Inhaltsverzeichnis

[2 Softwareentwicklung 1](#_heading=h.3rdcrjn)

[2.1 Programmierung und Debugging 1](#_heading=h.1ksv4uv)

[2.1.1 ISP über SPI 1](#_heading=h.z337ya)

[2.1.2 Debugging over Serial 2](#_heading=h.3j2qqm3)

[2.2 Versionsverwaltung und Style – Guide 3](#_heading=h.1y810tw)

[2.2.1 Standard - Header 3](#_heading=h.4i7ojhp)

[2.2.2 Variablen- und Funktionsnamen 3](#_heading=h.2xcytpi)

[2.3 Software-Design Controller 4](#_heading=h.1ci93xb)

[2.3.1 Ansteuerung der Status-LED 4](#_heading=h.x7zqdnlgs943)

[2.3.2 Auslesen der Elektronik 4](#_heading=h.3whwml4)

[2.4 Software-Design Roboter 4](#_heading=h.2bn6wsx)

[2.4.1 Ansteuerung der Status-LED 4](#_heading=h.ijzubndyngy1)

[2.4.2 Berechnung der Motoransteuerung 5](#_heading=h.qsh70q)

[2.4.3 Ansteuerung des Motorshields 6](#_heading=h.3as4poj)

[2.4.4 Einstellen des Watchdog-Timers 7](#_heading=h.y06j5djhvze3)

[2.4.5 Debug – Mode 7](#_heading=h.1pxezwc)

[2.5 Networking 8](#_heading=h.dd4ixy9f8ebd)

[2.5.1 Initialisierung des Moduls 8](#_heading=h.vfx5ax9r30wy)

[2.5.2 Standard - Datenpakete 9](#_heading=h.49x2ik5)

# Softwareentwicklung

In diesem Kapitel wird die Softwareentwicklung für dieses Projekt erklärt. Diese reicht von: „Wieladen wir den Code hoch?“, „Wie dokumentieren wir Änderungen an den Codes?“, „Wie benennen wir Variablen und Funktionen“ bis hin zu „Wie funktioniert der Code, was soll er tun?“

Alle Codes, die in diesem Kapitel verwendet werden, sind an C angelehnte „Pseudocodes“. Diese sind nicht in einer Programmiersprache verfasst und sollen nur die Funktionsweise des tatsächlichen Codes erklären. Dabei können gewisse Funktionsaufrufe, Variablendeklarationen, Typecasting und andere Kleinigkeiten, die der Erklärung der eigentlichen Funktionsweise des Codes im Weg stehen könnten, ausgelassen werden. In Fußnoten wird teilweise auf die echte Implementierung eingegangen.

Im Anhang ist allerdings eine gut kommentierte Version der tatsächlich verwendeten Codes zu finden.

## Programmierung und Debugging

In diesem Kapitel wird der Style Guide definiert und die Weise, wie wir den Code hochladen und Textnachrichten vom µC bekommen, erklärt.

Der Roboter und Controller verwenden jeweils den atMega2560 und atMega328p µC. Das sind beides µC, die für verschiedene Arduino Boards verwendet werden, weshalb für die Softwareentwicklung die gewöhnliche Arduino Umgebung verwendet werden kann. Ein gewöhnliches Arduino-Board wird mit einem USB – Kabel mit dem Computer verbunden. Ein zweiter µC ermöglicht „Serial over USB“, eine (virtuelle) serielle Verbindung über die USB – Schnittstelle. Auf dem Arduino läuft ein Bootloader, ein kleines Programm welches beim Hochfahren des Arduinos startet, und das Programmieren über die Serielle Verbindung ermöglicht. Über diese können auch Debug – Nachrichten an den Computer übertragen werden. Wir haben allerdings weder den zweiten µC noch den Bootloader, da wir die µC direkt vom Händler kaufen. Deshalb mussten für Programmierung und Debugging andere Methoden verwendet werden, die folgend näher beschrieben werden.

### ISP über SPI

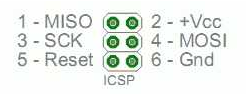
Um Funktionalitäten von Arduino-Boards verwenden zu können schalten wir für Programmierung und Debugging einen zweiten Arduino dazwischen, wie folgt:

A picture containing text, screenshot, font, software

Description automatically generated

*Abbildung 1: Blockschaltbild neue Programmierung*

Programmiert kann der µC über den SPI – Bus werden. Für diesen Zweck haben Arduinos einen veränderten SPI - Bus am ICSP[[1]](#footnote-0) – Header ausgeführt:



*Abbildung 2: ICSP – Header der Arduinos*

Verändert, weil statt dem üblichen „Slave Select“, der „Reset“ Pin ausgeführt ist.

Das Programmieren erfolgt wie folgt:

1. Zunächst muss das Programmierer-Board selbst den Code „Programmer.ino“ hochgeladen bekommen.
2. Danach kann man die von uns erstellten ICSP – Stecker miteinander verbinden. Der Stecker mit zusätzlichem Jumper-Draht wird auf die ICSP - Schnittstelle des Programmierer-Boards angesteckt, das zusätzliche Kabel auf Pin 10. Dieser ist mit Pin 5: „reset“ verbunden. Die andere Seite wird auf das ICSP – Interface des zu Programmierenden Boards angesteckt. Die schwarz angemalte Seite zeigt nach oben, ist also näher an Pin 1 der Schnittstelle.
3. Nun kann das Programm hochgeladen werden. Als Board wird „Arduino Mega2560“ ausgewählt, als Programmer „Arduino as ISP“ und „Upload with Programmer“ betätigt.

### Debugging over Serial

Für das Debugging müssen einfach nur die seriellen Schnitstellen von µC und Arduino verbunden werden. Der Computer bekommt dann die Nachrichten des µC, als würden sie vom Arduino kommen.

## Versionsverwaltung und Style – Guide

Hier wird auf die Verwaltung der Codes eingegangen. Einerseits auf Versionierung und Change-Logs, andererseits auf einige Standards was Variablennamen, Funktionsnamen etc. betrifft.

### Standard - Header

Jedes Programm bekommt einen Header, in folgender Form:

/\*\*

\* Version: Projektversionsnummer.Programmversionsnummer

\* Author: Author

\*

\* Created: Datum der Erstellung

\* Last modified: Datum der letzten Änderung

\*

\* Desc: Kurzbeschreibung des Programms

\*/

Die Projekt- und Programmversionsnummern sind laufende, bei 0 beginnende Kennzahlen. Ändert man ein Programm, wird die Programmversionsnummer um eins erhöht. Verlangt eine Änderung auch Anpassung der anderen Programme, wird stadtessen die Programmversionsnummer rückgesetzt und die Projektversionsnummer um eins erhöht.

Verschiedene Codes sind also nur dann kompatibel, wenn die Projektversionsnummer übereinstimmt.

Zusätzlich wird ein Dokument geführt, welches alle Veränderung einer neuen Version beschreibt.

### Variablen- und Funktionsnamen

(XXX im jetztigen Code noch nicht so praktiziert)

Alle Variablen und Funktionen sind Englisch[[2]](#footnote-1) und werden im „Camel case“ benannt. Statt einem Abstand wird also das nächste Wort großgeschrieben (z.B. doDebug). Alle Pinnummern beginnen mit p\_. Einige Wörter werden so oft verwendet, dass sie mit einzelnen Buchstaben abgekürzt werden dürfen:

l … left, r … right

b … backwards, f … forwards

Die Reihenfolge ist Left / Right dann Backwards / Forwards. Der Pin, mit dem der Rechte Motor nach vorne angesteuert werden kann heißt somit:

p\_rf

In den Pseudo-Codes der Diplomarbeit sind diese Wörter der Klarheit halber oft ausgeschrieben.

## Software-Design Controller

In diesem Kapitel wird der Code des Controllers erklärt. Dieser soll die elektronische Komponente auslesen, die Daten kurz verarbeiten und anpassen, um diese dan an den Roboter weiterzusenden. Um denn Status zu erkennen soll eine gruene LED angesteuert werden.

### Ansteuerung der Status-LED

Die ersten Zeilen im Code des uC schalten die LED ein. Das zeig 1) das der richtige Code hochgeladen ist und 2) das der uC mit Strom versorgt ist. Danach wird das Funkmodul gestartet und es wird versucht, ein Paket an den Roboter zu senden. Wird diese angenommen, blinkt die LED kurz auf und leuchtet dann wieder konstant. Somit kann auf einem Blick erkannt werden, ob eine Verbindung hergestellt wurde.

### Auslesen der Elektronik

Der Controller muss die verschiedenen elektrischen Komponente, also den JoyStick und die Schultertasten auslesen und an den Roboter weiterschicken. Unnabhaengig von der Hardware wurden die Wertebereiche in Kapitel 1.5.2. definiert. Die Schultertasten entsprechen diesem Standard bereits, der Wertebereich des Joysticks muss jedoch darauf angepasst werden. (XXX richtiger code in lukas laptop)

loop() {

array[0] = analogRead(joyX)

array[1] = analogRead(joyY)

array[2] = analogRead(l\_schulter)

array[3] = analogRead(r\_schulter)

radio.write(array)

}

“radio.write()” verwendet das Funkmodul, um diese Daten an den Roboter zu senden. Die Ansteuerung des Funkmoduls wird im Kapitel 1.5.1. genauer beschrieben.

## Software-Design Roboter

In diesem Kapitel wird der Code des Roboters erklärt. Dieser soll Daten des Controllers empfangen, und mit diesen die Motoren ansteuern. Er soll außerdem in einem Intervall von 250ms Daten von verschiedenen Sensoren, beschrieben in Kapitel XXX, an einen Raspberry PI weiterleiten. Um den Status zu erkenn soll eine RGB-LED angesteuert werden.

### Ansteuerung der Status-LED

Die ersten Zeilen im Code des uC schalten die LED auf Rot. Das zeigt, dass 1) der richtige Code hochgeladen ist und 2) das der Controller mit Strom versorgt ist. Danach wird das Funkmodul gestartet und es wird versucht, ein Paket vom Controller einzulesen. Ist das geschehen,, blinkt die LED kurz auf und leuchtet dann konstant gruen. Somit kann auf einem Blick erkannt werden, ob eine Verbindung hergestellt wurde.

### Berechnung der Motoransteuerung

Als Input – Größen bekommt der Roboter die Position des Joysticks und der zwei Schultertasten. Die Ansteuerung der Motoren wird anhand dieser berechnet. Die Schultertaste bestimmt die Geschwindigkeit der Motoren, das Verhältnis zwischen den Motoren bleibt aber unverändert. Je nachdem ob die rechte oder linke Schultertaste gedrückt wird, fährt der Roboter nach vorne oder hinten. Die Schultertaste, die stärker gedrückt wird, „gewinnt“. Durch die Wrapper – Funktion, auf die später noch eingegangen wird, laufen Motoren durch negative Werte rückwärts. Die linke Schultertaste gibt also dieselbe Magnitude, aber ein negatives Vorzeichen.

Damit kommen wir auf folgenden Code:

if (r\_schulter > l\_schulter)

speed = r\_schulter

else

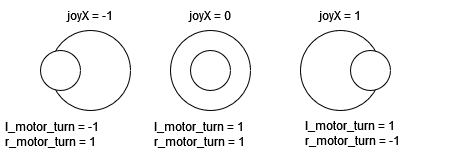
speed = -l\_schulter

l\_motor = l\_motor\_turn \* speed

r\_motor = r\_motor\_turn \* speed

l und r\_motor\_turn geben das Verhältnis zwischen linkem und rechtem Motor an. Sie sind also Variablen, die von -1 bis 1 gehen und sich aus der x-Achse des Joycons berechnen lassen.

Nimmt man an das „joyX“ eine floating-point-Zahl ist, die von -1 (ganz links) bis +1 (ganz rechts) geht[[3]](#footnote-2), kommen wir auf folgendes Diagramm:



Ist der Stick auf ganz links positioniert, soll der Roboter sich im Stand nach links drehen. Deshalb soll sich die linke Kette Rückwärts und die rechte Vorwärts bewegen. Die Anderen Positionen können mit einer ähnlichen Argumentation verstanden werden. Zwischen diesen Extrempositionen sollen die Variablen linear alle Zwischenwerte einnehmen.

Mittels map – Befehl[[4]](#footnote-3) werden diese Wertebereiche aneinander angepasst.

if joyX < 0 {

l\_motor\_turn = map(joyX, -1, 0, -1, 1)

r\_motor\_turn = 1

}

else if (joyX >= 0) {

l\_motor\_turn = 1

r\_motor\_turn = map(joyX, 0, 1, 1, -1)

}

l\_motor = l\_motor\_turn \* speed

r\_motor = r\_motor\_turn \* speed

Steuert dieser Code die Motoren direkt an und falls der Fahrer nicht vorsichtig ist und schnelle Richtungsänderungen vornimmt, kann das die Motorbruecke ueberlasten (XXX Erklärung im Hardware teil?). Deshalb sollen diese Änderungen nur langsam übernommen werden. Der Controller ändert nur die „\_soll Werte“, tatsächlich an das Motorshield weitergegeben werden die „\_ist Werte“. Ist ein Soll-Wert und Ist-Wert nicht gleich, wird der Ist-Wert einen Schritt Richtung Soll-Wert verändert.

l\_motor\_soll = l\_motor\_turn \* speed

if (l\_motor\_ist > l\_motor\_soll)

l\_motor\_ist -= 1

else if (l\_motor\_ist < l\_motor\_soll)

l\_motor\_ist += 1

Im obigen Code wurde also einfach l\_motor mit l\_ motor\_soll ersetzt.

Dieser Code gilt genauso für den rechten Motor.

### Ansteuerung des Motorshields

Wie die Motorbrücke funktioniert wurde im Kapitel XXX bereits genauer beschrieben, wird hier aber noch einmal zusammengefasst (XXX notewendig?). Er hat für beide Motoren zwei Pins die ein PWM – Signal akzeptieren; eins, falls sich der Motor nach vorne drehen soll, eins fall sich der Motor rückwärts drehen soll. Diese Ansteuerung wird durch eine “Wrapper” - Funktion implementiert. Sie soll fuer linken und rechten Motor Werte von -255 bis 255 akzeptieren. Negative Werte stehen hier für einen sich rückwärts-drehenden Motor. Wie man im vorherigen Kapitel erkennen konnte, macht das die Mathematik ein wenig bequemer.

In der Funktion muss dann nur abgeprüft werden, ob der Wert negativ oder positiv ist, und je nachdem auf den Vorwärts- oder Rückwärtspin der absolute Wert[[5]](#footnote-4) des Arguments als PWM – Signal angelegt werden.

Wie im Kapitel XXX zu sehen ist der linke Motor des Roboters verkehrt herum eingebaut. Das kompensieren wir Softwaretechnisch, indem wir den linken Wert negieren und somit seine Fahrtrichtung umkehren.

drive(left, right) {

left = -left

if (left > 0) {

analogWrite(p\_left\_forward, |left|)

analogWrite(p\_left\_backward, 0)

}

else {

analogWrite(p\_left\_forward, 0)

analogWrite(p\_left\_backward, |left|)

}

}

Auch dieser Code gilt, abseits der Negation des Wertes, genauso für den rechten Motor.

### Einstellen des Watchdog-Timers

Der Watchdogtimer soll eingerichtet werden, um einen periodischen Interrupt alle 150ms zu erzeugen. In der dazugehoerigen Interrupt Service Routine werden Sensoren eingelesen und die eingelesenen Daten an den Server weitergeleitet. Wie die Sensoren eingelesen werden wird bereits in Kapitel XXX beschrieben, wie das Funkmodul angesteurt wird.

### Debug – Mode

Um unser Projekt debuggen zu können wurde ein Debug – Mode implementiert. Der Roboter bekommt die Daten des Controllers, steuert aber nicht die Motoren an sondern gibt die vom Controller bekommenen Daten und die sich ausgerechneten Daten an den Serial Monitor aus. Dieser besteht also nur aus einer if – Anweisung und einer Reihe an „Serial.print“ befehlen. Vor dem Wert wird der Name des Wertes und sein Wertebereich angegeben.

if (doDebug) {

Serial.print("X: ")

Serial.println(inputs[0])

Serial.print("Y: ")

Serial.println(inputs[1])

…

Serial.println("Kurve [-1..1]: ");

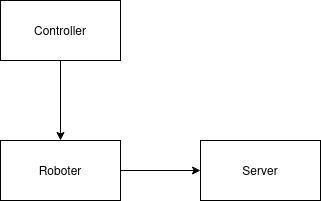
Serial.println(turn);

Serial.println();

}

## Networking

Es gibt drei verschiedene Komponente die ueber das im Kapitel XXX beschriebene Funkmodel miteinander kommunizieren. Das daraus resultierende Netzwerk kann durch folgendes Diagramm veranschaulicht werden.



XXX reference

Der Controller schickt dem Roboter die (verarbeiteten) Zustaende des Joysticks und der Schultertasten. Der Roboter schickt dem Server im 250ms Takt Daten seiner verschiedenen Sensoren.

### Initialisierung des Moduls

Software-seitig verwenden wir fuer die Ansteuerung des Funkmoduls die dazugehoerige Library “nRF24L01.h”. Einige der anfaenglichen Befehle zur Initialisierung des Moduls bleiben bei allen Komponenten gleich.

RF24 radio(p\_CE, p\_CSN);

radio.begin();

Ein Teilnehmer, der Sender, definiert die Verbindung mit “radio.openWritingPipe(0, address)”, der Empfaenger mit “radio.openReadingPipe(address)”, wobei die Addresse natuerlich auf beiden Seiten uebereinstimmen muss. Mit jeweils “radio.stopListening()” und “radio.startListening()” wird die Verbindung dann tatsaechlich aufgebaut.

Erwuenscht ist jedoch keine Punkt-zu-Punkt Verbindung, sondern ein Netzwerk mit 3 Teilnehmern. Es gibt Libraries, die basierend auf dem rf24 Modul ein Netzwerk-Protokoll (OSI Lvl 3) implementieren. Die Ausmase unseres Netzwerks sind jedoch relativ bescheiden, weshalb das Netzwerk eigens realisiert wurde.

Controller und Server haben jeweils nur eine Writing und ListeningPipe offen, ihr Code muss nicht angepasst werden. Einzig der Roboter muss Daten von Controller empfangen und an den Server schicken. Die Library ermoeglicht es jeodoch nicht, eine Writing und eine Reading -Pipe gleichzeitig offen zu haben. Wollen wir also zwischen diesen wechseln muss eine geschlossen und die andere geoeffnet werden. In unserem Fall soll der Roboter Daten der Sensoren periodisch an den Server schicken und dazwischen Daten vom Controller bekommen.

radio.openReadingPipe(controllerAddress)

radio.startListening()

radio.read(inputData)

if ( 250ms passed )[[6]](#footnote-5) {

radio.stopListening()

radio.openWritingPipe(serverAddress)

radio.write(sensorData)

radio.startListening()

}

Für die Verbindung zwischen Funkmodul und Roboter verwenden wir das nRF24L01+ Modul. Die elektrischen Charakteristiken sind im Kapitel XXX beschrieben, hier wird auf die Software-seitige Ansteuerung eingegangen.

Das Modul wird angesteuert über die nRF24L01.h library.

(XXX Quelle: <https://nrf24.github.io/RF24/classRF24.html#a025fcbad6f062d18252485c1d6ba574f>)

### Standard - Datenpakete

Zwischen Controller und Roboter werden Packete in folgender Form gesendet:

(XXX l und r\_schulter entsprechen nicht 1.2.2.)

[l\_schulter, r\_schulter, joyX, joyY]

l\_schulter 0 (losgelassen) … 255 (vollstaendig gedrueckt)

r\_schulter 0 (losgelassen) … 255 (vollstaendig gedrueckt)

joyX -255 (ganz links) … 255 (ganz rechts)

joyY -255 (ganz unten) … 255 (ganz oben)

Zwischen Roboter und Server werden Pakete in folgender Form gesendet:

[sensor0, sensor1 , …]

(XXX wie viele und was fuer sensoren wir haben noch nicht klar)

1. ICSP (= „In Circuit Serial Programming“), ist eine andere Abkürzung für ISP (= „In System Programming“) und meint das Programmieren eines uC, während dieser im System eingebaut is. [↑](#footnote-ref-0)
2. Die einzigen Ausnahmen sind die \_soll und \_ist Suffixe. [↑](#footnote-ref-1)
3. In Echt ist diese nach Kapitel 1.3.2. eine Integer-Zahl zwischen -255 und 255. Diese wird zuerst auf einen Float gecasted und dann durch 256 dividiert, um auf den Wertebereich [-1; 1] zu kommen. [↑](#footnote-ref-2)
4. Der Map – Befehl von Arduino verwendet nur Integer Mathematik. Deshalb ist im Code ein eigener floating-point-map Befehl implementiert. l\_motor und r\_motor werden dann wieder zurück auf einen Integer – Wert gecasted, um als PWM – Signal an das Motorshield weitergeleiten werden zu können. (XXX ob man in integer mathe bleiben kann?) [↑](#footnote-ref-3)
5. Im tatsaechlichen Code wurde der Einfachheit halber, der positive Wert gelassen und der Negative negiert. Da wir sowieso nach dem Vorzeichen abfragen spart uns das sogar eine implizite if – bedingung in der abs() Funktion. [↑](#footnote-ref-4)
6. Tatsaechlich wird das periodische Senden der Daten mit Interrupts des Watchdogs-Timers implementiert und nicht z.B. einer Zeitmessung mit der millis() Funktion, wie dieser Code suggestiert. Auf die Implementierung des Watchdogtimers wird in Kapitel XXX genauer eingegangen. [↑](#footnote-ref-5)