

Open Source Roboter Plattform

Lukas Reichwein

Yves Ehrlich

Nick Gnoevoj

University of Applied Science Fulda — 13. Februar 2020

Inhaltsverzeichnis

1 Vorwort	3
2 Überlegung	4
3 Mechanik	6
3.1 Montagemöglichkeiten	6
3.2 Antrieb	6
3.3 Verbesserung	6
4 Elektronik	7
4.1 Verfahren	7
4.2 Bestückung	8
4.3 Schaltung	8
5 Fernsteuerung	9
6 Code allgemein	10
7 SPI	11
7.1 Was ist SPI?	11
7.2 Einsatz von SPI innerhalb des Projektes	11
8 Drahtlose Verbindung	12
8.1 Allgemeine Verwendung von RF24	12
8.2 Verwendung von RF24 innerhalb dieses Projektes	13
9 Steuerung	15
9.1 Button Steuerung	15
9.2 Joystick Steuerung	15
9.3 Überführung der Joystick Steuerung in eine Library	16
9.4 Aufbau der Klassen	16

10 Arduino Libaries	18
10.1 OneWire	18
11 Gedruckte Bauteile	19
12 Motorsteuerung	21
13 Thermosensor	23
14 Ultraschallsensor	26
15 Beispiele für Spezielle LaTeX Strukturen	28

1 Vorwort

Motivation: Eine Plattform bieten ist etwas, das momentan sehr stark im Trend liegt, sei es im Software oder im Hardware Bereich. Im Softwarebereich zeigt sich dies meist durch Open-Source Bibliotheken, welche möglichst variabel einsetzbaren Code für jeden frei zugänglich machen. Ein solches Projekt war auch von einem der Projektmitglieder (Yves Ehrlich) als Privates Projekt geplant und so kam die Überlegung dies innerhalb des Modules Embedded Networking zu wählen.

Basis des Projektes: Als Basis des Projektes dient einer schon bereits von Yves Ehrlich angefertigter Arduino Nano Shield samt Code, [?] welcher als Fernsteuerung verwendet wird.

Ziel des Projektes: Ziel des Projektes ist ein ferngesteuerte, Open-Source basierende Roboterplattform.

2 Überlegung

Die urprüngliche Idee war die Entwicklung eines Solarroboters, also eines Roboters, der tagsüber mit Hilfe der Sonneneinstrahlung seinen Eigenbedarf decken kann und somit nur einen kleinen Pufferakku benötigt. Praktische Anwendungen könnten z.B. die Nutzung als kompakte Straßenkehrmaschine oder als Rasenmäher sein. Bedingt durch die eingeschränkte Leistungsabgabe der Solarzellen vermutlich nur für kleine zu bearbeitende Flächen. Da es nur wenige brauchbare und frei zugängliche Quellen für solche Plattformen gibt, wollten wir eine eigene schaffen. Daher sollte das Projekt exemplarisch als Grundlage für eine Quelloffene Plattform dienen, die auch von anderen genutzt und weiterentwickelt werden kann.

Da wir bei der Entwicklung der mechanischen Komponenten schnell feststellen mussten, dass das Fahrwerk recht komplex ist, insbesondere die Vorderachse, schwenkten wir schnell auf eine andere Bauweise um. Zunächst war der Aufbau mit 4 gleichen Rädern geplant, von denen zwei Stück (Hinterachse) von einem bürstenlosen Gleichstrommotor angetrieben werden sollten. Die Entwicklung eines Differenzials gestaltete sich noch als relativ einfach. Die Vorderachse aber mit dem Servo gestaltete sich dann, hinsichtlich des Zeitaufwandes, für zu komplex.



Abbildung 1: Hinterachse des ursprünglichen Aufbaus



Abbildung 2: Lagerschalen

Somit schwenkten wir kurzer Hand zu einem sehr viel einfacheren Aufbau um, um uns schneller den wichtigeren Komponenten widmen zu können. Der neue Aufbau sah eine schlichte Grundplatte als Plattform für die weiteren Bauteile, wie Motoren (Mabuchi N20 mit Getriebe), Ultraschallsensor, Akku, Hauptplatine und uvm. vor. Um möglichst viele Montagemöglichkeiten zu bieten, wurde die Grundplatte mit Bohrungen in regel-

mäßigen Abständen versehen. Diese Bohrungen können zum durchstecken für Schrauben oder zum versenken von Gewindemuffen genutzt werden. Zum anbringen unserer Komponenten haben wir auf Messinggewindebuchsen und passende M3 Schrauben gesetzt. Alle selbstentworfenen Komponenten haben wir mit einem FFF 3D-Drucker aus PLA gedruckt. Mit Ausnahme der Hauptplatine natürlich. Bedingt durch den Zeitaufwand des gesamten Projektes und anderer Studienmodule haben wir uns entschieden, die Implementierung eines MPPT Ladereglers fallen zu lassen, somit wurde die Idee des Solarroboters auf die Idee der Entwicklung eines Roboters reduziert.

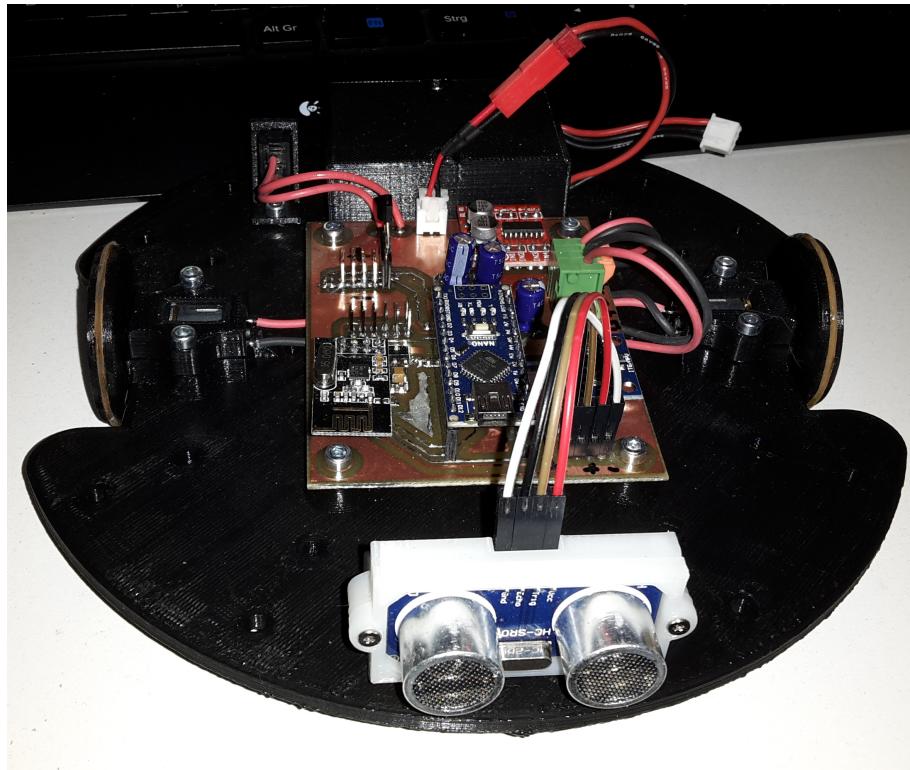


Abbildung 3: Der Roboter, aktueller Aufbau

3 Mechanik

Wie bereits unter „Überlegung“ erwähnt, gestaltete sich der Aufbau mit Achsträger, Radlager und Lagerschalen uvm. als zu Zeitintensiv um in der vorgegebenen Zeit bewältig werden zu können. Zum Entwurf der mechanischen Bauteile wurde das CAD Programm Fusion 360 von Autodesk verwendet. Dabei gab es kaum Schwierigkeiten, da Yves Ehrlich bereits sehr viel Erfahrung mit 3D-CAD Programmen gesammelt hat. Lediglich der Zwang Windows zu benutzen stellt eine ständige Unannehmlichkeit dar.

Alle Entwürfe wurden mit einem 3D Drucker gedruckt, einem HCmaker7, ein Klon des bekannten Creality CR-10S. Das Material der Wahl war PLA. Für einen Außeneinsatz ist das Material wegen der geringen Witterungsbeständigkeit kaum geeignet. Insbesondere die geringe Temperaturbeständigkeit (max. 60°C) disqualifizieren das Material für den Einsatz in der Sonne. Da dies aber ohnehin nicht mehr geplant war, ist das Material für unseren Zweck geeignet.

3.1 Montagemöglichkeiten

Die zur Befestigung von Bauteilen nötigen Gewindebuchsen in den Kunststoff einzusetzen ist relativ einfach, man benötigt dafür lediglich eine Schraube die man in die Buchse einschraubt und einen Lötkolben mit einer Temperatur von etwa 240°C. Ohne Druck auszuüben legt man die Spitze des Lötkolbens auf den Schraubenkopf. Wenn die Schraube ausreichend Wärme an die Muffe übertragen hat, fängt die Muffe an im Kunststoff zu versinken. Die Schwerkraft sorgt hier für alles nötige. Wenn die Muffe weit genug versunken ist, kann der Lötkolben entfernt werden und man sollte die Schraube schnell abkühlen, z.B. durch pusten oder ein kleines Gebläse wie z.B. einen Radiallüfter. Beim Design der Kunststoffteile muss man darauf achten, dass für Gewindemuffen stehts eine ausreichend große Fase vorhanden sein muss, damit die Muffe einerseits zentriert wird beim versinken und andererseits kein verdrängtes Material die Muffe beim versinken behindert oder verstopft. Letzterem wird aber auch durch die eingeschraubte Schraube vorgebeugt.

3.2 Antrieb

Der Aufbau sieht zwei Antriebsmotoren vor, welche jeweils ein Rad treiben. Zum abstützen verfügt der Roboter über ein selbstentworfenes Kugelrad. Dieses Kugelrad verwendet als Kugeln vier handelsübliche Airsoft Kugeln mit einem Durchmesser von 6mm. Die Gleiteigenschaften dieser Konstruktion sind nicht die besten, aber für diesen Prototypen ausreichend. Um die Reibung zu vermindern haben wir Silikonöl auf die Kugeln aufgetragen.

3.3 Verbesserung

Da Kunststoffe auf glatten Oberflächen sehr wenig Bodenhaftung haben, wurden die Räder mit einer Nut versehen. In diese Nut kann man später ein einfaches Haushaltsgummiband einlegen, somit hat der Roboter auch auf glatten Oberflächen relativ gute Haftung.

4 Elektronik

Lochrasterplatinen, wie sie häufig für Prototypzwecke genutzt werden, sind nicht nur unpraktisch für SMD Komponenten, auch wenn man ein 2,54mm Raster mit einigen Tricksereien durchaus auch für SMD Komponenten nutzen kann, sie sind auch relativ unzuverlässig und die Fehlersuche gestaltet sich z.T. als schwierig (je nach Menge und Komplexität der Verdrahtungslagen). Deshalb entschieden wir uns, eine eigene Platine zu entwickeln. Als Software nutzten wir dafür das kommerzielle Programm Sprint Layout 6 von der Firma Abacom.

Da die Herstellung der Platine von Hand erfolgen sollte, haben wir die Strukturen der Platine sehr grob gezeichnet. Was auch durch die geringe Anzahl an Komponenten und deren modulare Bauweise begünstigt wurde.

4.1 Verfahren

Als Verfahren zum beschichten der Platine diente die Tonertransfermethode. Dabei wird die Schaltung mit einem Laserdrucker auf geeignetes Papier gedruckt, in diesem Fall Photopapier, und dann mit Hitze auf die blanke Platine gedrückt. Für diesen Zweck hat Yves Ehrlich ein Laminiergerät umgebaut, welches den Vorgang erheblich erleichtert. Da unsere Platine doppelseitig ist, müssen zuvor die zwei Seiten, welche die Schaltung enthalten Deckungsgleich aufeinander ausgerichtet werden und dann an den Rändern verklebt werden, damit man am Ende eine Art Tasche erhält, in welche dann die Platine eingesteckt werden kann. Das ganze läuft dann einige Male durch das Laminiergerät. Nach dem das erfolgt ist, lässt man die Platine etwas abkühlen und ab einer Temperatur von etwa 30-40°C schneidet man die Tasche seitlich auf und zieht vorsichtig das Papier ab. Der Toner bleibt am Kupfer haften und schützt an diesen Stellen, wie ein Fotolack, das Kupfer davor weggeätzt zu werden. Anschließen gibt man die beschichtete Platine in eine Ätzküvette welche mit einer Natriumpersulfat-Lösung gefüllt ist. Nach einiger Zeit kann man beim durchleuchten der Küvette sehen, wie sich das Kupfer immer weiter auflöst, bis am Ende die Schaltung klar und deutlich zu sehen ist. Dann wird die Platine entnommen und gründlich mit Wasser abgespült. Zum entfernen des Toners benötigt man etwas Aceton und einen Lappen. Für die THT Komponenten müssen nun noch Bohrungen gesetzt werden. 1,2mm, passend für die Durchkontaktierungen von Bungard. Nach dem verpressen (mangels passender Presse eher mit einem Körner breitschlagen) der Nieten, müssen diese beidseitig verlötet werden. Danach können alle Komponenten bestückt werden.

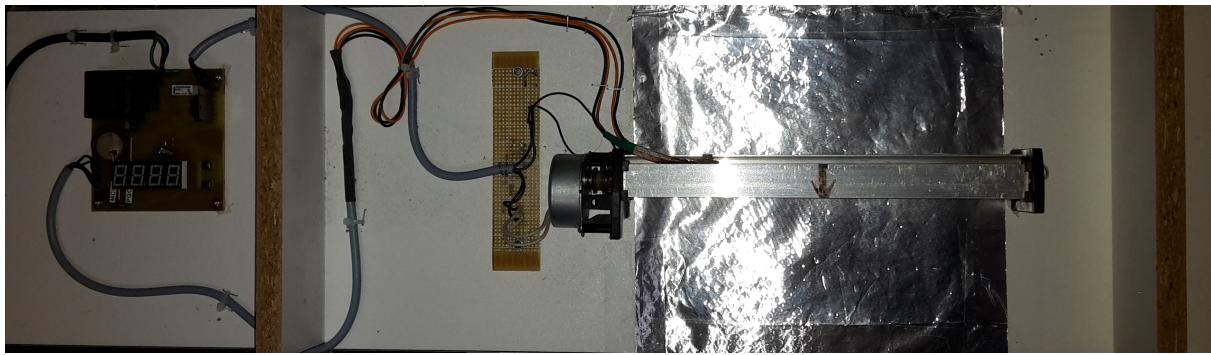
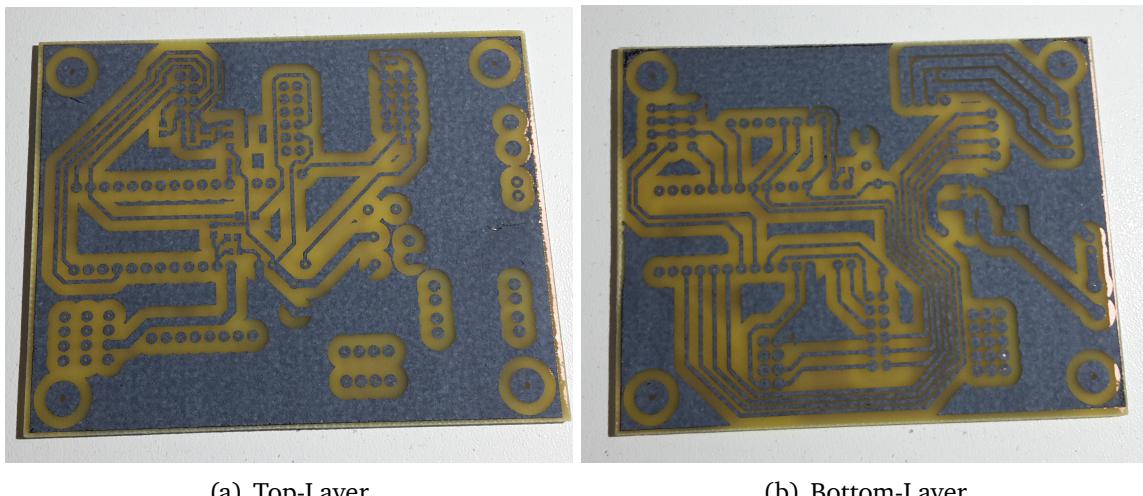


Abbildung 4: Laminiergerät



(a) Top-Layer

(b) Bottom-Layer

Abbildung 5: fertig geätzte Platine

4.2 Bestückung

Zum verlöten der Komponenten kam SMD-Lot mit Bleianteil zum Einsatz. Für SMD Komponenten wurde eine Pinzette als Hilfswerkzeug verwendet.

4.3 Schaltung

Die Schaltung ist relativ einfach gehalten. Im wesentlichen dient die Platine nur als Träger für die modularen Komponenten und für diverse Anschlussmöglichkeiten. Für den NRF24 wurde ein 3,3V Spannungsregler integriert. Um etwaige Motorenwechsel zu erleichtern, sind zu deren Anschluss Federzugklemmen verbaut worden. Wegen der Überschaubarkeit der Schaltung haben wir auf einen Schaltplan verzichtet, da das Layout selbsterklärend ist.

5 Fernsteuerung

Auf die Fernsteuerung soll an dieser Stelle nicht näher eingegangen werden, da diese zum Zeitpunkt der Erstellung dieses Projektes bereits „fertig“ war. Es wurden lediglich ein Joystick sowie ein NRF24 ergänzt. Detaillierte Informationen finden sich im dazugehörigen Repository[?].

Im Rahmen eines Arduino Workshops für Anfänger entwickelte Yves Ehrlich eine Platine, welche als Shield für einen Arduino Nano fungiert. Sie enthält grundlegende Dinge wie Tasten, einen Akku, eine stabile 5V Spannungsversorgung sowie ein monochromes LCD Display mit Hintergrundbeleuchtung, um nur die wichtigsten Sachen zu nennen.

Dank der Buchsenleiste auf der Rückseite, welche neben 3,3V und 5V Spannungsversorgung auch diverse I/Os und Analogeingänge bereitstellt, lässt sich das Shield „nanoGame“ hervorragend erweitern und für viele verschiedene Dinge nutzen, so auch als Fernsteuerung für unseren Roboter.

Bei dem Joystick handelt es sich um eine Platine welche ein original Ersatzteil der Sony PSP enthält. Dieser Joystick ist sehr kompakt und ließ sich dadurch recht einfach in unsere Projekt integrieren. Es musste lediglich eine Halterung konstruiert werden, um den Joystick an das Gehäuse der Fernsteuerung zu schrauben.

Beim Funkmodul sind wir kompromissloser vorgegangen. Dieses wurde mit ein wenig Silikonklebstoff an der Oberseite fixiert.

Alle Verbindungsleitungen wurden Freiluft auf der Rückseite der Fernsteuerung zum Schacht der bereits erwähnten Buchsenleiste gelegt und dort mit Steckerleistenangeschlossen.

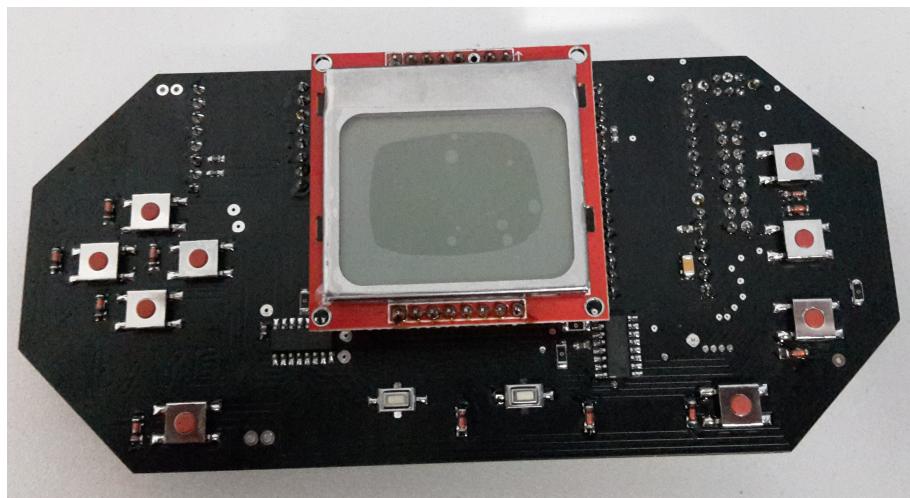


Abbildung 6: Fernsteuerung vorn

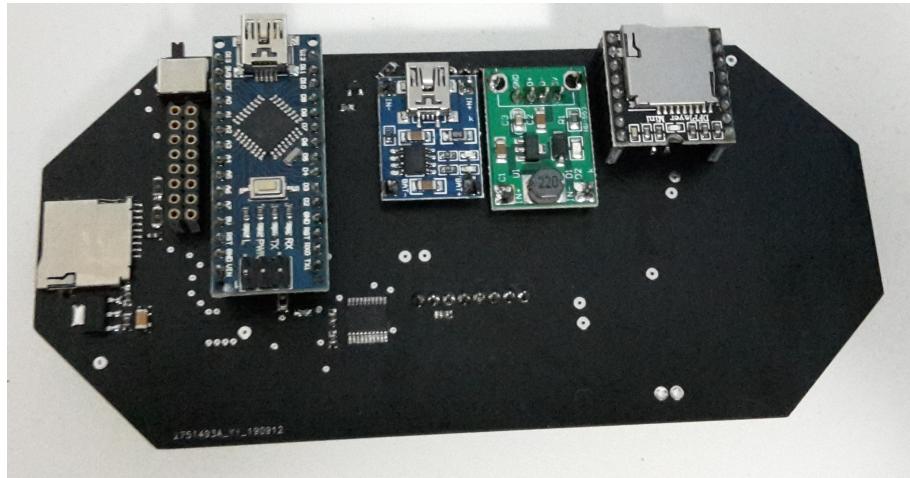


Abbildung 7: Fernsteuerung hinten

6 LCD

Als LCD kam, entsprechend dem nanoGame [?] Projekt, ein monochromes Grafik-LCD mit Hintergrundbeleuchtung und einer Auflösung von 84 x 48 Pixeln zum Einsatz. Beim integrierten Controller handelt es sich um den PCD8544. Angesteuert per SPI und diversen Steuerleitungen.

Da aktuell im Projekt keine Grafiken dargestellt werden, entschieden wir uns, das LCD einfach fest in 6 Zeilen einzuteilen. Um kollisionsfrei Texte ausgeben zu können, verwendeten wir ein global deklariertes Stringarray. Jeder Teil des Codes, der etwas ausgibt schreibt in ein Element bzw. in eine Zeile. die Methode „refreshLCD()“ liest dieses Stringarray ein und gibt die 6 Zeilen auf dem LCD aus.

7 Code allgemein

Um die Einsatzmöglichkeiten für den Code möglichst flexibel zu gestalten, haben wir uns bemüht, nicht blockierenden Quelltext zu schreiben. Somit lässt sich auch mit dem recht kleinen Atmega328p bei 16 MHz Takt eine ganze Menge bewältigen.

Aktuell ist der Roboter in der Lage auf Kommandos, die er von der Fernsteuerung erhält, zu reagieren. Aus Zeitgründen konnten wir leider nur einen manuellen Fahrmodus implementieren. Dieser beinhaltet das fahren mit digitalen Tasten sowie „analog“ mit dem ergänzten Joystick. Weiterhin implementiert wurden das erfassen von Temperaturen mit Hilfe eines DS1820 und eine Distanzmessung mit einem HC-SR04 Ultraschallsensor.

Die Übertragung von Kommandos ist relativ einfach gehalten. Das erste Byte im commands Array ist ein Kommando, gefolgt von 2 Bytes mit Daten. Diesem Muster folgend ist jedes dritte Byte ein Kommando, gefolgt von zwei Datenbytes. Da das Array eine feste Größe von 32 Byte hat, lassen sich folglich bis zu 10 Kommandos übertragen. Wie viele Kommandos jeweils übertragen werden und in welcher Reihenfolge diese im Array vorkommen, also ob Byte 0,3,6,9...usw., spielt keine Rolle.

8 SPI

8.1 Was ist SPI?

SPI - Serial Peripheral Interface ist der Standard für ein Synchronen Seriellen Datenbus. Dieser Datenbus kann zwei digitale Schaltungen mittels Master-Slave Prinzip verbinden [?]. Der Bus ermöglicht die Verwendung von mehreren Slaves, jedoch wird die Anzahl der Slaves durch die Anzahl der CS - ChipSelect Leitungen begrenzt. Die Datenübertragung geschieht über eine separate Leitung, auch SD (Serial Data) genannt.

Rolle des Masters Der Master gibt das Taktsignal über den SCK (Serial Clock) Ausgang an, des Weiteren wählt er einen Slave mit welchem er kommunizieren möchte über den CS (ChipSelect).

Rolle des Slaves Der Slave ist solange inaktiv bis dieser über den CS Pin aktiviert wird. Nach der Aktivierung wird über Serial Data kommuniziert, das Taktsignal wird über den SCK synchronisiert bzw vom Master übernommen.

8.2 Einsatz von SPI innerhalb des Projektes

Verbindung: Um die Verbindung zwischen dem nRF24L01 Chip und dem Arduino herzustellen wurden zunächst alle Verbindungen getätigt. Im Folgenden Graphik ist das Pinout vom nRF24L01 Chip abgebildet. Dabei sind die Kürzel wie folgt benannt: CE - Chip Enable dieser entspricht dem Chip Select Pin, CSN - Chip Select Not dieser ist die Invertierte Version des CE, SCK - Serial Clock, MISO - Master in Slave Out, MOSI - Master Out Slave In und die beiden Pins für die Stromversorgung VCC und Ground. Die beiden Pins MISO und MOSI sind die Datenleitungen und werden an den Gleichnamigen Pins des Arduinos angeschlossen. Des Weiteren ermöglicht der IRQ Interruptsteuerung dieser Pin ist optional zu verwenden.

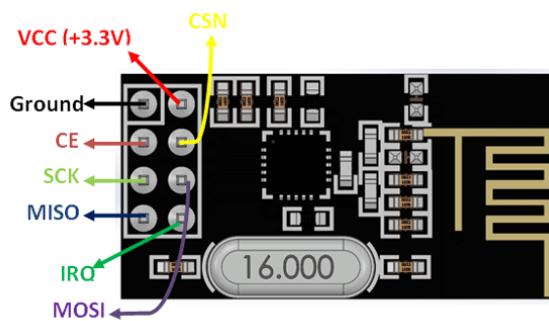


Abbildung 8: Pinout des nRF24L01 Chips [?]

9 Drahtlose Verbindung

Wie schon zuvor erwähnt wird für die Basis der Funksteuerung das Arduino Shield verwendet, welches mit dem nRF24L01 Chip erweitert wurde. Der Roboter wurde ebenfalls mit dem nRF24L01 ausgestattet.

9.1 Allgemeine Verwendung von RF24

Die Opensoucre Bibliothek RF24 [?] diente als Codebasis für die Funksteuerung. Da diese Bibliothek bei korrekter Verwendung genau auf die Kommunikation zwischen zwei nRF24L01 Chips abgestimmt ist. Zur Verwendung der Bibliothek muss sie nur inkludiert und Instanziert werden dabei werden die Beiden Pins CE und CSN für das Hardware-SPI konfiguriert.

RF24 initialisieren

```
#include <RF24.h>
RF24 radio(A2, A3); // CE, CSN
const byte address[6] = "00001";
```

Damit sind bereits Sämtliche Konfigurationen für die Hardware-SPI Kommunikation zwischen Arduino nano und dem nRF24L01 erledigt. Kommunizieren zwischen zwei dieser Setups wird dann durch die Funktionen read und wirite, jedoch muss vor dem Start einer Kommunikation die Methoden begin(), openWritingPipe(address) mit address als Kommunikationsleitung und stopListening() bei Sender bzw startListening() beim Empfänger aufgerufen werden. Die Methode setPAlevel() setzt den Power Amplifiyer Level umso höher dieser Wert umso Stärker das Signal, wodurch der Stromverbrauch Steigt. Der Anstieg des Stromverbrauchs kann auch bei einer nicht ganz konstanten Stromversorgung dazu führen das das Modul nicht korrekt funktioniert.

RF24 initialisieren

```
//An der Sender Seite
radio.begin();
radio.openWritingPipe(address);
radio.setPALevel(RF24_PA_MAX);
radio.stopListening();
radio.write(&payload, sizeof(payload));

//An der Empfaenger Seite
radio.begin();
radio.openReadingPipe(0, address);
radio.setPALevel(RF24_PA_MAX);
radio.startListening();
if (radio.available()) {
    radio.read(&payload, sizeof(payload));
    //Payload weiter verarbeiten.
}
```

9.2 Verwendung von RF24 innerhalb dieses Projektes

Am Anfang des Projektes: Zu Begin des Projektes wurde zuerst die Funkkommunikation zwischen zwei Arduino's mittels nRF24L01 getestet, dabei gab es schon die ersten Komplikationen. Da der Chip nicht korrekt mit den Hardware SPI Anschlüssen des Arduino verbunden wurde. Nach dem weiteren Einlesen in die Funktionsweise von SPI konnten die Verbindungen richtig gesetzt und diese Komplikation wieder aufgehoben werden.

Logik hinter der Kommunikation: Um den Roboter mit der Fernsteuerung auch Steuern zu können musste erst ermittelt werden welche Daten der Roboter benötigt um zu fahren. Diese Daten wurden dann in ein 32Byte großes Commands Array gespeichert, welches dann übertragen wurde. Das Array besteht aus Folgenden Inhalten:

1. speedA - Geschwindigkeit des rechten Motors.
2. dirA - Vorzeichen der Geschwindigkeit des rechten Motors.
3. speedB - Geschwindigkeit des linken Motors.
4. dirB - Vorzeichen der Geschwindigkeit des linken Motors.
5. goDrive - startet autodrive.
6. stopDrive - stoppt autodrive.
7. getTemp - löst Temperaturübertragung aus.

8. timeToDrive - Zeit die der Roboter fahren soll.
9. nothing - Stoppt den Roboter.

Übertragung der Commands Das Commands Array wurde dann wie in der Sektion allgemeine Verwendung von RF24 beschrieben Übertragen, dabei ist die Payload das Commands Array. Der Sender ist Die Fernsteuerung und der Roboter ist der Empfänger. Der Empfänger besitzt daher auch die commandInterpretation() Funktion, welche die entsprechenden Kommandos liest und umsetzt.

10 Steuerung

Der Roboter kann über ein Steuerkreuz mit 4 Buttons oder einen Joystick gesteuert werden. Begonnen wurde mit der Steuerung über die Buttons, dann wurde die Steuerung mittels Joystick realisiert und anschließend wurde diese dann noch stark variable als C++ Library Implementiert.

10.1 Button Steuerung

Die Button Steuerung ermöglicht entweder Vorwärts, Rückwärts, Links oder Rechts zu fahren. Dabei werden die Beiden Motoren mit den jeweilig maximalen Pwm-Werten angesprochen, sprich sie sind entweder eingeschaltet oder abgeschaltet.

Logik hinter der Steuerung Der Roboter soll für Folgende Tasten folgende Bewegungen durchführen:

- Oben - vorwärts fahren
- Unten - rückwärts fahren
- Rechts - Drehung im Uhrzeigersinn
- Links - Drehung gegen den Uhrzeigersinn

Realisiert wurde es folgendermaßen:

- vorwärts fahren - setzte PWM-Werte beider Motoren auf das Maximum
- rückwärts fahren - setzte PWM-Werte beider Motoren auf das minimum
- Drehung im Uhrzeigersinn - setze den PWM-Wert des rechten Motors auf das minimum und den des linken auf das maximum
- Drehung gegen den Uhrzeigersinn - setze den PWM-Wert des rechten linken auf das minimum und den des rechten auf das maximum

10.2 Joystick Steuerung

Die Steuerung über einen Joystick ermöglicht das kontrolieren über die Aussteuerung des Joysticks. Verwendet wurde ein klassischer Joystick, welcher aus dem Consolen bereich stammt.

Grundlegende Verwendung des Joysticks Der Joystick besitzt zwei analoge Anschlüsse und überträgt dort werte von 0 bis 511 je nach aussteuerung des Joysticks, jede Achse hat somit ihren eigenen Pin. Die Werte der jeweiligen Achsen können dann über AnalogRead() mit dem Pin der Achse als Argument abgefragt werden. Die Mittelstellung des Joysticks ist demnach bei 255.

Mapping: Da der Bereich der Werte nicht zu dem der benötigten PWM- Werten passt werden diese mithilfe der map() Funktion von 0 bis 511 auf -255 bis 255 gemappt. Nach dem Mapping kann die Auswertung über die Richtung durchgeführt werden um dann in den entsprechenden Fällen die gemappten Werte als Geschwindigkeit zu interpretieren.

Anbringung und dessen Auswirkung auf die Steuerung Der Joystick wurde so an das Shield angebracht, so dass durch ein nach Vorne drücken die Achsenwerte für die Y-Achse kleiner werden und ein nach Links drücken die X-Achse ebenfalls kleiner werden. Die Aussteuerung nach Rechts folgt demnach zu einer Erhöhung der Achsenwerte der X-Achse und die Aussteuerung nach unten eine Erhöhung der Achsenwerte der Y-Achse.

Logik hinter der Steuerung Zuerst wurde die Logik der Button Steuerung übernommen, jedoch mit folgenden Veränderungen. Vorwärtsfahren wird jetzt aktiviert, wenn der Achsenwert der Y-Achse kleiner als die Mittelstellung abzüglich eines Toleranzbereiches ist. Das Rückwärtsfahren wird aktiviert, wenn der Achsenwert der Y-Achse größer als die Mittelstellung zuzüglich eines Toleranzbereiches ist. Eine Drehung im Uhrzeigersinn wird ausgelöst, wenn der Achsenwert der X-Achse größer als die Mittelstellung zuzüglich eines Toleranzbereiches ist. Demnach wird eine Drehung gegen den Uhrzeigersinn ausgelöst, wenn der Achsenwert der X-Achse kleiner als die Mittelstellung abzüglich eines Toleranzbereiches ist. In dieser Version ist der Toleranzbereich im Code mit 20 festgeschrieben.

Umsetzung der Steuerung Die Umsetzung ist nichts anderes als die PWM-Werte für die beiden Motoren in jedem der Steuerungsfälle zu ermitteln. Da die Werte schon gemappt wurden können diese nun wieder verwendet werden. In der folgenden Graphik ist zu erkennen wann welche Werte gesendet werden und ab welcher Aussteuerung die PWM-Werte negativ bzw. positiv werden.

10.3 Überführung der Joystick Steuerung in eine Library

Hier wurde besonderes Augenmerk auf Wiederverwendbarkeit und Variabilität gelegt. Des Weiteren ist die Library komplett in C++ geschrieben. Wie genau eine C++ Library erstellt wird, wird in dieser Sektion nicht aufgefasst sondern nur die Besonderheiten betreffend der Joystick Library.

10.4 Aufbau der Klassen

Um eine höhere Variabilität und Wiederverwendbarkeit zu ermöglichen wurde die Library Objekt Orientiert programmiert. Dazu wurden drei Klassen implementiert: Joystick, Motor und Steuerung. Von diesen Klassen ist die Steuerung diejenige, welche die eigentliche Logik beinhaltet, jedoch benötigt man zur Verwendung der implementierten Steuerung zwei Motoren und einen Joystick. Um einen Motor zu erstellen reicht es den Höchsten und den Niedrigsten PWM-Wert anzugeben. Über das Attribut PWMValue lässt sich nach dem Aufruf der updateValues() Funktion der Steuerungsklasse der aktuelle PWM-Wert jedes Motors einzeln abrufen. Beim instantiiieren der Joystick-Klasse müssen

Joystick
<pre> -x_pin: uint16_t -y_pin: uint16_t +lowestValue: int16_t +highestValue: int16_t +spaceing: uint16_t +xAxisReading: int16_t +yAxisReading: int16_t +Joystick(x_pin: uint16_t, y_pin: uint16_t, lowestValue: int16_t, highestValue: int16_t, spaceing: uint16_t) +checkJoystickInput() </pre>

Motor
<pre> +highestValue: int16_t +lowestValue: int16_t +PWMValue: int16_t +Motor(lowestValue: int16_t, highestValue: int16_t) </pre>

mehr Daten angegeben werden. Die ersten beiden Parameter legen die Pins fest an welchen die jeweilige Achse angeschlossen ist. Die nächsten zwei sind wieder der niedrige und Höchste Wert. Zum Schluss benötigt der Joystick noch einen Wert, der einen Abstand zum Mittelpunkt von der Aktivierung der Steuerung ausschließt das sogenannte spaceing. Es wird benötigt, da der Joystick selbst in der Mittelstellung werte sendet, die über bzw unter 0 liegen.

Steuerung
-joystick: Joystick
-left: Motor
-right: Motor
-xValue: int16_t
-yValue: int16_t
+Steuerung(joystick: Joystick, left: Motor, right: Motor)
+updateValues(): void
-mapReadingsToMatchPWMValues(): void
-applyPWMValuesDependingOnReadings(): void

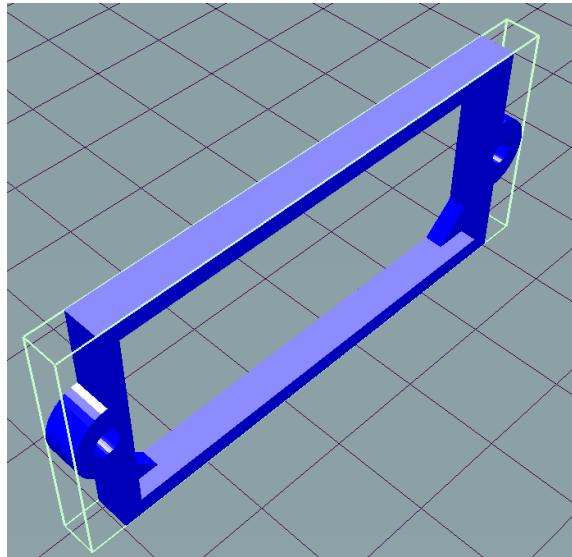
11 Arduino Libaries

11.1 OneWire

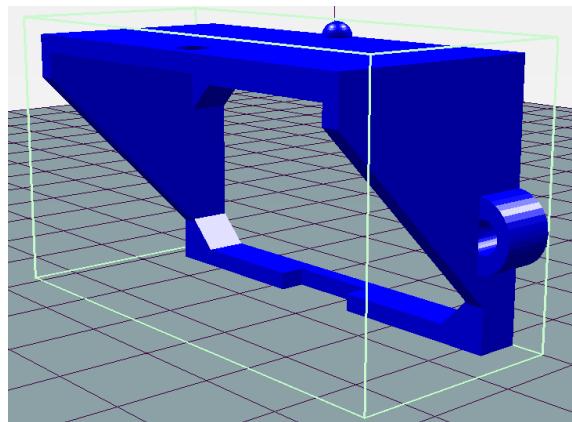
OneWire ist eine serielle Schnittstelle, die nur mit einer Datenleitung auskommt. Jedes angeschlossene Gerät hat eine eigene 64bit ROM Adresse, die man gezielt ansprechen kann. Alternativ, kann man Befehle an alle Geräte senden indem man den Skip ROM-Command (0xCC) benutzt. Anschließend kommt der eigentliche Befehl mit write(), der ausgeführt werden soll. Danach werden die Daten auf ein Scratchpad-Speicher geschrieben von wo sie per read() gelesen werden können.

12 Gedruckte Bauteile

Alle gedruckten Bauteile haben wir selbst entworfen. einsehbar sind diese im Repository als STL oder aber auch unter den öffentlichen Fusion 360 Links.



(a) Rahmen [?]



(b) Winkel [?]

Abbildung 9: HC-SR04 Halterung

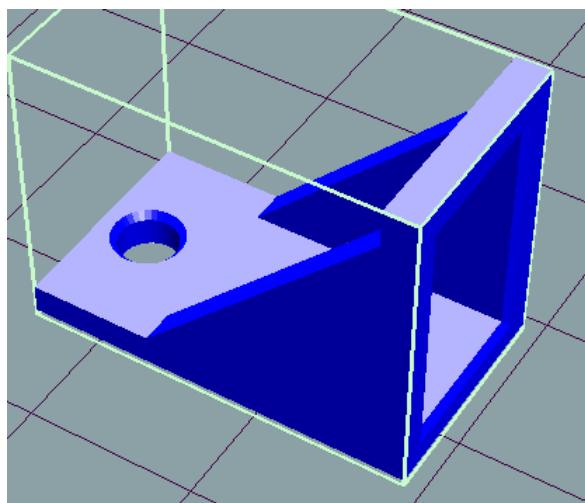


Abbildung 10: Schalterhalterung [?]

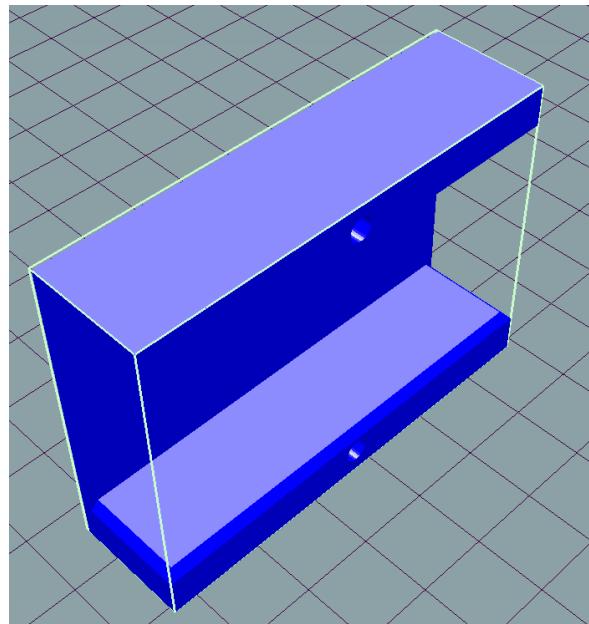


Abbildung 11: Akkuhalterung [?]

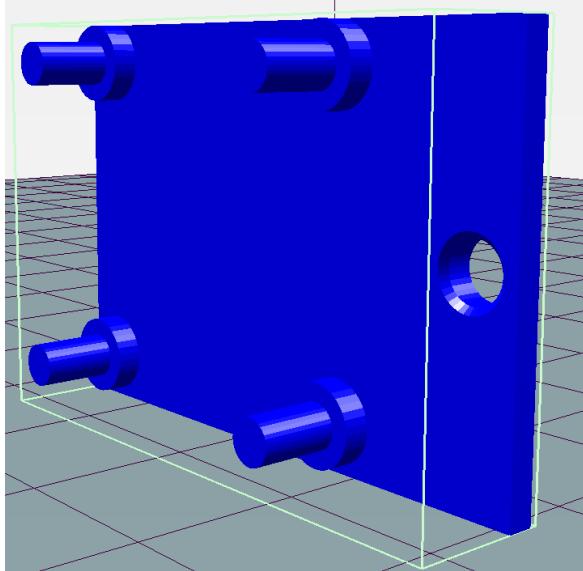


Abbildung 12: Joystick Halterung [?]

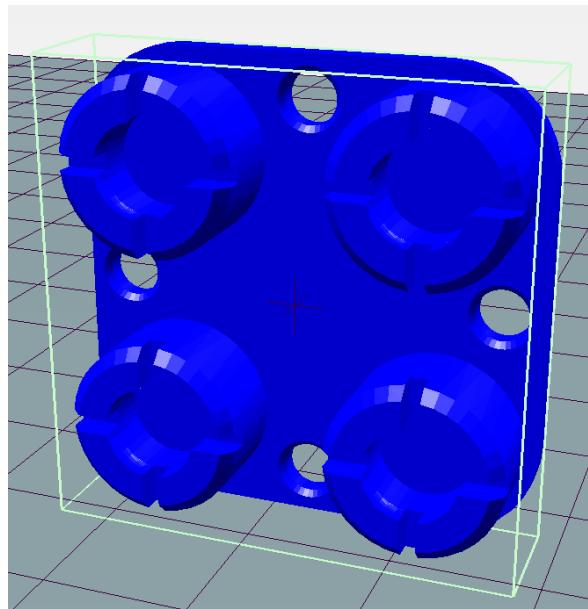


Abbildung 13: Kugelrad [?]

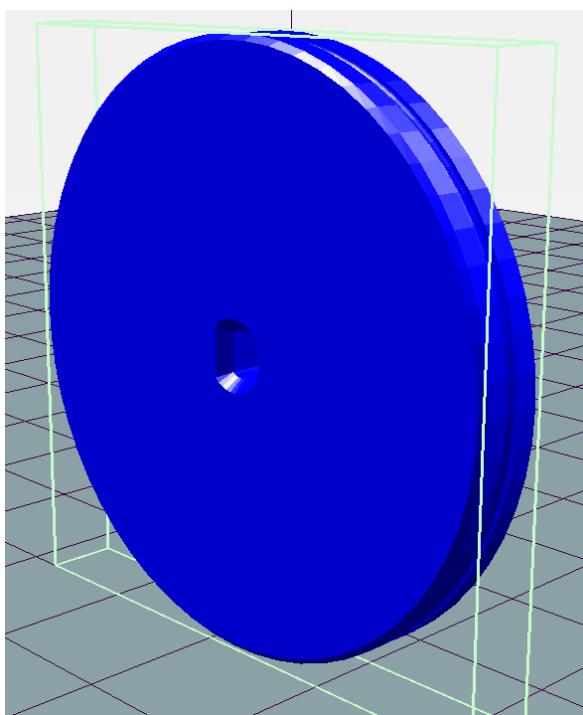


Abbildung 14: Räder [?]

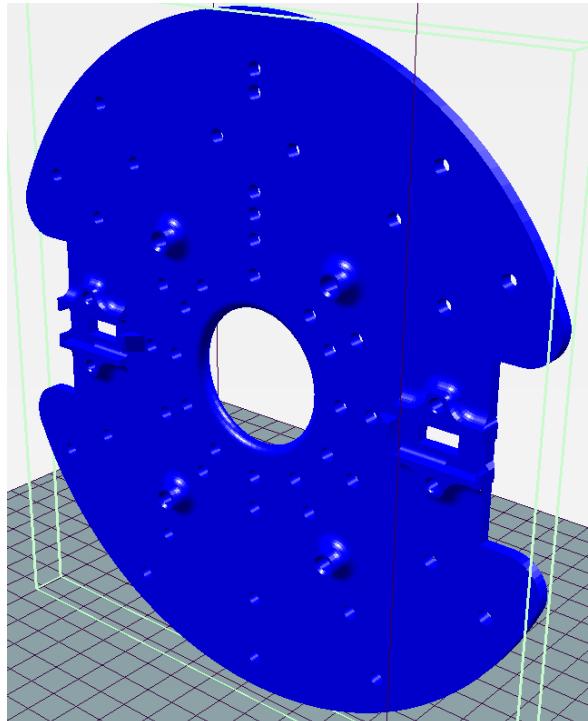


Abbildung 15: Roboter Plattform [?]

13 Motorsteuerung

13.1 Funktionsweise eines PWM

Die Pulsbreitenmodulation oder engl. Pulse Width Modulation (PWM) dient zur Variation der ausgegebenen Effektivspannung eines Ausgangs. Umgesetzt wird ein PWM überlicherweise mit einem Timer, welcher zählt bis er überläuft und wieder von vorn beginnt, sowie einem Komparator als auch einem Vergleichswert, welcher vorgibt, wie lang der Impuls ausgegeben werden soll. Ist der Wert vom Timer und dem PWM Sollwert identisch, so wird das Ausgangssignal abgeschalten. Durch Variation der Impulsbreite im Verhältnis zur Periodendauer ergibt sich ein Effektivwert $< =$ der max. Signalspannung. Das Tastverhältnis hängt von der Auflösung des Timers ab. Im Falle unseres Projektes arbeitet Timer1 als 8-Bit Timer. Somit ergibt sich ein Tastverhältnis von 1:256. Die Grafik aus dem Atmega328p Datenblatt von Seite 133 illustriert dies noch einmal.

Figure 16-7. Fast PWM Mode, Timing Diagram

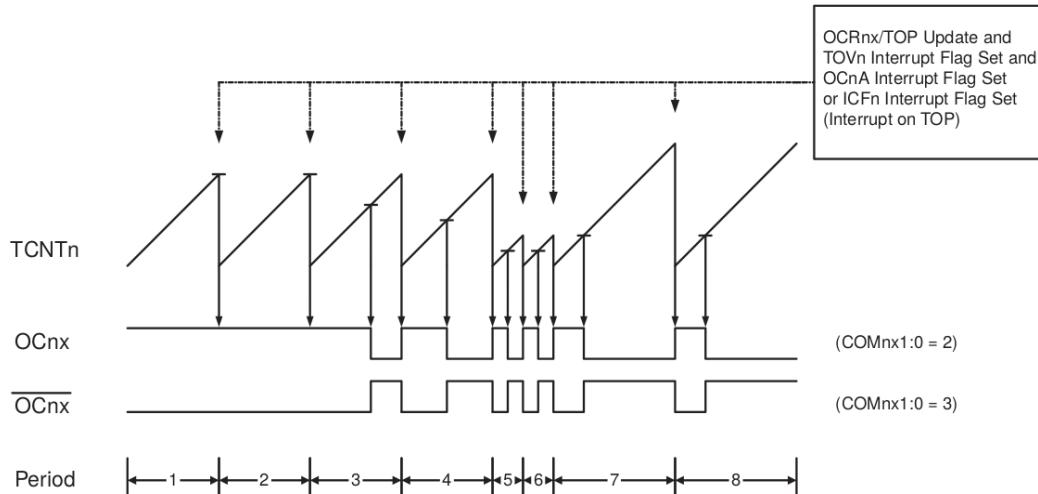


Abbildung 16: Funktion des PWM Moduls im Atmega328p
[?]

13.2 Implementierung

Die Steuerung der Motoren wurde über den IC MX1508 realisiert. Dieser ist sehr günstig als Modul für etwa 1€ erhältlich. Mit seinen Spezifikationen von max. 10V Versorgungsspannung und maximal 1,5A war er perfekt für dieses Projekt geeignet. Leider konnten wir kein Datenblatt zu diesem IC finden, die Angaben stammen lediglich aus den Beschreibungen der Händler die dieses Modul verkaufen. Es scheint sich um einen Treiber mit H-Brücke bzw. doppelter H-Brücke zu handeln, er verfügt über jeweils zwei Kanäle mit je zwei Eingängen. Je nach Polung der Eingänge kann ein Motor vorwärts oder rückwärts drehen. Um die Geschwindigkeit der Motoren regulieren zu können schrieben wir eine Bibliothek mit dem Namen L298N (Achtung, nicht mit der gleichnamigen Bibliothek im Librarymanager der Arduino IDE zu verwechseln!). Der Name ist angelehnt an den gleichnamigen IC, welcher dem MX1508 in seiner Ansteuerung ähnelt. Die Bibliothek nutzt Timer1 als 8-Bit Timer für die im Atmega328p integrierten PWM-Module OC1A

und OC1B. Ein PWM Pin ist mit jeweils einem der beiden Eingangspins eines Kanals des MX1508 verbunden. Der jeweils andere Pin muss kein PWM Pin sein, muss aber je nach Drehrichtung korrekt gepolt sein. Für diese nicht-PWM-Pins haben wir die Portpins B0 (Kanal A) und D7 (Kanal B) verwendet.

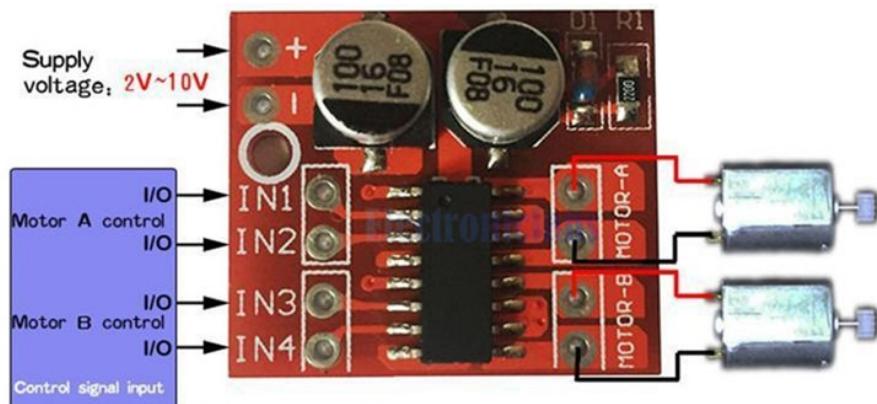
Im folgenden Code ist exemplarisch zu sehen, wie das setzen eines PWM Wertes für Kanal A abläuft. Valide PWM-Werte sind -255 bis +255.

```
MX1508

L298N::setPWM_A(int16_t pwmA) {
    if(pwmA < 0) {
        PWMA = 0xFF + pwmA;
        PORT_A |= PIN_A;
    } else {
        if(pwmA == 0) {
            PWMA = 0;
            PORT_A &= ~PIN_A;
        } else {
            PWMA = pwmA;
            PORT_A &= ~PIN_A;
        }
    }
}
```

Zunächst wird geprüft, ob der PWM-Wert „pwmA“ negativ ist. Davon hängt die Polung von „PIN_A“ ab und ob der PWM-Wert negiert werden muss.

Anschließend werden PWM-Wert und der Zustand des nicht-PWM-Pins entsprechend gesetzt.



Two independent drive DC motor;

INx control signal input, signal voltage range 1.8–7V;

IN1, IN2 control the motor A; IN2, IN3 control motor B;

Abbildung 17: MX1508 Anschluss [?]

14 Thermosensor

Wie bereits erwähnt benutzen wir einen Dallas DS18b20 1-Wire Digital Thermometer, um die Motortemperatur zu messen. Der DS18b20 speichert die Temperatur als einen 9bit - 12bit Wert und kann Temperaturen im Bereich von -55°C bis 125°C messen. Für unsere Bedürfnisse reicht es wenn wir in 1-er Schritten messen. Der Sensor kommt aber auch mit kleineren Inkrementen zu recht. Der Sensor benötigt die OneWire-Library um angesprochen zu werden.

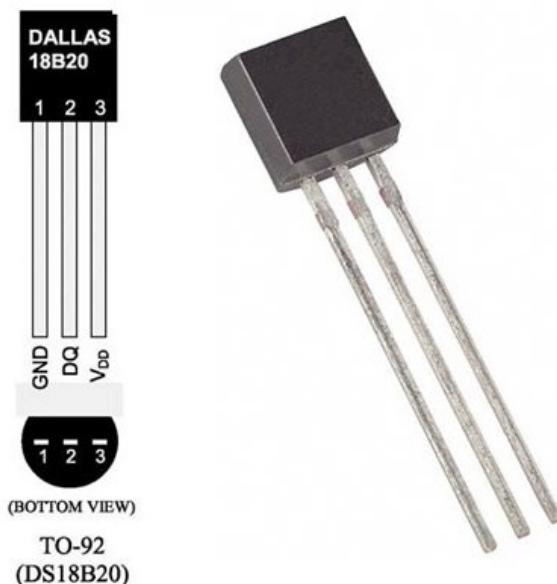


Abbildung 18: Der benutzte Sensor
1) Erdung 2) Datenleitung 3) Spannung

Eine Messung wird durchgeführt indem der Sensor Takte, innerhalb eines Zeitfensters zählt. Der Zähler startet bei einem Wert, der -55°C darstellt. Erreicht der Zähler 0 schneller als das vorgegebene Zeitfenster wird das Temperaturregister inkrementiert. Der Zähler wird dann auf einen vom Slope Accumulator Circuitry abhängigen wert gestellt und wieder runtergezählt. Dies wiederholt sich solange bis das festgelegte Zeitfenster abgelaufen ist.

Es gibt verschiedene Wege den Code für den Sensor zu implementieren. Der einfachste Weg wäre die DallasTemperature-Library zu benutzen.

DS18B20

```
#include <OneWire.h>
#include <DallasTemperature.h>

OneWire oneWire(x);
DallasTemperatures sensors(&oneWire);

void setup(){
    Serial.begin(9600);
    // Die Library starten
    sensors.begin();
}

void loop(){
    // Temperaturmessung anfordern
    sensors.requestTemperatures();
    // Temperatur anzeigen
    Serial.print(sensors.getTempCByIndex(y));

}
```

OneWire wird auf Pin 4 initialisiert, dann wird OneWire an sensors() übergeben. Damit weiß die Library wo der Sensor angeschlossen ist. Mit sensors.begin() startet man die Library. Eine Messung fordert man mit requestTemperatures() an und mit getTempCByIndex(y) kann man einen bestimmten Sensor, wenn man mehrere hat, nach der Temperatur in Grad Celsius abfragen. Mit der Library ist das ganze also sehr einfach.

Das war auch der erste Ansatz. Der Clue an der Sache ist jedoch, dass die Prozedur auf die Weise etwa 600ms dauert und das den Roboter viel zu lange blockieren würde, da er in der Zeit keine weiteren Befehle verarbeiten kann. Also wurde diese Idee mitsamt der Library gestrichen und nach einer effizienteren Alternative gesucht.

Für unsere Implementierung haben wir uns an <https://www.scargill.net/reading-dallas-ds18b20-chips-quickly/> orientiert. Dieses Beispiel ermöglicht es eine Messung in nur 5ms durchzuführen. Es wurden jedoch einige Änderungen in der Umrechnung der Temperatur gemacht. Da ein Einsatz in negativen Temperaturen nicht vorgesehen ist, machen wir eine viel simplere Umrechnung als die im Blog.

DS18B20

```
#include <OneWire.h>

int16_t dallas(int x){
    // Initialisierung
    OneWire ds(x);
    byte data[2];
    int16_t result = 0;
    // Auf Scratchpad schreiben
    ds.reset();
    ds.write(0xCC);
    ds.write(0xBE);
    // Vom Scratchpad lesen
    for(int i = 0; i < 2; i++){
        data[i] = ds.read();
    }
    result = (data[1]<<8)|data[0];
    result = result/16;
    // Messen
    ds.reset();
    ds.write(0xCC);
    ds.write(0x44, 1);
    return result;
}
```

Wir initialisieren im Beispiel OneWire auf einen beliebigen Pin und setzen den Sensor mit reset() zurück. Die ROM Adresse wird dabei mit write(0xCC) übersprungen, da wir nur einen Temperatursensor haben und es somit reicht, dass wir alle Geräte auf dieser OneWire-Verbindung ansprechen. Mit write(0xBE) wird dem Sensor gesagt, dass er die Werte auf dem Scratchpad speichern soll, die wir dann mit read() in einen Array einlesen. Die Werte im Array führen wir dann zusammen und teilen sie durch 16. Die 16 ergibt sich aus einer Tabelle aus dem Datenblatt. Bei genauerer Betrachtung stellt man fest, dass die Temperatur Werte über 0 einfach nur die Binärwerte geteilt durch 16 sind. Da es für unsere Bedürfnisse genügt, dass wir nur die positiven Werte messen, belassen wir es bei dieser einfachen Umrechnung. Dann wird der Chip wieder resetet, die Adressierung übersprungen und mit write(0x44) eine Messung durchgeführt. Es empfiehlt sich in der setup() Funktion die dallas() Funktion aufzurufen, damit er in der loop() den richtigen Wert erhält.

Um den Roboter so wenig wie möglich zu belasten und somit eine schnelle Reaktionszeit zu ermöglichen, wird die Temperaturmessung nur alle 100ms durchgeführt und nur jede Sekunde durch den Funkcontroller abgefragt. Die Abfrage wird, aus Übersichtsgründen, abseits der Steuerung gemacht.

15 Ultraschallsensor

Zur Messung der Distanz zu einem Objekt vor dem Robotor benutzen wir einen HC-SR04 Ultraschallsensor. Dieser kann einen Abstand bis ca 450cm messen. Das ganze funktioniert, durch ein Signal, das der Sensor von sich gibt und die Zeit zählt bis er ein Echo zurückbekommt.



Abbildung 19: Der benutzte Sensor
1) Spannung 2) Trigger Pin 3) Echo Pin 4) Erdung

Die Implementation ist relativ einfach. Wir setzen den Trigger Pin als Ausgang und den Echo Pin als Eingang. Wir senden mit dem Trigger Pin, dann ein Signal, welches vom Echo Pin empfangen wird und rechnen dann die Zeit die dafür gebraucht wurde in cm um.

Zum Empfangen des Echo nutzen wir einen Pinchange Interrupt. Diese ISR wird ausgeführt, wenn ein Signal an einen bestimmten Pin kommt. Dazu aktivieren wir die nötigen PCINTs im PCIRC Register und PCMSK2 das bit für unseren Pin.

In der ISR messen wir die Zeit bis ein Echo ankommt und teilen es, wie im Datenblatt beschrieben, durch 58 um die Distanz in Centimetern zu bekommen.

HC-SR04

```
int trig;
int echo;
//ISR for PCINT2
ISR(PCINT2_vect) {
    if((PIND & 64) > 0) {
        pulseStart = micros();
    } else {
        startNewMeasurement = true;
        pulseLength = micros() - pulseStart;
        newResult = true;
    }
    PCIFR = 0;
}

void measureDistance(){
    digitalWrite(trig, HIGH);
    // ... wait for 10 µs ...
    delayMicroseconds(10);
    // ... put the trigger down ...
    digitalWrite(trig, LOW);
    //Serial.println("messe... ");
    startNewMeasurement = false;
}
```

In einer setup() Funktion müssen wir den Trigger Pin als Ausgang und den Echo Pin als Eingang definieren und ein initiales Signal mit dem trigger Pin senden. Im Anschluss aktivieren wir die nötigen Bits im PCICR und PCMSK2 Register um den Pinchange Interrupt für unseren Echo Pin zu aktivieren.

Die measureDistance() Funktion, leitet die Messung ein. Sie gibt aus dem Trigger mit digitalWrite(trig, HIGH) Pin ein Signal an den Sensor, der dann ein Ultraschallsignal von sich für 10 Mikrosekunden abgibt, welches mit digitalWrite(trig, LOW) wieder abgestellt wird. Die ISR führt dann die eigentliche Messung durch. Da die ISR bei Anfang und bei Ende des Signals aufgerufen wird, haben wir eine Fallunterscheidung eingebaut die zu Beginn des Signals die Startzeit speichert und am Ende des Signals, daraus die Pulslänge berechnet, mit der wir die Distanz bestimmen können. Dazu müssen wir die Pulslänge, wie im bereits erwähnt, durch 58 teilen.

Mit der Distanz die wir nun haben können wir den Roboter stoppen lassen, wenn er zu nah an einen Gegenstand ranfährt. Ihn aber komplett zu stoppen wäre unsinnig, da er sich überhaupt nicht mehr bewegen könnte. Also gucken wir nach den Motorgeschwindigkeiten und setzen diese nur auf 0 wenn sie mit einer Vorwärtsbewegung übereinstimmen. Somit kann sich unser Roboter immer noch auf der Stelle drehen oder rückwärts fahren um das Hindernis zu umgehen.

16 Beispiele für Spezielle LaTeX Strukturen



| **Info:** benutze den Info block um wichtige informationen hervorzuheben.

Algorithm 1: FastTwoSum

Input: (a, b) , two floating-point numbers

Result: (c, d) , such that $a + b = c + d$

```
if  $|b| > |a|$  then
    | exchange  $a$  and  $b$  ;
end
 $c \leftarrow a + b$  ;
 $z \leftarrow c - a$  ;
 $d \leftarrow b - z$  ;
return  $(c, d)$  ;
```

```
hello.py
```

```
#!/usr/bin/python

import sys
sys.stdout.write("HelloWorld!\n")
```

Command Line

```
$ chmod +x hello.py
$ ./hello.py
```

```
Hello World!S
```



| **Notice:** Warungen könnten auch nützlich sein, immerhin braucht der RF24 3.3V und nicht 5V

