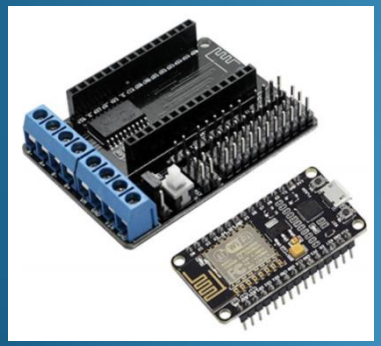
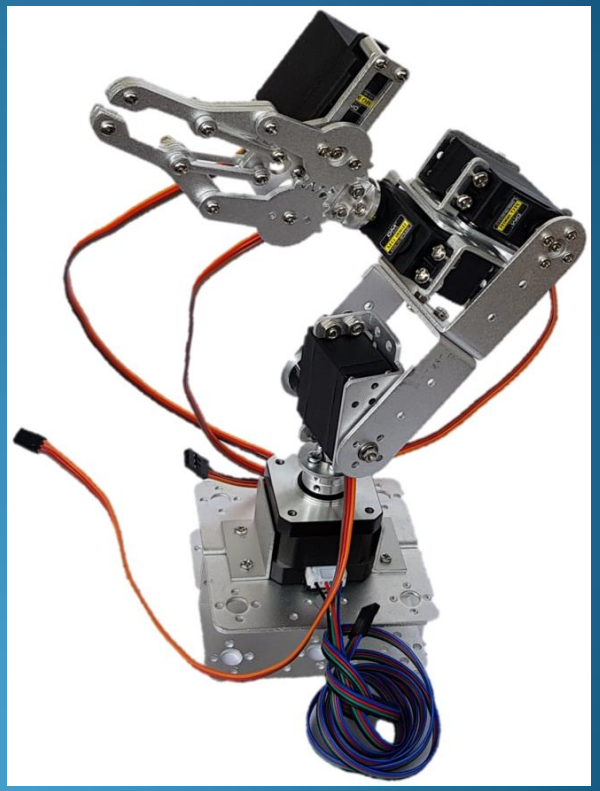
GTA Mechatronik Teil 3

Wir bauen und programmieren Montageroboter
und lernen dabei viel über Roboter, Schrittmotoren und serielle Datenübertragung





Inhaltsverzeichnis

Roboter Impressionen

- https://www.youtube.com/watch?v=J3gvpaNFvZU
- https://www.youtube.com/watch?v=OTU0z7Q08xQ
- https://www.youtube.com/watch?v=l_1uijyIeU8

- https://www.youtube.com/watch?v=P7fi4hP_y80

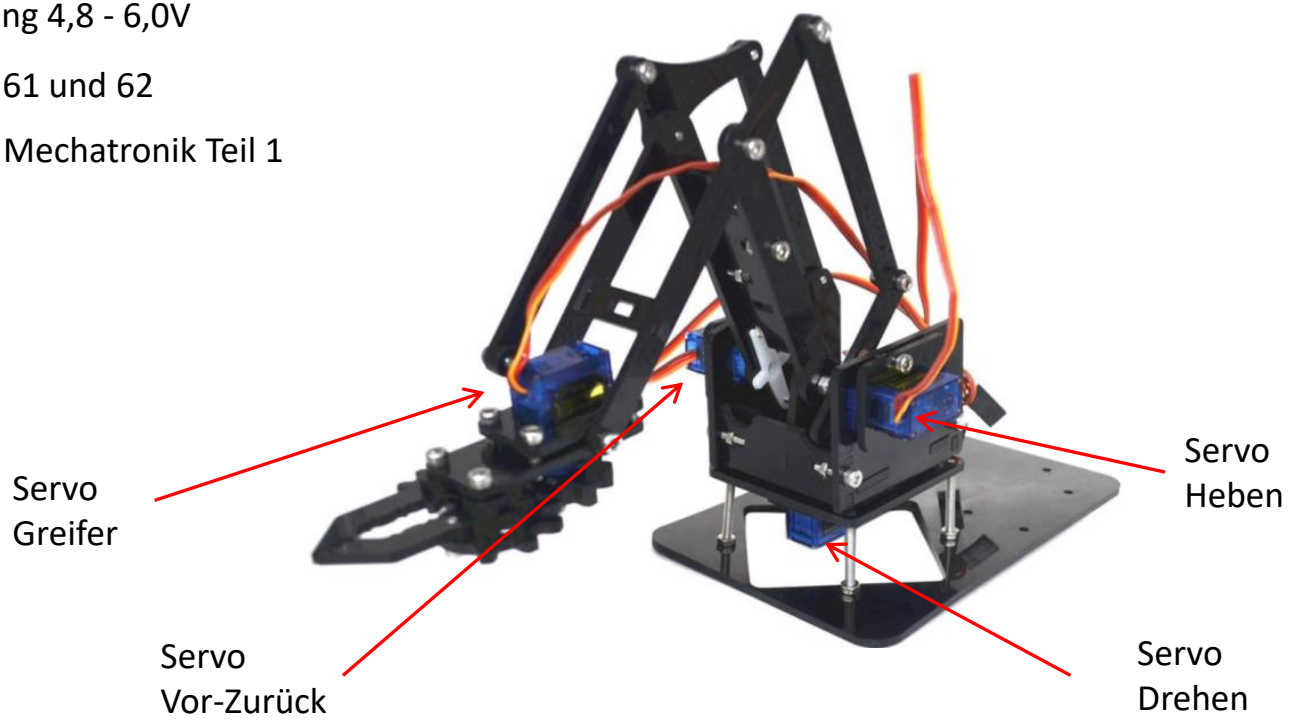
Roboter 4 Achsen mit Servos SG90

Der Roboter hat 4 Bewegungs-Achsen, die alle von Servos des Typs SG90 angetrieben werden.
Diese Servos haben nur eine geringe Leistung (kleiner Motor, Getriebe aus Kunststoff).
Wenn sie immer nur einzeln angesteuert werden, kann die Stromversorgung direkt vom Arduino Nano erfolgen (Anschluß VCC) .

Zuläss.Spannung 4,8 - 6,0V

Siehe Sketche 61 und 62

Servos: Siehe Mechatronik Teil 1



Montageanleitung siehe: https://smallhammer.cc/docs/4dof

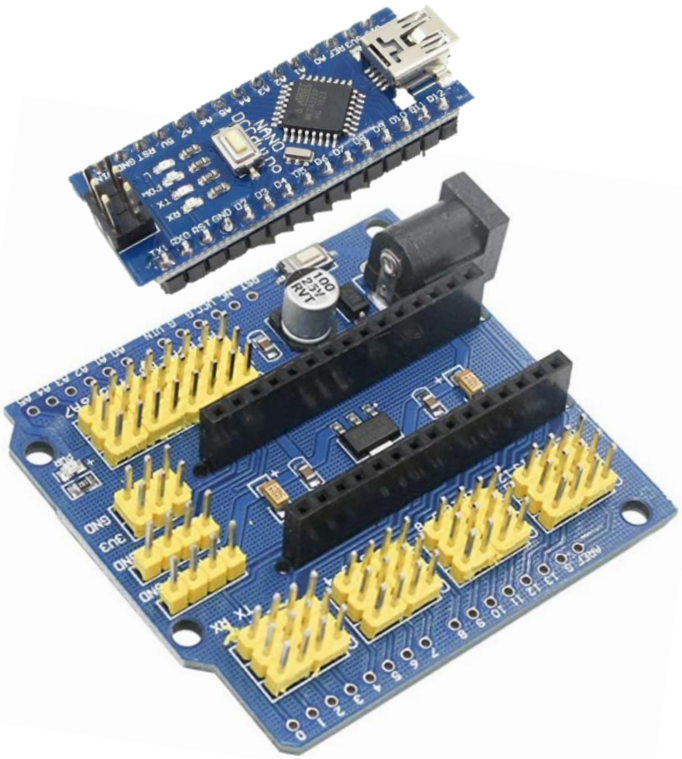
I/O Erweiterungs Sensor Shield für Arduino Nano

IZOKEE 3 Stück I/O Erweiterungs Sensor Shield Modul für Arduino Nano

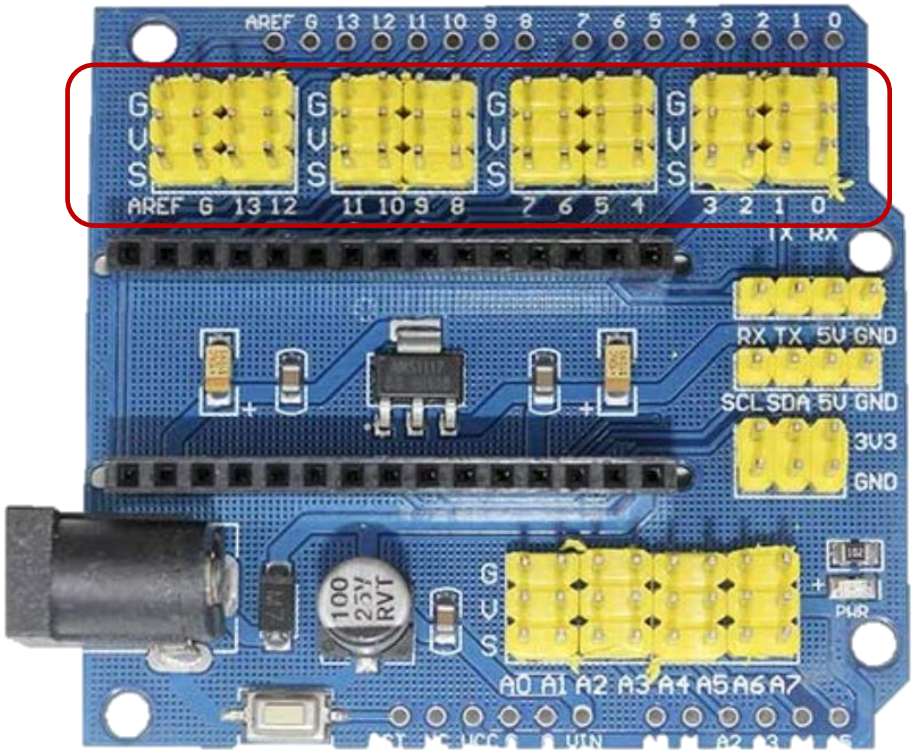
von IZOKEE

★★★★☆ 26 Sternebewertungen

Preis: 10,99 € **GRATIS-Lieferung** für qualifizierte Erstbestellung nach Deutschland und Österreich. Wählen



Die Servos SG90 können hier angeschlossen werden:



Roboter 5 Achsen mit Servos MG996R und Schrittmotor

Der Roboter hat 5 Bewegungs-Achsen.
4 Achsen werden von Servos MG996R angetrieben (siehe Mechatronik Teil 1).

Diese Servos haben wesentlich höhere Leistung (größerer Motor) und Belastbarkeit (Getriebe aus Metall) als die SG90.

Die 5.Achse „Drehung“ treibt ein Schrittmotor Typ 17HS13-0404S an.

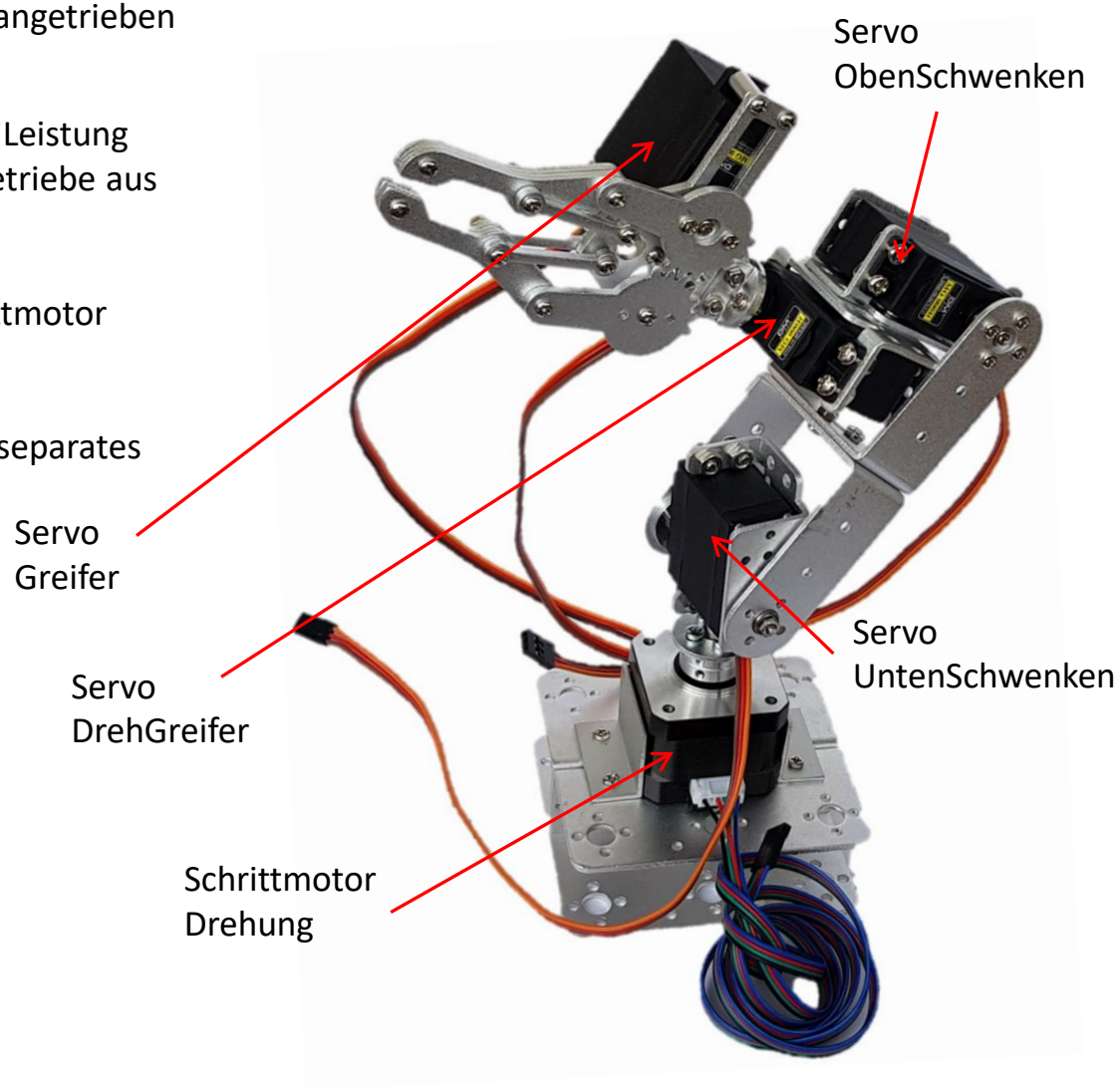
Die Stromversorgung erfolgt durch ein separates Netzteil.

Zuläss. Spannung für Servos:
4,8 ... 7,2 V

Strom pro Servo :
bei Bewegung: ca. 0,5 A
bei Blockierung @ 6V: 2,5 A

Zuläss.Spannung für Schrittmotor:
Bis 12 V auch bei Dauerbetrieb

Siehe Sketche 66 bis 72.

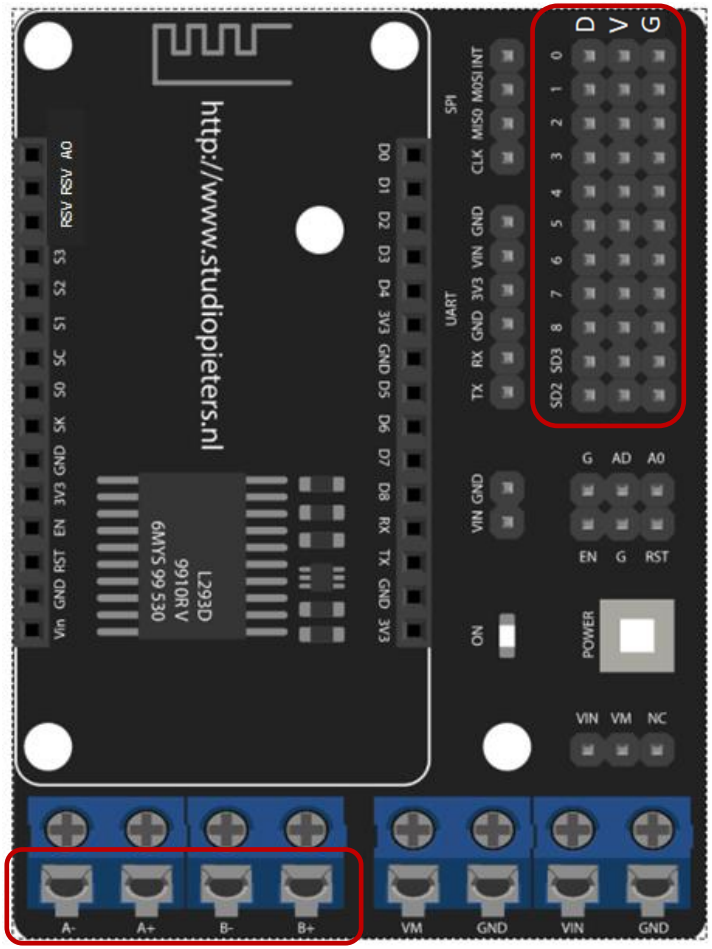


NodeMCU V2 – L293D Motor Shield

DollaTek NodeMcu ESP8266 ESP-12E Entwicklungsboard und L293D WiFi Motor Drive Erweiterungskarte
Motor power supply (VM): 4,5V-36V
Control Power (VIN): 4,5V-9V (max 10V)
Treiber: Dual H-bridge driver L293D
Max 1200 mA
ESP12E Dev Kit Control Port: D1, D3 (winding A); D2, D4 (winding B)
Man kann VM und VIN verbinden (dann zulässig nur 5V ... 9V)

https://www.youtube.com/watch?v=sEjhM3cMlhc

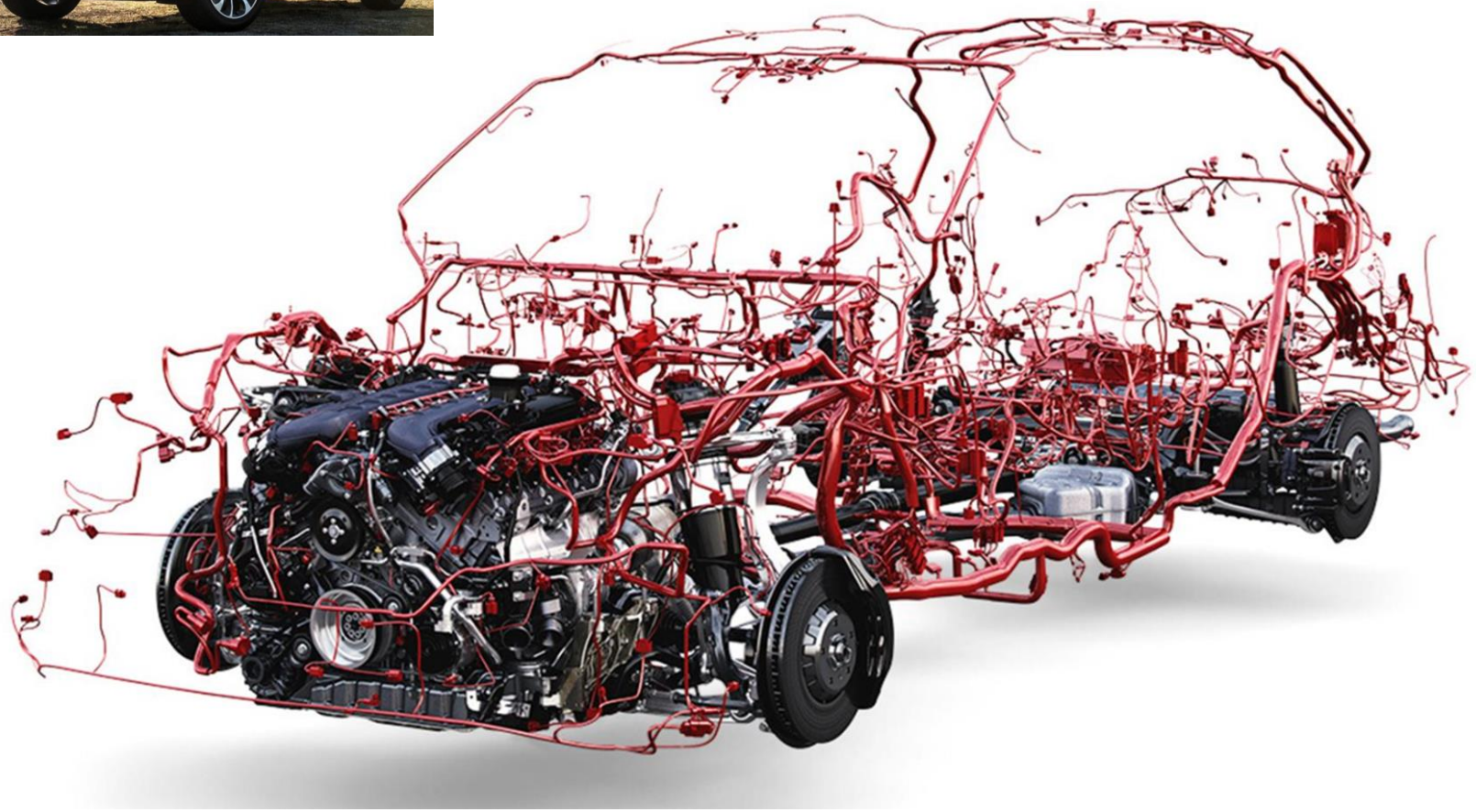
Es gibt auch Shields die sind hier nicht verwendbar.



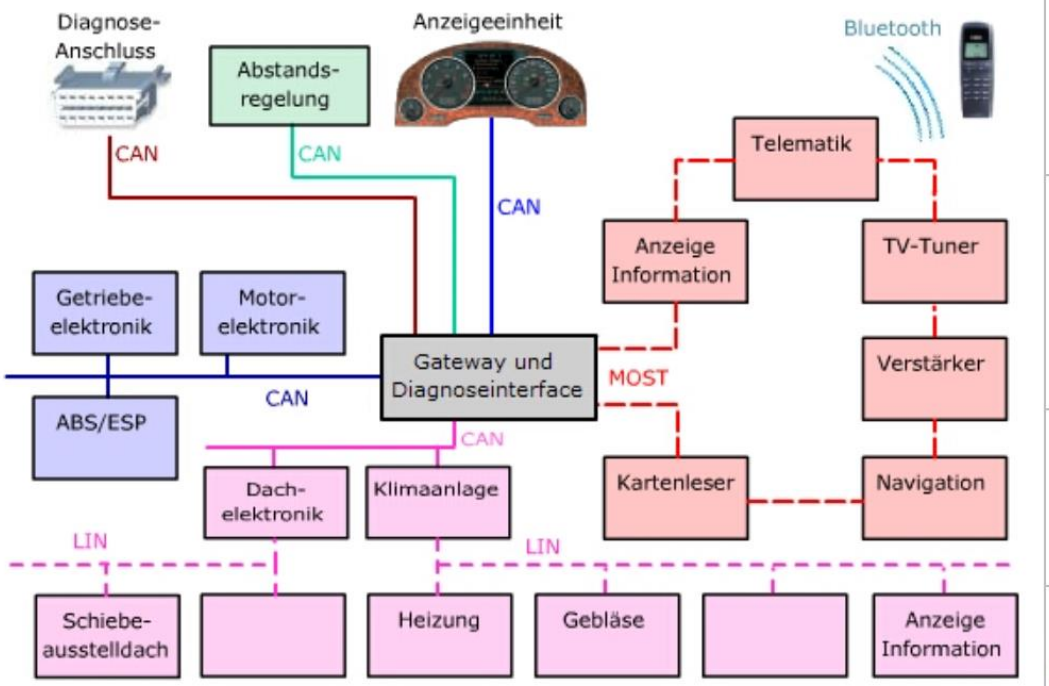
Geeignet für NodeMCU V2.
Das Modul NodeMCU V3 ist breiter und benötigt ein NodeMCU V3 – L293D Motor Shield.

Kabelbaum im Automobil

Bentley Bentayga



Serielle Bussysteme im Automobil

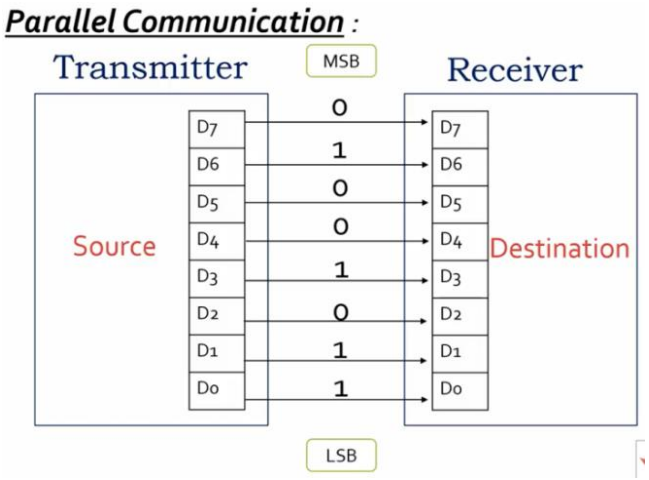


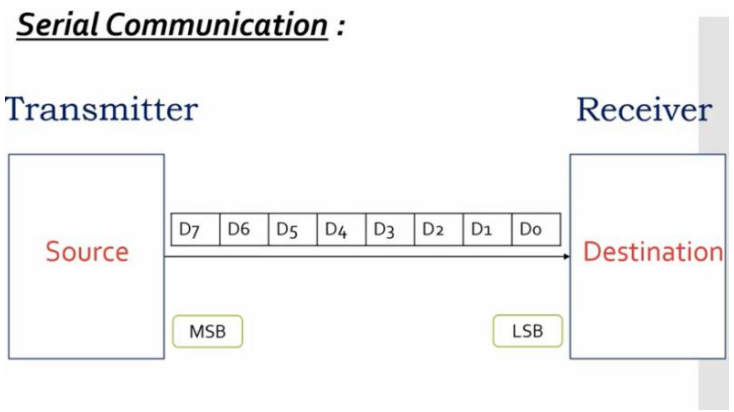
CAN-Bus	Der **CAN-Bus** existiert in verschiedenen Varianten. In Abhängigkeit von der Geschwindigkeit, mit der er Daten übertragen kann, wird er in den Bereichen Antrieb, Komfort und zur Diagnose eingesetzt. Low Speed bis High Speed, 100 bis 500 kBit/s incl. Diagnose CAN
LIN	Der **LIN-Bus** (Local Interconnect Network) ist ein preiswertes dem CAN-Bus untergeordnetes Sub-Bussystem; er wird für die Ansteuerung bestimmter lokaler Komfort-Komponenten oder für Sensoren verwendet. - max. 20 kBit/s
MOST	(Media Oriented Systems Transport) optisches Hochgeschwindigkeits-Multimedia-Bussystem, meist ringförmig, Sehr hohe Übertragungsrate (23 MBit/s)
Bluetooth	**Bluetooth-Funkstrecke** erfordert auf kurzen Distanzen keine Verkabelung und dient zur sicheren Übertragung von Sprach-Signalen.

Wichtige Kriterien:
- Geschwindigkeit Datenübertragung
- Sicherheit gegen Störungen
- Erkennen und Korrigieren von Übertragungsfehlern

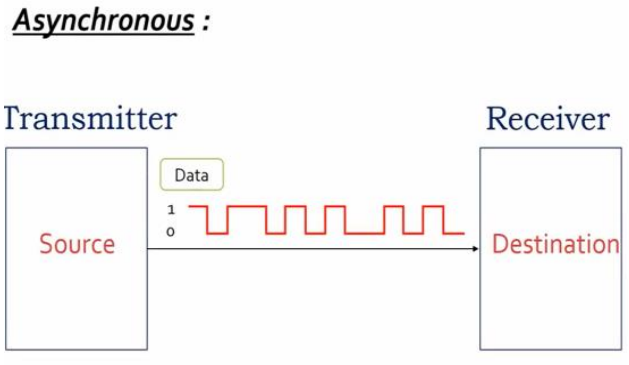
Parallele und Serielle Datenübertragung

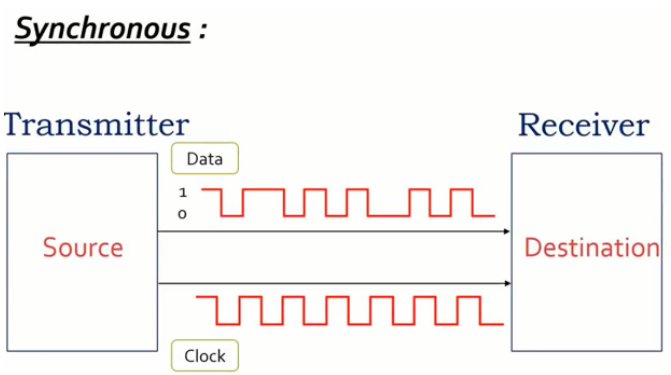
Gundbegriffe der Datenübertragung einfach erklärt: https://www.youtube.com/watch?v=_3oKSK3sfd8

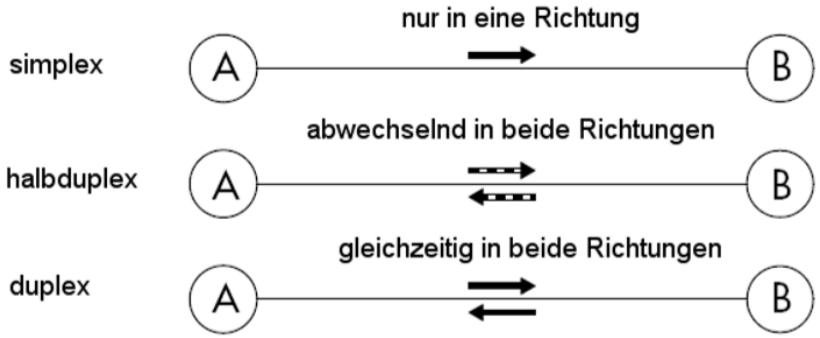




Serielle Schnittstelle (Serielle Datenübertragung, Serieller Bus)





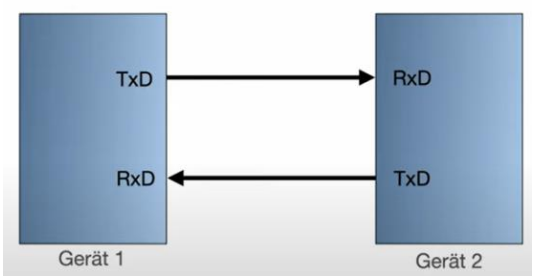


http://www.netzmafia.de/skripten/hardware/Control/schnittstellen.pdf

Serielle Schnittstellen an Arduino und ESP8266 NodeMCU

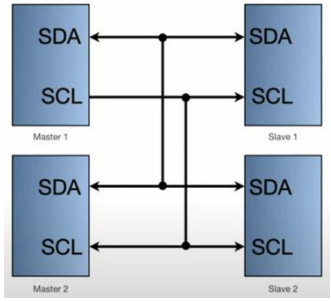
UART Universal Asynchronous Receiver Transmitter

https://www.youtube.com/watch?v=t-K1jHKactw



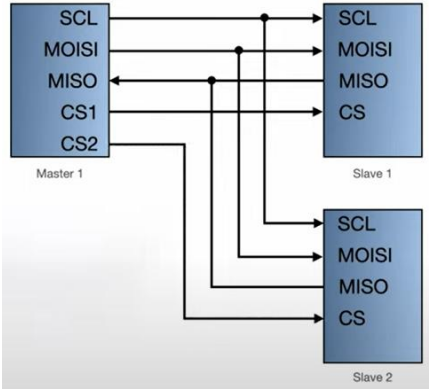
I2C Inter-integrated Circuit

https://www.youtube.com/watch?v=3wlITceWQBw

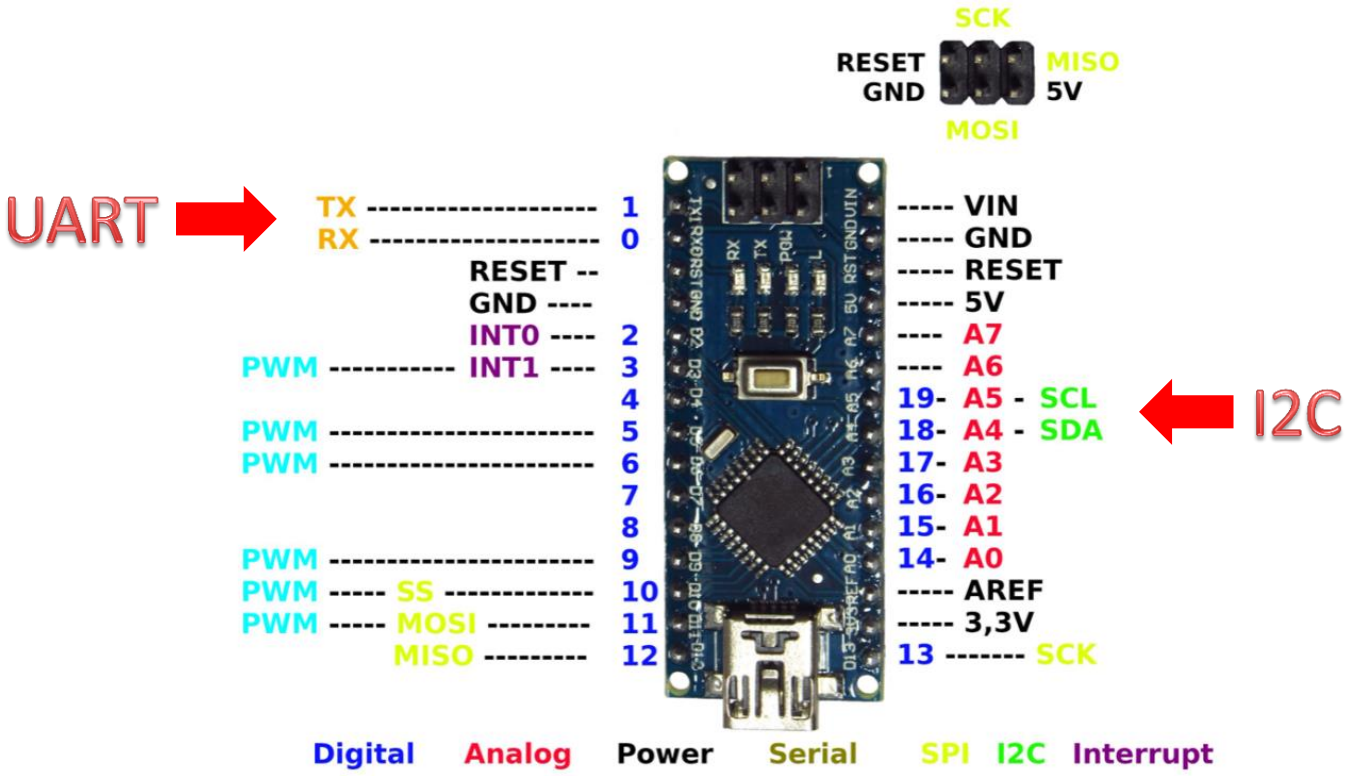


SPI Serial Peripheral Interface

https://www.youtube.com/watch?v=k3v_i1YmCo4

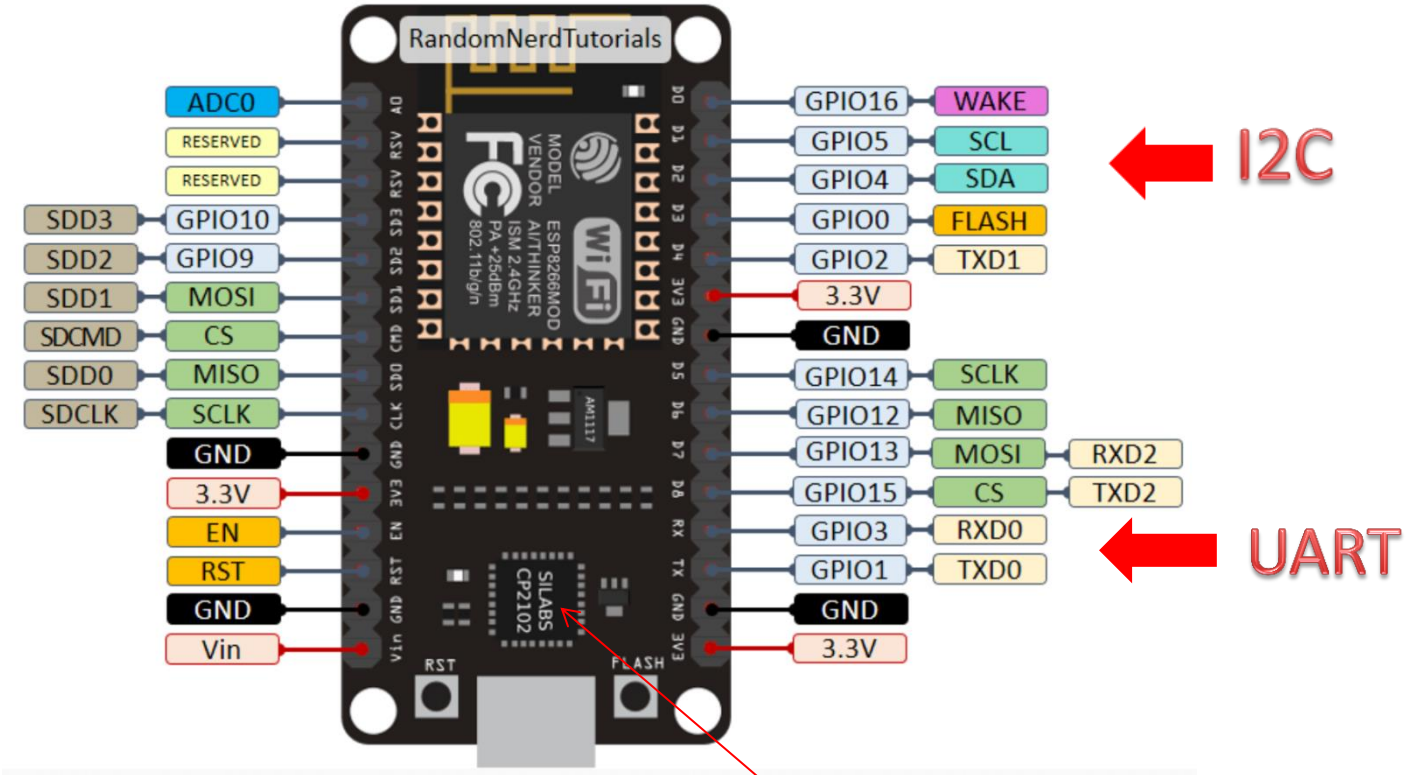


UART und I2C am Arduino Nano



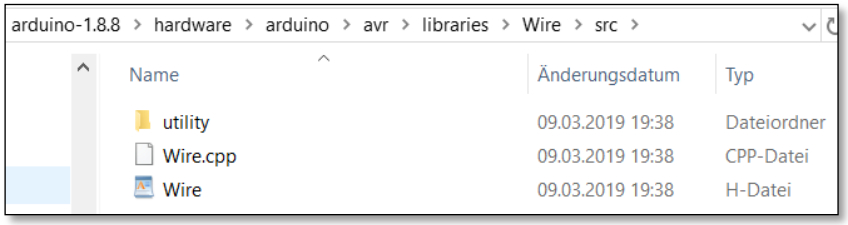
Chip CH341G (auf Unterseite): Verbindung USB mit UART

UART und I2C an ESP8266 NodeMCU



Bibliotheksprogramm (Library) Wire.h

- Das Bibliotheksprogramm findet man unter:



- Es ermöglicht die Kommunikation über den I2C-Bus.
- Die Wire-Bibliothek stellt folgende Befehle zur Verfügung (siehe auch http://html.szaktilla.de/arduino/6.html) :

- Wire.begin(*Adresse*)
- Initialisiert die Bibliothek und meldet den Arduino mit der angegebenen Adresse am I²C-Bus an.
- Soll der Arduino als Busmaster angemeldet werden, entfällt die Adresse.

- Wire.requestFrom(*Adresse, Anzahl[, stop]*)
- Fordert vom angegebenen Gerät die angegebene Anzahl von Bytes an.
- Der dritte (optionale) Parameter gibt an, ob der I²C-Bus nach dem Senden der Anforderung wieder freigegeben wird: *true* (default) bedeutet, dass der Bus wieder freigegeben wird, *false* sorgt dafür, dass der Bus „besetzt“ bleibt.
- Sinnvoll ist zweiteres dann, wenn man mehrfach Daten abfragen und verhindern möchte, dass ein anderer Busteilnehmer „dazwischenfunkt“.

- Wire.beginTransmission(*Adresse*)
- Bereitet eine Datenübertragung an das Gerät mit der angegebenen Adresse vor.
- Anschließend werden die zu übertragenden Daten mit einem oder mehreren *Wire.write()*-Befehl(en) in einen Puffer geschrieben.
- Mit dem Aufruf von *Wire.endTransmission()* werden die Daten übertragen.

- Wire.endTransmission()
- Überträgt die gepufferten Daten an das mit *Wire.beginTransmission()* angegebene Gerät.

Bibliotheksprogramm (Library) Wire.h

- Wire.write()
- Mit diesem Befehl befüllt man den Sendepuffer mit den Dingen, die beim Aufruf von *Wire.endTransmittion()* gesendet werden sollen. Beispiele:
  Wire.write(5); schreibt die Zahl 5 in den Puffer. Wire.write("Hallo"); schreibt „Hallo“ in den Puffer.

- Wire.available()
- Gibt die Anzahl der Bytes zurück, die sich im Empfangspuffer befinden. Man kann diesen Befehl zum Beispiel im Kopf einer while-Schleife benutzen.
- Die Schleife wird dann durchlaufen, so lange sich etwas im Empfangspuffer befindet. Beispiel:
  while(Wire.available()) { Serial.print(Wire.read()); }

- Wire.read()
- Liest ein Zeichen aus dem Empfangspuffer. Siehe das Beispiel bei *Wire.available()*.

- Wire.onReceive(*Funktion*)
- Bestimmt die Funktion, die aufgerufen wird, wenn Daten über den I²C-Bus empfangen wurden.
- Die Funktionsdefinition kann einen Parameter enthalten, mit dem die Zahl der eingetroffenen Bytes übergeben werden kann.
  Beispiel: Wire.onReceive(I2C_receive);
  Die Definition dieser Funktion beginnt dann beispielsweise mit void I2C_receive(int num_bytes) {…
  Achtung: Der Typ des Parameters der Funktion muss *int* sein, sonst gibt es beim Compilieren eine Fehlermeldung.

- Wire.onRequest(*Funktion*)
- Bestimmt Funktion, die aufgerufen wird, wenn Daten angefordert werden (siehe *Wire.requestFrom()*).
- Diese Funktion erhält keine Daten und gibt auch nichts zurück.
  Beispiel: Wire.onRequest(requestHandler);
  Die Definition dieser Funktion sieht dann beispielsweise so aus: void requesthandler() {…}

Schrittmotoren

Die bisher verwendeten Elektromotoren (DC Motoren, Servos, siehe auch Mechatronik Teil 1) erzeugen eine kontinuierliche Drehbewegung.

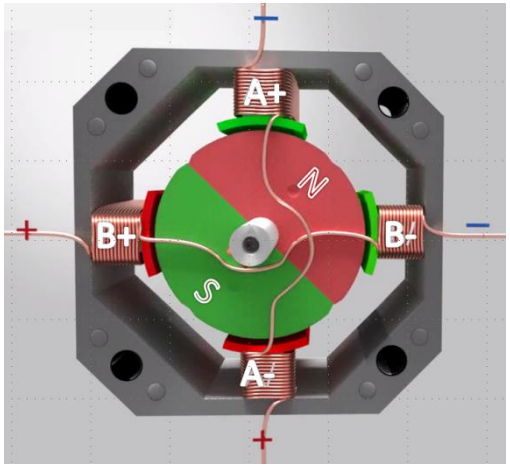
Die Drehzahl ist abhängig von der angelegten Spannung und der mechanischen Belastung, die Drehrichtung ist durch Umpolung der Spannung änderbar.

Schrittmotoren dagegen erzeugen eine diskontinuierliche „schrittweise“ Drehung. Die Schrittweite beträgt je nach Bauform 0,9° ... 15°.

Durch Aufruf einer definierten Anzahl an Schritten läßt sich eine Positionierbewegung errreichen.

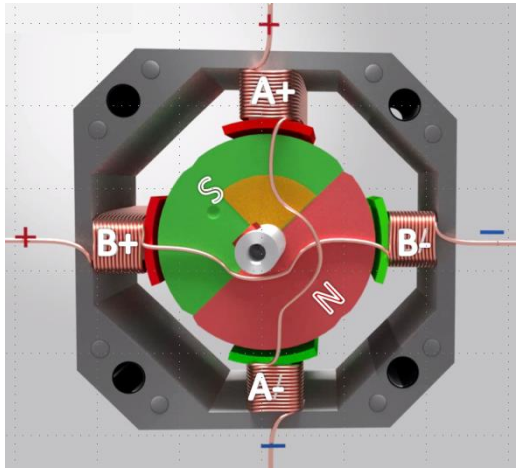
Man kann z.B. die Drehung eines Roboters sehr wiederholgenau steuern, auch aus verschiedenen Richtungen.

Die Steuerung erfolgt durch Umschalten der Stromrichtung in zwei Spulen (Wicklungen) des Motors nach einem bestimmten Schema.



Beispiel:
Ausführen eines Schrittes durch Umschalten des Stroms in einer Spule

(Prinzipdarstellung, tatsächliche Schrittwinkel sind < 90°)



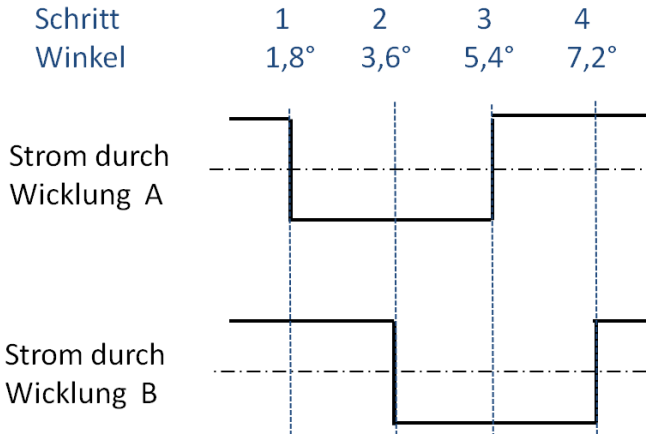
Siehe auch:
www.youtube.com/watch?v=Vtbd80FksuM

www.youtube.com/watch?v=gt09CDu2K_c

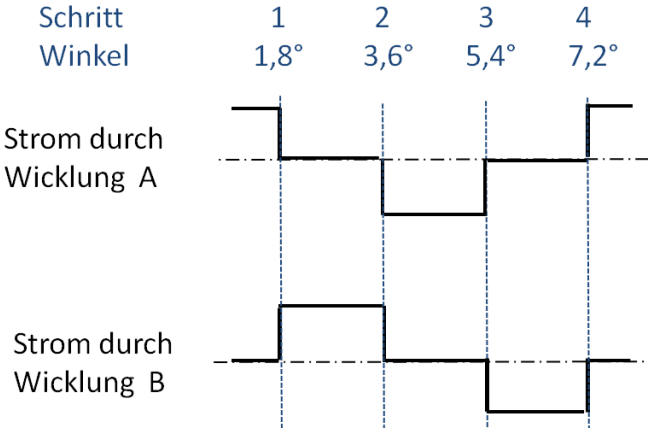
Schrittmotoren Ansteuerung

Stromverlauf durch die Wicklungen für 4 Schritte.

A) Beide Spulen sind immer eingeschaltet, in jeder Spule fließt Strom.



B) Nur jeweils eine Spule ist eingeschaltet. Der gesamte Strom, aber auch das Drehmoment werden geringer.



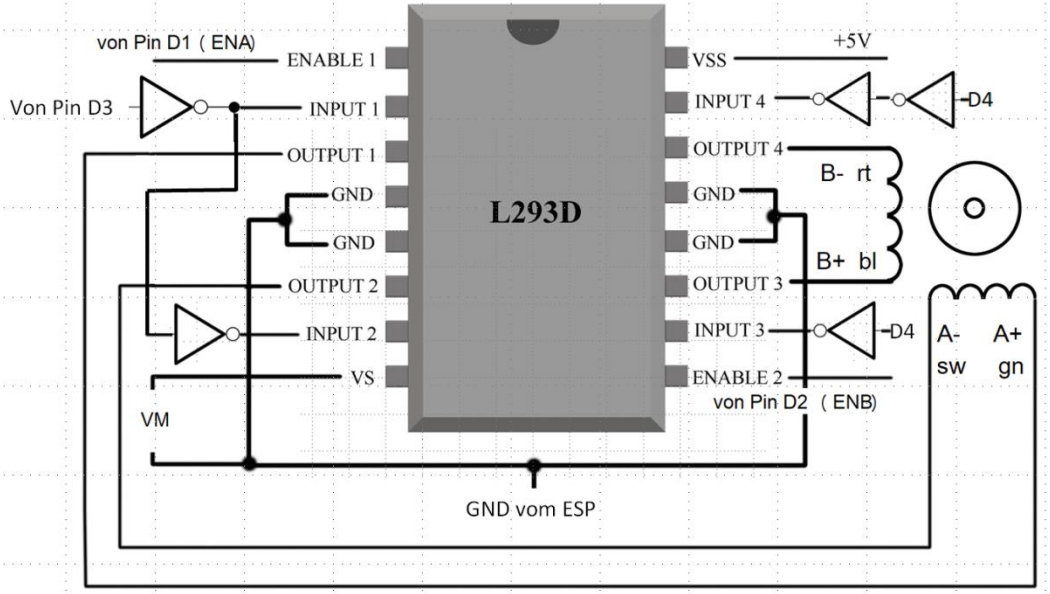
Schrittmotoren Ansteuerung

Die Steuerung wird in einem Sketch auf dem ESP8266 NodeMCU programmiert.
Ausgangssignale digitalWrite() gehen auf einen Motortreiber-Schaltkreis L293D, dessen Leistungsendstufen mit dem Schrittmotor verbunden sind.

Der L293D befindet sich auf einem „NodeMCU V2 – L293D Motor Shield“, das ESP8266 NodeMCU wird aufgesteckt.

Hier können auch noch Servos angeschlossen werden.

Der Schrittmotor wird an die Terminal-Klemmen A- A+ (Spule A) und B- B+ (Spule B) angeschlossen.
Siehe Sketch 68 bis 72.



Schrittmotor 17HS13-0404S

Für die Drehung des Roboters wird dieser Schrittmotor verwendet:

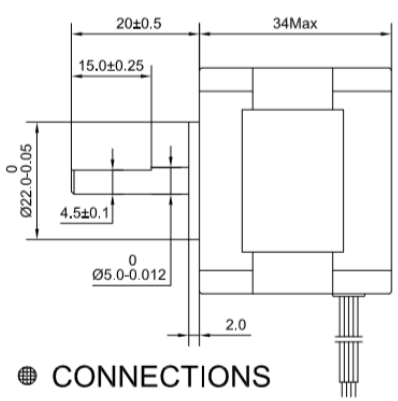
COMMON RATINGS

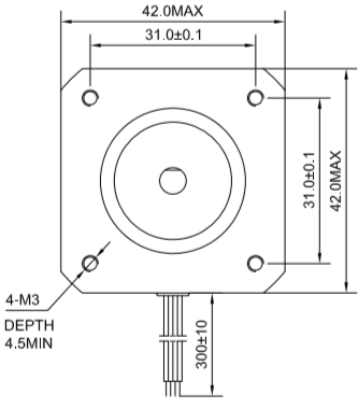
Step angle :	1.8°	Dielectric strength :	500VAC
Positional accuracy :	±5%	Insulation resistance :	100Mohm(500VDC)
Number of Phase :	2	Ambient Temperature :	-10℃~50℃
Temperature rise :	80℃MAX	Insulation class :	B
Rotor Inertia :	38gcm²	Weight :	0.28Kg

SPECIFICATIONS

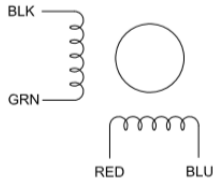
Holding Torque (2 phases on) Kg.cm	Rated Current/Phase (Amps DC)	Phase Resistance (ohms) ±10%	Voltage Current/Phase (V DC)	Phase Inductance (mH)±20%(1KHz) Typical
2.6	0.4	30	12.0	37

DIMENSIONS unit=mm





CONNECTIONS



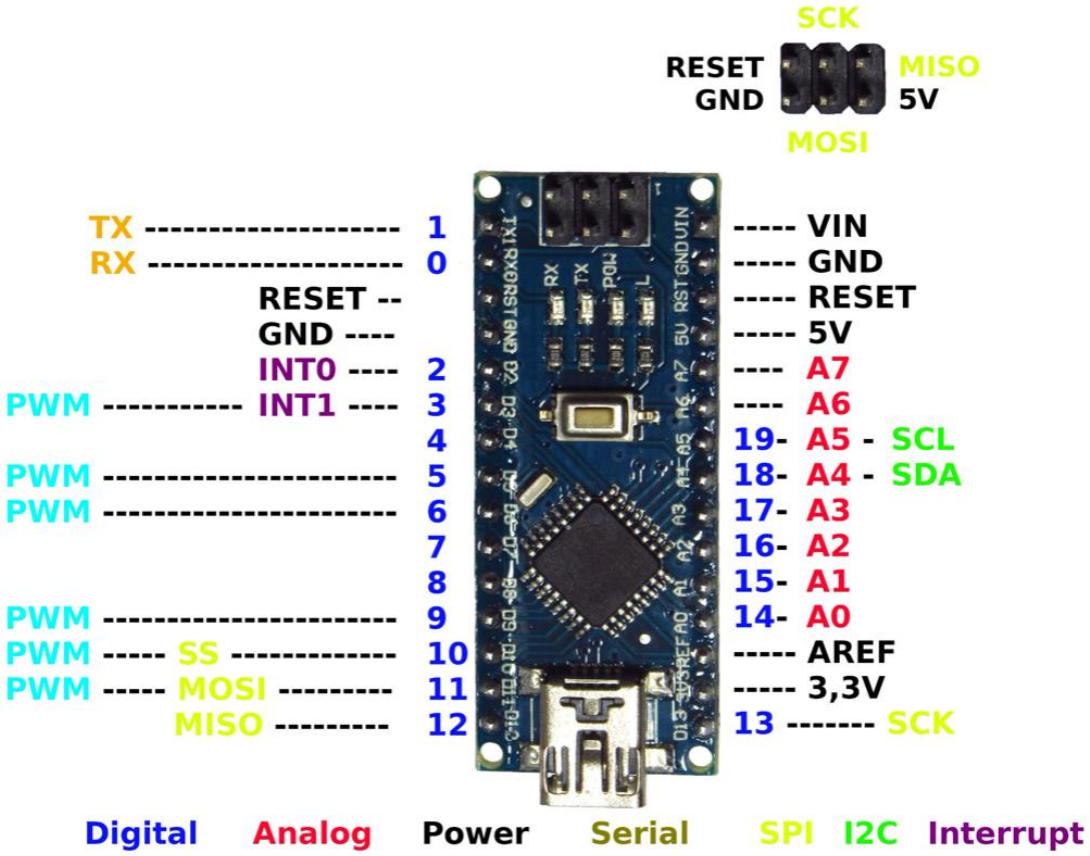
Sketche

Inhaltsverzeichnis

Zuordnung Pins am Arduino Nano

ESP	Funktion	Eing/Ausg		
A4 = D18	SDA	Eing+Ausg		I2C-Bus „Serial Data“
A5 = D19	SCL	Eing+Ausg		I2C-Bus „Serial Clock“
D2	Servo DR	Ausgang		Drehung
D4	Servo GR	Ausgang		Greifer
D6	Servo HE	Ausgang		Heben
D8	Servo VZ	Ausgang		VorZurueck

Arduino Nano

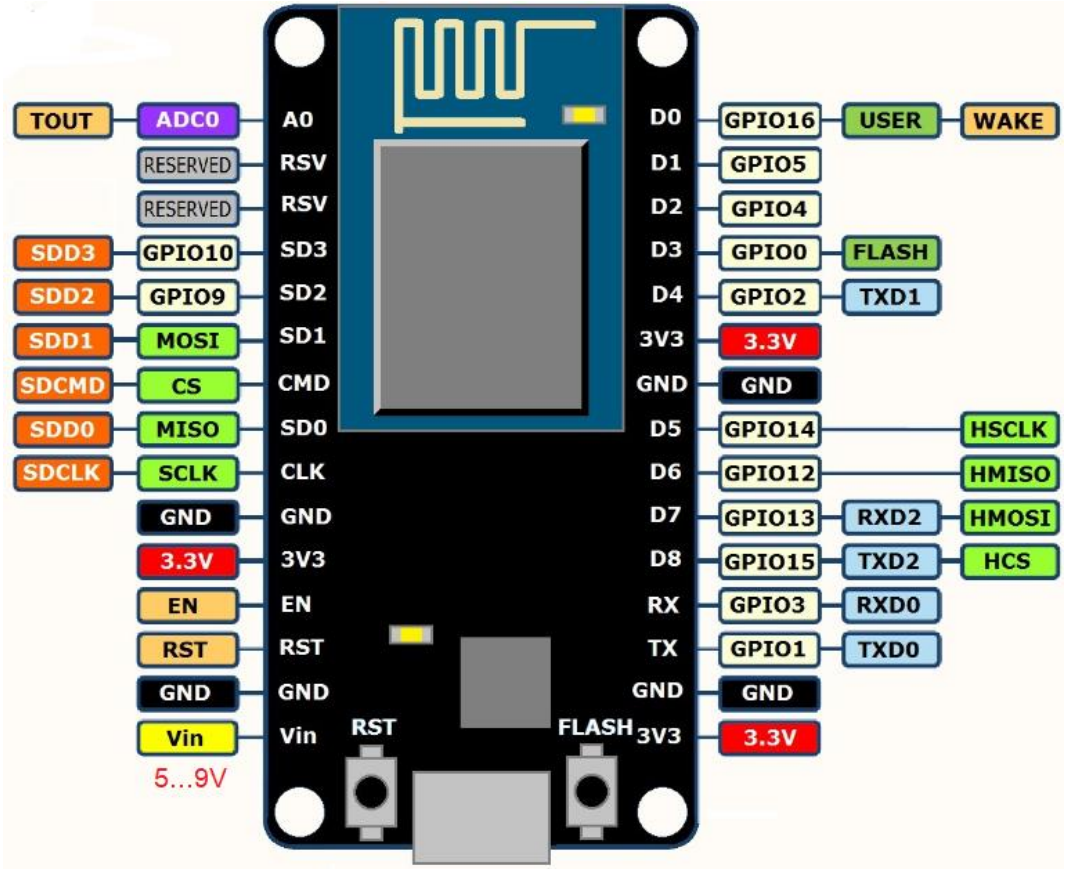


Zuordnung Pins am ESP8266 NodeMCU

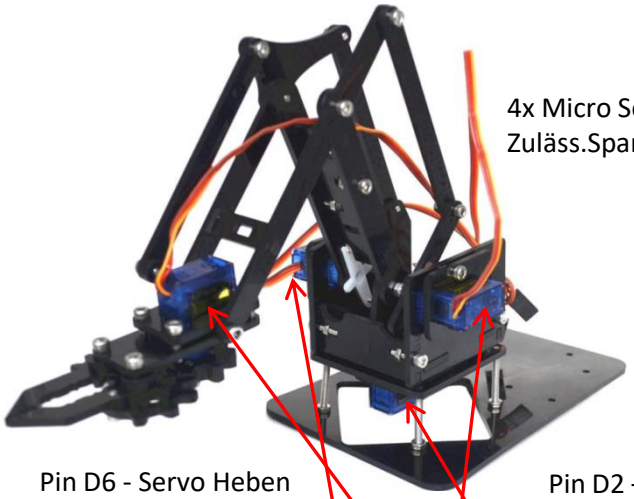
ESP	Funktion	Eing/Ausg		
D2~	SDA ENB	Eing+Ausg Ausgang		I2C-Bus „Serial Data“ Enable Wicklung B (Schrittmotor)
D1~	SCL ENA	Eing+Ausg Ausgang		I2C-Bus „Serial Clock“ Enable Wicklung A (Schrittmotor)
D3~	DA	Ausgang		Drehrichtung Wickl. A (Schrittmotor)
D4~	DB	Ausgang		Drehrichtung Wickl. B (Schrittmotor)
D5~	Servo GR	Ausgang		Greifer
D6~	Servo DG	Ausgang		DrehGreifer
D7~	Servo OS	Ausgang		Gelenk ObenSchwenken
D8~	Servo US	Ausgang		Gelenk UntenSchwenken

~ Ausgang PWM möglich

ESP8266 NodeMCU



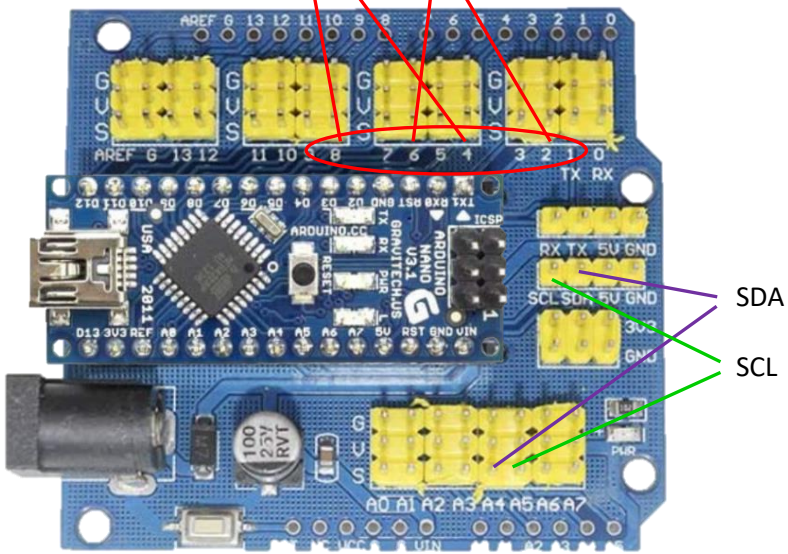
Sketch 61 Servos Roboter schwenken (Arduino Nano)



4x Micro Servos SG90
Zuläss.Spannung 4,8 - 6,0V

Pin D6 - Servo Heben
Pin D8 – Vor-Zurück

Pin D2 – Drehung
Pin D4 - Greifer



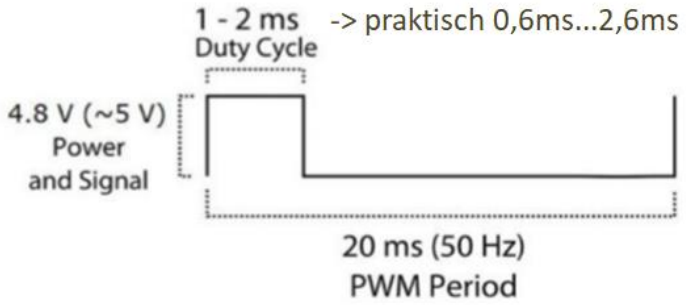
Es wird Arduino Nano auf I/O Erweiterungs Shield verwendet.
Die Spannungsversorgung der Servos geht direkt von 5V oder von 3,3V (drehen langsamer) des Nano.
Nur möglich bei Micro Servos SG90 und wenn immer nur ein Servo angesteuert wird
(die anderen Servos stromlos).
Aufgrund der Reibungen bleiben auch die stromlosen Servos in ihrer Position.

Sketch 62 Servos steuern mit dem Serial Monitor (Nano)

Es wird ebenfalls Arduino Nano auf I/O Erweiterungs Shield Modul verwendet.

Es wird immer nur 1 Servo des Roboters bewegt, die anderen bleiben stromlos (sonst ist die Strombelastung zu hoch).

Die Zeitdauer des High-Signals für den Schwenkbereich -90°...+90° ist theoretisch 1...2ms, aber bei SG90 praktisch etwa 0,6ms...2,6ms (600 ... 2600 Mikrosekunden).
Für die Mittelstellung muß der High-Puls theoretisch 1,5ms (=1500 Mikrosekunden) sein, kann aber abweichen (experimentell ermitteln).

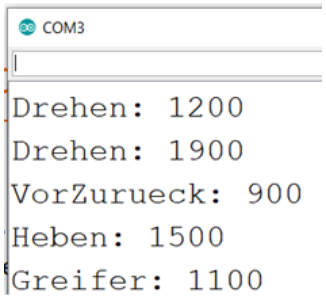


Die Bewegungen werden über Eingaben im seriellen Monitor aktiviert , das Programm verwendet den Befehl `Serial.read` bzw. `Serial.parseInt`.

Es erfolgt auch eine Anzeige im seriellen Monitor:

Zum Beispiel Eingabe "d1500" bewegt den Servo „Drehung“ in die Mittelstellung, "g600" den Servo „Greifer“ in eine Endstellung.

Nach Bewegungsende werden die Servos stromlos, man kann sie von Hand verdrehen. Aufgrund der Reibung bleiben sie aber in Position, driften nicht ab.



Bei 4,8V braucht der Servo SG90 etwa 300ms für Drehung um 60 Grad.

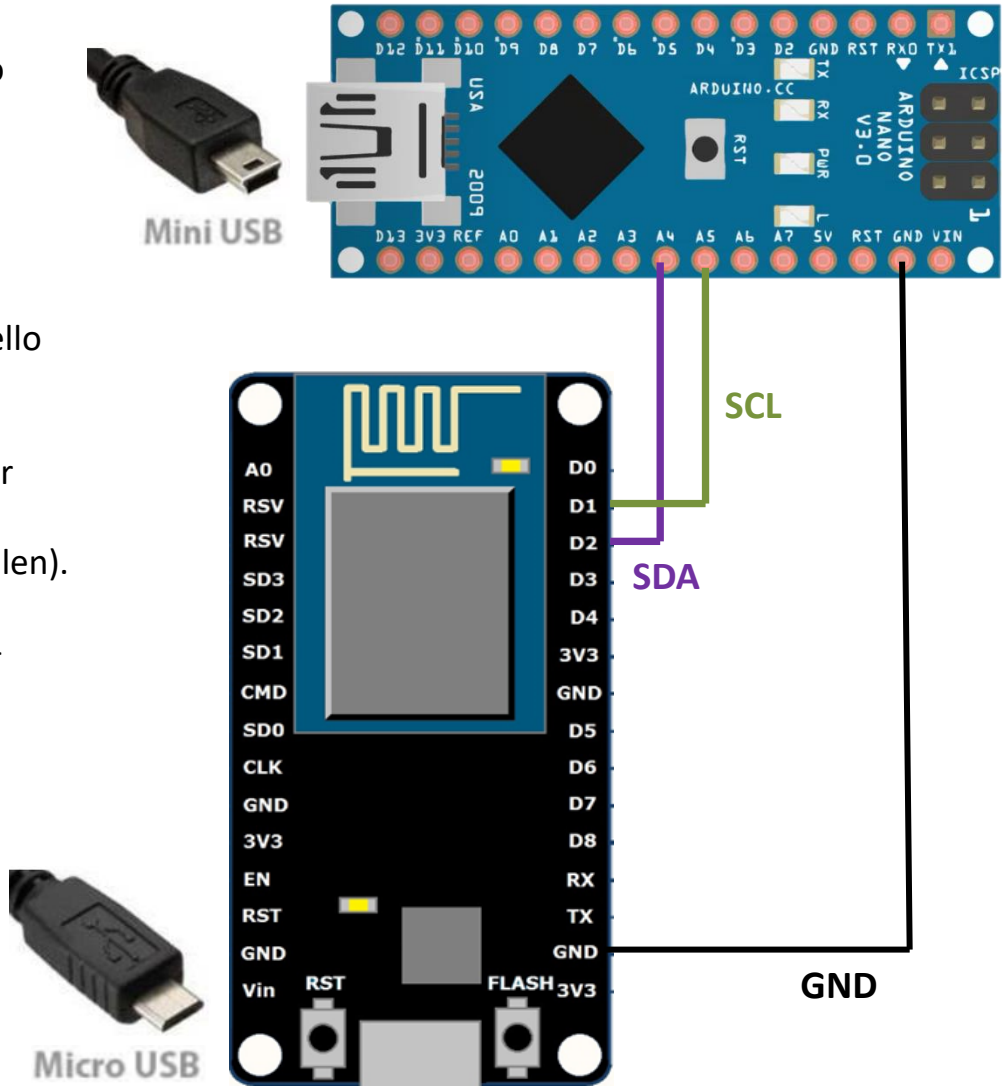
Sketch 64+65 I2C Bus zwischen ESP und Nano - zählen

Senden (write) von Daten über die I2C serielle Schnittstelle von ESP8266 NodeMCU zu Arduino Nano und zurück.
ESP fungiert als Master, Nano als Slave.

ESP sendet jede Sekunde die Zeichenfolge "Wert ist jetzt: " sowie eine Zahl 0...255.
Im Gegenzug sendet Nano die Zeichenfolge "Hello ESP from Nano! ".

Diese Daten werden jeweils im seriellen Monitor angezeigt
(für die Anzeige das jeweilige COM-Port auswählen).

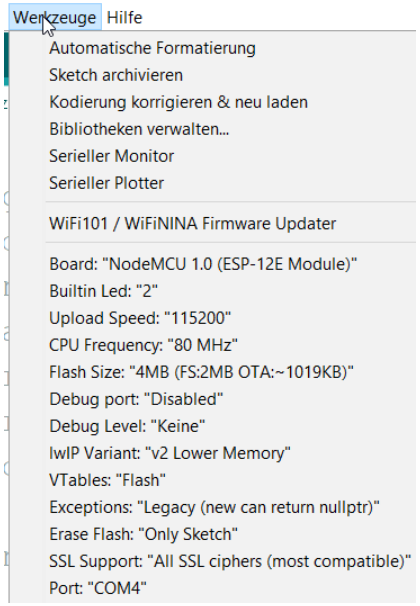
Die Spannungsversorgung erfolgt über die USB-Kabel.

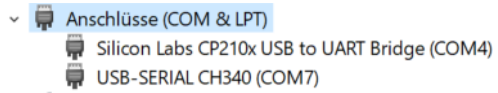


Siehe auch:
https://www.electronicwings.com/nodemcu/nodemcu-i2c-with-arduino-ide

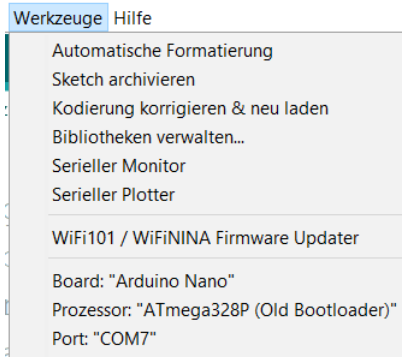
Sketch 64+65 I2C Bus zwischen ESP und Nano - zählen

Auf ESP8266 NodeMCU laden:
sketch_64_esp_I2C_zaehlen



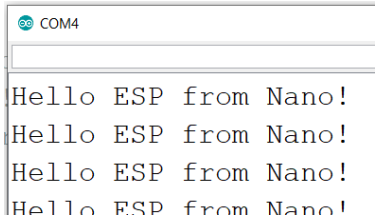


Auf Arduino Nano laden:
sketch_65_nano_I2C_zaehlen

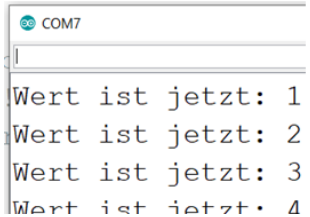


Ausgabe am Serial Monitor

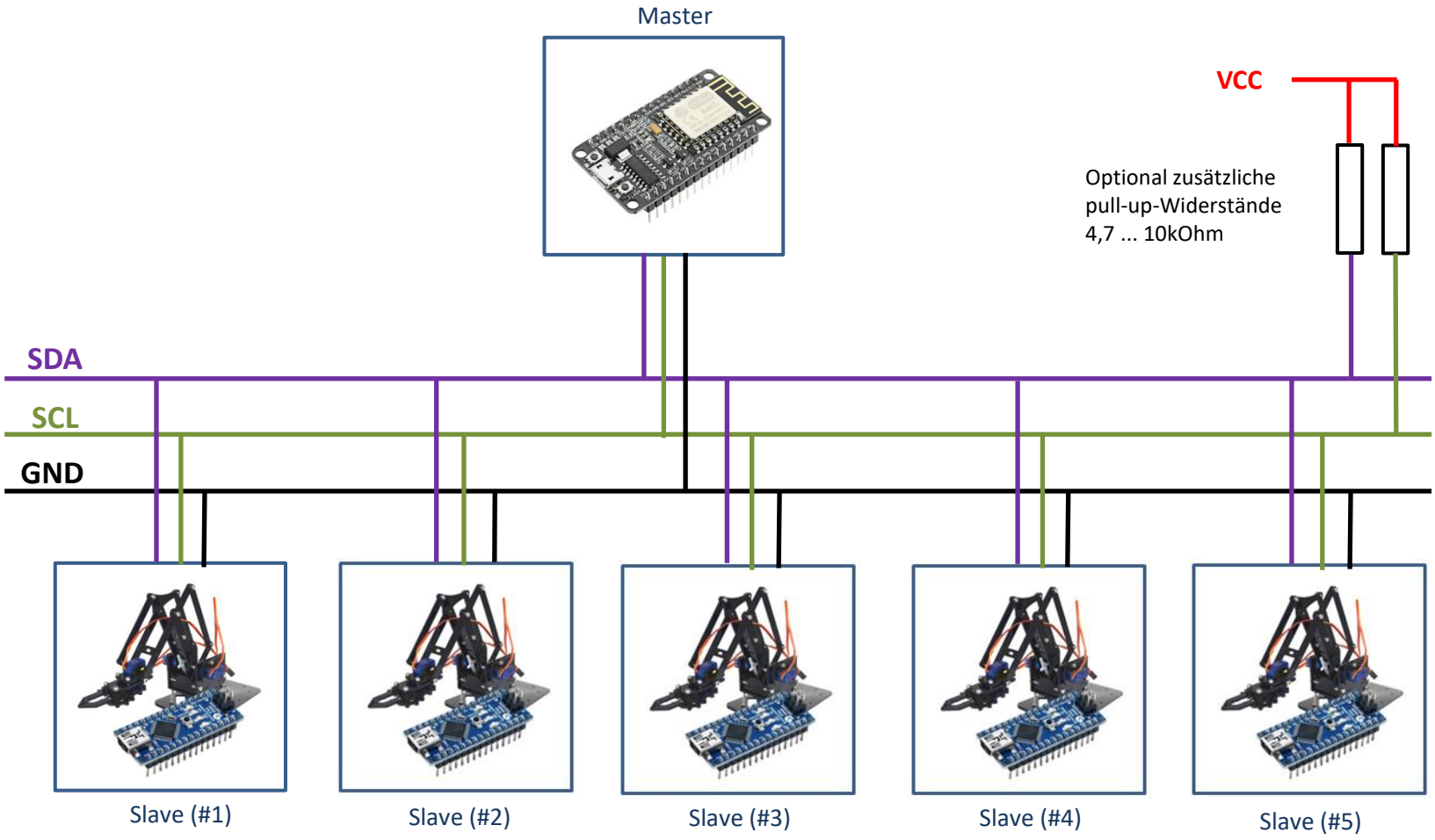
Unter „Werkzeuge“ Einstellung COM4:



Unter „Werkzeuge“ Einstellung COM7:



Sketch 66+67 Servos Roboter ansteuern über I2C Bus



Sketch 66+67 Servos ansteuern über I2C Bus

Über die I2C serielle Schnittstelle ist ein ESP8266 NodeMCU (Master) mit mehreren Arduino Nano (Slaves) verbunden.

Jeder Nano steuert einen Roboter, mit jeweils 4 Servos.

Im Sketch des Masters (ESP) wird über den seriellen Monitor mit `Serial.read` eingegeben, welcher Nano, d.h. welcher Roboter, jeweils über den I2C Bus angesprochen werden soll.

Siehe auch:  Projektlabor Robotik MINTgrün **Übermittlung von Kommandos und Daten vom PC zum Arduino**

http://www.mintgruen.tu-berlin.de/robotikWiki/doku.php?id=techniken:datenaustausch:serialchars

Die Spannungsversorgung vom ESP als auch der Nanos erfolgt über die USB-Kabel.

Der Chip ATmega328 auf dem Arduino Nano V3.0 als auch der Chip ESP8266 auf der NodeMCU haben interne Pull-up Widerstände integriert: Nano 20...50kOhm , ESP 30...100kOhm.

Sie lassen sich aktivieren mit dem Befehl: `pinMode (pin, INPUT_PULLUP)`

Das Bibliotheksprogramm `Wire.h` enthält bereits diesen Befehl.

Damit funktioniert der I2C Bus.
Für eine Übertragung über längere Leitungen sind zusätzliche (parallel geschaltete) niederohmigere externe Widerstände 4,7kOhm oder 10kOhm günstig (im LOW-Zustand fließt höherer Strom – dadurch störsicherer).

Sketch 66+67 Servos ansteuern über I2C Bus

Im seriellen Monitor der Nanos als auch des ESP erfolgen Anzeigen zum Ablauf, u.a. auch der "Rueckgabewert" vom Slave:

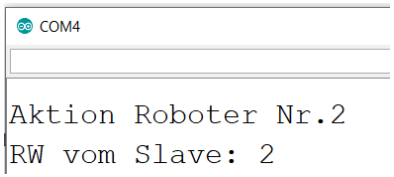
Rückgabewert	Byte, gibt Hinweis auf den Status der Übertragung 0 - alles ok 1 - Daten zu lang; passen nicht in den Übertragungspuffer 2 - NACK erhalten bei Übertragung der Adresse 3 - NACK erhalten bei Übertragung von Daten 4 - sonstiger Fehler

Beachte: Die Ausgabe im seriellen Monitor für den Slave ist nur möglich, wenn er an einem anderen Computer angeschlossen ist (Neu-Aufruf des Monitors löst immer einen Reset aus).

Master:

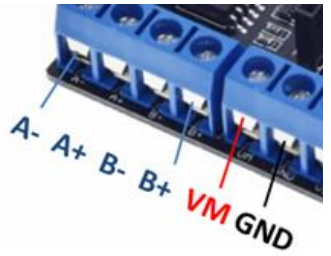
Slave:

wenn Roboter Nr. 2 nicht angeschlossen ist:



Sketch 68 Schrittmotor Roboter um +-90° drehen

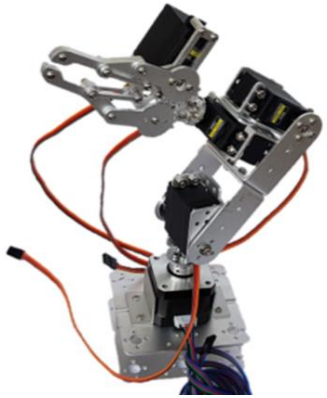
Anschluss des Schrittmotors an ESP8266 NodeMCU V2 L293D Motor Shield.
Terminal-Klemmen A- A+ und B- B+.
Anschluß Netzteil 6V / 7,5V oder Akkupack 7,2 V (tatsächl.Spannung bis 8V)
an VM und GND.
Drehen hin und her um +-90°, mit konstanter Drehzahl.

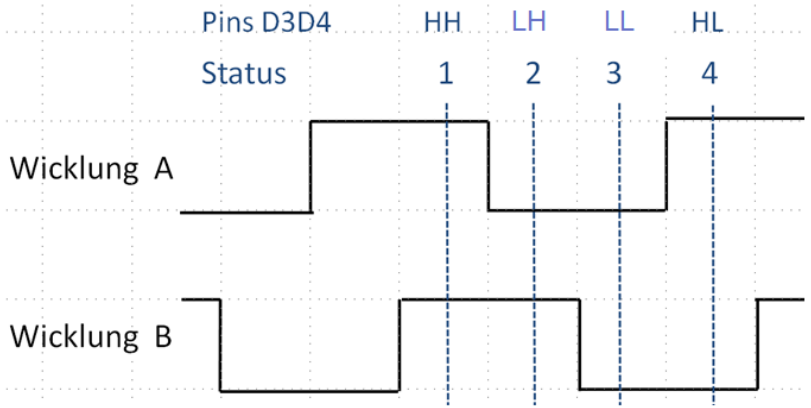


Die Drehzahl kann in der Variablen „Schrittzeit“ eingestellt werden.
Eine Schrittzeit von z.B. 10ms entspricht der Schrittfrequenz 100 Schritte/s und der Drehzahl 30 U/min .

Einstellung der effektiven Spannung an den Wicklungen und damit Strom+Drehmoment über PWM-Ausgänge Pins D1,D2 (max Wert ist 1023 -> volle Spannung liegt an).
Aufgrund der sehr niedrigen PWM-Frequenz von nur 1000Hz pfeift der Motor allerdings , wenn < 1023 eingestellt wird.

Beide Spulen sind immer eingeschaltet, in jeder Spule fließt Strom.
Der gesamte Strom, der aus der Batterie in den Anschluß VM fließt, beträgt etwa 290mA (wenn die Batteriespannung 8V und die Schrittfrequentz 100Hz beträgt).



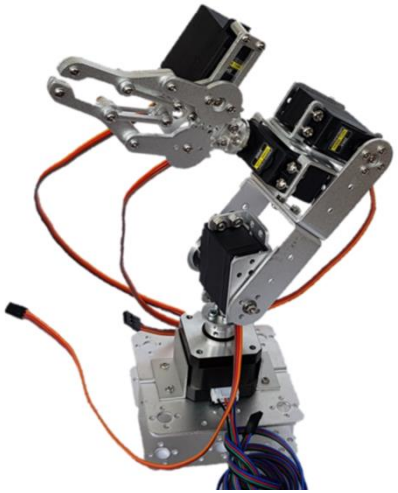


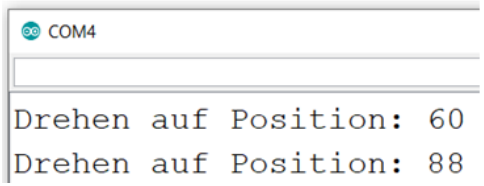
Sketch 69 Schrittmotor Roboter steuern

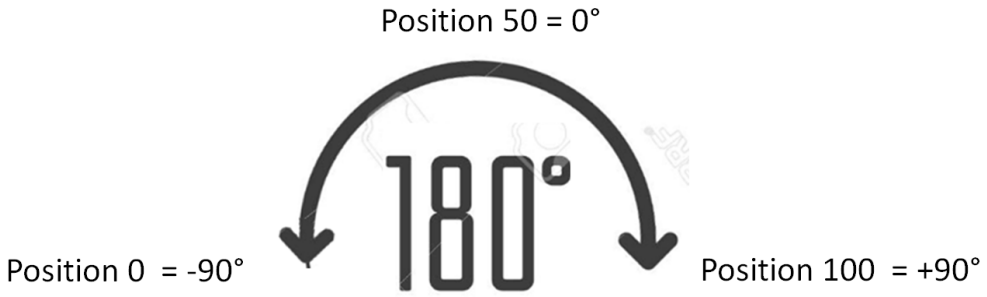
Der Schrittmotor (Roboter-Drehung) kann in Positionen zwischen "0" und "100" bewegt werden
(100 Schritte a 1,8°).
"50" ist die Nullstellung, auch die Startposition.
Der Roboterarm muss vor Einschalten manuell in diese Stellung gedreht werden.
Im seriellen Monitor wird die neue Position eingegeben, z.B. "s100" ("s" für SchrittmotorDrehung).
Eingabe "s0" fuehrt zu -90° und "s100" zu +90°.

Durch Anwendung der Variablen "Position_alt" und "Position_neu" wird gewährleistet, daß bei jedem neuen Drehbefehl kein Schritt verloren geht (stets richtiger Schaltzustand der Ausgänge des L293D).
Es wird der Variablentyp `byte` verwendet, dadurch Begrenzung des Drehwinkels bei versehentlicher Fehleingabe.

Die jeweilige neue Position wird auch im Serial Monitor angezeigt.







Sketch 70 Schrittmotor und Servos steuern

Der Roboter mit Schrittmotor (Drehen Hauptachse) sowie 4 Servos wird über Eingaben im seriellen Monitor gesteuert (siehe auch Sketch 62).

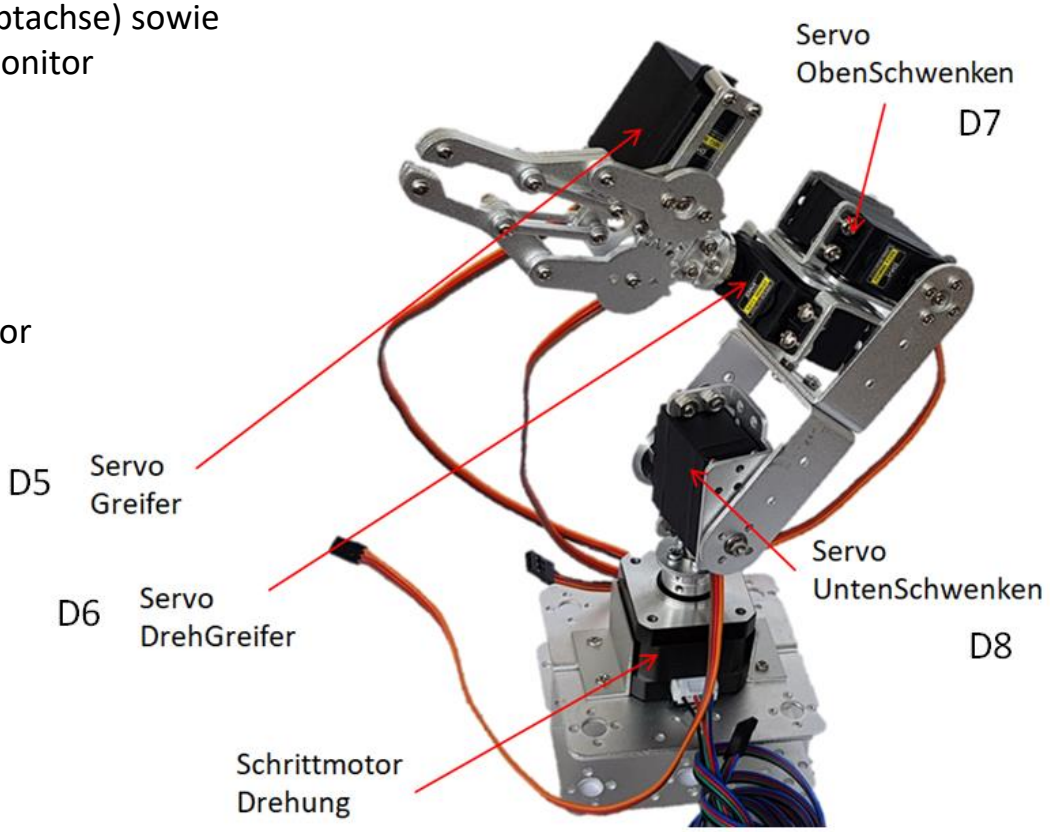
Spannungsversorgung:

5V von Netzteil an VIN für die Servos
9V von Akkupack an VM für den Schrittmotor

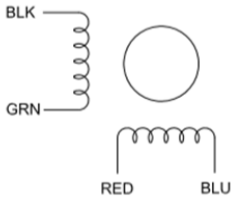
Schrittmotor-Drehung wie in Sketch 69.

Die Servos werden mit Eingabewerten zwischen ca. 600 und 2600 gesteuert, z.B. Eingabe "g1000" fuer die Stellung des Greifers.

Es wird immer nur ein Servo des Roboters bewegt, die anderen bleiben stromlos.



D1 (ENA)	D2 (ENB)
D3 (DA)	D4 (DB)
A- A+	B- B+
BLK GRN	RED BLU



Sketch 70 Schrittmotor und Servos steuern

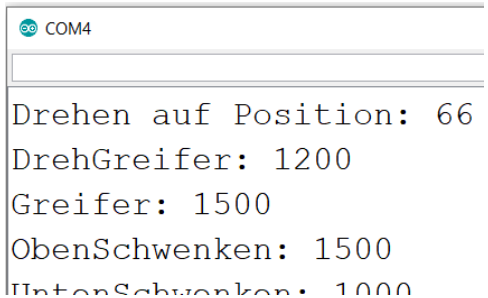
Die "for"-Schleife sorgt dafür, dass der Servo so lange aktiviert bleibt bis er seine Zielstellung erreicht hat.

for () www.arduino.cc/reference/de/language/structure/control-structure/for/

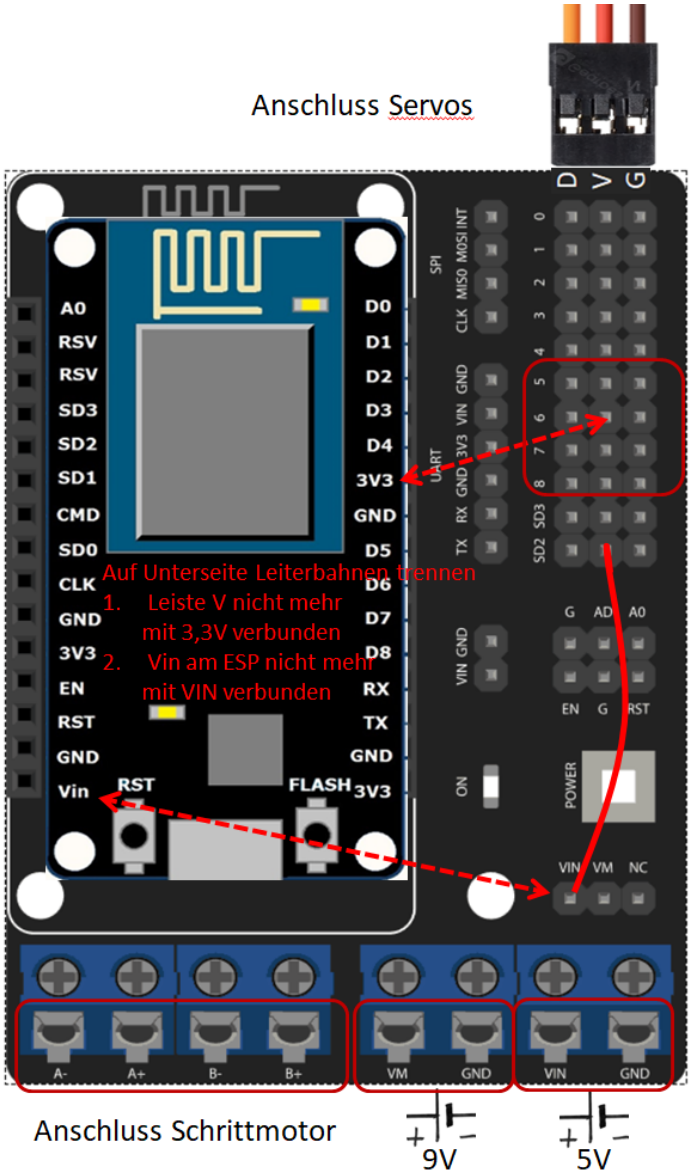
Danach hält er die Position nicht mehr, man kann von Hand verdrehen.

Deshalb können insbesondere die Achsen „Oben Schwenken“ und "UntenSchwenken" nach Abschalten durch das Gewicht des Roboters wegdriften.

Die jeweilige neue Position wird auch im Serial Monitor angezeigt:



Sketch 70 Schrittmotor und Servos steuern



Zuläss.Spannung für L293D (VM):	4,5 ... 36 V
Zuläss. Spannung für Servos:	4,8 ... 7,2 V
Zuläss. Spannung für ESP (VIN):	5,0 ... 9,0 V
Strom pro Servo bei Bewegung:	ca. 0,5 A
und bei Blockierung @ 6V:	2,5 A

Die Spannungsversorgung für den L293D und damit für den Schrittmotor erfolgt an VM. Wir verwenden 9V vom Akku.

Die Spannungsversorgung für die 4 Servos erfolgt an VIN. Wir verwenden 5V von einem Netzteil, damit die Drehmomentbelastung für den Greifer nicht zu gross wird.

Die Spannungsversorgung für den ESP erfolgt über die USB-Verbindung.
Um Störungen von VIN der Servos zum ESP zu vermeiden, wird die Verbindung an der Leiterplattenunterseite getrennt.

Achtung:
Auf dem Motor-Shield ist ein Pin 3V3 des ESP mit der Leiste „V“ verbunden.
Wenn an Leiste „V“ die Versorgungsspannung für die Servos gelegt werden soll, muß diese Verbindung auf der Unterseite des Shields durchtrennt werden.

Es gibt auch Shields 

, die sind hier nicht verwendbar.

Funktionen millis() und micros()

Bisher wurde für die Ansteuerung der Servos die delay-Funktion benutzt.
Während des delay ist der Mikroprozessor aber faktisch abgeschaltet.
Dadurch läßt sich immer nur ein Servo bewegen, die anderen müssen stromlos bleiben.
Der Nachteil ist, dass der Roboterarm bei stromlosem Servo driftet (absinkt).

Mit den Funktionen millis() bzw. micros() kann man dieses Problem lösen.

`millis()` gibt die Anzahl von Millisekunden (ms) zurück, seit der Sketch gestartet wurde.
Diese Zahl läuft nach etwa 50 Tagen über (geht auf Null zurück).

Man kann die Zahl in eine Variable laden, sie muss das Format unsigned long haben;

```
unsigned long time = millis();
```

Entsprechend gibt `micros()` die Anzahl von Mikrosekunden (µs) zurück.
Diese Zahl läuft nach etwa 70 Minuten über.
Die zeitliche Auflösung beträgt 4 µs bei Arduino mit 16MHz und 0,8 µs bei ESP8266 mit 80MHz.

millis()	www.arduino.cc/reference/de/language/functions/time/millis/
micros()	www.arduino.cc/reference/en/language/functions/time/micros/

Sketch 72 Roboter Bewegungsablauf – nur Schrittmotor

In diesem Sketch wird eine Ablaufsteuerung, d.h.kompletter Bewegungsablauf, für den Roboter programmiert.
Aber nur fuer den Schrittmotor (Drehung Roboter), nicht für die Servos.

Der Schrittmotor wird mit Schrittfrequenz 50 Schritten/s betrieben, dann kann jeweils am Ende des 20ms Servo-Zyklus ein neuer Schritt ausgeführt werden (vereinfacht den Sketch wenn auch die Servos mit gesteuert werden sollen).

Es wird jeweils 2 Sek Zeit gegeben um die neue Position sicher zu erreichen.

Der Roboter muss manuell vor dem Start in die Startposition 0° gedreht werden (Referenzposition "50").

Der Schrittmotor hat eine Nennspannung von 12V und ist besonders bei niedriger Spannung sehr schwach.
Wir verwenden 9V von einem Akku.
Die Stromaufnahme des Schrittmotors ist relativ niedrig.

Die neuen Positionen werden in ein Array eingetragen und im Sketch ausgelesen.

Siehe auch: https://starthardware.org/lektion-15-array/

array [ ] www.arduino.cc/reference/de/language/variables/data-types/array/

Es werden die Funktionen millis() und micros() angewendet.

Sketch 73 Roboter kompletter Bewegungsablauf

Siehe Video dazu auf der Webseite.

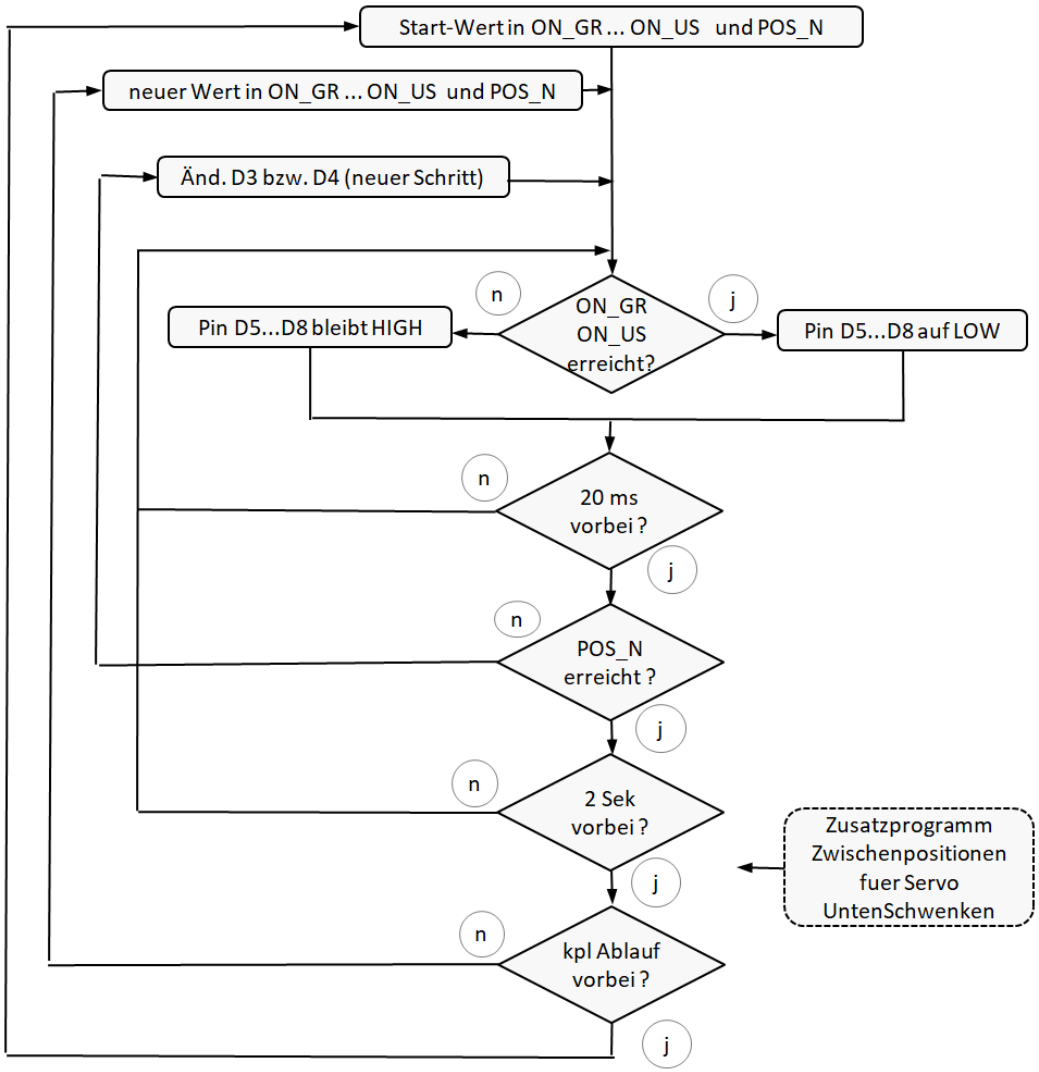
In diesem Sketch wird eine Ablaufsteuerung, d.h.kompletter Bewegungsablauf, für den Roboter programmiert.
Es werden die 4 Servos als auch der Schrittmotor (Drehung Roboter) gesteuert.

Der Schrittmotor wird mit Schrittfrequenz 50 Schritten/s betrieben, dann kann jeweils am Ende des 20ms Servo-Zyklus ein neuer Schritt ausgeführt werden (vereinfacht den Sketch).
Es wird jeweils 2 Sek Zeit gegeben um die neue Position sicher zu erreichen.

Die neuen Positionen werden aus Arrays ausgelesen.

Der Roboter muss manuell vor dem Start in die Startposition 0° gedreht werden (Referenzposition "50").

Sketch 73 Programmablaufplan



Servos:
Wert für Dauer des HIGH-Pulses
(in die Variablen ON_GR ... ON_US)

Schrittmotor:
Startwert für neue Position
(in die Variable POS_N)

Servos:
Zeitdauer des HIGH-Pulses erreicht ?

Servos:
kompletter Zykluszeit 20ms erreicht ?

Schrittmotor:
Pulszeit 20ms erreicht ?

Schrittmotor:
neue Position erreicht ?

Servos:
2 Sek vorbei ? Soviel Zeit wird gegeben um
die nächste Position sicher zu erreichen

Ist der komplette Bewegungsablauf abgeschlossen
(alle Positionsaufrufe abgearbeitet) ?
Wenn ja - Wiederholung des gesamten Bewegungsablaufs

Anhang

Inhaltsverzeichnis

Begriffe und Abkürzungen

BIT	Binary Digit
CAN	Controller Area Network
CS	Chip Select (SPI-Bus)
I2C	Inter Integrated Circuit
LIN	Local Interconnect Network
LSB	Least Significant Bit (niederwertigstes Bit)
MISO	Master-In Slave-Out (SPI-Bus)
MOSI	Master-Out Slave-In (SPI-Bus)
MSB	Most Significant Bit (höherwertigstes Bit)
SCL	Serial Clock (SPI-Bus, I2C-Bus)
SDA	Serial Data (I2C-Bus)
SPI	Serial peripheral Interface
SS	Slave Select (SPI-Bus)
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus

Hexadezimalsystem

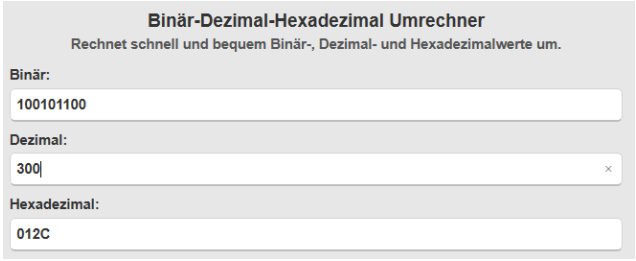
Dezimalsystem = 10 Ziffern -> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Binärsystem = 2 Ziffern -> 0, 1

Hexadezimalsystem = 16 Ziffern -> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

0x (Null x) zeigt an, daß es sich um eine Hexadezimalzahl handelt

https://bin-dez-hex-umrechner.de/



https://www.youtube.com/watch?v=nbUiXka4tj0

Ende

Inhaltsverzeichnis