
Diese Handreichung ist verlinkt.
Über einen Klick auf Textteile des Inhaltsverzeichnisses oben gelangt man zur angegebenen Seite. Von dort führt ein Klick auf die Nummerierung einer jeden Seite wieder hierher zurück .

Gerd Stein * Glüsinger Str. 7 * 21481 Schnakenbek * http://mint-unt.de

Werkzeuge, die so oder so ähnlich für die in dieser Anleitung beschriebenen Arbeiten gebraucht werden.

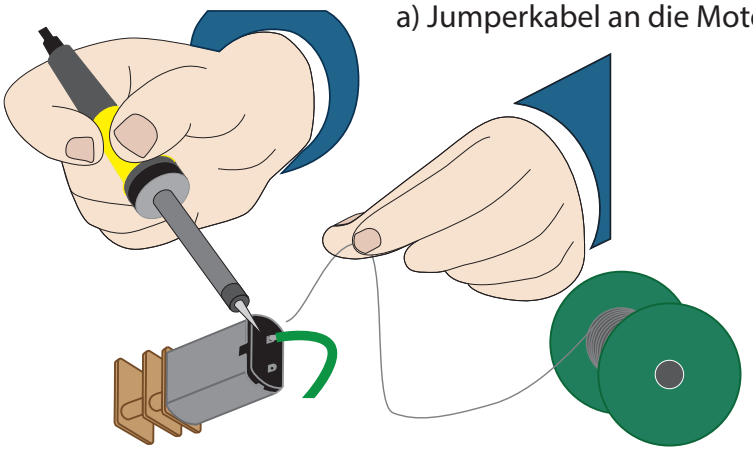


Zusammenschau der zum Bau des Robino benötigten Materialien (Bezugsquellen am Ende dieser Handreichung).

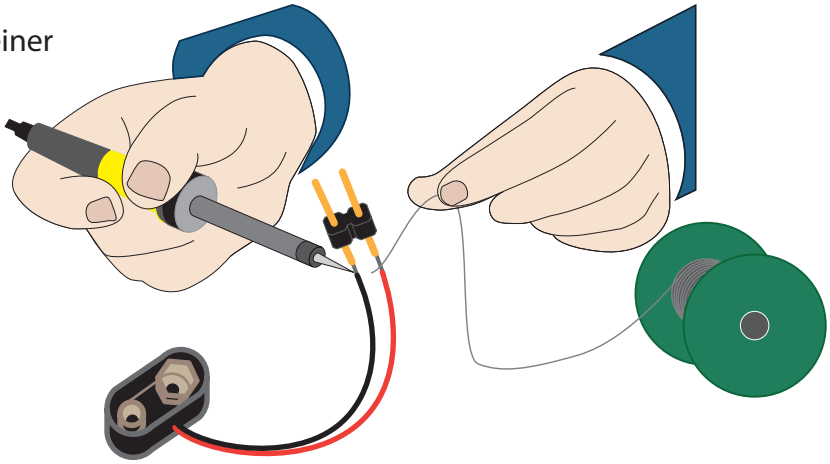


Löt- und Sicherungsarbeiten

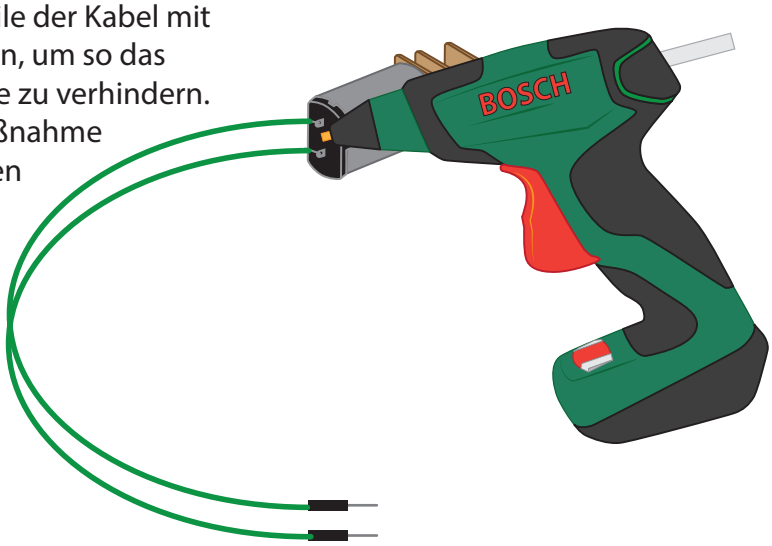
a) Jumperkabel an die Motorösen löten



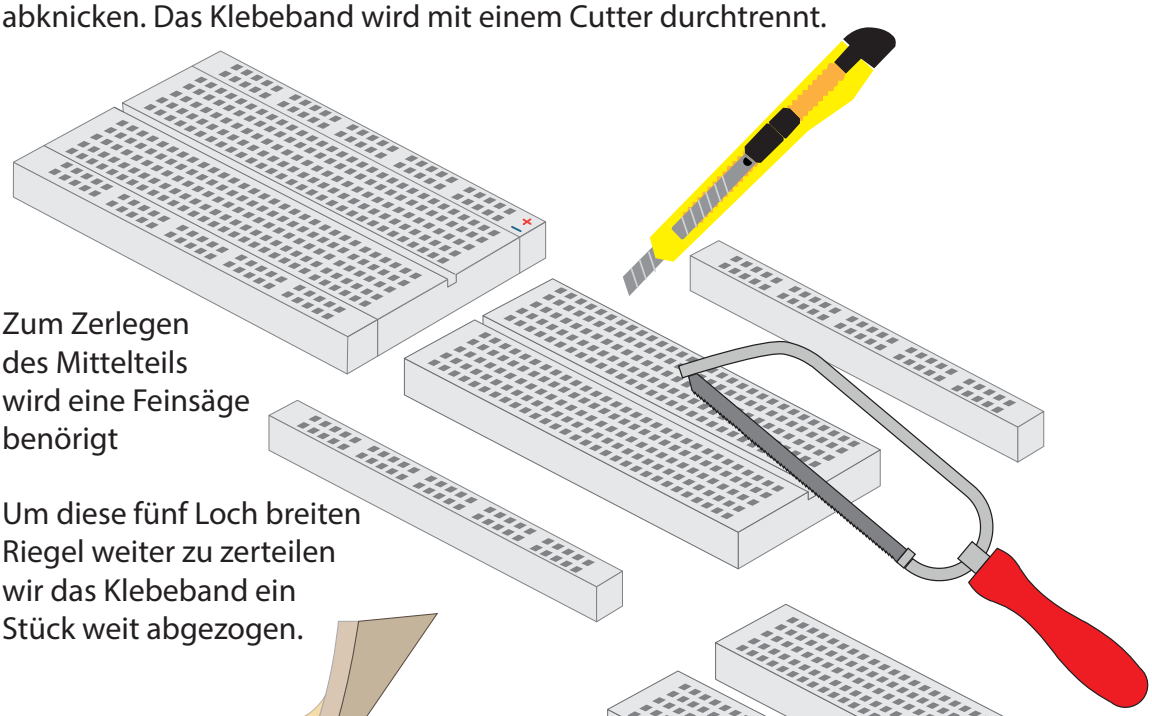
b) An die beiden Kabelenden des Batterieclips ein 2-poliges Stück einer Stiftleiste löten.



c) Die Lötstellen und Teile der Kabel mit Heißkleber überziehen, um so das Abbrechen der Drähte zu verhindern. Diese Sicherungsmaßnahme auch an den Lötstellen der Stiftleiste durchführen.

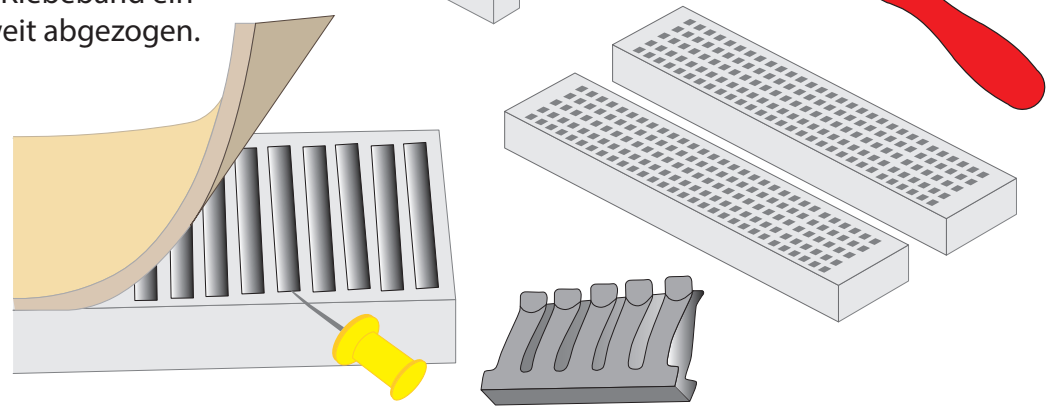


Um den Ultraschallsensor zu befestigen, wird ein Stück von einem Breadboard benötigt, das man wie folgt zerteilen kann: Die Seiten lassen sich wie Schokoriegel abknicken. Das Klebeband wird mit einem Cutter durchtrennt.



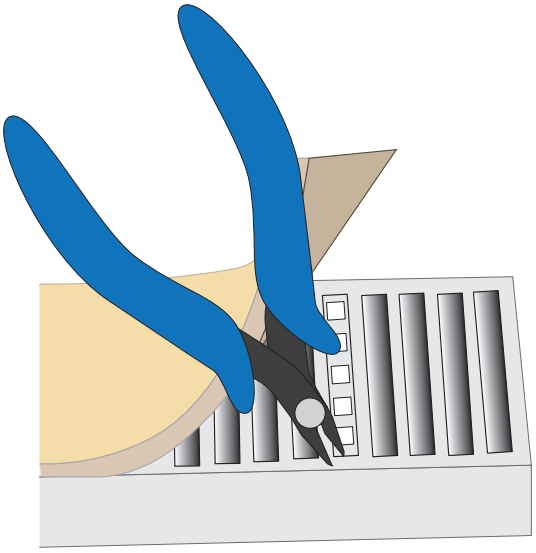
Zum Zerlegen des Mittelteils wird eine Feinsäge benörigt

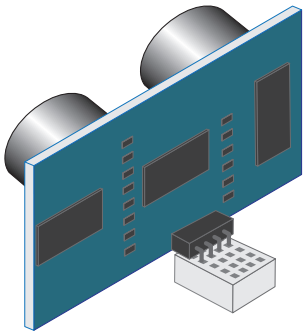
Um diese fünf Loch breiten Riegel weiter zu zerteilen wir das Klebeband ein Stück weit abgezogen.



Mit einem spitzen Gegenstand kann das Kontaktblech herausgehebelt werden.

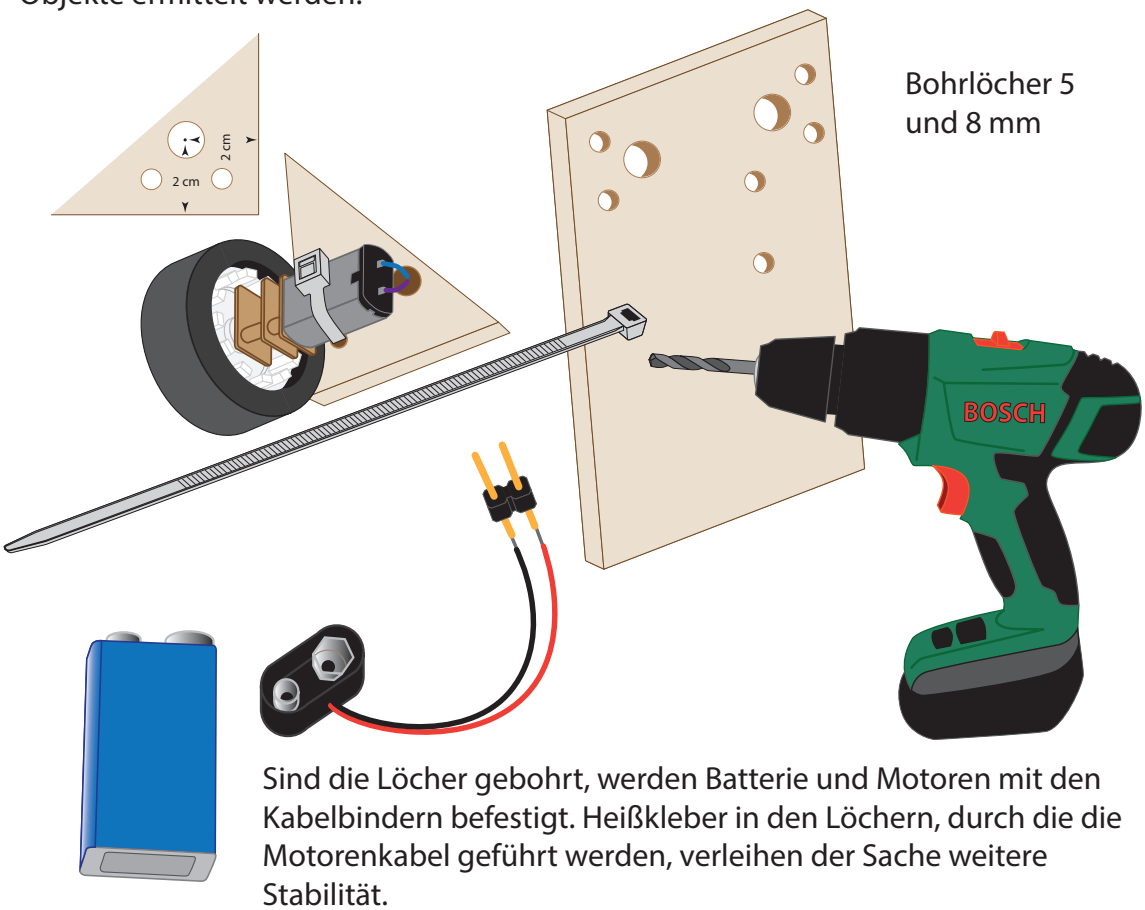
In der entstehenden Lücke lässt sich ein Seitenschneider ansetzen. Das gewünschte Stück wird abgetrennt ...





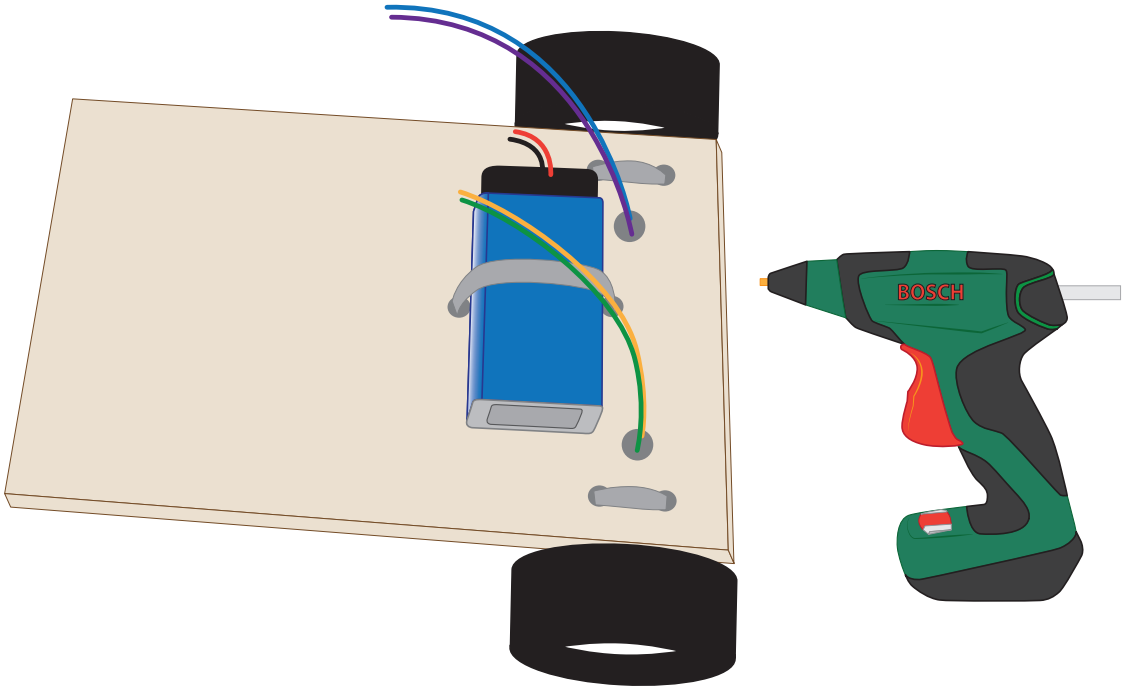
... und auf die Kontaktstifte des Sensors gesteckt. So kann dieser später leicht auf dem Brettchen des Robino befestigt werden.

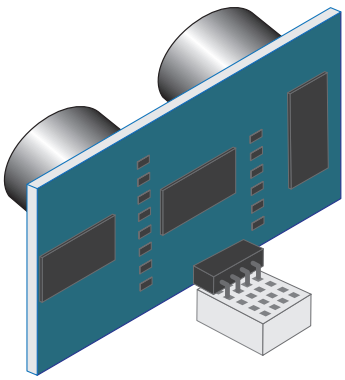
Für den Zusammenbau des Robino werden zuerst zwei 8 mm Löcher im Abstand von 2 cm von den Kanten in das Brettchen gebohrt. Die Lage der 5 mm Löcher zum Durchführen der Kabelbinder kann über die Platzierung der zu befestigenden Objekte ermittelt werden.

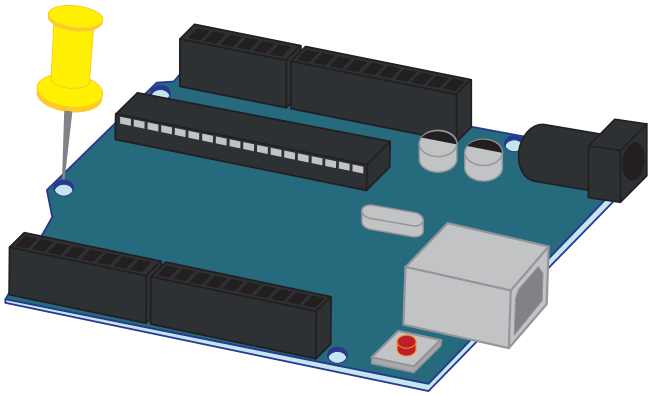


Bohrlöcher 5 und 8 mm

Sind die Löcher gebohrt, werden Batterie und Motoren mit den Kabelbindern befestigt. Heißkleber in den Löchern, durch die die Motorenkabel geführt werden, verleihen der Sache weitere Stabilität.

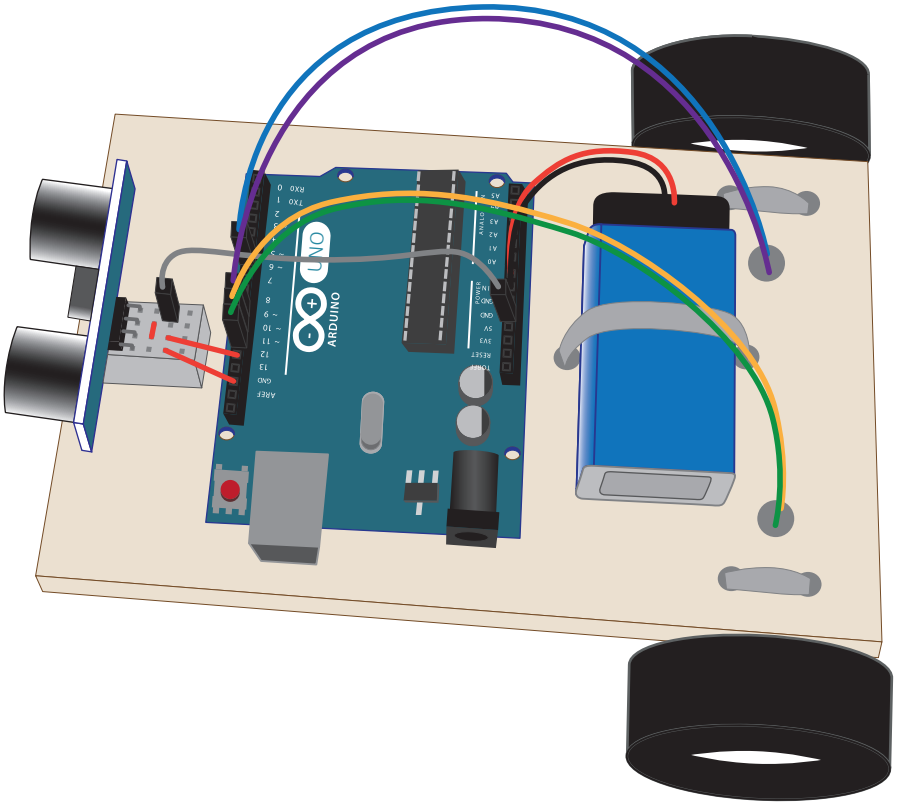




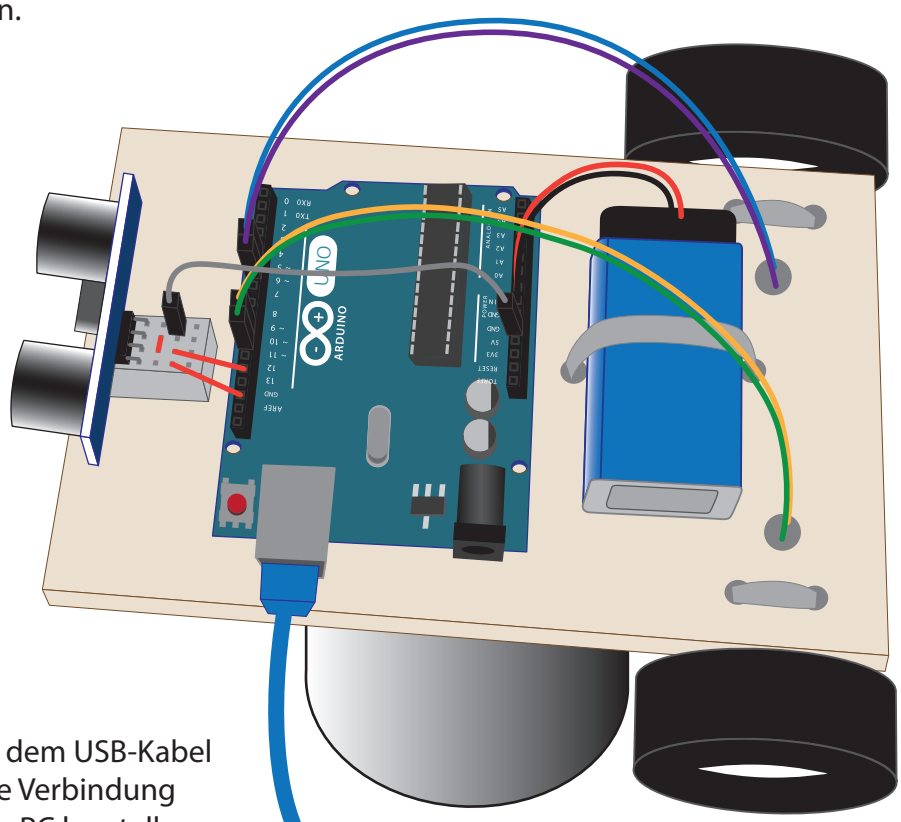


Sodann Arduino auf dem Brettchen platzieren, ggfs. Lage der Öffnungen mithilfe eines spitzen Werkzeugs markieren und mit kleinen Schrauben festsetzen.
Motor hinten rechts über die Jumperkabel mit den Pins 11 und 10, Motor hinten links mit den Pins 9 und 6 verbinden.
Ultraschallsensor mit dem Abschnitt eines Steckbretts festkleben und den Sensor mit Steckbrücken wie folgt anschließen:

GND GND
Echo } überbrücken und mit
Trig } Pin 12 verbinden
VCC 5V

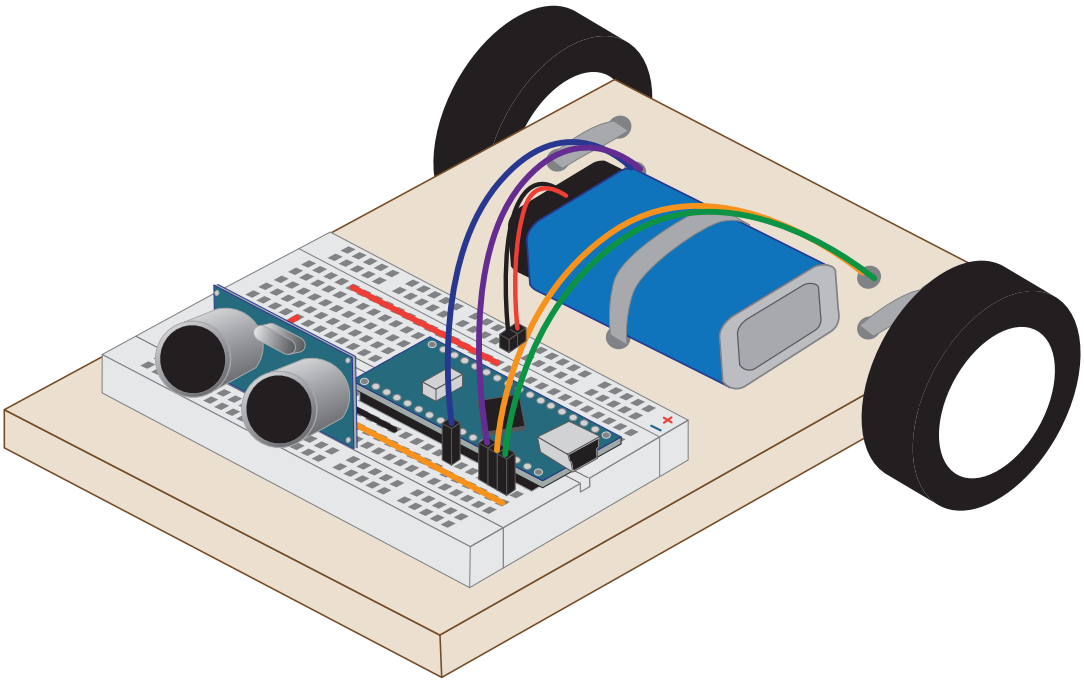


Für erste Programmierübungen Fahrzeug so aufbocken, dass die Räder frei drehen können.

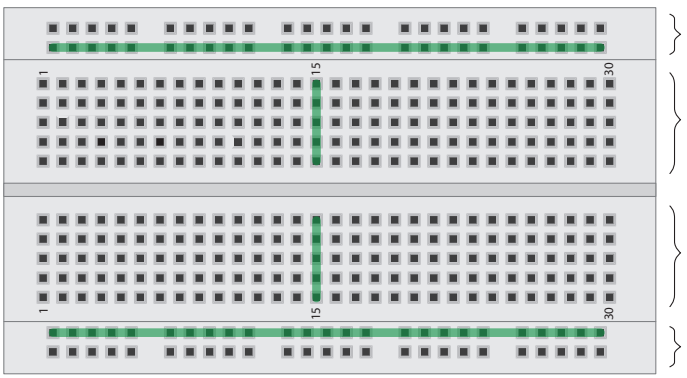


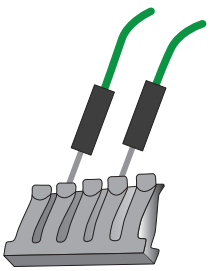
Mit dem USB-Kabel eine Verbindung zum PC herstellen und ihn starten.

Alternativ zum Arduino UNO könnte auch der preisgünstigere Arduino Nano zur Realisierung des programmierbaren Fahrzeugs eingesetzt werden. Bei Verwendung eines Steckbretts würde das Fahrzeug dann so aussehen:



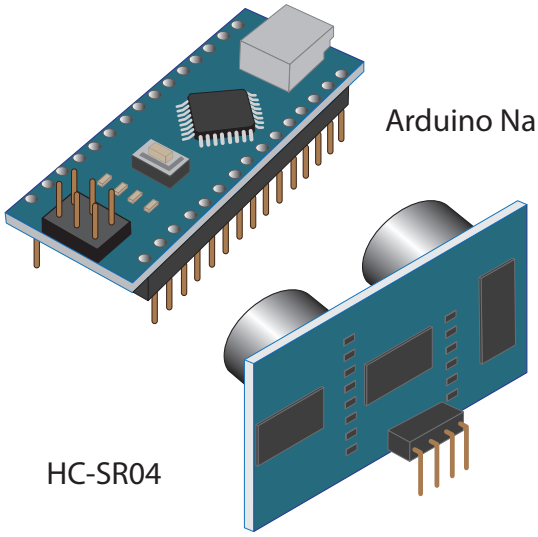
Steckbretter (Breadboards) gibt es in unterschiedlichen Größen. Das hier abgebildete hat 400 Löcher. Die Lochabstände sind genormt auf ein Maß von 2,54 mm. Unter den Löchern im isolierenden Kunststoff eingebettet finden sich Kontaktbleche, die bestimmte Löcher in einer Reihe elektrisch leitfähig miteinander verbinden. Die grünen Linien sollen verdeutlichen, wie diese Bleche ausgerichtet sind. An den Rändern sind das jeweils zwei Reihen. Jede Reihe hat 25 Löcher, die miteinander verbunden sind. Im Mittelteil finden sich zwei Abschnitte mit jeweils 30 Fünferreihen. Die sehr viel kürzeren Bleche sind in einem Winkel von 90 Grad zu den vorher beschriebenen angeordnet und liefern so 5 Kontaktpunkte (vgl. Abb. unten rechts).





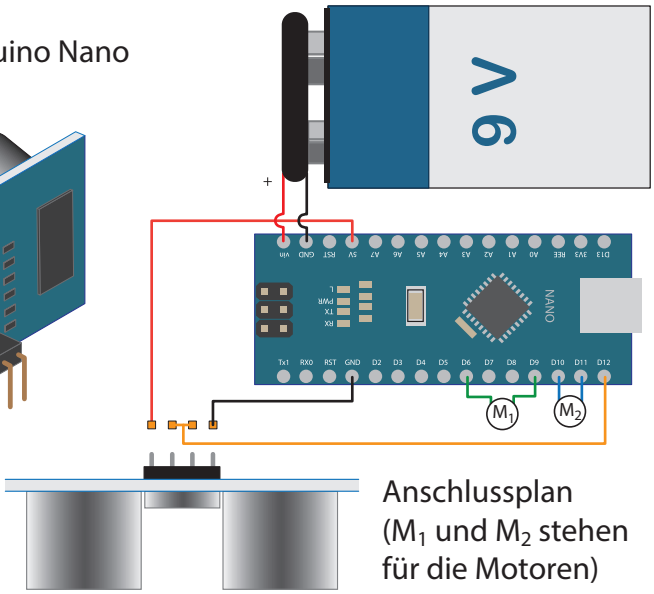
Kontaktblech

Die Verwendung eines solchen Steckbretts erklärt sich aus dem Umstand, dass beim Arduino Nano die Anschlüsse nicht über Buchsen-, sondern über Stiftleisten nach außen geführt sind. So können er und der ebenfalls mit Stiften ausgestattete Ultraschallsensor auf dem 400er Breadboard zusammengeführt werden.



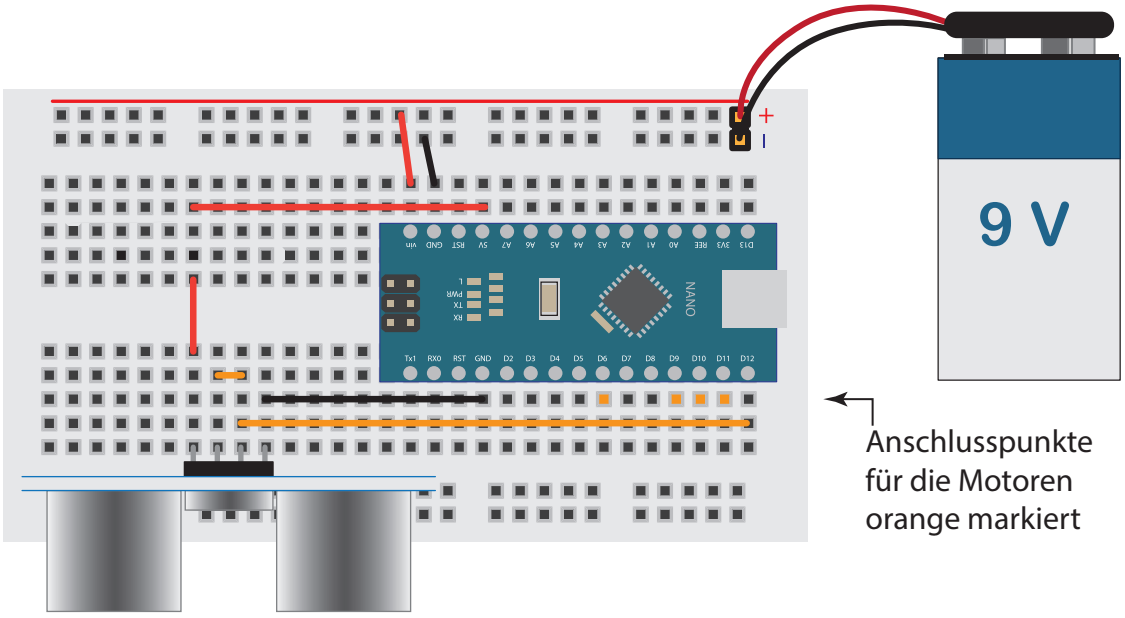
Arduino Nano

HC-SR04



Anschlussplan (M_1 und M_2 stehen für die Motoren)

Die nachfolgende Abbildung zeigt, wie Arduino Nano und Ultraschallsensor auf dem Breadboard platziert und die Anschlüsse realisiert werden können.



Anschlusspunkte für die Motoren orange markiert

Zum Verbinden konkreter Kontaktpunkte auf dem Breadboard müssen die vorgefertigten Drahtbrücken oft in der Länge angepasst werden. Die nachfolgenden Abbildungen zeigen, wie das gelingt. Die ausgewählten Kontaktstellen sind markiert (Abb.1) und eine hinreichend lange Drahtbrücke zur Hand. Sie wird an einem Punkt eingesteckt und 2 - 3 Löcher über das Ziel hinaus abgezwackt (Abb.2).

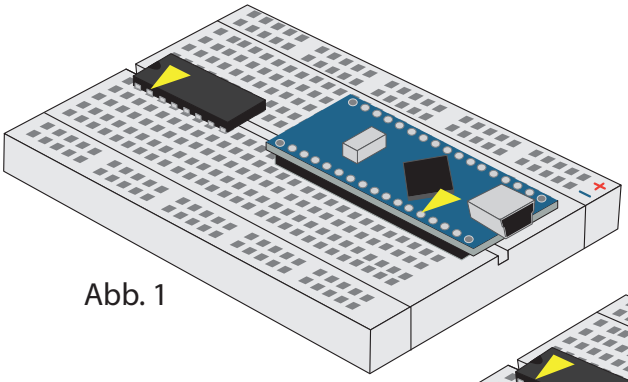


Abb. 1

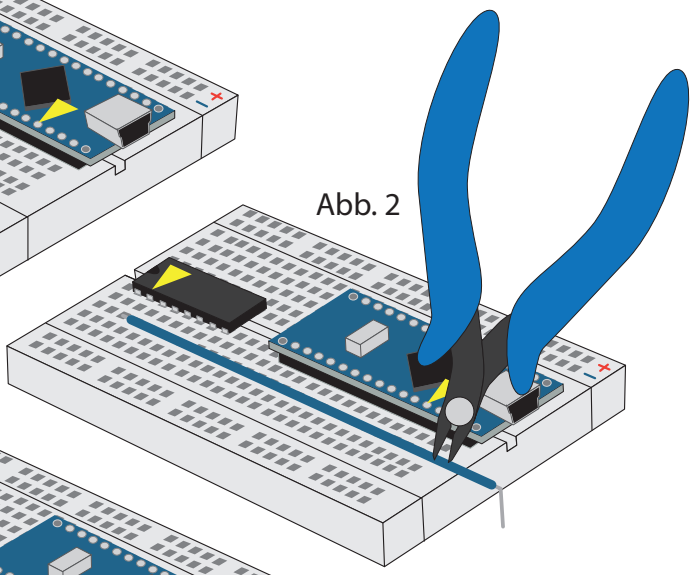


Abb. 2

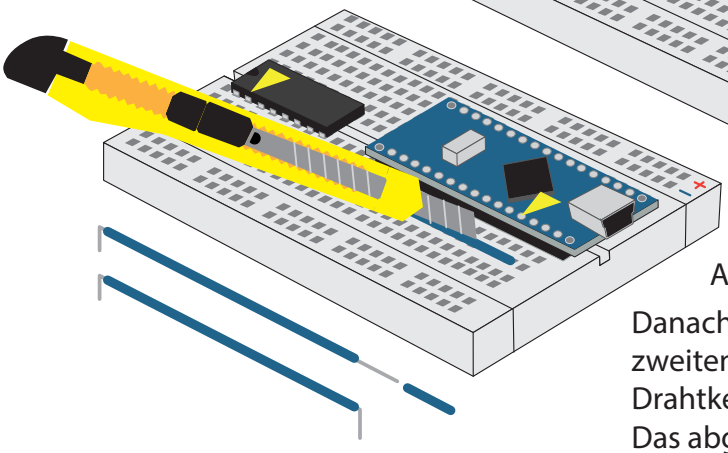


Abb. 3a

Danach wird die Isolierung an dem zweiten Loch vorsichtig bis auf den Drahtkern durchtrennt (Abb. 3a). Das abgetrennte Stück wird abgezogen und das herausragende Drahtende umgebogen (Abb.3b).

Abb. 3b

So eingekürzt kann die Steckbrücke die ausgesuchten Kontakte punktgenau miteinander verbinden (Abb. 4).

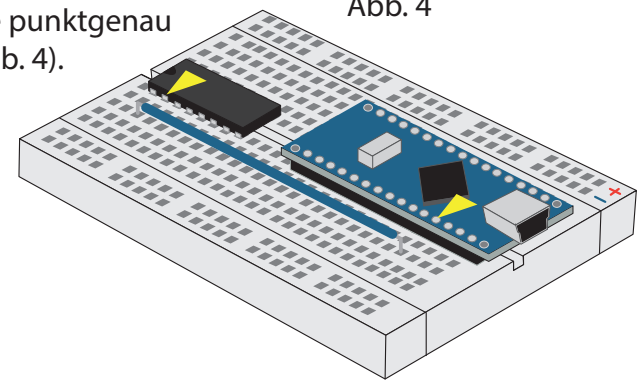
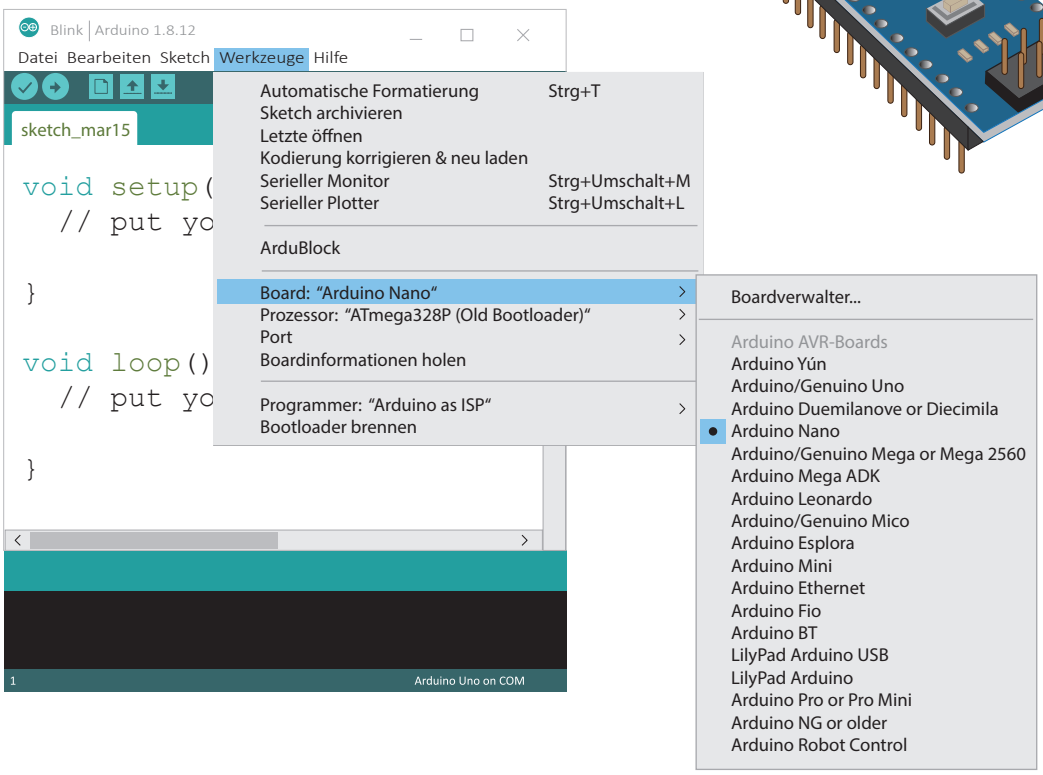
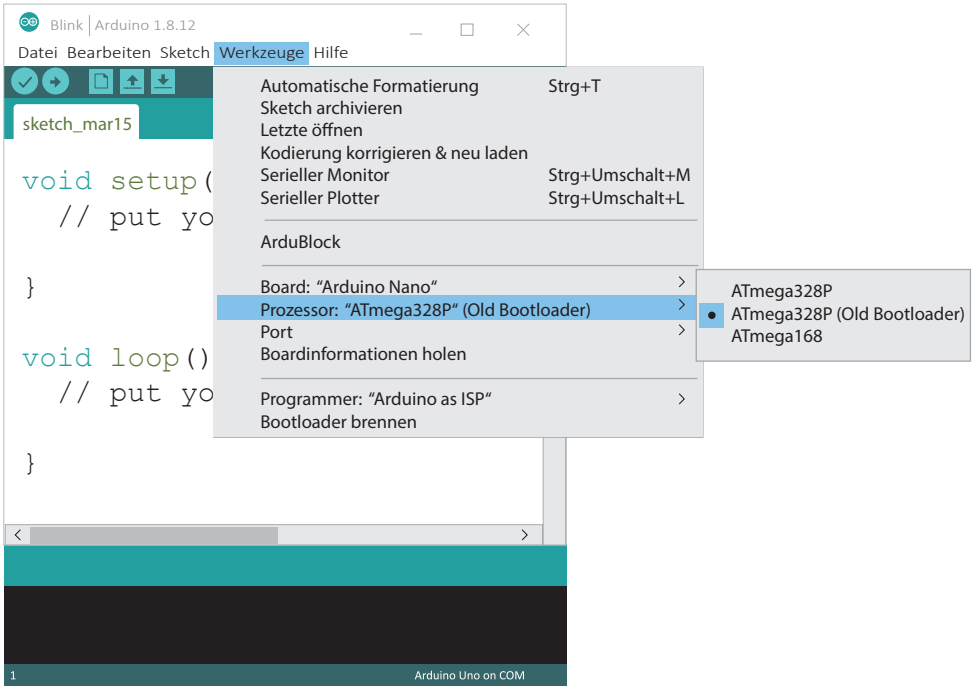


Abb. 4

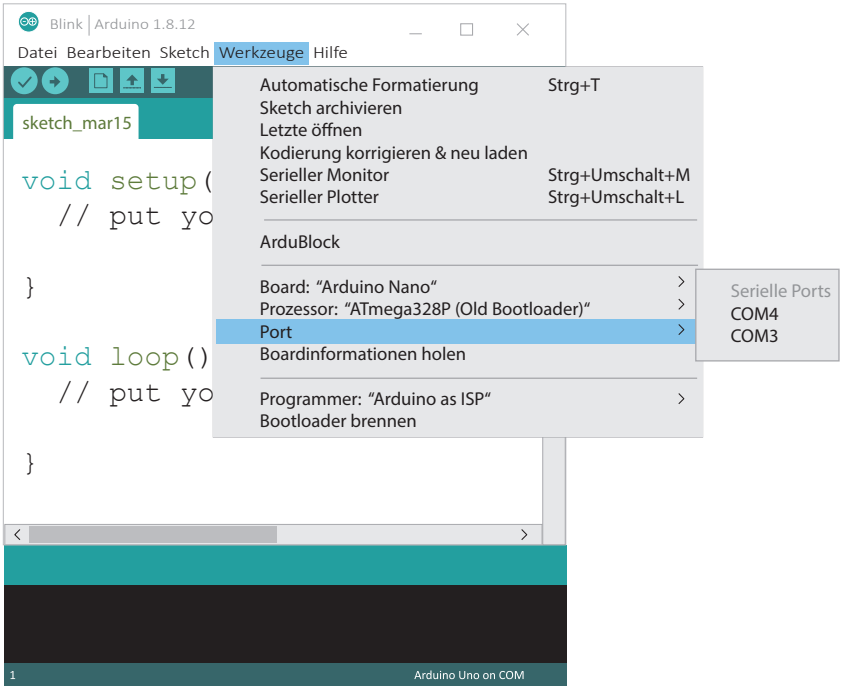
Zum Anschließen an einen PC ist der Arduino Nano mit einer mini USB Buchse ausgestattet. Zur Programmierung dienen die gleichen Editoren wie für den UNO. Anders als dort wird der Nano nicht automatisch erkannt, sondern muss händisch eingerichtet werden. Dazu den Nano mit dem PC verbinden und die Arduino IDE starten. Über den Menüpunkt „Werkzeuge“ und „Board“ im Menü „Arduino Nano“ auswählen.



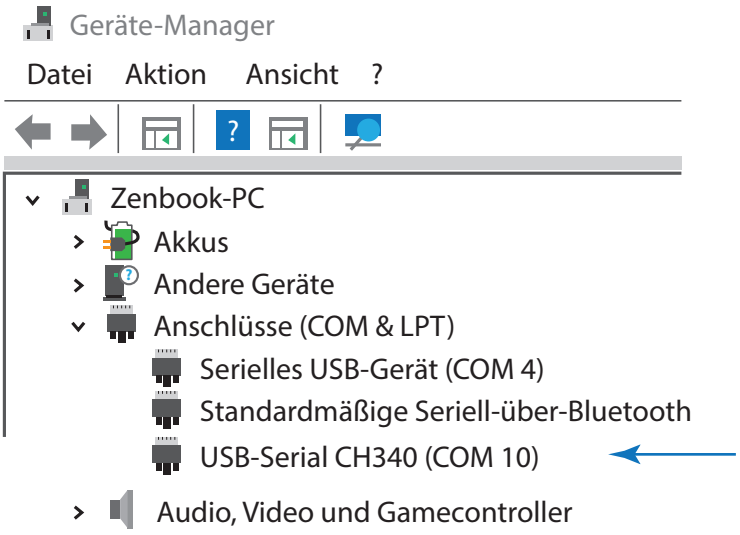
Der übliche Prozessortyp für den Nano ist ein ATmega328P. Wer nicht weiß, welcher Bootloader dort aufgespielt ist, muss unter „Werkzeuge“ und „Prozessor“ im Menü entweder „ATmega328P“ oder „ATmega328P (Old Bootloader)“ einrichten. Ist der falsche eingestellt, bricht eine Übertragung von Code mit einer Fehlermeldung ab.



Anders als beim Arduino UNO, wird der COM-Port, über den Daten an den Nano übermittelt werden können, nicht durch einen Namens-Zusatz im Menü der IDE kenntlich gemacht.

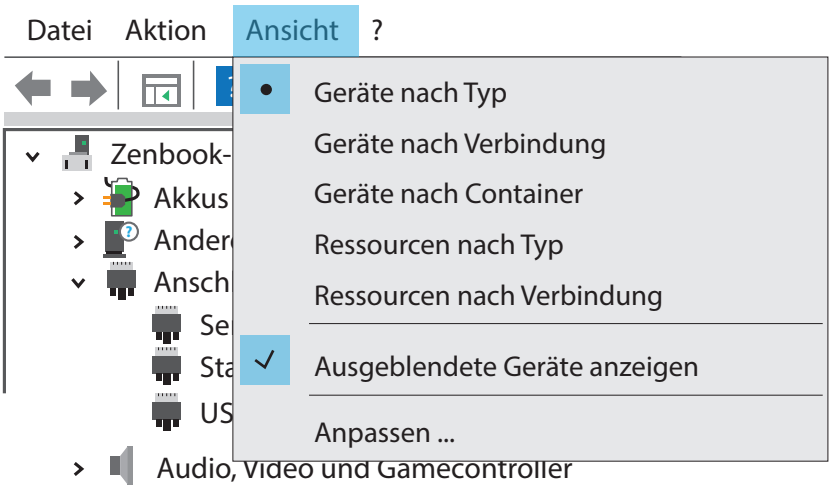


Hier ist es hilfreich zu wissen, dass der zuletzt zugewiesene COM-Port die höchste Zahl trägt. Um sicherzustellen, dass der so ausgewählte COM-Port tatsächlich zum Arduino-Nano gehört, kann der Gerätemanager aus der Systemsteuerung zurate gezogen werden. Er zeigt unter „Anschlüsse (COM&LPT)“ alle Geräte an, die aktuell einen solchen Port belegen. Wird ein Gerät - hier also der der Nano - vom PC getrennt, erlischt die Anzeige. Beim erneuten Herstellen der Verbindung erscheint sie wieder.

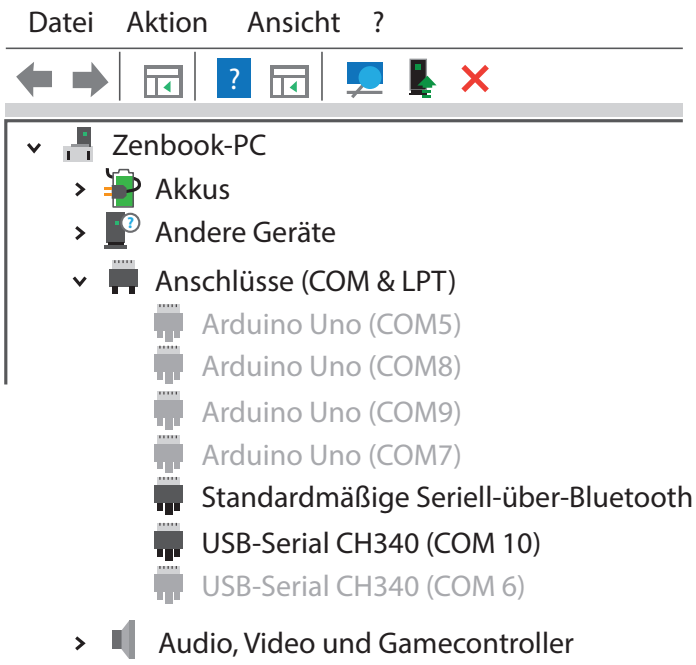


COM-Port, den das Betriebssystem vermutlich dem Arduino Nano zugewiesen hat

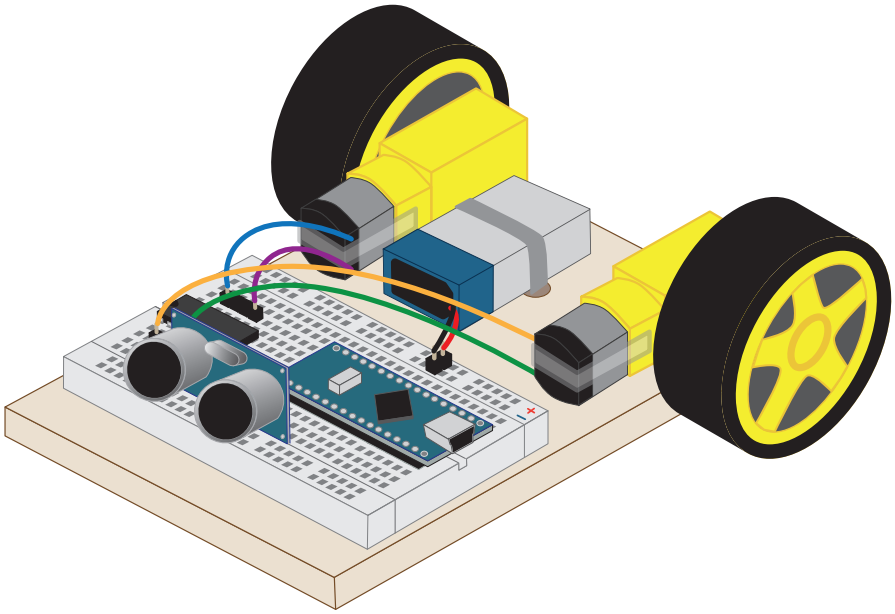
Wen es überrascht, dass so hohe Nummern vergeben werden, sollte wissen, dass Windows sich alle in der Vergangenheit erzeugten Ports merkt. Diese unbenutzten, aber als belegt geführten Schnittstellen werden nach den üblichen Voreinstellungen nicht angezeigt und müssen erst sichtbar gemacht werden. Dafür im Geräte-Manager auf „Ansicht“ klicken und in dem erscheinenden Menü „Ausgeblendete Geräte anzeigen“ auswählen.



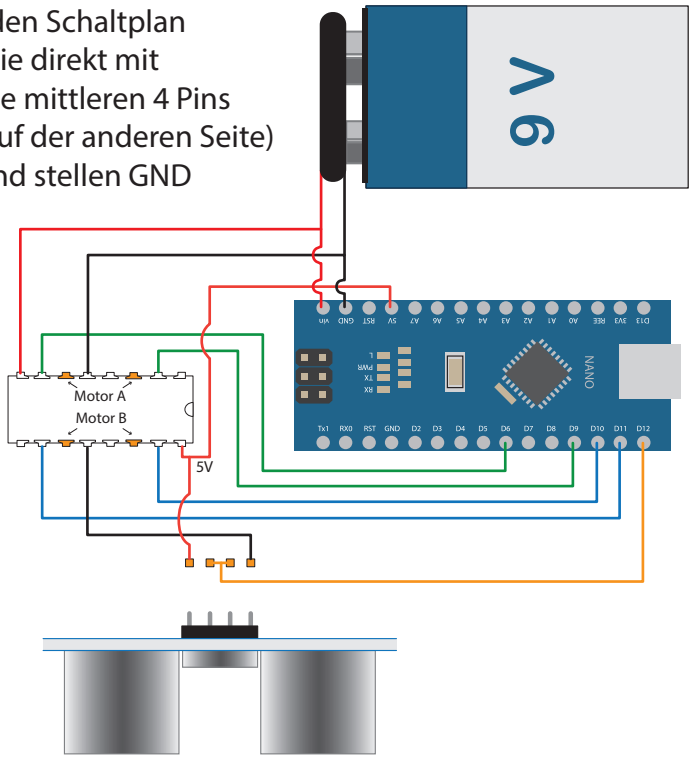
Mit dieser Einstellung werden die als belegt geführten Schnittstellen sichtbar. Zur Unterscheidung von den aktiven Geräten sind Symbole und Schrift blasser. Wird ein solcher Eintrag angewählt, kann er über das rote Kreuz in der Menüleiste oder die „Entf“-Taste gelöscht werden.



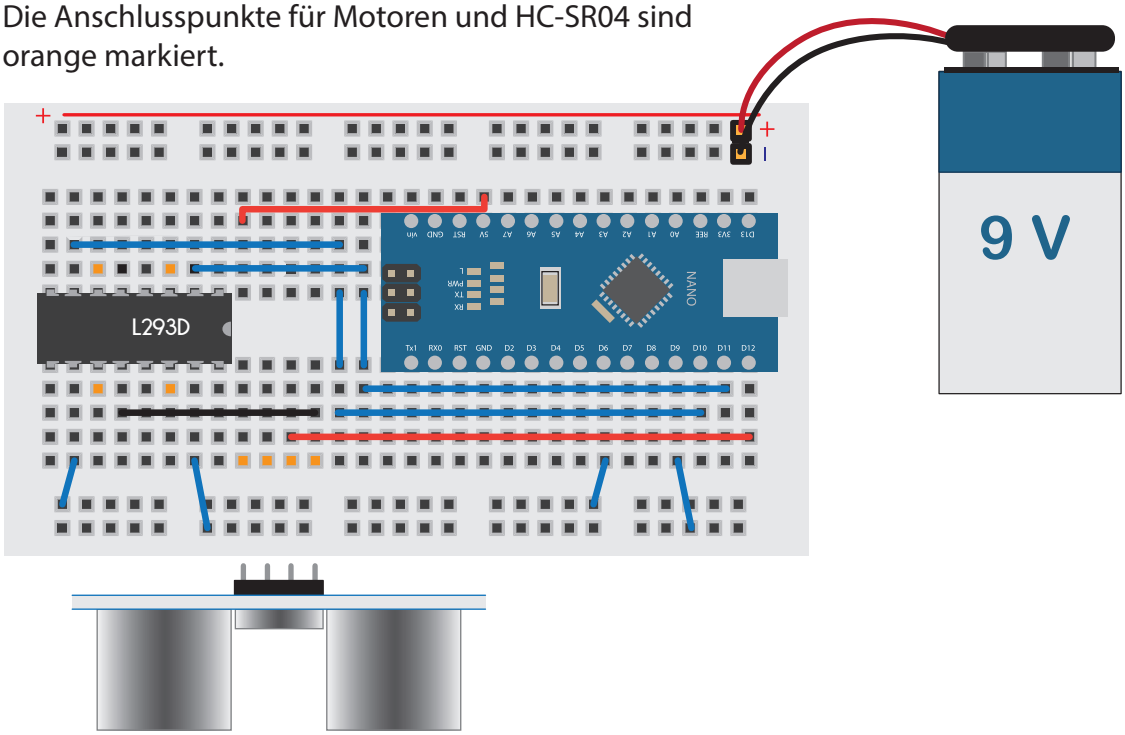
Wer sich vor etwas Mehrarbeit nicht scheut, der kann mit anderer Hardware die Kosten für das Fahrzeug nach gegenwärtigem Stand auf etwa 10 Euro begrenzen. Die größte Ersparnis liegt dabei bei den Motoren, die es schon für etwas mehr als 1 € pro Motor und Rad zu kaufen gibt.



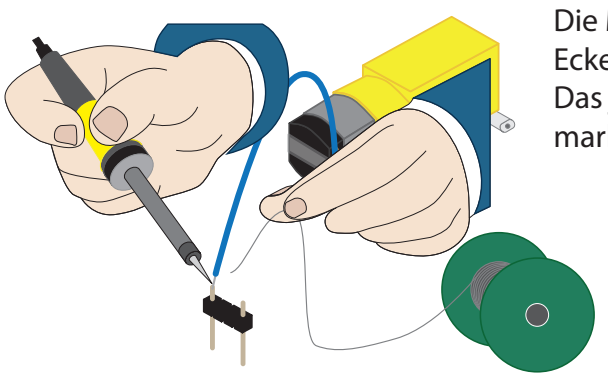
Der Mehraufwand hängt damit zusammen, dass diese Motoren sich nicht mit dem Strom aus den Arduino-Pins begnügen. Zur Lösung des Problems muss ein Motorentreiber - hier ein L293D - mit eingeplant werden, der es ermöglicht, den Strom direkt aus dem 9V Block zu ziehen. Zusätzlicher Programmieraufwand ist dazu nicht erforderlich.
Ein Blick auf den nebenstehenden Schaltplan zeigt links besagte Hardware, die direkt mit dem 9V Block verbunden ist. Die mittleren 4 Pins (zwei auf der einen und zwei auf der anderen Seite) sind miteinander verbunden und stellen GND bereit. Die Anschlüsse unmitelbar daneben sind Ein- und Ausgänge zur Ansteuerung der Motoren. Die Pluspole (9 und 5V) liegen sich diagonal gegenüber. Die übrigen beiden sich diagonal gegenüberliegenden Pins sind unbelegt. Eine Kerbe macht es möglich, die Richtung zu erkennen, in die der L293D in den Schaltplan eingebunden ist.



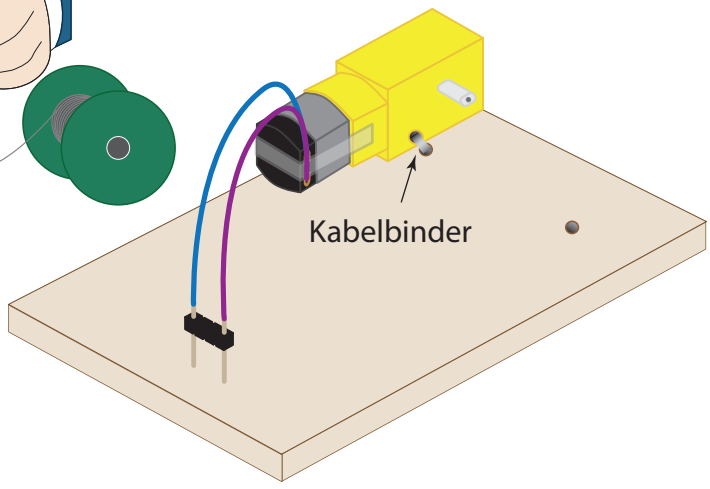
Die zur Steuerung des Fahrzeugs benötigten Komponenten finden auf einem 400er Breadboard Platz. Wie unten gezeigt, wird der Nano ganz rechs und der L293D ganz links platziert. Die Kerbe zeigt dabei nach rechts. Zwischen den beiden Teilen bleibt so Platz für den Ultraschallsensor. Die Verdrahtung ist der Zeichnung zu entnehmen. Die Anschlusspunkte für Motoren und HC-SR04 sind orange markiert.



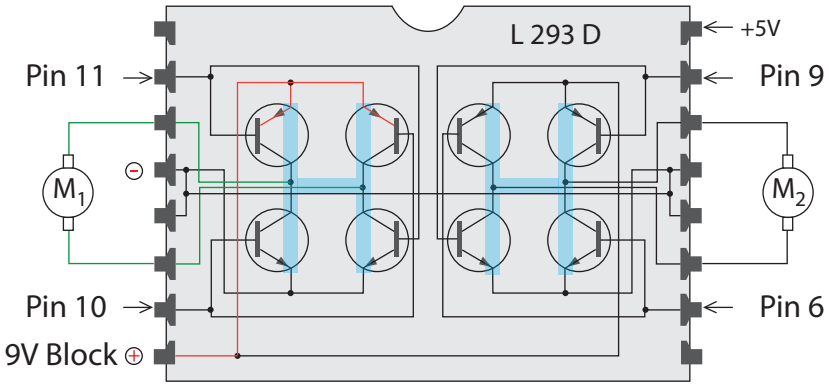
Um die Motoren auf einfache Weise an den vorgesehenen Kontaktpunkten mit dem Breadboard verbinden zu können, empfiehlt sich die Verwendung von Stiftleisten. Aus einem Stück mit vier Stiften werden die mittleren entfernt, die Motorenkabel an den endständigen angelötet und die Lötpunkte mit Heißkleber abgesichert.



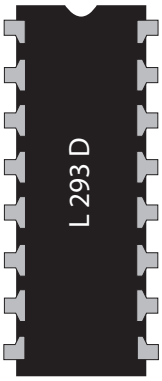
Die Motoren werden randscharf in die Ecken des Tragebrettchens gesetzt. Das jeweils zu bohrende Loch an der Stelle markieren, wo das Motorengehäuse einen Durchlass hat. Kabelbinder (200 x 2,4mm) von unten durch das Loch im Brettchen, dann durch die Öffnung im Gehäuse führen und auf der Unterseite zusammenzurren.



Damit die Motoren vom Arduino Nano gesteuert werden können, wird - wie gesagt - ein Treiber benötigt. Wie er funktioniert, illustrieren die nachfolgenden Skizzen:



Schematische Darstellung des „Innenlebens“



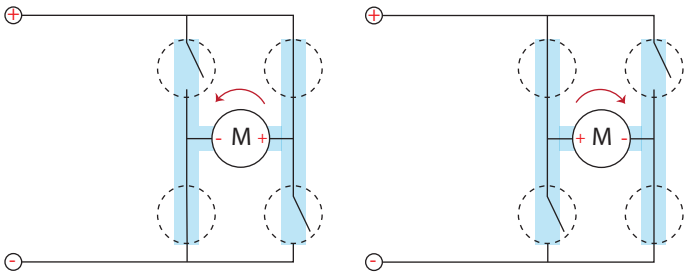
Realitätsnahe Darstellung

Kernstücke des L293D sind 8 Transistoren (= elektronische Schalter), die zu zwei „H-Brücken“ vereint sind. Der Name „H-Brücke“ erklärt sich aus der Anordnung der Transistoren und ihrer Verbindung zum jeweiligen Motor, die einem „H“ ähneln. Jeweils zwei diagonal gegenüberliegende Schalter sind mit dem Plus- bzw. Minuspol der Spannungsquelle (hier der 9V Block) verbunden. Angesteuert werden sie über die Arduino-Pins, hier - wie immer in dieser Handreichung - repräsentiert durch die Pins 11, 10, 9 und 6.

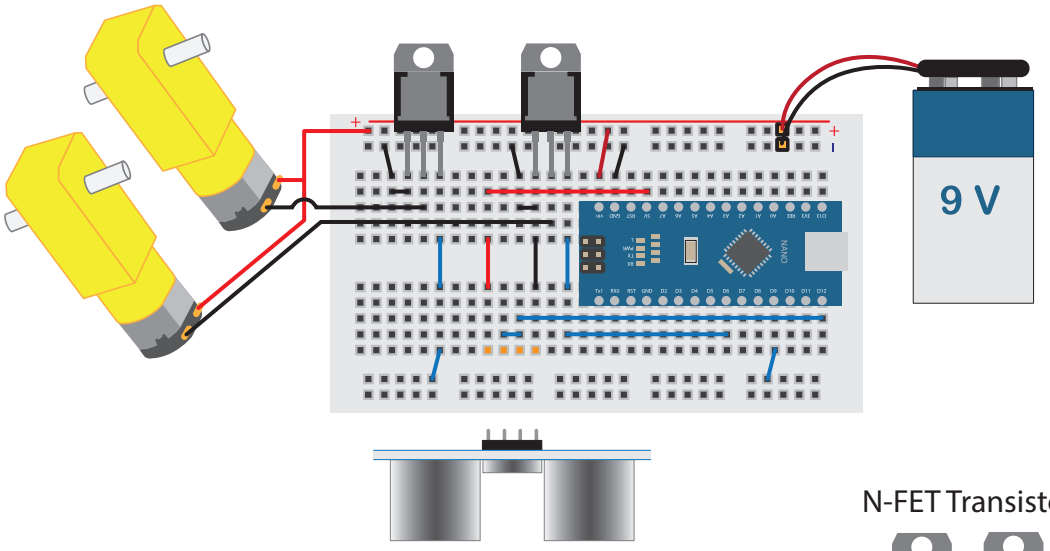
Auf der Programmierebene werden die Schaltvorgänge ausgelöst mit den Funktionen digitalWrite() zusammen mit der Pin-Nummer und den Werten „HIGH“, „LOW“ oder über analogWrite() mit entsprechenden Zahlwerten.

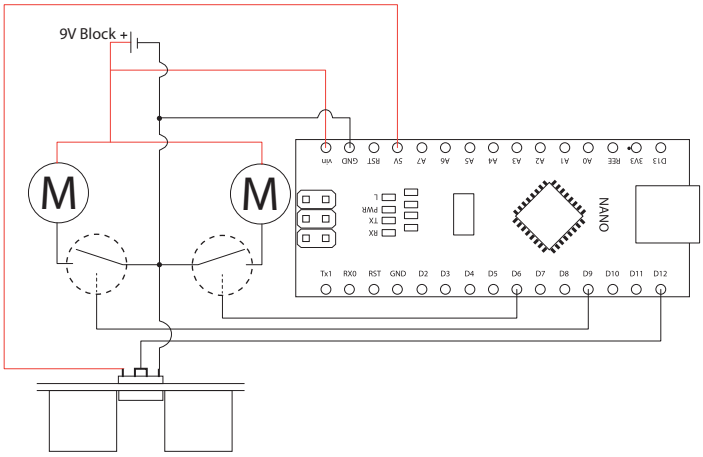
Da ein solcher Befehl aufgrund der Verdrahtung immer zwei Schalter auslöst, können die Motoren auf vier verschiedene Weisen gepolt werden: Plus/Minus, Minus/Plus, Minus/Minus und Plus/Plus. Die erste Polung bewirkt, dass der Rotor ihn in die eine, die zweite, dass er in die andere Richtung bewegt wird. Eine gleichnamige Polung lässt den Rotor stillstehen.

Die nachfolgende Skizze illustriert die beiden Schaltstellungen, die einen Rotor in die eine bzw.die andere Richtung drehen lässt.

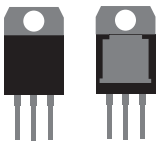


Wem ein Motorentreiber zu unanschaulich erscheint, kann die Steuerung auch mit zwei N-FET Transistoren realisieren. In Kauf genommen werden muss dabei, dass die Motoren nur an- und abgeschaltet werden. Ein Umpolen ist nicht möglich.





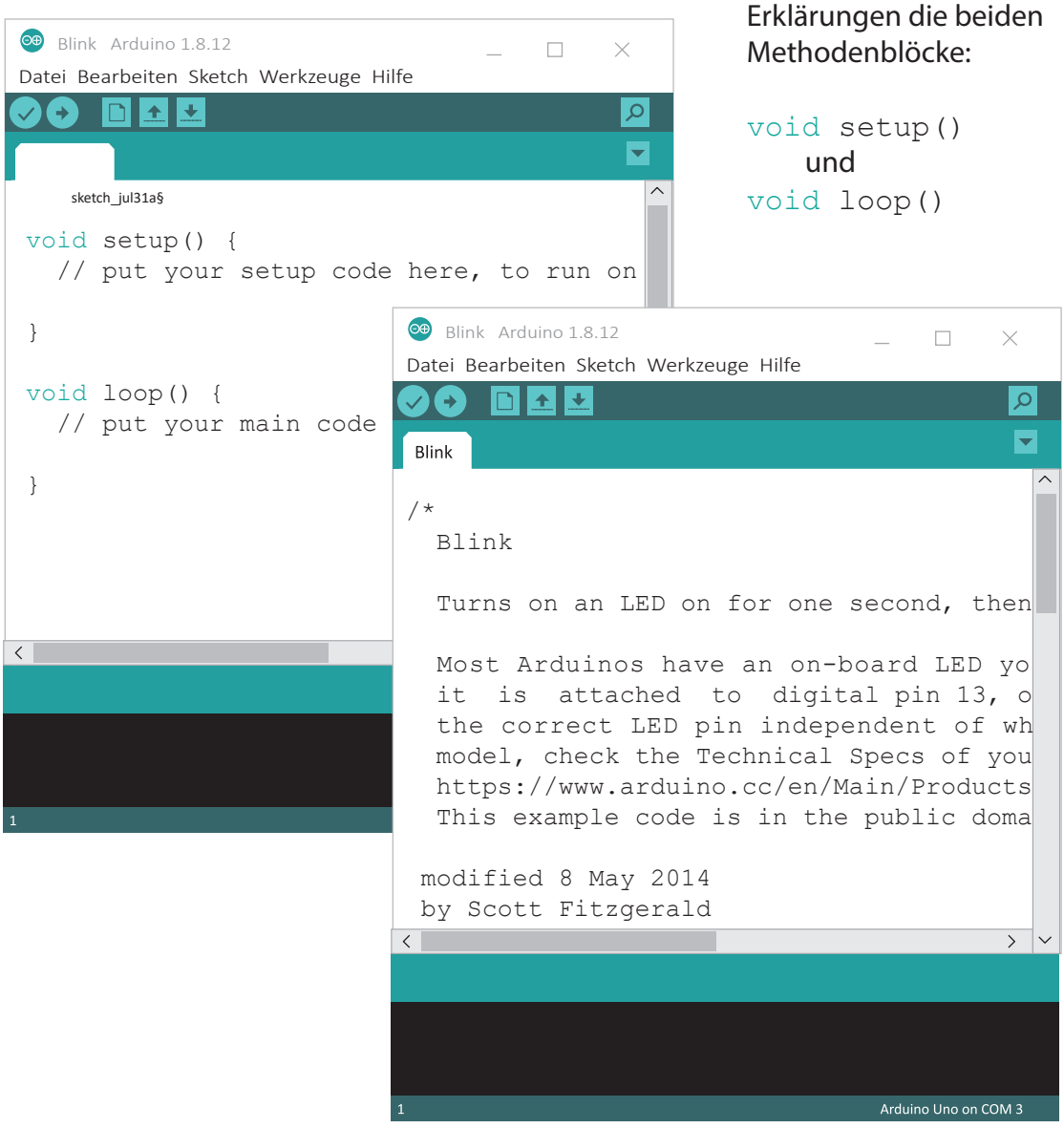
N-FET Transistor



vorn hinten

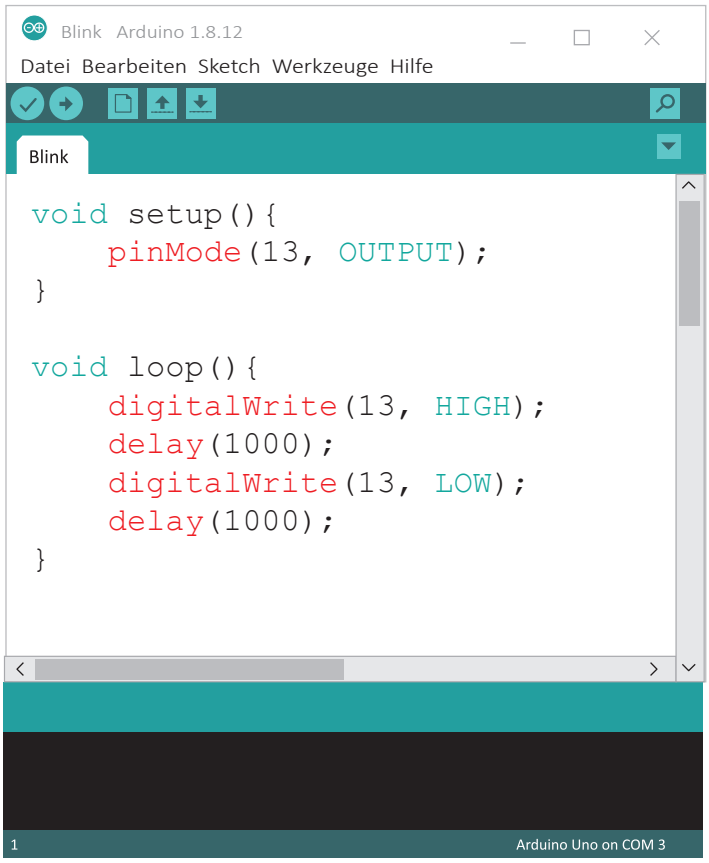
Nach dem Start des Betriebssystems mithilfe eines Internet-Browsers die zum Programmieren erforderliche IDE herunterladen, die unter ´arduino.cc´ zum Download bereitsteht.
Nach erfolgreicher Installation kann die IDE über das Icon gestartet werden. Es öffnet sich ein Fenster, das einer Textverarbeitung ähnelt. Der Text enthält neben Erklärungen die beiden Methodenblöcke:

```
void setup()
    und
void loop()
```

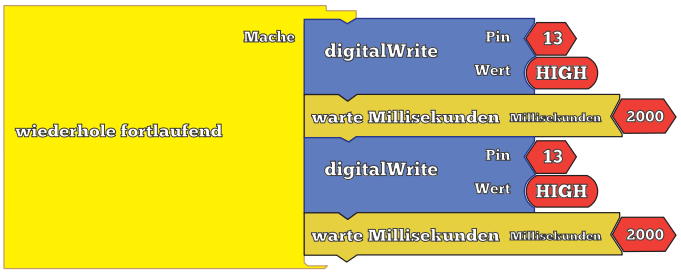


Da weitere Programmfunktionen fehlen, bietet sich an, ein für die hier verfolgten Zwecke geeignetes Programm zu öffnen. Es findet sich unter Datei > Beispiele > 01.Basics > Blink. Ein Klick öffnet ein zweites Fenster, das die Kennung „Blink“ auf dem Reiter trägt. Der eigentliche Programmcode ist hinter einer mehrzeiligen Erklärung auf Englisch verborgen. Eingeleitet wird sie von [/*] und abgeschlossen von [*/]. Wird das gelöscht, finden sich weitere Kommentare in einzelnen Zeilen, denen jeweils zwei Schrägstriche [//] vorangestellt sind. Werden auch diese Kommentare entfernt, bleiben wenige Zeilen Programmcode übrig.

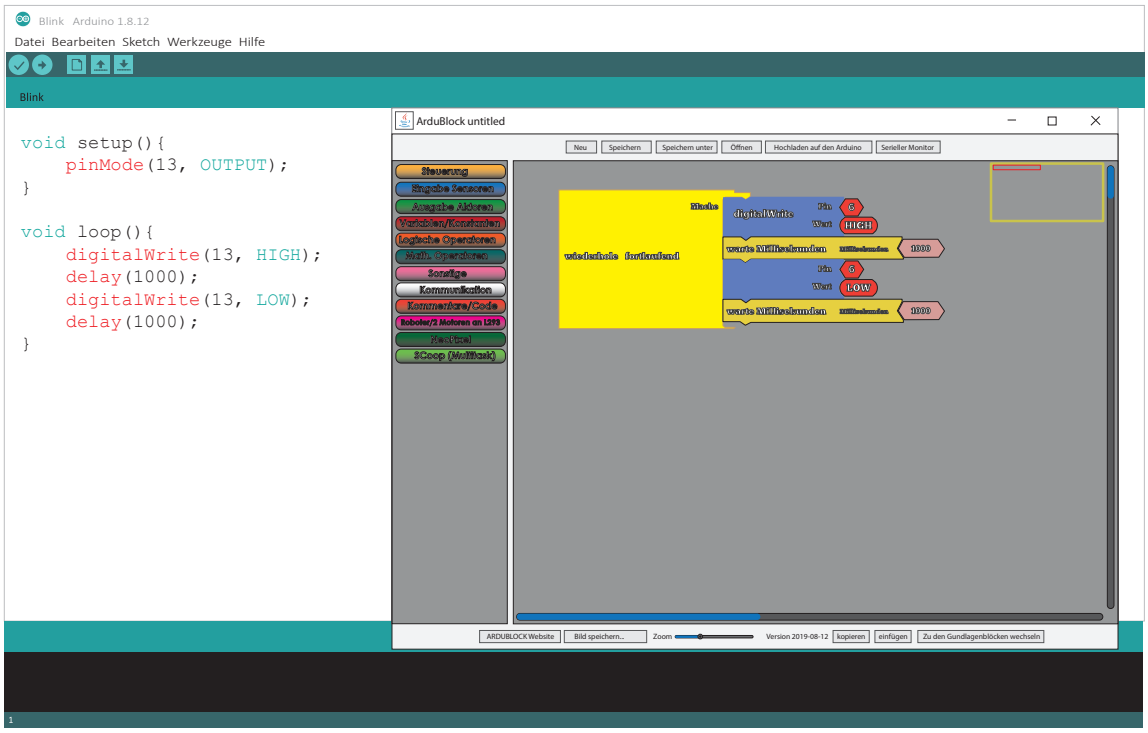
Die geschweiften Klammern hinter dem Methodenblock `void setup()` sorgen für die Eingrenzung der zur jeweiligen Methode gehörenden Funktion, hier `pinMode(13, OUTPUT);`
In dem Methodenblock `void loop()` sind es vier Teile:
`digitalWrite(13, HIGH);`
`delay(1000);`
`digitalWrite(13, LOW);`
`delay(1000);`
Die eingefärbten Codewörter, `pinMode`, `OUTPUT`, `digitalWrite`, `HIGH`, `delay` und `LOW` sind dem Programm bekannt. Ein Rechtschreibfehler, beispielweise das „M“ in „pinMode“ klein zu schreiben, verhinderte, dass das Wort als Teil des Codes erkannt würde. Es bliebe dann schwarz. Solch automatisierte Hilfen erleichtern das Programmieren. Auch kann jederzeit mit einem Klick auf ✓ im Programmfenster oben links der Code auf Richtigkeit geprüft werden. Wird ein Fehler entdeckt, färbt sich die Folgezeile orange ein. Der Fehler findet sich dann in den Zeilen darüber. Zudem werden Hinweise auf Englisch im unteren Drittel des Programmfensters orange auf schwarz eingeblendet.



Wegen der Vielzahl möglicher Rechtschreib- und Interpunktionsfehler wird textuelles Programmieren gemeinhin als schwieriger erachtet als visuelles. Eine visueller Editor kann in die Arduino IDE eingebunden werden. Das Programm heißt „Ardublock“ und ist mit Installationsanleitungen unter mehreren Adressen im Internet zu finden.
Nach erfolgreicher Einrichtung ist es in der Arduino IDE unter „Werkzeuge“ zu finden. Die dem Blink-Code entsprechende Programmierung stellt sich dort so dar:



Beim Download auf den Mikrocontroller wird der visuelle in Textcode umgewandelt und in der Arduino IDE ausgegeben. Beide Editoren sind dann verfügbar und können - wie in der Zeichnung unten - nebeneinander dargestellt werden.



Im Vergleich zeigen sich die gemeinsamen Strukturen.

```
void setup(){

}
```

Ein leerer (engl., ´void´) Rahmen für das Setup (engl., ´to set up´, ´aufbauen, montieren´)

```
void loop(){

}
```

Ein weiterer leerer Rahmen für eine Dauerschleife (engl., ´loop´)

```
void setup(){
pinMode(13, OUTPUT);
}
```

Im Setup wird Pin 13 wird als Ausgabe (engl., ´output´), festgelegt, hier Pol einer Spannungsquelle

```
void loop(){
digitalWrite(13, HIGH);
delay(1000);
digitalWrite(13, LOW);
delay(1000);
}
```

In der Dauerschleife finden sich vier Funktionen:

-> Pin 13 wird auf ´HIGH´ gesetzt (wird zum Pluspol),

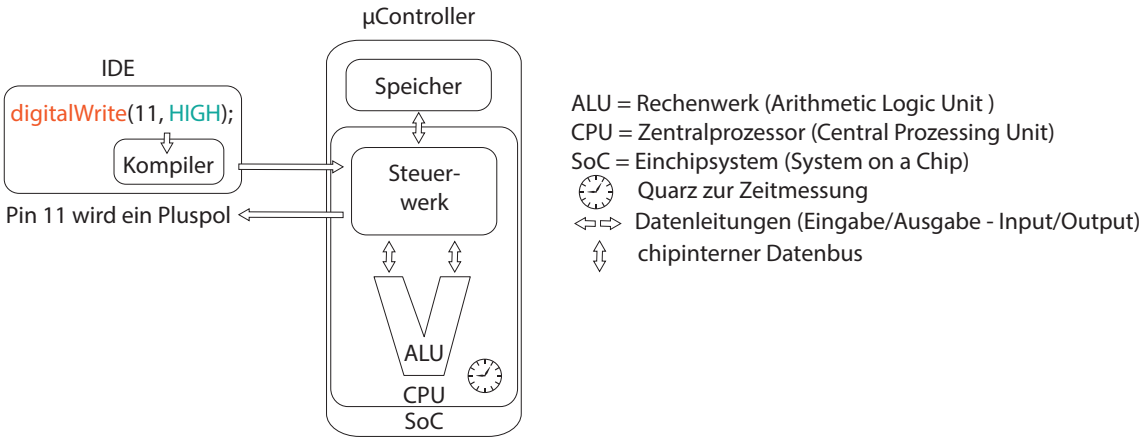
-> nach einer Verzögerung von 1000 Millisekunden

-> wird der gleiche Pin auf ´LOW´ gesetzt (Minuspol)

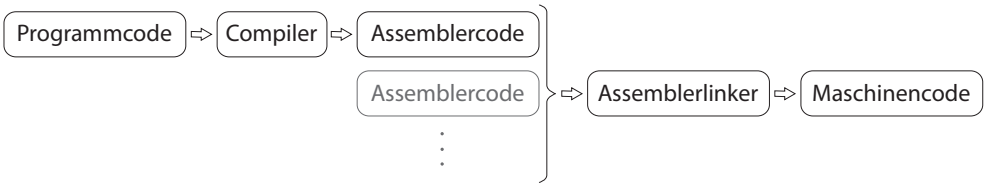
-> nach einer weiteren Sekunde springt das Programm an den Anfang und setzt Pin 13 wieder auf ´HIGH´ usw. (Endlosschleife)

Fehlerfreier Code kann mit einem Klick auf  im Programmfenster kompiliert und über das USB-Kabel zum Mikrocontroller auf dem Arduino-Board übertragen werden.

Das Kompilieren ist ein komplexer Vorgang und den Augen des Anwenders weitgehend verborgen. Im Kern geht es darum, den Arduino-Code in eine maschinenlesbare binäre Sprache zu übersetzen. Das Ergebnis ist „ausführbarer Code“, der auch als Objektcode oder Zielsprache bezeichnet wird. Erst in dieser Form ist der Mikrocontroller in der Lage, die Programmierbefehle umzusetzen.

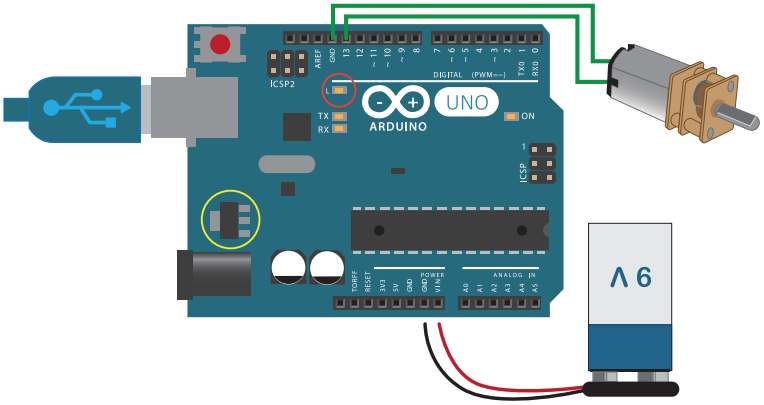


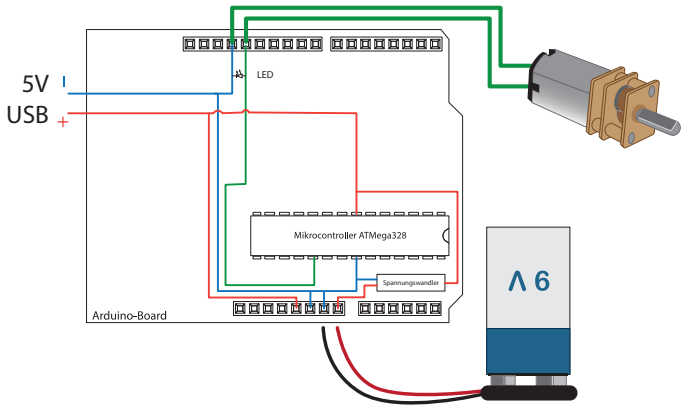
In der Arduino IDE wird zum Kompilieren der freie C-Kompiler „avr-gcc“ (avrdude) genutzt, der zur Erzeugung des Maschinencodes einen Zwischenschritt mit einer Assemblersprache macht.



Da bei diesem Vorgang der Quellcode verlorengeht, ist die Arduino IDE so angelegt, dass der originäre Code als .ino-Datei in einen Ordner gleichen Namens gespeichert werden muss. Von dort kann er dann bei Bedarf wieder hochgeladen werden.

Ist das Arduino-Board mit dem Rechner verbunden, erfolgt die Stromversorgung über den USB-Port. Er liefert eine Spannung von 5 Volt, die über die Pins GND und 5V direkt abgegriffen werden können. Die Mehrzahl der übrigen Pins überwacht der µController. Wird er in der Programmierung über die Funktion pinMode() angewiesen, einen Pin als OUTPUT zu behandeln, legt ihn das darauf fest, dort entweder den Plus- oder Minuspol hinzuschalten. Zur Ausführung wird die digitalWrite() Funktion benutzt, mit der besagtem Pin entweder der Wert HIGH oder LOW zugeordnet wird.





Ist ein Motor an GND und Pin 13 angeschlossen und der Controller mit dem oben vorgestellten Blink-Code programmiert, startet und stoppt der Motor unablässig im Sekundentakt.
Eine kleine, auf dem Board integrierte LED (roter Kreis in der Abbildung) blinkt dazu im gleichen Takt. Sie ist, wie der Motor, mit Pin 13 und GND verbunden.
Um das Motorenprogramm ohne Rechner betreiben zu können, wird eine externe Spannungsquelle benötigt. Tauglich dafür ist ein 9V Block. Ein auf dem Arduino-Board verbauter Wandler (gelber Kreis) regelt die Batteriespannung herunter.

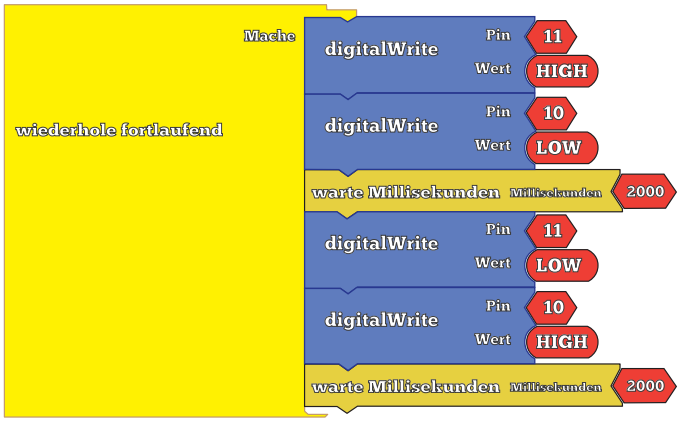
Um den Motor vom Mikrocontroller überwachen zu lassen, können neben dem oben beschriebenen Weg (Anschlüsse an GND und Pin 13), auch die übrigen Pins genutzt werden. Beim Robino sind die beiden Motoren mit den Pins 11, 10, 9 und 6 verbunden, die auf dem Board mit einer „Welle“ (~) gekennzeichnet sind. Das eröffnet die Möglichkeiten, sie nicht nur an- und auszuschalten, sondern sie sowohl vor- und rückwärts als auch mit verminderter Drehzahl laufen zu lassen. Auf er folgenden Seite finden sich die dazu passenden textuellen und visuellen Codes, die bewirken, dass ein Motor andauernd:

- die Drehrichtung ändert
- die Drehgeschwindigkeit an- und abschwellen lässt.

Um das zu zeigen, den Robino über USB mit dem PC verbinden und - wie weiter vorn gezeigt - so aufbocken, dass die Räder frei drehen können.

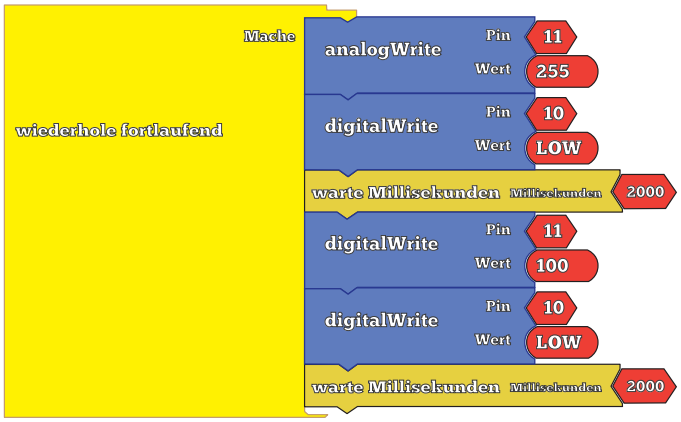
Drehrichtung ändern

```
void setup(){
    pinMode(11, OUTPUT);
    pinMode(10, OUTPUT);
}

void loop(){
    digitalWrite(11, HIGH);
    digitalWrite(10, LOW);
    delay(2000);
    digitalWrite(11, LOW);
    digitalWrite(10, HIGH);
    delay(2000);
}
```



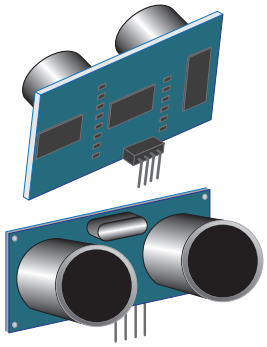
Drehgeschwindigkeit variieren

```
void setup(){
    pinMode(11, OUTPUT);
    pinMode(10, OUTPUT);
}

void loop(){
    analogWrite(11, 255);
    digitalWrite(10, LOW);
    delay(2000);
    analogWrite(11, 100);
    digitalWrite(10, LOW);
    delay(2000);
}
```



Nur die Pins 11, 10, 9, 6, 5 und 3 sind mit `analogWrite()` programmierbar. Die in der Klammer anzugebenden Werte können zwischen 0 und 255 variiert werden. Eine Hundert beispielsweise bewirkt, dass der Motor in einem gegebenen Zeitintervall 100-mal an- und ausgeschaltet wird. Sind die Intervalle beim An- und Abschalten kürzer (größere Zahl) hat das zur Folge, dass der Motor schneller dreht. Sind sie länger (kleinere Zahl) wird der Motor langsamer. Diese Methode zum Steuern von Motoren wird Pulsweitenmodulation (PWM) genannt.

Der Ultraschallsensor HC-SR04, mit dem der Robino ausgestattet ist, wird zur Entfernungsmessung genutzt. Es kann für den Menschen unhörbaren Schall senden und empfangen. Die Sensoren, die das ermöglichen, sind auf einer kleinen Platine platziert, die alle für das Erzeugen und Übertragen der Signale benötigten elektrischen und elektronischen Komponenten beherbergt. Im Innern eines jeden Sensors findet sich ein Piezo-Kristall in Form eines dünnen Plättchens, das sich beim Auftreffen von Schallwellen elastisch verformt und so messbare elektrische Spannungsimpulse erzeugt (Abb.1). Umgekehrt kann das Plättchen durch in rascher Folge erzeugte elektrische Impulse verformt und so Schallsignale jenseits der Hörschwelle aussenden (Abb. 2).



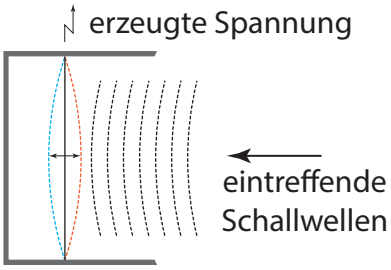


Abb. 1

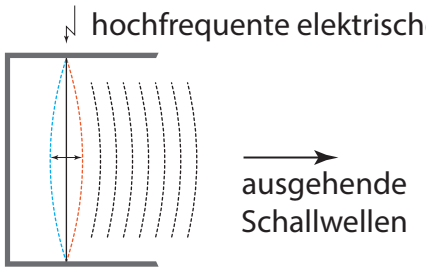
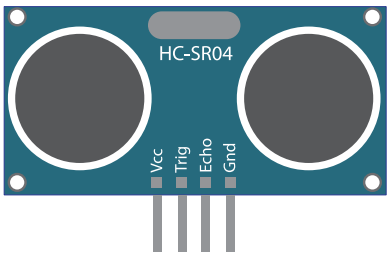
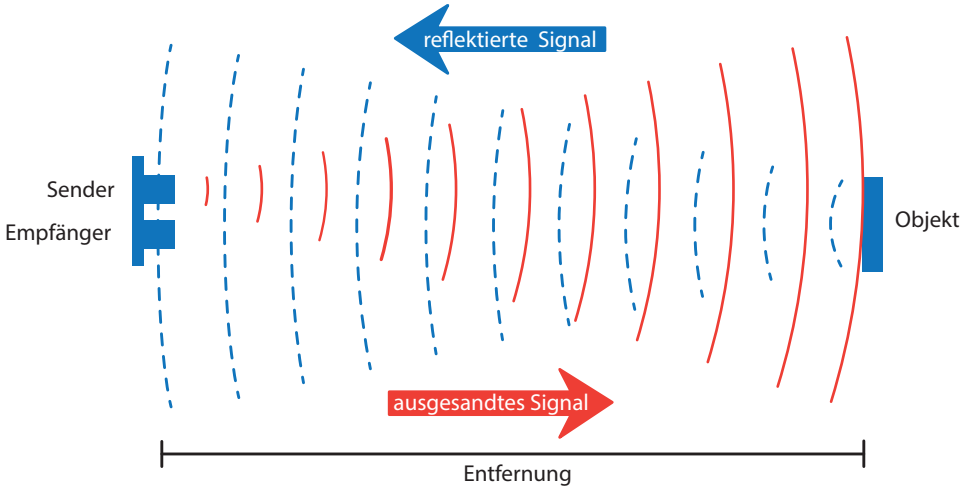


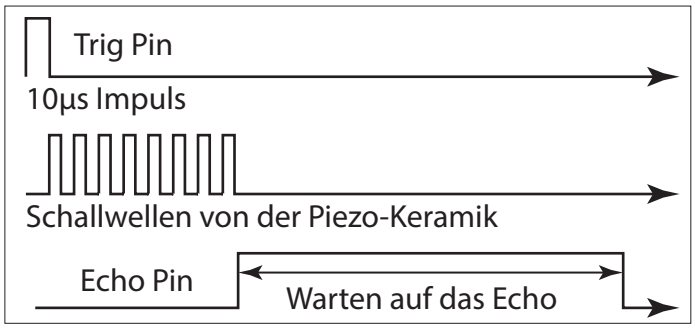
Abb. 2

Das Piezoplättchen elektrisch in Schwingung zu versetzen wird „Triggern“ genannt. Der Pin am HC-SR04, über das die dazu nötigen Impulse übertragen werden, trägt deshalb die Bezeichnung „Trig“. Der Pin, über den die vom Schall über das Piezoplättchen erzeugte Spannung an den µController weitergeleitet wird, ist mit „Echo“ bezeichnet. Über die Pins „Vcc“ und „Gnd“ wird das Modul mit Strom versorgt. Wie die Entfernungsmessung funktioniert, verdeutlicht die Abbildung unten. Gemessen wird die Zeit zwischen dem Aussenden und Empfang des Signals.





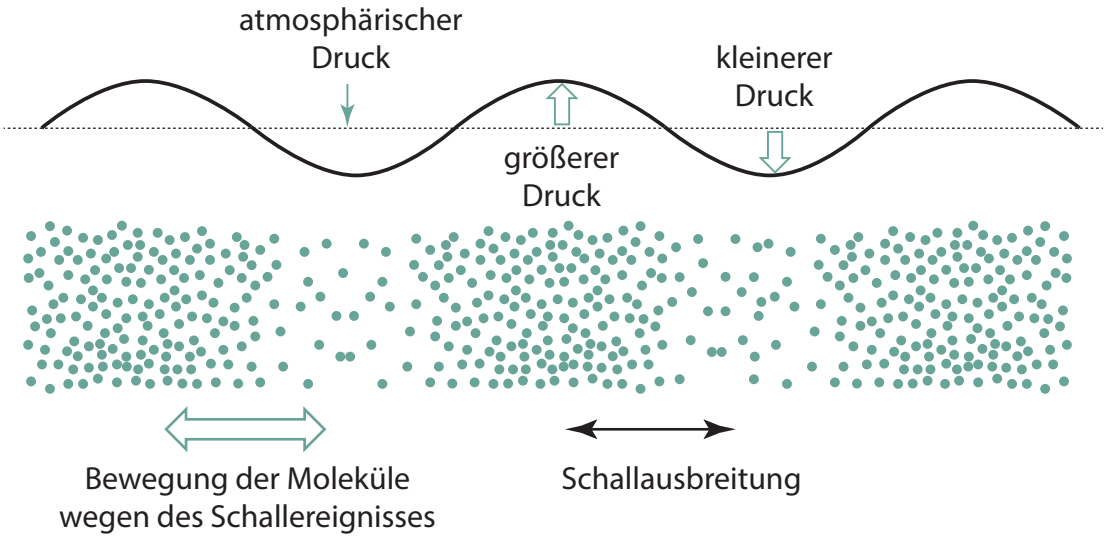
Die Laufzeitmessung wird über den Ausgang gestartet. Das Auslösen geschieht durch einen mindestens 5 Mikrosekunden langen HIGH-Impuls, der den Sensor aktiviert. Er sendet daraufhin ein 40-kHz-Signal aus acht Impulsen. Danach geht der Eingang sofort auf HIGH und wartet auf das zurückkehrende Signal. Gleichzeitig speichert das Programm den Zeitpunkt, bei dem der Eingang auf HIGH gesetzt wird. Detektiert der Empfänger das reflektierte Ultraschallsignal, wird der Eingang wieder auf LOW gesetzt. Diesen Pegelwechsel registriert das Programm ebenfalls. Die Laufzeit ergibt sich aus dem Zeitunterschied zwischen Senden und Empfangen des Ultraschallsignals. Alle 20 ms nach einer Triggerung kann eine weitere Messung stattfinden. Wird kein Echo empfangen, bleibt der Ausgang für insgesamt 38 ms auf HIGH und zeigt so den Misserfolg an. Da all diese Vorgänge nacheinander ablaufen, genügt eine Datenleitung. Auch wäre ein Sensor ausreichend, wie das beispielsweise bei PKWs üblich ist.



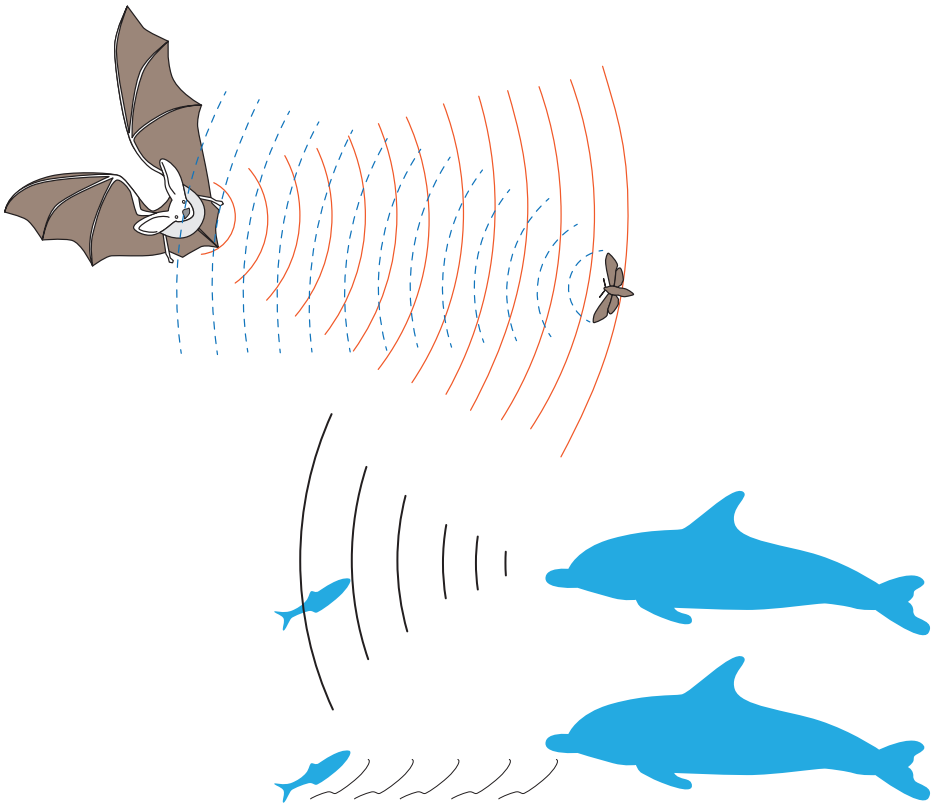

```
int sensorPin = 12;

void setup() {
  pinMode(11, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(6, OUTPUT);
}

void loop() {
  long Laufzeit, Entfernung;
  pinMode(sensorPin, OUTPUT);
  digitalWrite(sensorPin, LOW);
  delayMicroseconds(2);
  digitalWrite(sensorPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(sensorPin, LOW);
  pinMode(sensorPin, INPUT);
  Laufzeit = pulseIn(sensorPin, HIGH);
  Entfernung = Laufzeit/58.2;
  if (Entfernung < 12) {
    digitalWrite(11, LOW);
    digitalWrite(10, LOW);
    delay(2000);
  }
  else {
    digitalWrite(11, HIGH);
    digitalWrite(10, LOW);
    digitalWrite(9, HIGH);
    digitalWrite(6, LOW);
  }
}
```

Die Berechnung der Entfernung kann nach folgender Formel erfolgen: **Entfernung = (Schallgeschwindigkeit * Laufzeit)/2** [ms]- Durch zwei muss geteilt werden, weil das Signal den doppelten Weg zurücklegt: hin zum Objekt und wieder zurück. Die Schallgeschwindigkeit ist von der Temperatur abhängig. Bei Raumtemperatur (20° C) beträgt sie 343,5 Meter pro Sekunde (m/s).
Da für die Steuerung des Robino Abstände im Zentimeterbereich (cm) bedeutsam sind, muss die Zeit in Mikrosekunden (µs) gemessen werden, also 343,5 * 100 = 34350 [cm/s] und 34350 / 1000000 = 0,03435 [cm/µs]. Werden die Werte eingesetzt, ergibt sich folgende Formel: **Entfernung = Laufzeit * 0,03435 / 2**.
Häufig wird statt malzunehmen durch den Kehrwert (1 / 0,03435 = 29,1) geteilt, also **Entfernung = Laufzeit / 29,1 / 2**.
Natürlich kann dann gleich in einem Schritt die Laufzeit durch 58,2 geteilt werden, also **Entfernung = Laufzeit / 58,2**. Ebenso üblich ist es, die Zahlen noch zu runden.

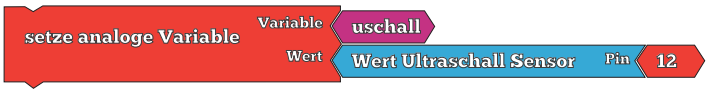
Wie sich die Schwingungen in der Luft fortpflanzen, greift die nachfolgende Zeichnung auf: Das vorschwingende Piezoplättchen erhöht den Luftdruck, das zurückschwingende verringert ihn. Umgekehrt biegt ein erhöhter Luftdruck das Plättchen in die eine, geringerer Druck in die andere Richtung.



Bleibt noch zu bemerken, dass dieses Prinzip der Echoortung beispielsweise von Fledermäusen und Delphinen bei der Jagd nach Beutetieren angewendet wird.



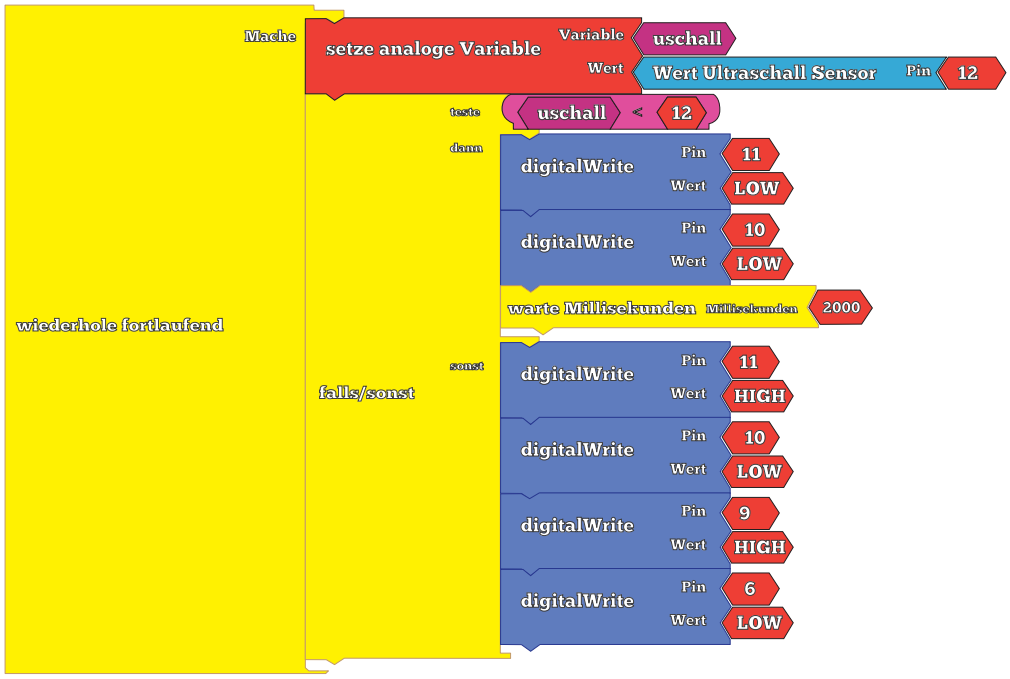
Ardublock reduziert die Sensor-Programmierung auf einen Codeblock, der als „analoge Variable“ gekennzeichnet ist und dem eine Name und eine Pin-Nummer zugeordnet werden kann.



Was sich an Programmierschritten hinter dem Baustein verbirgt, zeigt sich bei der Umwandlung in Arduino-Code. Er entspricht - wie nicht anders zu erwarten - überwiegend dem Code von der vorherigen Seite. Abweichungen davon erscheinen weitgehend selbsterklärend.
Der Kontrollparameter „return cm“ beendet eine Funktion und gibt - wenn gewünscht - einen Wert in Zentimetern an die aufrufende Funktion zurück.

```
int UltraschallSensor(int SensorPin){
  long duration, cm;
  pinMode(SensorPin, OUTPUT);
  digitalWrite(SensorPin, LOW);
  delayMicroseconds(2);
  digitalWrite(SensorPin, HIGH);
  delayMicroseconds(5);
  digitalWrite(SensorPin, LOW);
  pinMode(SensorPin, INPUT);
  duration = pulseIn(SensorPin, HIGH);
  cm =  duration / 29 / 2;
  return cm;
}

int uschall = 0;
```

Der vollständige Programmcode für den Robino zeigt sich in ArduBlock wie folgt:

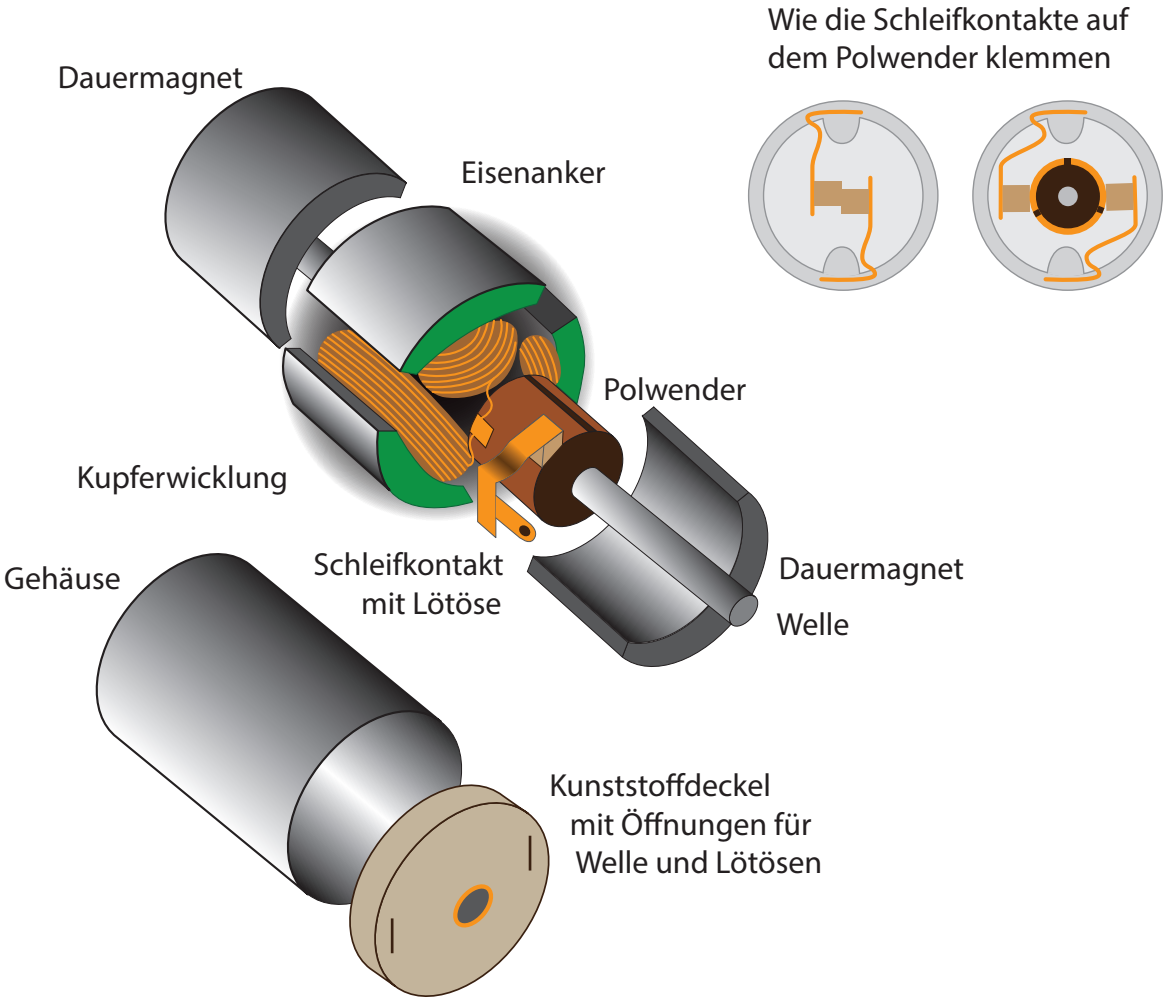


Dieser visuelle Code wird in der Arduino IDE so dargestellt:

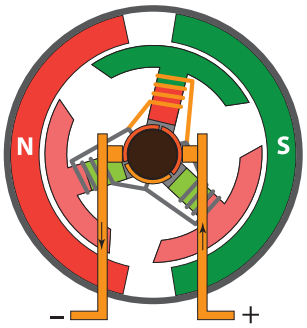
```
int UltraschallSensor(int SensorPin){
  long duration, cm;
  pinMode(SensorPin, OUTPUT);
  digitalWrite(SensorPin, LOW);
  delayMicroseconds(2);
  digitalWrite(SensorPin, HIGH);
  delayMicroseconds(5);
  digitalWrite(SensorPin, LOW);
  pinMode(SensorPin, INPUT);
  duration = pulseIn(SensorPin, HIGH);
  cm =  duration / 29 / 2;
  return cm;
}

int uschall = 0;

void setup() {
  digitalWrite( 12 , LOW );
  pinMode(11, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(6, OUTPUT);
}

void loop() {
  uschall = UltraschallSensor( 12 );
  if (( uschall < 12 )) {
    digitalWrite( 11 , LOW );
    digitalWrite( 10 , LOW );
    delay( 2000 );
  }
  else {
    digitalWrite( 11 , HIGH );
    digitalWrite( 10 , LOW );
    digitalWrite( 9 , HIGH );
    digitalWrite( 6 , LOW );
  }
}
```

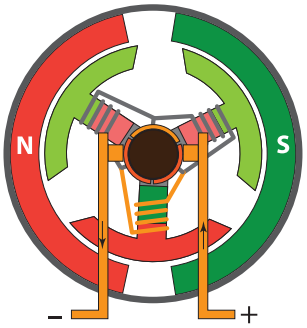
Die im Robino verwendeten Metallgetriebemotoren werden zahlreich im Internet angeboten. Obwohl es sich offensichtlich um baugleiche Motoren handelt, werden diese von den Anbietern unterschiedlich spezifiziert. Die Angaben variieren zwischen 3 und 12 V . Zum Anschließen an ein Arduino-Board scheinen sie allesamt geeignet zu sein. Geachtet werden sollte allerdings auf die angegebene Drehgeschwindigkeit. Auch hier sind die Angaben nur bedingt verlässlich, aber deutlich mehr als 100 rpm sollten es schon sein, besser 300 und mehr. Herunterregeln lassen sich die Drehzahlen immer.
In der Bauanleitung wird davon ausgegangen, dass passende Räder zusammen mit den Motoren geordert werden. Solche Angebote gibt es nur vereinzelt.

Gleichstrommotoren lassen sich zumeist leicht zerlegen, allerdings nicht die hier verwendeten. Zwar lässt sich das Getriebe abschrauben und die hintere Abdeckung durch Aufbiegen der beiden Klammern lösen, aber herausnehmen lassen sich die Teile nicht, weil ein Zahnrad auf die Welle gepresst ist. Zu Anschauungszwecken sollte deshalb ein einfacher DC-Motor demontiert werden, der für wenige Cent käuflich erworben werden kann. Alle wesentlichen Details eines solchen Motors zeigt die nachfolgende Abbildung:



Erklärungen, wie die Drehung des Rotors beim Anlegen einer Gleichspannung zustandekommt, finden sich zuhauf im Internet. Selten werden dabei Abbildungen verwendet, die dem Innenleben der hier verwendeten Motoren entsprechen.

1. 

2. 

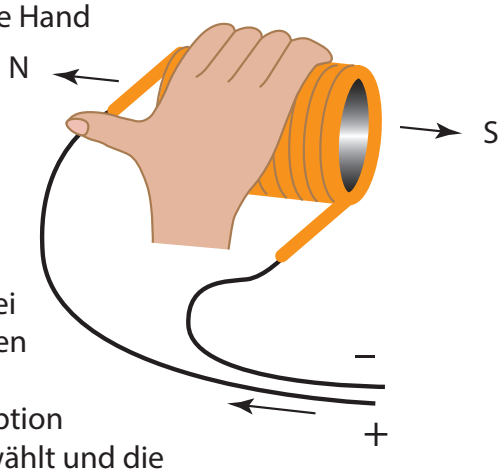
technische Stromrichtung

Bei der Drehung des Rotors lassen sich zwei Stellungen unterscheiden, an denen gezeigt werden kann, dass beim Anlegen einer Gleichspannung der Rotor aufgrund der entstehenden Anziehungs- und Abstoßungskräfte in eine Drehbewegung versetzt wird:

1. Der Südpol des obige Ankers wird nach links zum Nordpol des Dauermagneten gezogen. Stellung und Polarisierung der übrigen beiden Anker unterstützen die Bewegung.
   - Umpolung durch den Polwender
2. Nun wird der Nordpol des unteren Ankers zum Südpol des Dauermagneten gezogen. Auch hier wird die Bewegung von der Stellung und Polarisierung der übrigen Anker unterstützt.
   -Umpolung durch den Polwender und weiter wie am Anfang.

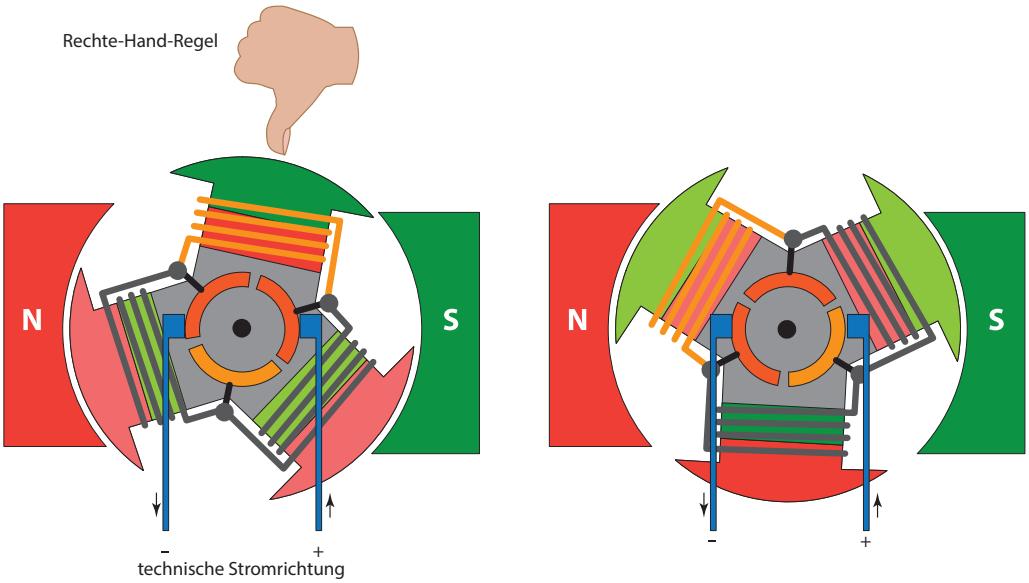
Interessant ist, dass sich die Magnetwirkung der Anker aufbaut, einen Höhepunkt erreicht und sich dann abbaut, ehe die Umpolung erfolgt.

Die Polung der Anker, die von der Fließrichtung des Stroms in der Spule abhängig ist, kann ermittelt werden, indem die rechte Hand so auf die Spule gelegt wird, dass die Finger in Richtung des durch die Spule fließenden Stroms zeigen. Der Daumen weist dann in die Richtung des Nordpols des Magnetfelds (Rechte-Hand-Regel).

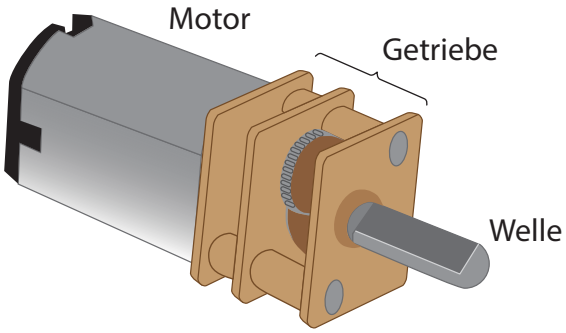


Die Animierung der Rotorbewegung gelingt mit Powerpoint, indem man auf zwei Folien je eine der beiden Schnittzeichnungen oben punktgenau kopiert, dann unter <Bildschirmpräsentation einrichten> die Option <Wiederholen bis „Esc“ gedrückt wird> anwählt und die Präsentation startet. Will man den Weg eines Ankers verfolgen, braucht man sechs (3x2) Schnittbilder auf denen der zu verfolgende Anker mit einem Punkt markiert wird.

Alternative Darstellungsweise von Rotor, Stator und Polwender

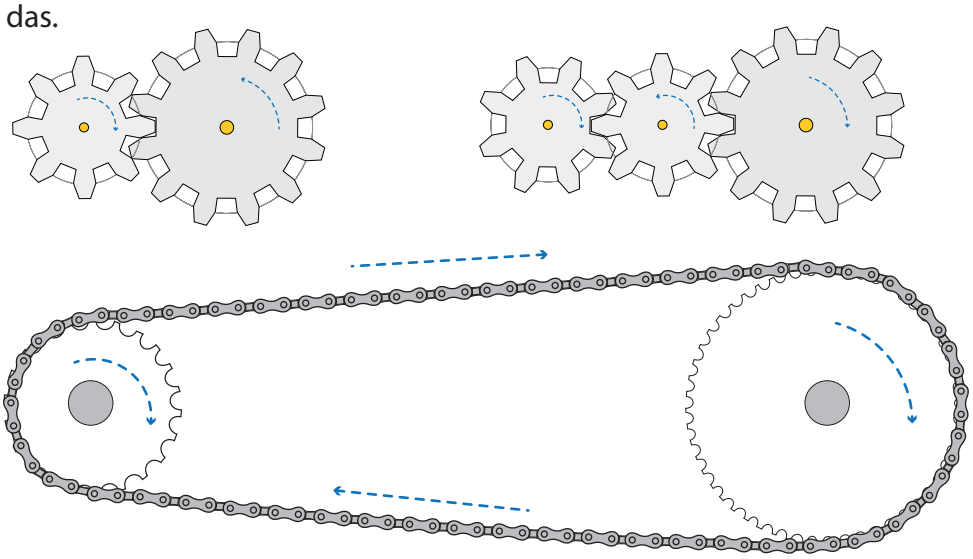


Über die Getriebe an den Motoren des Robotfahrzeugs werden sowohl die Drehzahlen der Wellen und damit die zu erwartende Geschwindigkeit der Fahrzeuge als auch deren Durchzugskraft (Drehmoment) maßgeblich bestimmt.



Als Getriebe bezeichnet man eine solche Konstruktion, in der mehrere Zahnräder miteinander kombiniert sind. Zwei gleich große Räder wirken dabei stets mit gleich großer Kraft aufeinander ein. Sind die Radien der Zahnräder unterschiedlich groß, so werden Drehzahl und Drehmoment erhöht oder verringert.

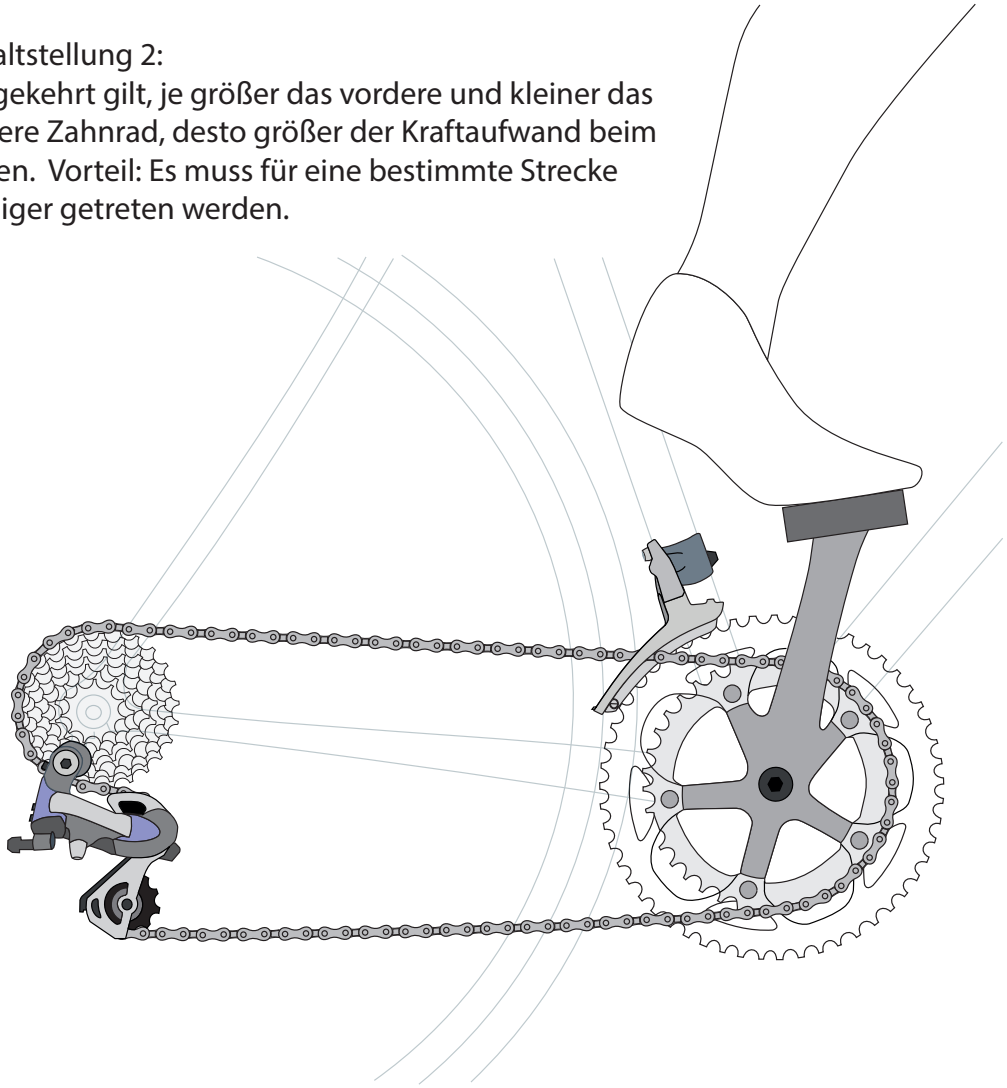
Ein Zahnrad ist ein Rad, entlang dessen Umfang Zahnungen eingearbeitet sind. Mittels dieser Zahnungen kann ein Zahnrad ein wirkendes Drehmoment beispielsweise auf eine Kette oder ein anderes Zahnrad übertragen. Eine Kette kommt dort zum Einsatz, wo Zahnräder nicht unmittelbar ineinander fassen können, wie beispielsweise beim Fahrrad. Dort hat die Kette zudem den Vorteil, dass beide Zahnräder in die gleiche Richtung bewegt werden. Das ist anders, wenn zwei Zahnräder direkt ineinander greifen. Sie drehen sich dann in entgegengesetzte Richtungen. Erst ein drittes Rad ändert das.



Unmittelbar erfahrbar ist die Wirkung eines Getriebes beim Radfahren, wenn das Rad mit einer Gangschaltung ausgestattet ist und widrige Fahrbedingungen herrschen wie Gegenwind, ansteigendes Gelände, rauher oder weicher Untergrund. Wird die Anstrengung beim Treten zu groß, kann auf einen kleineren Gang zurückgeschaltet werden, was den Kraftaufwand verringert. Um die Geschwindigkeit zu halten, muss allerding schneller getreten werden. Geht das Treten dagegen leicht, kann auf einen größeren Gang hochgeschaltet werden, sodass bei gleichbleibender Trittgeschwindigkeit schneller gefahren wird. Warum das so ist, soll an zwei Schaltstellungen verdeutlicht werden:

Schaltstellung 1:
Je kleiner das vordere und größer das hintere Zahnrad, desto geringer der Kraftaufwand beim Treten. Nachteil: Es muss für eine bestimmte Strecke mehr getreten werden.

Schaltstellung 2:
Umgekehrt gilt, je größer das vordere und kleiner das hintere Zahnrad, desto größer der Kraftaufwand beim Treten. Vorteil: Es muss für eine bestimmte Strecke weniger getreten werden.

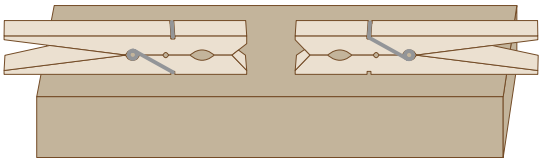


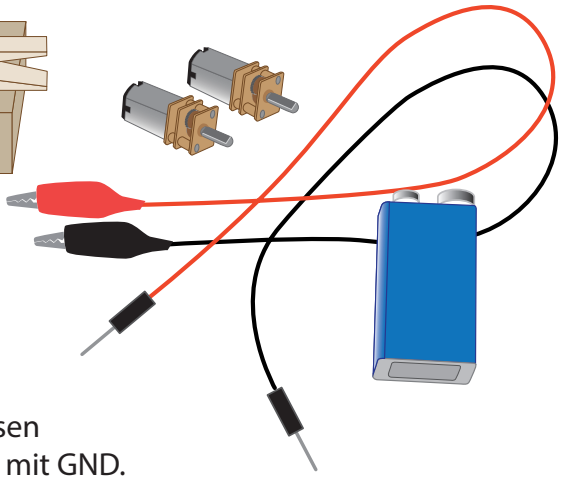
Beim Bergauffahren oder Fahren gegen den Wind empfiehlt sich Schaltstellung 1
Bei ebener Fahrbahn ohne Gegen- oder gar Rückenwind ist Schaltstellung 2 richtig.
Bergrunter muss gar nicht getreten werden. Das Rad rollt von selbst.

Anhang

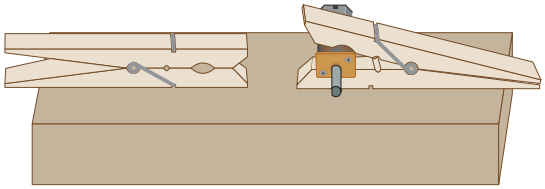
Aufgaben und Lösungen

Du brauchst:
Block mit Klammern, Motor, zwei Krokodilklemmen, PC mit Arduino UNO, Batterie

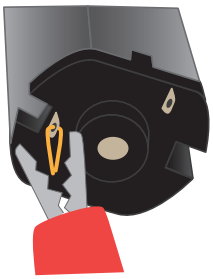




So gehst du vor:
Spanne den Motor wie dargestellt in eine Klammer. Ein Motoranschluss ist mit einem Plus gekennzeichnet. Mithilfe der Krokodilklemmen verbindest du diesen mit Pin 13 am Arduino und den anderen mit GND.



Dann schließt du den Arduino am PC an und startest die Arduino IDE. Über > Datei > Beispiele > Basics öffnest du das Programm „Blink“ und überträgst es zum Mikrocontroller.



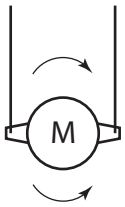
Die Klemmen finden besseren Halt, wenn ein Stück Draht durch die Ösen der Motoranschlüsse gesteckt und umgebogen wird.

Was kannst du beobachten, wenn das Hochladen des Blink-Programms abgeschlossen ist?

Erklärung?

Arduino

GND Pin 13



In welche Richtung deht die Welle? Streiche durch, welche Pfeilrichtung in der Abbildung nicht zutrifft!

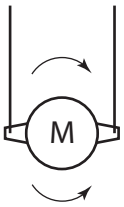
Vergleiche das Ergebnis mit einer Batterie, indem du den mit Plus markierten Anschluss am Motor mit dem Pluspol der Batterie und den anderen mit dem Minuspol verbindest. Streiche auch hier die nicht zutreffende Drehrichtung durch!

linksherum

rechtsherum

Batterie

(-) (+)



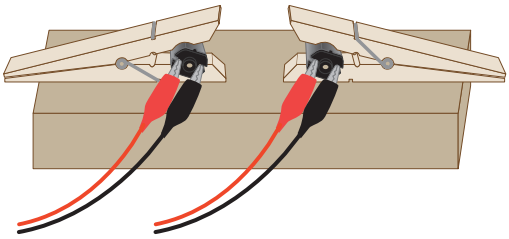
Aufgabe 2:

Suche im Internet nach dem Begriff „technische Stromrichtung“ und versuche aufzuklären, wie Strom- und Drehrichtung von Motoren zusammenhängen. Schreibe eine kurze Zusammenfassung deiner Einsichten!

Aufgabe 3:

Löse die Steckverbindung an GND und verbinde sie mit Pin 12 und passe die Programmierung so an, dass das Motorenverhalten gleich bleibt.

Aufgabe 4:



Schließe einen zweiten Motor an zwei weitere digitale Pins an und programmiere sie so, dass beide Motoren im Wechsel starten und stoppen. Wenn der eine an ist, ist der andere aus und umgekehrt.

Aufgabe 5:

Anschlüsse unverändert lassen, aber das Programm so umschreiben, dass beide Motoren gleichzeitig anlaufen und nach 5 Sekunden stoppen.

Aufgabe 6:

Anschlüsse bleiben, Programm aber so verändern, dass beide Motoren gleichzeitig anlaufen, einer der beiden nach 4 Sekunden für 8 Sekunden deutlich langsamer dreht, dann wieder Fahrt aufnimmt und mit dem anderen für 2 Sekunden weiterläuft, ehe beide stoppen.

Lösungen mit Bezug zur Arduino IDE

Aufgabe 1:
Die Motoren starten und stoppen im Sekundentakt. Aufgrund der Programmierung wird Pin 13 laufend zwischen HIGH (+) und LOW (-) umgeschaltet. Bei GND und (-) stoppt der Motor, bei GND und (+) läuft er.

Aufgabe 2:
Die „technische“ Stromrichtung besagt, dass Gleichtrom vom Plus- zum Minuspol fließt. Tatsächlichen bewegen sich die Elektronen aber in die entgegengesetzte Richtung. Die Motoren drehen in Fließrichtung.

Aufgabe 3:

```
void setup(){
    pinMode(13, OUTPUT);
    pinMode(12, OUTPUT);
}
void loop(){
    digitalWrite(13, HIGH);
    digitalWrite(12, LOW);
    delay(2000);
    digitalWrite(13, LOW);
    digitalWrite(12, HIGH);
    delay(2000);
}
```

Aufgabe 4:

```
void setup(){
    pinMode(13, OUTPUT);
    pinMode(12, OUTPUT);
    pinMode(9, OUTPUT);
    pinMode(8, OUTPUT);
}
void loop(){
    digitalWrite(13, HIGH);
    digitalWrite(12, LOW);
    delay(1000);
    digitalWrite(13, LOW);
    digitalWrite(9, HIGH);
    digitalWrite(8, LOW);
    delay(1000);
    digitalWrite(9, LOW);
    digitalWrite(13, HIGH);
}
```

Aufgabe 5:

```
void setup(){
    pinMode(13, OUTPUT);
    pinMode(12, OUTPUT);
    pinMode(9, OUTPUT);
    pinMode(8, OUTPUT);

    digitalWrite(13, HIGH);
    digitalWrite(12, LOW);
    digitalWrite(9, HIGH);
    digitalWrite(8, LOW);
    delay(5000);
    digitalWrite(9, LOW);
    digitalWrite(13, LOW);
}
void loop(){
}
```

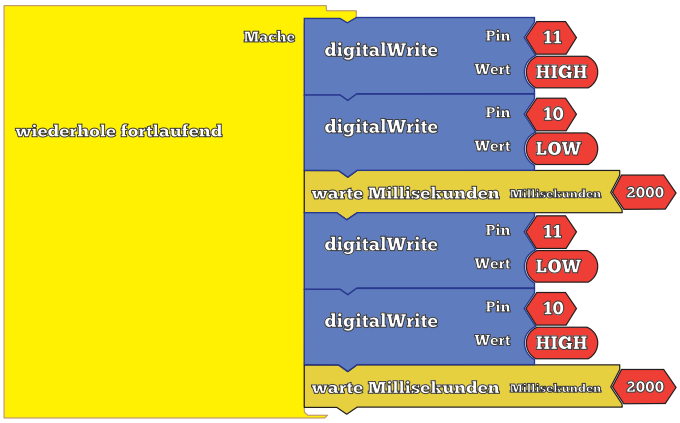
Aufgabe 6:

```
void setup(){
    pinMode(13, OUTPUT);
    pinMode(12, OUTPUT);
    pinMode(9, OUTPUT);
    pinMode(8, OUTPUT);

    digitalWrite(13, HIGH);
    digitalWrite(12, LOW);
    digitalWrite(9, HIGH);
    digitalWrite(8, LOW);
    delay(4000);
    analogWrite(13, 125);
    delay(8000);
    digitalWrite(13, HIGH);
    delay(2000);
    digitalWrite(9, LOW);
    digitalWrite(13, LOW);
}
void loop(){
}
```

Lösungen zu den Programmieraufgaben bei Verwendung von Ardublock unterschiedlicher Entwickler.

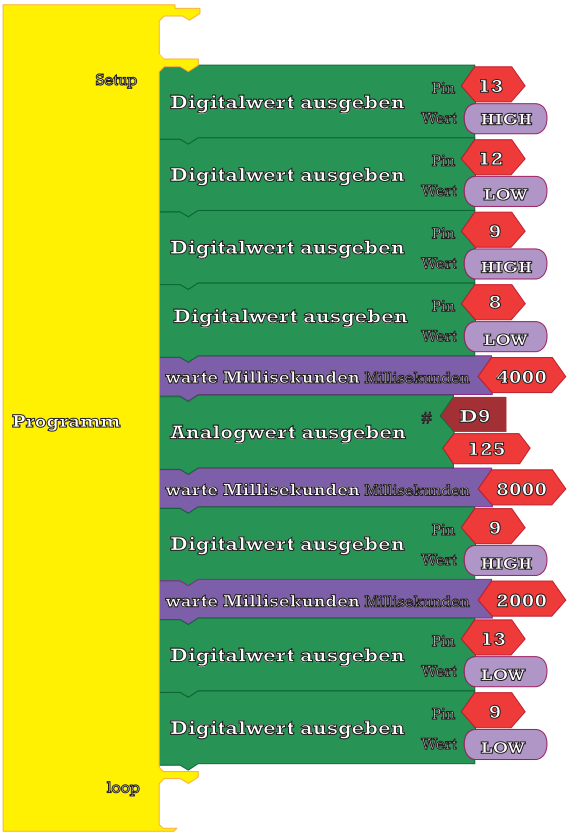
Lösungen mit Ardublock letsgoING Version 2.12
„ ardublock_letsgoing_21.jar“

Aufgabe 3:



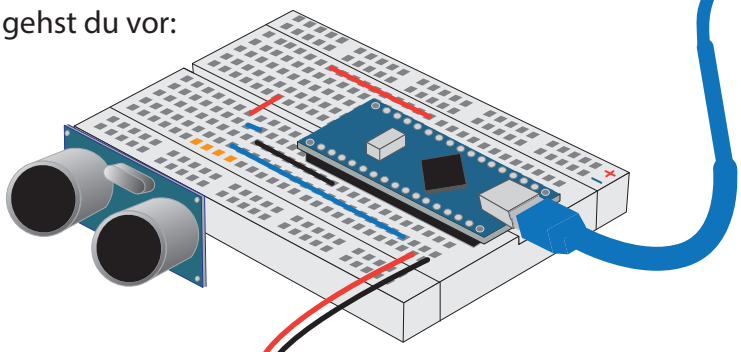
„ ardublock-all-2019-08-13.jar“

Aufgabe 6:

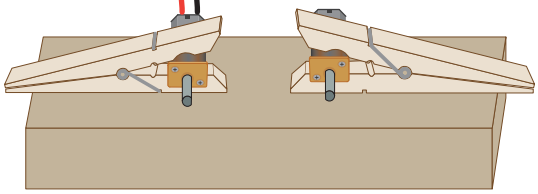


Du brauchst:
PC auf dem die Arduino IDE und Ardublock installiert
Arduino Mikrocontroller (hier ein Arduino Nano auf einem Breadboard) HC- SR04 Ultraschallsensor, Klammerblock mit Motoren, Krokoklemmen, Steckbrücken

So gehst du vor:



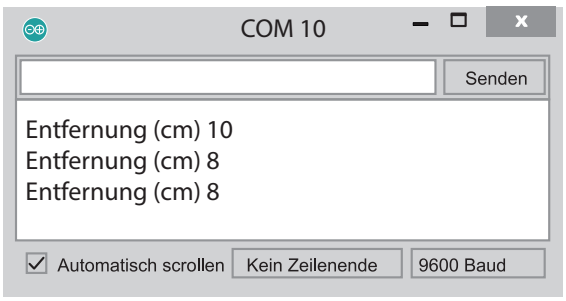
Verbinde die Teile so, wie aus der Abbildung ersichtlich. Die Kontakte am Arduino und Sensor sind gekennzeichnet: VCC an 5V , Trig und Echo überbrücken und an Pin 10 anschließen, Gnd an GND. Wo die Stifte des Sensors ihren Platz auf den Breadboard finden sollen ist orange markiert . Ein Beispielprogramm mit Erklärungen zur Funktionsweise des Sensors finet sich auf der Seite 21. Zur Vertiefung der dort vermittelten Einsichten hier einige Aufgaben:



Aufgabe 1:
Schließe einen oder beide Motoren an den Arduino an und übertrage das Beispiel-programm über Copy & Paste in die Arduino IDE. Stimme die Pinbelegung im Code auf die tatsächlich verwendeten Anschlüsse ab und lade das Programm auf den Mikrocontroller. Wenn alles richtig ist, solte ein Motor stoppen, wenn die Hand an den Sensor herangeführt wird.

Aufgabe 2:
Ziehe die Motoren vom Board ab und ersetze in dem Beispielprogramm alle Teile, die sich auf die Steuerung der Motoren beziehen und durch die abgebildeten Funktionn. Sie sollen das Programm so verändern, dass die Ergebnisse der Messungen mithilfe des HC-SR04 im seriellen Monitor der IDE angezeigt werden.

```
Serial.begin(9600);


Serial.print("Entfernung (cm)   ");
Serial.print(Entfernung);
Serial.println();
```

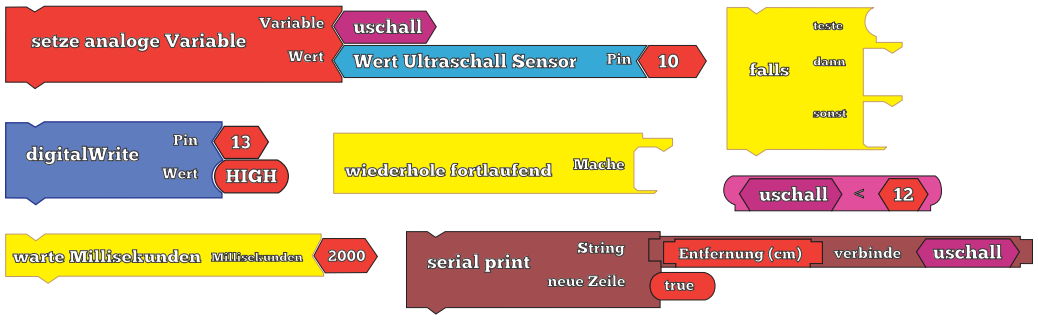


Den Monitor ruft man in der IDE mit der Tastenkombination <Shift+Strg+M> auf. Zu einer Anzeige kommt es nur, wenn sich ein Gegenstand dichter als 12 cm vor dem Sensor befindet. .

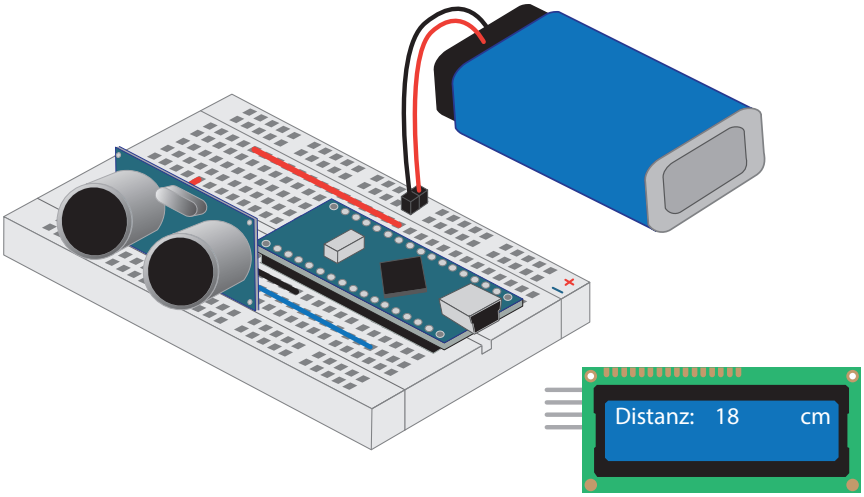
Aufgabe 3:
Wenn nicht versehentlich gelöscht, sollte neben der „if“-Bedingung auch das „else“ noch in dem Programm vorhanden sein. Füge hier ein, dass die eingebaute LED auf dem Board an Pin 13 und GND blinken soll, solange kein Objekt erkannt wird.

Aufgabe 4:
Programmiere den Mikrocontroller ohne die „if - else“ Bedingung und versuche herauszufinden, wie weit der Sensor sehen kann.

Aufgabe 5:
Realisiere alle Programmierung mit Ardublock. Nutze zur Lösung folgende Blöcke:



Zusatzaufgabe:
Baue einen Entfernungsmesser, der handlich und unabhängig vom PC ist.
Näheres unter:
http://mint-unt.de/lcd-display-2.html> - http://mint-unt.de/distanzmessung-2.html



1602 I2C Display mit
2 Zeilen á 16 Zeichen

Lösungen mit Bezug zur Arduino IDE

```
int sensorPin = 10;

void setup() {

}

void loop() {
  long Laufzeit, Entfernung;
  pinMode(sensorPin, OUTPUT);
  digitalWrite(sensorPin, LOW);
  delayMicroseconds(2);
  digitalWrite(sensorPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(sensorPin, LOW);
  pinMode(sensorPin, INPUT);
  Laufzeit = pulseIn(sensorPin, HIGH);
  Entfernung = Laufzeit/58.2;
  if (Entfernung < 12) {

    delay(2000);
  }
  else {

  }
}
```

Beispielprogramm ohne Steuerbefehle für die Motoren

Beispielprogramm mit den Funktionen zur Anzeige der Messungen unterhalb von 12 cm im seriellen Monitor der IDE. Die eingebaute LED blinkt, wenn kein Objekt detektiert wird.

Startet den seriellen Monitor mit 9600 Baud

Limitiert die Messungen auf 12 cm

Sorgt für den Begleittext

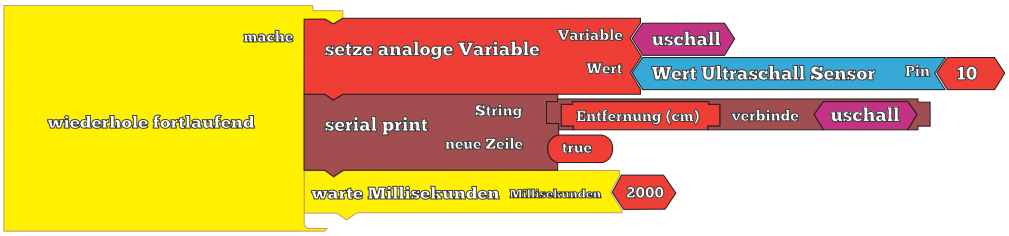
Fügt den Messwert ein

Erzeugt eine neue Zeile

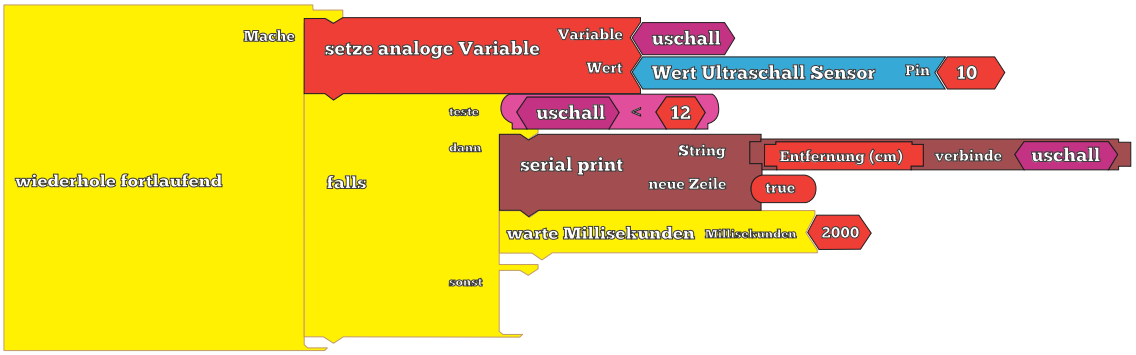
Lässt die eingebaute LED blinken

```
int sensorPin = 10;

void setup() {
  Serial.begin(9600);
}

void loop() {
  long Laufzeit, Entfernung;
  pinMode(sensorPin, OUTPUT);
  digitalWrite(sensorPin, LOW);
  delayMicroseconds(2);
  digitalWrite(sensorPin, HIGH);
  delayMicroseconds(5);
  digitalWrite(sensorPin, LOW);
  pinMode(sensorPin, INPUT);
  Laufzeit = pulseIn(sensorPin, HIGH);
  Entfernung =  Laufzeit / 29 / 2;
  if (Entfernung < 12) {
  Serial.print("Entfernung (cm)  ");
  Serial.print(Entfernung);
  Serial.println();
  delay( 2000 );
  }
  else {
    digitalWrite(13, HIGH);
    delay( 2000 );
    digitalWrite(13, LOW);
    delay( 2000 );
  }
}
```

Bringt alle Messungen auf den seriellen Monitor



Bringt nur Messungen unterhalb von 12 cm auf den seriellen Monitor

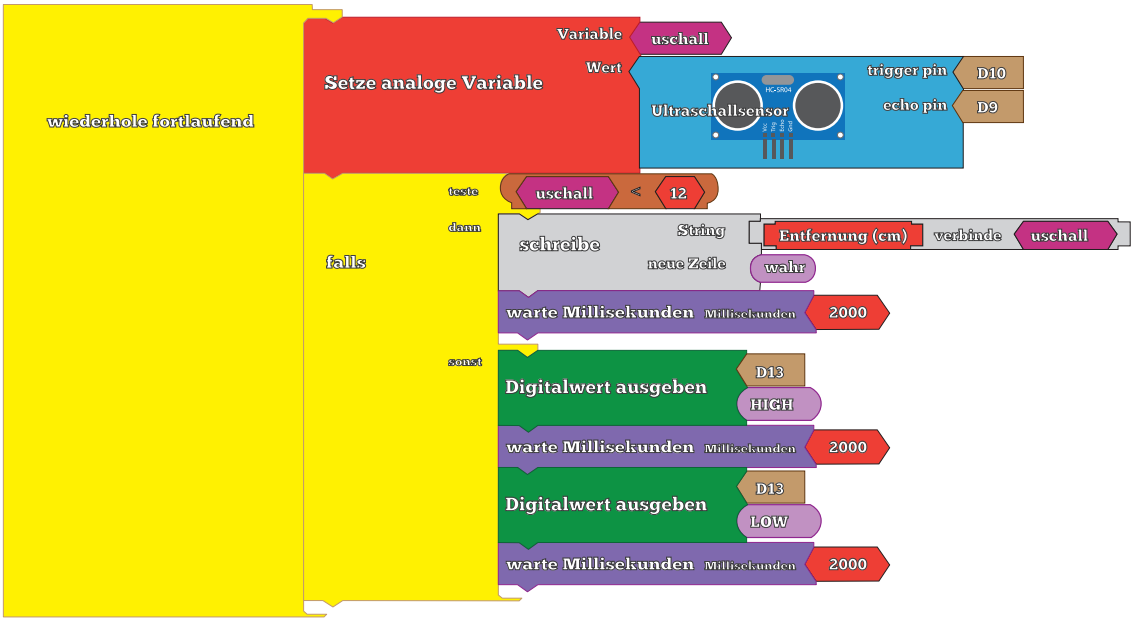


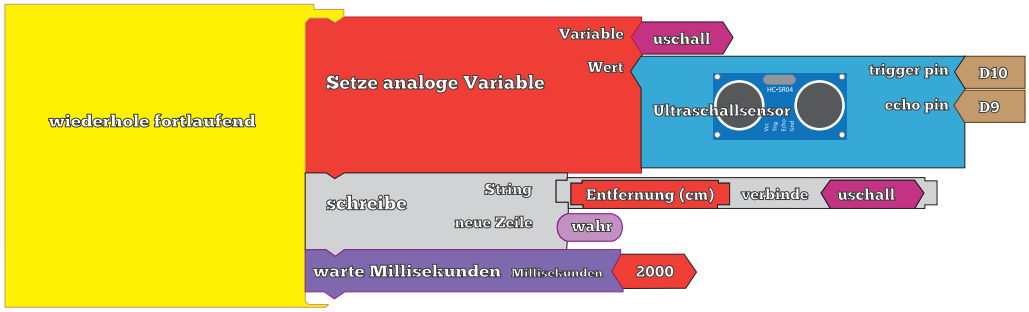
Wie oben, lässt aber die eingebaute LED blinken, wenn keine Objekte unterhalb der festgelegten Grenze erkannt werden

Ergänzender Hinweis:

Soweit bekannt, ist nur die Ardublock letsgoING Version darauf ausgelegt, dass der Ultraschallsensor HC-SR04 seine Daten über eine einzige Datenleitung sendet und empfängt. Die Versionen anderer Anbieter tun das nicht und bedienen getrennte Output/Input Pins. In einem solchen Fall müssen Trig und Echo mit verschiedenen Pins am Arduino verbunden werden. In dem hier gezeigten Beispiel sind das die Pins 9 und 10.

Software:: ardublock-all-2019-11-20





Der zu dieser Ardublock-Version gehörige Arduino-Code:

```
int ardublockUltrasonicSensorCodeAutoGeneratedReturnCM(int trigPin, int echoPin) {
  long duration;
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(20);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  duration = duration / 59;
  if ((duration < 2) || (duration > 300)) return false;
  return duration;
}

int uschall = 0;

void setup() {
  pinMode(13, OUTPUT);
  digitalWrite( 10 , LOW );

  Serial.begin(9600);
}

void loop() {
  uschall =       ardublockUltrasonicSensorCodeAutoGeneratedReturnCM( 10 , 9 );
  if (( uschall < 12 )) {
    Serial.print("Entfernung (cm) ");
    Serial.print(uschall );
    Serial.println();
    delay( 2000 );
    }
  else {
    digitalWrite( 13 , HIGH);
    delay( 2000 );
    digitalWrite( 13 , LOW );
    delay( 2000 );
  }
}
```


mint-unt.de


BY
SA