Motor-Steuerung

Einen Gleichstrom-Motor kann man nicht direkt an das Arduino anschließen. Zum einen stellt ein Elektro-Motor eine induktive Last dar, das bedeutet, dass in ihm selbst eine Spannung induziert wird, die ihrer Ursache entgegen wirkt und unter ungünstigen Bedingungen sogar höher sein kann als die ihm zugeführte. Zum anderen kann ein Motor weit mehr Strom brauchen als ein Arduino liefern kann:
max. 60mA pro Ausgangs-Pin, max. 600mA insgesamt
Man würde das Arduino damit unweigerlich zerstören, wenn man einen Gleichstrom-Motor direkt an einen digitalen Ausgangs-Pin anschließen wir!

Um dieses Projekt meistern zu können solltest du vorher unbedingt folgende technische Dokumentation angesehen und verstanden haben

→ **Motor-Teiber, H-Brücke** **(ganz wichtig !!!)**

Ziel des Projekts

Du sollst den prinzipiellen Aufbau und die Funktion eines Motor-Treibers (einer H-Brücke) verstehen und nachvollziehen.

Die Motor-Brücke holt sich vom Arduino nur Spannungs-Signale zur Steuerung des Motors. Der Strom kann bei Bedarf von einer anderen, zusätzlichen Spannungs-Versorgung kommen. Auch laufen nicht alle Motoren mit 5V, wie sie das Arduino ausgibt. Der von uns verwendete Motor-Treiber kann bis zu zwei Motoren mit bis zu 12V und insgesamt 4A (also bis insgesamt 48W) betreiben …

Für jeden Motor braucht man 2 digitale Ausgans-Pins, mit denen man die Dreh-Richtung angibt. Jeweils ein weiterer PWM-Pin kann benutzt werden, um die Drehzahl des Motors zu regeln.

Hardware

Du benötigst

1 Motor-Treiber (H-Brücke)
1 Gleichstrom-Motor *(hier ein Getriebe-Motor)*
(am besten für 5V, damit die weitere Spannungs-Quelle entfallen kann)
1 Arduino (Uno)
1 Sensor-Shield
mehrere *Kabel* (male/female , female/female)

Für die einzelnen Projekte sind noch zusätzliche Bauteile erforderlich, die dort genannt werden.

Prinzipieller Aufbau

Für die prinzipielle Beschaltung wirfst du bitte einen Blick auf die Dokumentation der H-Brücke!

Bei diesen Versuchen kann auf den Anschluss einer zusätzlichen, externen Stromversorgung verzichtet werden.

Für die Stromversorgung kannst du beliebige G- bzw. V-Pins des Sensor-Shield verwenden.

Für die Regelung der Dreh-Richtung des Motors verwendest z.B. die digitalen Ausgangs-Pins 2 und 4.

Das PWM-Signal kannst du z.B. vom digitalen Pin 3 erhalten.

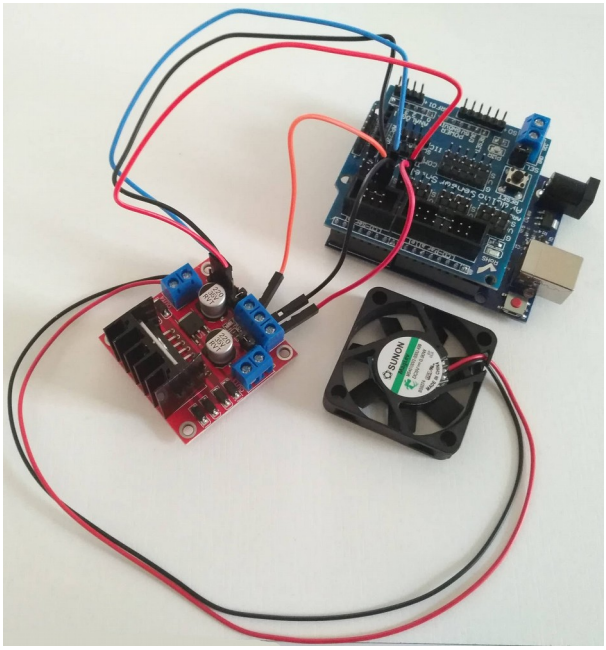
Ventilator

Technische Voraussetzungen

Der hier verwendete Ventilator kann – bedingt durch seine Bauart – nur ein und aus geschaltet werden.

Deshalb bleibt der PWM-Jumper-Stecker bei diesem Versuch auf der H-Brücke.

Streng genommen brauchst du für diesen Versuch auch nur ein Steuer-Kabel, z.B. von Pin 2. Der Grund ist, dass der Motor-Treiber intern seine Eingänge auf GND „zieht, wenn kein anderes Signal empfangen wird.



Die Signale, die du vom Arduino an die Eingänge des Motor-Treibers schickst, werden genauso *(nur mit mehr Strom und gegebenenfalls anderer Spannung)* an die Motor-Ausgänge übertragen. Sendest du also ein HIGH auf den Eingang, an dessen entsprechendem Ausgang das rote Kabel des Ventilators angeschlossen ist, so läuft der Ventilator auch schon …

Die Software

1. Erzeuge interaktiv durch Rechtsklick in BlueJ ein Objekt der Klasse PinOutput. Steuere mit diesem Pin den Eingang des Motor-Treibers, dessen entsprechender Motor-Ausgang mit dem roten Kabel des Ventilators verbunden ist.
   Schalte damit den Ventilator ein und wieder aus.
2. Wenn du das Projekt „Temperatur-Messung“ schon durchgemacht hast, so kannst du nun eine ***Klasse Ventilator*** schreiben, die ein Referenz-Attribut auf den Ventilator hat und ein weiteres Referenz-Attribut auf einen analogen Eingangs-Pin, an dem du den Temperatur-Sensor anschließt. Erzeuge beide Objekte in deinem **Konstruktor** an den entsprechenden Pins.
   Lies den Temperatur-Sensor vorher interaktiv aus, damit du einen Wert für „Zimmer-Temperatur“ hast.
   Schreibe nun eine ***Methode*** `aktivieren()` , in deren Rumpf eine Endlos-Wiederholung folgendes immer wieder tut:
   Wenn der Wert des Temperatur-Sensors den Wert für „Zimmer-Temperatur“ um 50 überschreitet, dann soll der Ventilator eingeschaltet werden.
   Wenn der Wert unter „Zimmer-Temperatur + 50“ sinkt, dann wird der Ventilator wieder ausgeschaltet. *(Experimentiere etwas mit dem Wert 50)*
   Alternativ kannst du auch die fertige, selbst geschriebene Klasse *TempMessung* verwenden und die Fallunterscheidung in °C vornehmen.
   Teste deine Schaltung, indem du den Temperatur-Sensor mit warmen Fingern erwärmst. Du kannst auch VORSICHTG ein Feuerzeug nähern, aber verbrenne den Sensor nicht!

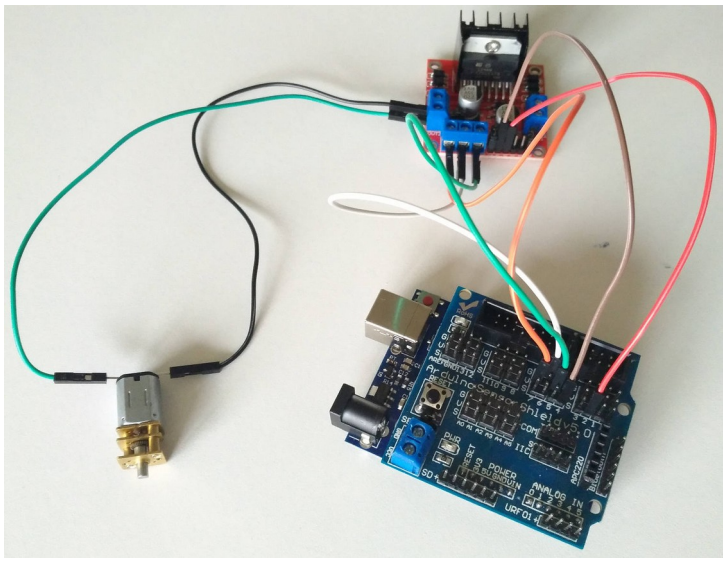
Getriebe-Motor

Technische Voraussetzungen

Der hier verwendete Getriebe-Motor kann vorwärts und rückwärts betrieben werden.
Später im Projekt werden wir ihn auch in der Geschwindigkeit regeln.

In diesem Projekt bleibt der PWM-Jumper-Stecker erst mal auf der H-Brücke, aber du verwendest nun zwei Steuerungs-Kabel *(IN1 und IN2)*, z.B. auf den Pins 2 und 4.

Sendest du auf einem Pin ein HIGH und auf dem anderen ein LOW, so dreht sich der Motor in die eine Richtung, vertauschst du HIGH und LOW, so dreht er sich in die andere Richtung.



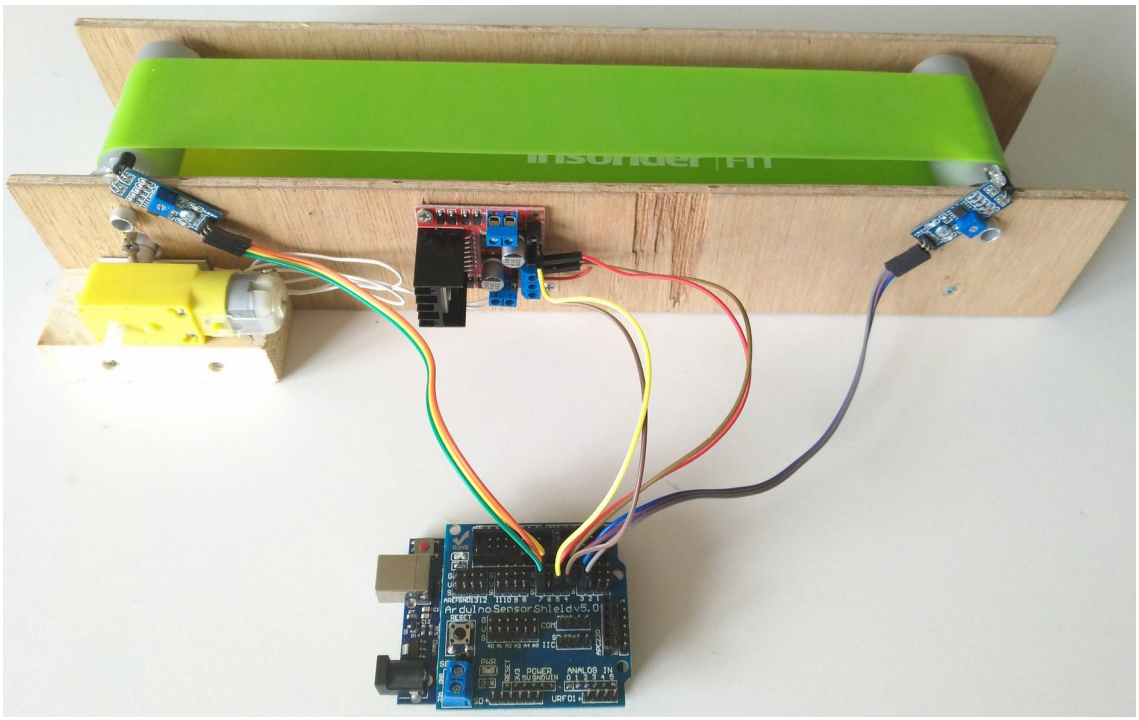
Die Software

3. Schreibe eine ***Klasse Motor***, die zwei Referenz-Attribute des Typs PinOutput hat.
   Erzeuge die beiden Objekte im ***Konstruktor*** an den entsprechenden Pins.
4. Schreibe eine ***Methode*** `vorwaerts()`, die den einen Pin auf HIGH und den anderen auf LOW setzt.
   Schreibe analog eine ***Methode*** `rueckwaerts()`.
   Schreibe auch eine ***Methode*** `stopp()`, die beide Pins auf LOW setzt.
5. Teste deine Klasse interaktiv in BlueJ.
6. Nimm nun den entsprechenden ***Jumper-Stecker*** vom dem Motor-Treiber *(hebe ihn gut auf)* und verbinde z.B. den PWM-Pin 3 des Arduino mit dem vorderen ENA-Pin des Motor-Treibers.
   Der hintere Pin bleibt frei.
7. ***Ergänze deine Klasse Motor*** um ein Referenz-Attribut des Typs PinPWM und erzeuge dieses Objekt in deinem Konstruktor am entsprechenden Pin.
8. ***Ergänze deine Methoden*** `vorwaerts()` und `rueckwaerts()` im Rumpf um je eine Zeile, die den PWM-Pin auf 100 setzt. Verfahre bei der Methode `stopp()` entsprechend und setze den PWM-Pin auf 0.
9. Schreibe zwei ***weitere Methoden*** `vorwaerts( int geschwindigkeit )` und `rueckwaerts( int geschwindigkeit )`, die genauso funktionieren wie die bisherigen Methoden, nur dass der Übergabe-Wert an den PWM-Pin weiter geleitet wird.
10. Teste deine Klasse wieder interaktiv in BlueJ.
    Aufgrund der Bauart des Motor-Treibers dreht der Motor bei `vorwaerts(100)` nicht so schnell wie bei `vorwaerts()` . *(analog* `rueckwaerts()` *)*

Projekt: Fließband

Als Anwendung kannst du nun ein Fließband mit zwei Lichtschranken steuern:

Technische Voraussetzungen



Der hier verwendete Getriebe-Motor wird genauso angesteuert wie der in den vorangehenden Projekten. Da er aber eigentlich 6V benötigt, reicht seine Kraft unter Nutzung von PWM oft leider nicht aus, das Fließband zu bewegen. Deshalb lassen wir hier den Jumper-Stecker wieder auf dem Motor-Treiber und betreiben es ohne Geschwindigkeits-Regelung, als „Vollgas“ … was Bauart-Bedingt nicht all zu schnell ist :-)

Die Software

Über das Programm kannst du dir selbst Gedanken machen.

Einige Szenarien sind:

- Es wird immer links aufgelegt und rechts entnommen.
  Das Band soll sich in Bewegung setzen, wenn links ein Gegenstand erkannt wird und stoppen, wenn dieser Gegenstand rechts angekommen ist.
- Wenn links und rechts gleichzeitig etwas liegt, soll sich das Band nicht in Bewegung setzen. Erst wenn der rechte Gegenstand entfernt wird, soll das Band wieder anfangen, sich zu bewegen.
- Kann dein Fließband auch sinnvoll reagieren, wenn ein weiterer Gegenstand aufgelegt wird, solange es den ersten noch befördert. *(Was geschieht, wenn du den ersten Gegenstand rechts entfernst?)*

- Ein völlig anderes Szenario ist, dass das Band erkennt, ob links oder rechts aufgelegt wird ...