Die H-Brücke

Möchte man Projekte mit Motoren umsetzen, die mehr Strom brauchen oder variabel in der Geschwindigkeit sein sollen, so greift man meist zu Modulen mit einer sogenannten H-Brücke. Zu finden sind solche Module unter dem Oberbegriff **Motor Driver** oder **DC Stepper**. In vielen Fällen, gerade wenn man sie bei Onlineshops sucht, findet man diese als sogenannte Doppel H-Brücke bzw. Dual H-Bridge. Aber was genau ist eine H-Brücke und wie setzt man diese genau ein, diese Grundfrage und auch alle weiteren Fragen zum Einsatz dieser Bauteile soll dieser Blogbeitrag klären.

Die Hard- und Software für diesen Blog

Die Bauteile aus Tabelle 1 dienen für diesen Blog als Referenzhardware, jedoch kann vom Prinzip jede H-Brücke oder Micro Controller genutzt werden.

Pos	Anzahl	Bauteil
1	1	Nano V3.0 with Atmega328 CH340 oder Nano V3.0 with FT232RL chip and ATmega328
2	1	ESP32 Dev Kit C V4 unsoldered oder ESP32 Dev Kit C unsoldered
3	1	MB 102 Breadboard Kit - 830 Breadboard, Netzteil Adapter 3,3V 5V, 65Stk Steckbrücken für Arduino
4	2	Netzteil 12V
5	1	Dual H-Brücke
6	1	2 Sätze DC-Zahnrad-Motor und Reifen-Rad für DC 3V-6V Arduino intelligente Auto-Roboter-Projekte

Tabelle 1: Hardware für Test

Damit dieser Blog auch von Ihnen nachgebaut werden kann, benötigen Sie folgende Software und Bibliotheken:

- Arduino IDE (https://www.arduino.cc/en/Main/Software), hier am besten die aktuellste Version herunterladen

Die H-Brücke oder doch Vierquadrantensteller

Fängt man im Internet mit der Suche nach der H-Brücke an, wird man meist zwei Definitionen finden. Die eine ist die sogenannte Brückenschaltung, die andere der Vierquadrantensteller. Beide sind H-Brücken, unterscheiden sich aber in ein paar Details. Streng genommen ist ein Motor Driver oder DC Stepper keine H-Brücke, sondern ein Vierquadrantensteller. Warum das so ist, soll im Vergleich geklärt werden.

Die H-Brücke - Brückenschaltung

Die erste H-Brücke in diesem Blog wird die Brückenschaltung sein. Diese ist auch in Fachkreisen unter den Begriffen H-Schaltung oder Viertel-, Halb- oder Vollbrücke bekannt. Fälschlicherweise wird Sie

auch in einigen Foren als Wheatstone‘sche Messbrücke bezeichnet, was so aber nicht korrekt ist. Die gibt es auch, aber damit ist etwas anderes gemeint.

Die H-Brücke, im weiteren Verlauf Brückenschaltung genannt, hat Ihren Namen aufgrund der Anordnung der elektrischen Bauteile, meist Widerstände, siehe Abbildung 1.

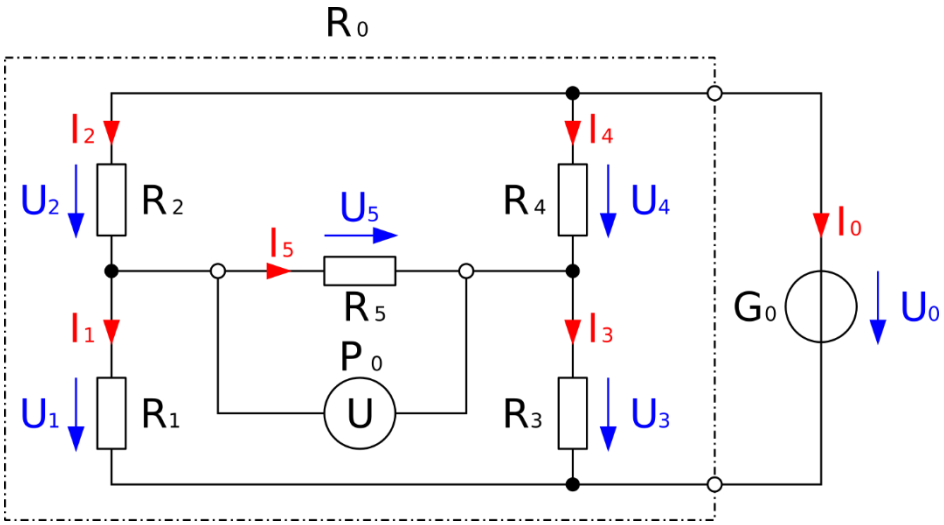

Abbildung 1: Grundaufbau Brückenschaltung, von Wikipedia MovGP0

Nimmt man die oben gezeigte Schaltung als Grundlage, ist eine Brückenschaltung nichts anderes wie eine Parallelschaltung zweier Spannungsteiler. Zwischen beiden Spannungsteilern befindet sich aber ein Abzweig, weswegen die Grundschaltung einem „H“ gleicht und somit der Name H-Brücke abgeleitet wurde. Diese Brücke kann sowohl für Gleichspannung als auch für Wechselspannung genutzt werden, es kommt aber darauf an, ob man Widerstände, Kondensatoren oder Spulen messen will. Eine weitere Eigenschaft der Brückenschaltung ist, dass man je nach Einstellung der Widerstände nicht nur den Strom und die Spannung variieren, sondern auch die Polarität verändern kann! Je nachdem wie viele Widerstände variabel sind, nennt man die Brückenschaltung auch Viertel- (ein Widerstand), Halb- (zwei Widerstände) oder Vollbrücke (alle Widerstände).
Das Problem an dieser Stelle ist, es ist nicht so einfach, die Widerstände für einen Motor so schnell zu verändern.

Die H-Brücke – Vierquadrantensteller

Zum oben genannten Problem kommen wir direkt zur nächsten H-Brücke in unserem Blog. Diese wird auch Vierquadrantensteller genannt und übernimmt genau die Aufgabe, die zum Ansteuern eines Motors gebraucht wird. Diese H-Brücke, im weiteren Vierquadrantensteller genannt, finden Sie in vielen Geräten mit einem Motor. Vom kleinsten Fahrzeug bis hin zum Beispiel im ICE der Deutschen Bahn wird diese verwendet.
Schaut man sich den Vierquadrantensteller genau an, siehe Abbildung 2, ist der Grundaufbau ähnlich zu der Brückenschaltung.

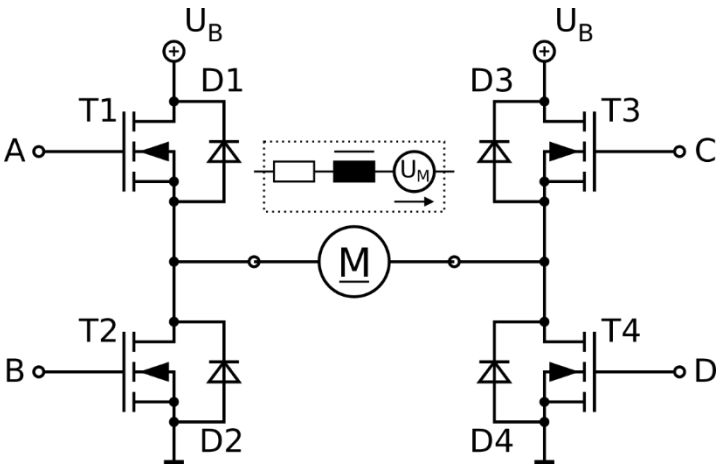

Abbildung 2: Grundaufbau Vierquadrantensteller, von Wikipedia Biezl

Der Unterschied liegt darin, dass die Widerstände der Brückenschaltung durch MOSFETs und Sperrdioden ausgetauscht wurden. In der Zwischenverbindung der beiden Parallelschaltungen liegt der Motor. Dieser kann, ähnlich wie beim Verändern der Widerstände der Brückenschaltung, die Polarität verändern. Die Spannung und der Strom müssen an den Anschlussklemmen korrekt eingestellt werden.
Damit ist schon einmal geklärt, wie der Vierquadrantensteller grob funktioniert. Jetzt muss noch geklärt werden, wie der Vierquadrantensteller zu seinem Namen kommt. Je nachdem wie die MOSFETs (T1 bis T4) geschaltet werden, kann man einen der vier Zustände, die Quadranten, für den Motor nutzen. Tabelle 2 listet alle vier Quadranten auf, wobei für T1 bis T4 die MOSFETs aus Abbildung 2 gemeint sind.

Quadrant	T1	T2	T3	T4	Motorlauf
1	Geschaltet	Offen	Offen	Geschaltet	Vorwärts beschleunigen
2	Offen	Geschaltet	Offen	Geschaltet	Vorwärts bremsen
3	Offen	Geschaltet	Geschaltet	Offen	Rückwärts beschleunigen
4	Geschaltet	Offen	Geschaltet	Offen	Rückwärts bremsen

Tabelle 2: Die vier Quadranten:

Je nach geschalteten MOSFETs kann man so jede Drehrichtung und die Art der Beschleunigung des Motors auswählen. Wie schon oben geschrieben, müssen die Spannung und der Strom entsprechend von der Spannungsquelle kommen und können nicht im Vierquadrantensteller selbst verändert werden.

Der Motor Driver in der Praxis

Da jetzt die Theorie erklärt ist, soll nun der Motor Driver in der Praxis zusammengebaut und verwendet werden. Die Bauteile dafür finden Sie in Tabelle 1. Grundsätzlich sind alle unsere Micro Controller mit Ausnahme des ESP-01 für diese Aufgabe geeignet.

Den Anfang macht zunächst der Nano V3.0. Dabei ist es egal, ob Sie den Nano nun über das USB-Kabel, siehe Abbildung 3, oder über eine externe Spannungsversorgung, z.B. auch über die

verfügbaren 5V des hier verwendeten Motor Driver, siehe

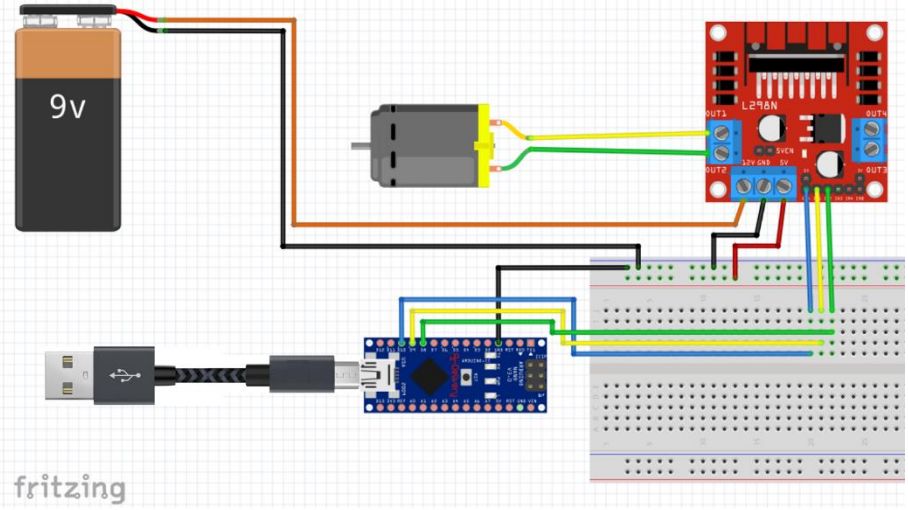

Abbildung 4, betreiben. Die Spannungsversorgung der Motoren muss in jedem Fall über eine Batterie oder einen Akku erfolgen.

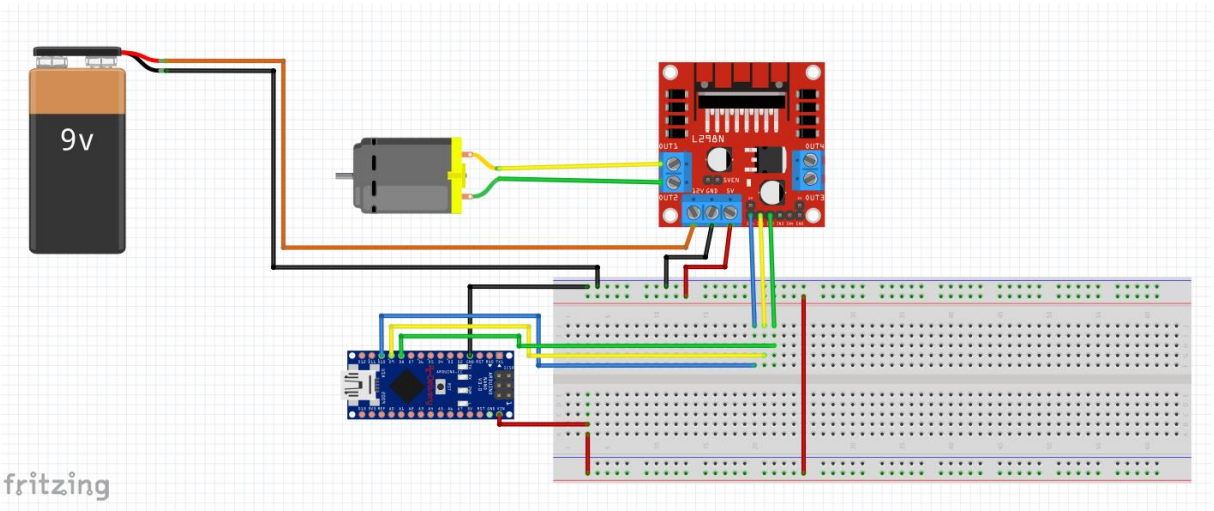

Abbildung 3: Nano mit USB-Versorgung

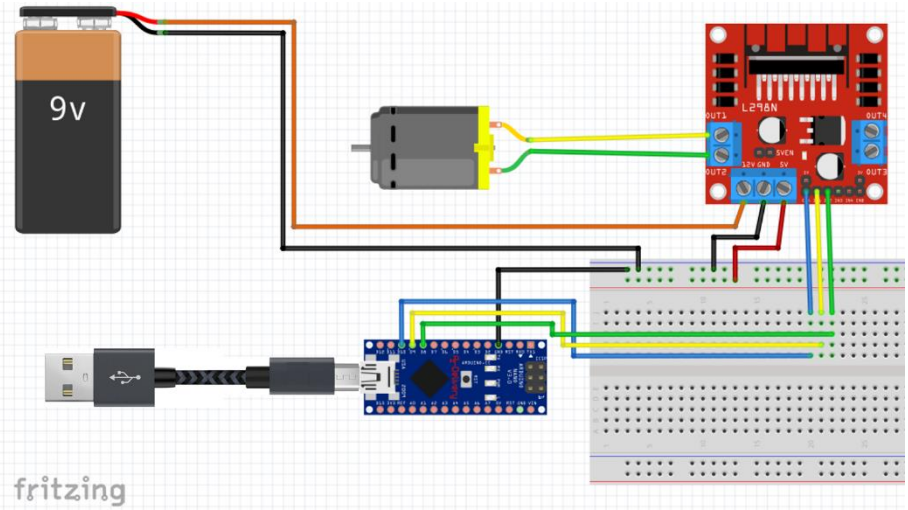

Abbildung 4: Nano mit 5V Versorgung der H-Brücke

Bitte beachten Sie, sollten Sie Spannungen über 12V am Motor Driver verwenden, muss der 5V-Jumper auf dem Motor Driver entfernt werden und eine separate Spannungsquelle für den Micro Controller ist notwendig.

Für die einfache Demonstration, wie Sie einen kleinen Motor ansteuern können, können Sie Code 1 verwenden.

```
//-----------------------------------------------------
// Controlling a DC-Motor with Nano V3.0
// Autor:   Joern Weise
// License: GNU GPl 3.0
// Created: 23. Feb 2021
// Update:  23. Feb 2021
//-----------------------------------------------------

//PWM and motor configuration
// Motor A
const int motor1Pin1 = 9;
const int motor1Pin2 = 8;
const int enable1Pin = 10;

void setup() {
  Serial.begin(115200);
  while (!Serial) {
    ; //Wait until serial is available
  }
  Serial.println("Motor control with NANO V3.0");
  Serial.println("(c) Joern Weise for AZ-Delivery");
  Serial.println("--------------------");
  Serial.println("Set outputs");

  //Set pins as outputs
  pinMode(motor1Pin1, OUTPUT);
  pinMode(motor1Pin2, OUTPUT);
  pinMode(enable1Pin, OUTPUT);

  Serial.println("Setup finished");
}

void loop() {
  Serial.println("----- Motor clockwise speedup-----");
  digitalWrite(motor1Pin1,HIGH); // A = HIGH and B = LOW means the motor will turn right
  digitalWrite(motor1Pin2,LOW);
  for (int i=0; i<256; i+=5)
  {
    Serial.println("Speed:" + String(i));
    analogWrite(enable1Pin, i);
    delay(100);
  }

  Serial.println("----- Motor clockwise break-----");
  for (int i=255; i>0; i-=5)
  {
    Serial.println("Speed:" + String(i));
    analogWrite(enable1Pin, i);
    delay(100);
```

```
 }

 Serial.println("----- Motor counterclockwise speedup-----");
 digitalWrite(motor1Pin1,LOW); // A = HIGH and B = LOW means the motor will turn right
 digitalWrite(motor1Pin2,HIGH);
 for (int i=0; i<256; i+=5)
 {
  Serial.println("Speed:" + String(i));
  analogWrite(enable1Pin, i);
  delay(100);
 }
 Serial.println("----- Motor counterclockwise break-----");
 for (int i=255; i>0; i-=5)
 {
  Serial.println("Speed:" + String(i));
  analogWrite(enable1Pin, i);
  delay(100);
 }
}
```

Code 1: Democode für Nano V3.0

Vom Prinzip macht der Code nicht viel Spektakuläres. Er startet den Motor einmal im Uhrzeigersinn und beschleunigt diesen bis zur Maximalgeschwindigkeit. Direkt danach wird der Motor wieder bis zum Stillstand abgebremst. Um zu zeigen, dass dies nicht nur in eine Richtung funktioniert, wird entsprechend gegen den Uhrzeigersinn dieselbe Ansteuerung noch einmal durchgeführt.
Ist die loop-Funktion einmal durchlaufen, beginnt der Ablauf wieder von Neuem. Damit Sie schnell verstehen, wo was passiert, sind im Quellcode reichlich Kommentare eingefügt worden.

Das bringt uns an dieser Stelle zu der ESP-Familie. Egal ob nun ein Controller aus der ESP32- oder ESP8266-Familie, der Anschluss ist auch hier nahezu identisch. Die Verkabelung für die Stromversorgung mit USB-Kabel, siehe Abbildung 5, oder mit dem 5V Anschluss des Motor Driver, siehe Abbildung 6, sind ebenfalls schnell erledigt.

Noch ein Wort zu dem hier verwendeten Motor Driver Board L298N. Gegenüber den L293D Boards, die ebenfalls geeignet sind, ist hier bereits ein Kühlkörper für höhere Ströme und damit größere Motoren verbaut. Nachdem wir anfänglich für die Micro Controller von Espressif einen Logic Level Converter benutzt haben, um die Ausgangsspannung an den 3,3V GPIOs der ESPs auf 5V anzupassen, haben wir bei weiteren Versuchen festgestellt, dass die Spannung am Enable Pin des MotorDrivers eine Art Referenzspannung für die PWM-Signale ist; also ein Tastgrad/Duty Cycle von 100% auch bei 3,3V die volle Ausgangsspannung für die Motoren liefert. Deshalb kann auf den LLC verzichtet werden.

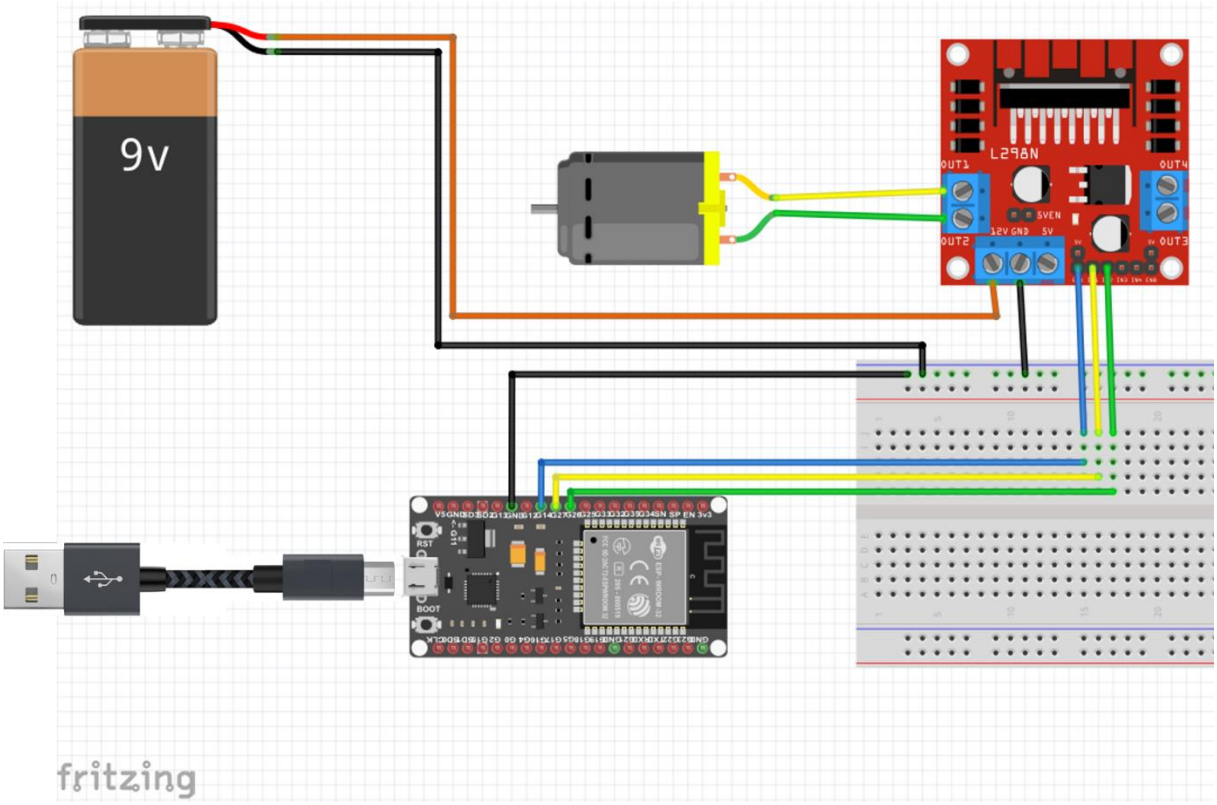

Abbildung 5: MicroController der ESP32-Familie mit USB-Versorgung

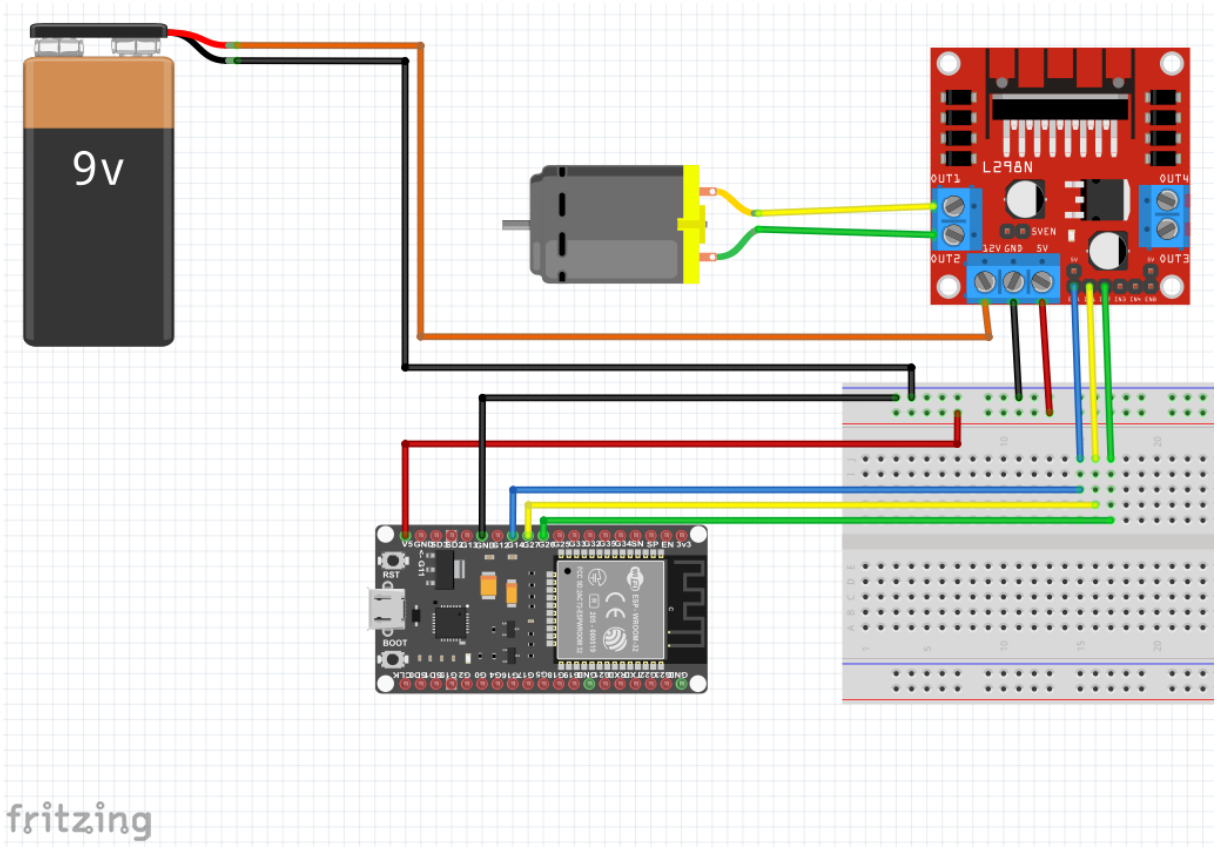

Abbildung 6: MicroController der ESP32-Familie mit 5V-Versorgung

Auch an dieser Stelle gilt, bei Spannungen über 12V am Motor Driver, ist der 5V-Jumper zu entfernen und eine separate Spannungsquelle für den Micro Controller notwendig.

Das Beispiel vom Nano V3.0 können wir leider nicht komplett übernehmen, siehe Code 2, da das PWM-Signal anders programmiert werden muss und wir zudem auch andere Pins nutzen.

```
//----------------------------------------------------
// Controlling a DC-Motor with ESP
// Autor:   Joern Weise
// License: GNU GPl 3.0
// Created: 23. Feb 2021
// Update:  23. Feb 2021
//----------------------------------------------------

//PWM and motor configuration
// Motor A
const int motor1Pin1 = 27;
const int motor1Pin2 = 26;
const int enable1Pin = 14;
const int motor1channel = 0;

// Setting PWM properties
const int freq = 30000;
const int resolution = 8;

void setup() {
  Serial.begin(115200);
  while (!Serial) {
    ; //Wait until serial is available
  }
  Serial.println("Motor control with ESP");
  Serial.println("(c) Joern Weise for AZ-Delivery");
  Serial.println("--------------------");
  Serial.println("Set outputs");

  //Set pins as outputs
  pinMode(motor1Pin1, OUTPUT);
  pinMode(motor1Pin2, OUTPUT);
  pinMode(enable1Pin, OUTPUT);

  Serial.println("Configurate motorchannel");
  ledcSetup(motor1channel, freq, resolution); //Configurate PWM for motor 1
  ledcAttachPin(enable1Pin, motor1channel); //Attach channel 1 to motor 1
  Serial.println("Setup finished");
}

void loop() {
  Serial.println("----- Motor clockwise speedup-----");
  digitalWrite(motor1Pin1,HIGH); // A = HIGH and B = LOW means the motor will turn right
  digitalWrite(motor1Pin2,LOW);
  for (int i=0; i<256; i+=5)
```

```
 {
  Serial.println("Speed:" + String(i));
  ledcWrite(motor1channel, i);
  delay(100);
 }
 Serial.println("----- Motor clockwise break-----");
 for (int i=255; i>0; i-=5)
 {
  Serial.println("Speed:" + String(i));
  ledcWrite(motor1channel, i);
  delay(100);
 }

 Serial.println("----- Motor counterclockwise speedup-----");
 digitalWrite(motor1Pin1,LOW); // A = HIGH and B = LOW means the motor will turn right
 digitalWrite(motor1Pin2,HIGH);
 for (int i=0; i<256; i+=5)
 {
  Serial.println("Speed:" + String(i));
  ledcWrite(motor1channel, i);
  delay(100);
 }
 Serial.println("----- Motor counterclockwise break-----");
 for (int i=255; i>0; i-=5)
 {
  Serial.println("Speed:" + String(i));
  ledcWrite(motor1channel, i);
  delay(100);
 }
}
```

Code 2: Democode für MicroController der ESP32-Familie

Wie schon beim Nano V3.0 sind auch in diesem Code viele Kommentare eingefügt, sodass Sie schnell sehen können, wo was an welcher Stelle passiert. Achten Sie vor allem auf die setup-Funktion, bei der die PWM deklariert wird. Durch „ledcSetup(motor1channel, freq, resolution);" deklariert man einen PWM-Kanal mit einer 8bit-Auflösung und einer Frequenz von30kHz. Direkt danach wird mittels "ledcAttachPin(iPWMPin, motor1channel);" der PWM-Kanal mit dem Ausgangspin verknüpft.

Wie sie sehen, ist das Ansteuern eines Motors mittels einer H-Brücke bzw. besser eines Motor Driver kein großes Hexenwerk. Das Prinzip hinter der Ansteuerung ist einfach, rein die Logik, wann der Motor was machen soll, muss in Ihrem Code sauber verarbeitet werden. Ein einfaches Beispiel dafür finden Sie in dem Blogbeitrag Mit MQTT einen Roboter steuern - [Teil 3]. Eine solche H-Brücke ist aber nötig, um Motoren in einer beliebigen Geschwindigkeit und Richtung drehen zu lassen.
Freuen Sie sich jetzt schon auf die weiteren Beiträge zum Schwerpunktthema Robot Cars, wo Sie eine H-Brücke bzw. Motor Driver mit großer Wahrscheinlichkeit wieder finden werden.

Dieses und weitere Projekte finden sich auf GitHub unter https://github.com/M3taKn1ght/Blog-Repo.