



ulm university universität  
**u**lm

Fakultät für  
Ingenieurwissenschaften  
und Informatik  
Institut für Organisation und  
Management von Informations-  
systemen

November 2015

# Entwicklung eines Fußballroboters

Bachelorarbeit an der Universität Ulm

**Vorgelegt von:**

Alexander Ulbrich

**Gutachter:**

Prof. Dr. Stefan Wesner,  
Prof. Dr.-Ing. Klaus Dietmayer

**Betreuer:**

Dipl.-Inf. Lutz Schubert

Fassung 27. November 2015

© 2015 Alexander Ulbrich

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 2.0 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/2.0/de/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

Satz: PDF-LAT<sub>E</sub>X 2<sub>c</sub>

# **Abstract**

Inhalt dieser Arbeit ist der Entwicklungsprozess eines fernsteuerbaren Fußballroboters. Der Roboter wurde für das Fußballspielen mit einem Tischtennisball entworfen und seine Plastikteile mit einem 3D-Drucker gedruckt. Um eine sensorgestützte Steuerung des Roboters zu realisieren war eine Kamera vorgesehen.

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
<b>2 Aufgabenstellung</b>	<b>2</b>
2.1 Anforderungen an den Roboter . . . . .	2
<b>3 Planung</b>	<b>3</b>
3.1 Antriebssystem . . . . .	3
3.2 Funkverbindung . . . . .	4
<b>4 Komponenten</b>	<b>6</b>
4.1 Motoren und Räder . . . . .	6
4.2 Motortreiber . . . . .	9
4.3 Schuss . . . . .	9
4.4 Akku . . . . .	10
4.5 Kamera . . . . .	10
4.6 Mikrocontroller . . . . .	11
4.7 WLAN-Modul . . . . .	12
<b>5 Hardware</b>	<b>14</b>
5.1 3D Design . . . . .	14
5.1.1 Einschränkungen durch 3D-Druck . . . . .	14
5.1.2 Version 1 . . . . .	15
5.1.3 Version 2 . . . . .	16
5.1.4 Ladestation . . . . .	17
5.2 Board Design . . . . .	18
<b>6 Software</b>	<b>22</b>
6.1 Übersicht . . . . .	22
6.2 Treiber Implementierung . . . . .	24
6.2.1 Kameratreiber . . . . .	24
6.2.2 WLAN Treiber . . . . .	29
6.2.3 Motortreiber . . . . .	30
6.3 Video Encoder . . . . .	30
6.4 Netzwerkprotokoll . . . . .	30
6.5 Videotest . . . . .	31
<b>7 Diskussion</b>	<b>33</b>
<b>8 Anhang</b>	<b>34</b>

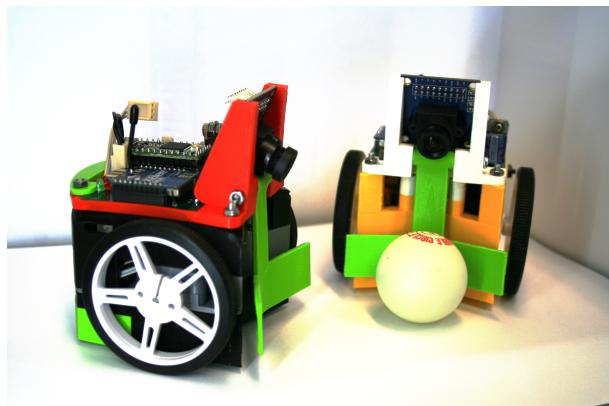
**Literaturverzeichnis**

**39**



# 1 Einleitung

Rapid Prototyping bedeutet übersetzt soviel wie schnelles Anfertigen von Prototypen. Es wird benötigt, um schon vor der eigentlichen Fertigung Prototypen zu testen und Fehler frühzeitig zu erkennen. Dadurch können die Entwicklungszeit und somit auch Entwicklungskosten gesenkt werden. Für die Realisierung der meisten Produkte ist die Entwicklung einer Mechanischen Hardware, einer Steuerungselektronik und deren Software nötig. Durch die Erfindung der gedruckten Leiterplatten ist dies in der Elektroindustrie schon weit fortgeschritten. Schon nach 48 Stunden kann für wenig Geld eine Platine gefertigt und geliefert werden. Durch die Verwendung von fertigen Elektronikmodulen mit Bibliotheken für Arduino oder Ähnlichem lässt sich auch die Softwareentwicklung auf das Wesentliche, die eigentliche Problemstellung, reduzieren. Bei der Herstellung von mechanischen Bauteilen ist das eine andere Sache. Für die Produktion von komplexen Teilen werden oft sehr teure CNC-Maschinen benötigt, deren Möglichkeiten oft stark eingeschränkt sind. Durch 3D-Druck kann auch dieser Schritt wesentlich günstiger, schneller und flexibler werden. So kann der in dieser Arbeit vorgestellte Roboter innerhalb von 24 Stunden vollständig gedruckt werden.



**Abbildung 1.1:** Fußballroboter mit Ball

Interessant werden diese neuen Technologien in der Robotik. Oft sind die Problemstellungen so speziell, dass alle existierenden Lösungen nicht ausreichend funktionieren. So auch in dieser Arbeit. Es gab weder fertige Fahrbasen, die den Anforderungen entsprachen, noch Steuerungssysteme, die die benötigte Größe hatten. So wurde der komplette Aufbau neu entworfen und die Elektronik aus bestehenden Modulen zusammengesetzt. Das Vorgehen bei der Entwicklung dieses Fußballroboters mittels 3D-Drucker und Arduino ist in dieser Arbeit erläutert.

## 2 Aufgabenstellung

Ziel war die Entwicklung eines Roboters, der auf einer Ausstellung von den Besuchern gesteuert werden kann. Der Roboter sollte mit einer App für Smartphones steuerbar sein. So kann jeder Besucher die App herunterladen und sich mit dem Roboter verbinden. Um das Ganze interessanter zu gestalten, sollten die Roboter auf einem Spielfeld mit Toren Fußball spielen. Es spielen jeweils zwei Besucher gegeneinander. Da die Entwicklung des Roboters und das Programmieren der App den Rahmen einer Bachelorarbeit sprengen würde, ist sie auf zwei Arbeiten aufgeteilt. Diese Arbeit befasst sich mit der Entwicklung der Hardware und Software des Roboters.

### 2.1 Anforderungen an den Roboter

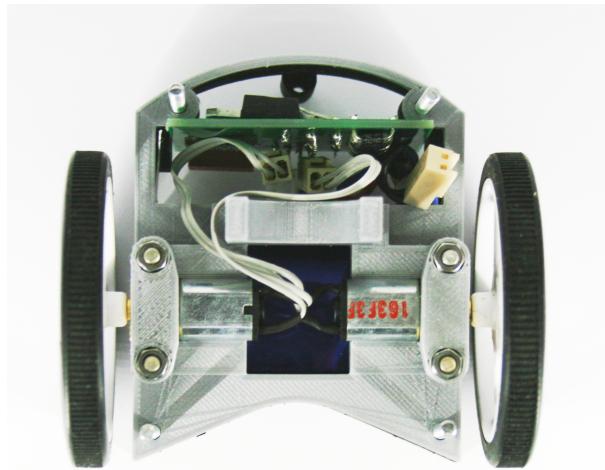
Die Herausforderung dieser Arbeit war, einen auf die Aufgabenstellung spezialisierten Roboter zu entwickeln. Er durfte nicht größer als 100 Millimeter im Durchmesser sein, um die Größe des Spielfeldes im Rahmen zu halten. Zum Fußballspielen benötigte er ein Antriebssystem und einen Schussmechanismus um Tore zu schießen. Für die Kontrolle des Roboters war eine Funkverbindung vorgesehen. Eine integrierte Kamera lässt den Roboter über das Internet oder mit einem Server autonom steuerbar werden. Aus diesem Grund musste das Videosignal über die Funkverbindung übertragbar sein. Damit der gesamte Spielaufbau mit Roboter ohne Eingriff von außen über längere Zeit funktionieren kann, war das Laden des Akkus zu automatisieren. Ein Infrarotdetektor sollte hierbei für das Finden der Ladestation zuständig sein.

# 3 Planung

Da ein Roboter wie in Abschnitt 2.1 beschrieben noch nicht existiert, war eine neue Roboterplattform zu entwickeln. Aus diesem Grund wurde das gesamte Gehäuse 3D-gedruckt. Dadurch lassen sich in kurzer Zeit maßgeschneiderte Halterungen herstellen und korrigieren, wodurch ein kompakter Aufbau des Roboters möglich wird. Um mit der Auswahl der Komponenten fortzufahren, waren grundlegende Fragen zu Antriebssystem und der verwendeten Funkverbindung zu klären.

## 3.1 Antriebssystem

Für das Antriebssystem gab es mehrere Optionen. Alle Mechanismen, die keine Räder verwenden, schieden aufgrund ihrer Komplexität aus. Ebenso war ein Aufbau mit fahr-dreh-Modulen nicht möglich, da diese in der geforderten Größe nicht verfügbar waren. Es kamen nur ein omnidirektonaler Antrieb mit speziellen Omnidräder oder ein differentieller Antrieb wie in Abbildung 3.1 in Frage. Ein differentieller Antrieb besteht aus zwei unabhängig voneinander steuerbaren Rädern. Bei gleicher Drehzahl der Räder fährt der Roboter nach vorne oder nach hinten. Durch unterschiedliche Drehzahlen lässt er sich in beliebig großen Kreisen fahren. Dabei ist die Achse der Räder immer parallel zum Radius des Kreises, den der Roboter fährt. Seitliches Fahren ist mit diesem Antrieb nicht möglich. Der omnidirektionale Antrieb hätte den Vorteil, dass der Roboter ohne vorher drehen zu müssen, in jede beliebige Richtung fahren kann. Dadurch kann der Roboter schneller seitlich fahren, um einen an ihm vorbeirollenden Ball abzublocken. Um solche Manöver fahren zu können ist eine komplexe, sensorgestützte Regelung sowie Rundumsicht zur Lokalisierung des Balles nötig. Der geplante Roboter besitzt nur einen nach vorne gerichteten Kamerasensor. Aus diesem Grund und weil Omnidräder in der benötigten Größe nicht verfügbar waren, war ein differentieller Antrieb die bessere Wahl. Um diesen zu realisieren, war eine Motor-Rad-Kombination zu finden, die nicht länger als der Radius des Roboters ist und den Roboter schnell genug fortbewegt. Außerdem musste der Motortreiber genügend groß dimensioniert sein, um die Stromspitzen bei schnellen Richtungsänderungen der Räder auszuhalten.



**Abbildung 3.1:** Differentialantrieb des Roboters

## 3.2 Funkverbindung

Für die Videoübertragung war eine hohe Datenrate über die Funkverbindung nötig. Im Gegenzug war aufgrund des Akkubetriebs der Stromverbrauch möglichst klein zu halten. Außerdem war es erforderlich, dass die Funkverbindung mit den meisten gängigen Smartphones kompatibel ist. Aus diesem Grund kamen nur Bluetooth und WLAN in Frage. In Tabelle 3.1 sind die drei möglichen Standards aufgelistet. Bluetooth v4.0 mit BLE (Bluetooth Low Energie Technologie) war hierbei die Option mit dem geringsten Stromverbrauch. Bluetooth V2.1 mit EDR (Enhanced Data Rate) ist der am meisten verwendete Bluetooth-Standard und bot die höchste Datenrate bei relativ niedrigem Stromverbrauch. Bluetooth V3.0 verwendet bei hohen Datenraten eine WLAN-Verbindung und hatte aus diesem Grund den gleichen Stromverbrauch wie WLAN. Deshalb ist anstelle dieses Bluetooth-Standards nur WLAN aufgelistet.

Standard	Modul	Datenrate	Stromverbrauch
Bluetooth v4.0 + BLE	BC118	60–270 kbps	12 mA
Bluetooth v2.1 + EDR	RN-42 (v6.15)	0,721–2,0 Mbps	30 mA
WLAN 802.11b/g/n	XBee S6B	1–72 Mbps	< 309 mA

**Tabelle 3.1:** Datenrate und Stromverbrauch der verschiedenen Funkstandards [4][7][17]

Um herauszufinden, welche Datenrate für den Videostream nötig ist, wurde diese für Auflösungen mit JPEG-Codierung und ohne betrachtet. Eine Übersicht ist in Tabelle 3.2 zu sehen. Die Bildfrequenz beträgt hier 24 fps.

Auflösung	keine Codierung	JPEG mit 1Bit/Pixel
VGA 640x480	117.9648 Mbit/s	7.3728 Mbit/s
CIF 352x288	38.928384 Mbit/s	2.433024 Mbit/s
QVGA 320x240	29.4912 Mbit/s	1.8432 Mbit/s
QCIF 176x144	9.732096 Mbit/s	0.608256 Mbit/s
QCIF schwarz/weiß	4.866048 Mbit/s	0.608256 Mbit/s

**Tabelle 3.2:** Datenrate bei verschiedenen Auflösungen mit und ohne JPEG-Codierung

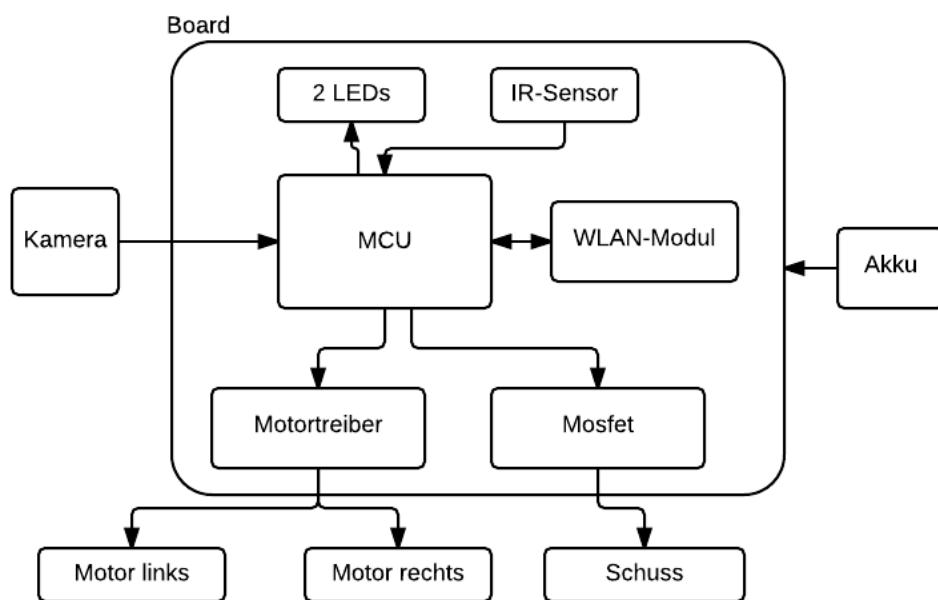
Beide Bluetooth-Standards können bei maximaler Datenrate höchstens JPEG-codiertes QVGA übertragen. Ob diese Übertragungsgeschwindigkeit auch erreicht wird, hängt zusätzlich von der Schnittstelle zwischen Controller und Funkmodul ab.

Rechnet man im Gegenzug den Anteil aus, den der Stromverbrauch des Funkmoduls ausmacht, so zeigt sich schnell dessen geringe Einfluss. Die Werte für den Strom der Motoren  $I_{Motor}$ , des WLAN Modul  $I_{XBee}$ , des Schuss  $I_{Schuss}$  und des Controllers  $I_{Controller}$  sind aus Kapitel 4 zu entnehmen.

Der Maximalstrom des XBee von 309 mA würde 10% des gesamten Stroms  $I_{max}$  ausmachen. Um zum einen viel Spielraum bei der Übertragungsgeschwindigkeit zu haben und zum anderen, weil der höhere Stromverbrauch nicht zu sehr ins Gewicht fällt, war WLAN die beste Option.

# 4 Komponenten

Um die nötigen Maße für das Gehäuse zu bekommen, mussten zuerst alle Komponenten ausgesucht werden. In Abbildung 4.1 sind die einzelnen Bauteile schematisch aufgezeichnet. Dabei sind die Bauteile, welche auf der Hauptplatine befestigt werden und welche im Gehäuse des Roboters verbaut sind, festgelegt. Die Pfeile geben die Richtung des Informations- und Stromflusses an. Die Kamera, das Controller-Board sowie das WLAN-Modul sollten als Breakout-Board (eine Platine die alle Pins des Bauteils auf eine Stiftleiste mit 2,54 mm Raster herausführt) verfügbar sein, um schon vor dem Fertigen der Platine erste Software-Tests durchführen zu können und die Bauteile bei möglichen Defekten einfach austauschen zu können.



**Abbildung 4.1:** Übersicht über alle Funktionen

## 4.1 Motoren und Räder

Nachdem das Antriebssystem feststand, war die Wahl der Motor-Rad-Kombination durch die Größe des Roboters stark eingeschränkt. Die Länge der Motoren mit Rad und Kabel durften 50

mm nicht überschreiten, da sonst der Roboter zu groß werden würde. Weitere Kriterien wie eine große Auswahl an Getrieben, gute Verfügbarkeit sowie Gleichstrombetrieb setzten die Micro Metal Gear Motoren von Pololu von allen anderen Motoren ab. Die Wahl der Getriebe so wie des Raddurchmessers war abhängig vom Gewicht und der Bauform des Roboters. Beim Zeichnen des Roboters zeigte sich, dass Räder mit 60 mm Durchmesser am geeignetsten waren. Mit ihnen war noch genügend Platz unter den Motoren für Akku oder Motortreiber, ließen den Roboter aber trotzdem nicht zu groß werden. In Tabelle 4.1 sind die verschiedenen Getriebetypen mit Geschwindigkeit und Drehmoment aus dem Datenblatt der Motoren aufgelistet.

Getriebe	Geschwindigkeit v [m/s]	Drehmoment M [Nm]
5:1	18,84	0,01412
10:1	9,42	0,0282
30:1	3,14	0,06355
50:1	1,96	0,10592
75:1	1,26	0,15535
100:1	1,00	0,21184
150:1	0,628	0,28246
210:1	0,38	0,35307
250:1	0,31	0,42369

**Tabelle 4.1:** Geschwindigkeit und Drehmoment bei verschiedenen Getriebeübersetzungen [16]

Da der Spielspaß mit der Geschwindigkeit der Roboter steigt, war das Ziel das Spielfeld mit einer Länge von ein bis zwei Meter in möglichst kurzer Zeit zu durchfahren. Dabei sollte der Roboter aus dem Stand starten. Das Gewicht  $m_{Robot}$  des Roboters wurde mithilfe der Gewichte der anderen Bauteile und einer sehr frühen Version des Gehäuses auf maximal 500 g festgelegt. Zusammen mit dem Raddurchmesser  $d_{Rad}$  von 60 mm konnte die Beschleunigung des Roboters berechnet werden. Der Rollwiderstand  $c_{Rad}$  wurde auf 0,01 festgelegt, was einem Autoreifen auf Beton entspricht [19].

$$F_{Motor} = F_{Rollreibung} + F_{Trägheit}$$

$$0,5Md_{Rad} = c_{Rad}m_{Robot}g + m_{Robot}a$$

$$a = \frac{0,5Md_{Rad} - c_{Rad}m_{Robot}g}{m_{Robot}} \quad (4.1)$$

Nun musste die Zeit  $t_g$  berechnet werden, die der Roboter benötigt, um eine bestimmte Strecke  $s_g$  aus dem Stand heraus zurückzulegen. Hierbei war zu unterscheiden, ob er die Endgeschwindigkeit  $v_{max}$  erreicht.  $s_1$  ist dabei die Strecke, die bis zur Maximalgeschwindigkeit zurückgelegt wurde und  $t_1$  die benötigte Zeit.

$$t_1 = \frac{v_{max}}{a} \quad (4.2)$$

$$s_1 = 0,5at_1^2 = \frac{v_{max}^2}{2a} \quad (4.3)$$

Falls  $s_1$  größer als  $s_g$  ist, wird die Maximalgeschwindigkeit nicht erreicht.

$$t_g(s_1) = \begin{cases} t_1 + \frac{(s_g - s_1)}{v_{max}} & \text{für } s_1 < s_g \\ \sqrt{\frac{s_g}{0,5*a}} & \text{für } s_1 \geq s_g \end{cases} \quad (4.4)$$

Setzt man die Werte aus Tabelle 4.1 mit  $s_g = 1$  m in Formel (4.4) ein, so kommt man auf die Zeiten für die verschiedenen Getriebe. Das Übersetzungsverhältnis 5:1 hatte eine negative Beschleunigung, was bedeutet, dass der Motor den Roboter nicht bewegen kann. An der Strecke  $s_1$  sieht man, dass erst ein Getriebe von 50:1 die Endgeschwindigkeit vor dem Ziel von einem Meter erreicht. Wichtig für die Wahl des richtigen Getriebes war jedoch die Zeit  $t_g$ , welche mit dem Getriebe 50:1 minimal ist.

Getriebe	$t_1$ [s]	$s_1$ [m]	$t_g$ [s]
5:1	-	-	-
10:1	10,67	50,24	1,5
30:1	0,97	1,52	0,79
50:1	0,32	0,32	0,67
75:1	0,13	0,08	0,86
100:1	0,08	0,04	1,03
150::1	0,04	0,01	1,61
210:1	0,02	0	2,28
250:1	0,01	0	2,66

**Tabelle 4.2:** Zeit für 1 m Strecke bei verschiedenen Getriebeübersetzungen

In Abbildung 4.2 ist der verwendete Motor mit Rad zu sehen.



**Abbildung 4.2:** Pololu Rad mit Motor

## 4.2 Motortreiber

Zum Steuern der zwei Gleichstrommotoren des Differentialantriebs ist ein Motortreiber nötig, da die meisten Microcontroller nur sehr geringe Ströme an den Pins aushalten. Der Motortreiber hatte den Blockierstrom der Motoren von 1600 mA [16] auszuhalten ohne Schaden zu nehmen. Dieser Strom kann, wenn der Roboter längere Zeit gegen die Wand fährt, auch über mehrere Sekunden fließen. Ein Ausschalten bei zu hohen Motorströmen ist nicht erwünscht, weil es den Spielfluss stören würde. Die Doppel-H-Brücke L298 mit einer Maximalbelastung von 2 A im Gleichstrombetrieb pro Kanal [18] war dafür bestens geeignet.

## 4.3 Schuss

Als Schussmechanismus kam nur ein Elektromagnet in Frage. Ein pneumatischer Mechanismus würde mit der Druckkammer, Ventilen und Kolben zu viel Platz einnehmen. Eine Spannvorrichtung mit Feder wäre zu fehleranfällig und benötigt unnötig lange Ladezeiten. Der Vorteil des Elektromagneten ist die hohe Schussfrequenz und die einfache Ansteuerung mittels Mosfet-Treiber. Die Spannung des Elektromagnets wurde nach der Motorspannung gewählt, da diese sowieso zur Verfügung stehen musste. Für die Hubkraft und den Hubweg wurde das Maximum ausgesucht. Die wichtigsten Daten wie Hubkraft, Gewicht und Leistung sind in Tabelle 4.3 zu sehen.

<b>Gewicht</b>	86 g
<b>Nennspannung</b>	6 V
<b>Leistung</b>	7 W
<b>Anfangskraft</b>	0,6 N
<b>Endkraft</b>	11 N

**Tabelle 4.3:** Technische Daten des Magnetschuss ITS-LZ-1949-D-6VDC [10]



**Abbildung 4.3:** Magnetschuss

## 4.4 Akku

Da die Motoren und der Schuss mit einer Spannung von 6 V betrieben werden, war ein zwei Zellen Lithium-Polymer-Akku (LiPo) nötig. Um eine möglichst lange Spielzeit zu garantieren, wurde der Akku so groß wie möglich gewählt. In Abbildung 4.4 ist der verwendete Akku zu sehen. Die wichtigsten Daten wie Kapazität, Gewicht und Spannung lassen sich aus Tabelle 4.4 entnehmen.

<b>Gewicht</b>	52 g
<b>Spannung</b>	7,4 V
<b>Maximalleistung</b>	48 A
<b>Kapazität</b>	800 mAh

**Tabelle 4.4:** Technische Daten des Akkus [6]



**Abbildung 4.4:** LiPo [6]

## 4.5 Kamera

Die Suche nach einer passenden Kamera stellte die größte Herausforderung dar. Fast alle Sensoren sind ausschließlich mit Lötanschlussflächen unter dem Gehäuse verfügbar. Diese Chips sind nur in speziellen Reflow-Öfen sinnvoll lötbar. Für den einfachen und schnellen Aufbau einer Testplatine sollte deshalb die Kamera auf ein Breakout Board vor gelötet sein. Da die meisten dieser Boards für Kameras konzipiert waren, die über eine für Videostream zu langsame UART-Schnittstelle angesteuert werden oder einen analogen Videoausgang besaßen, war die Auswahl sehr eingeschränkt. Der Kamerasensor OV7670 war einer der einzigen Sensoren, der ein paralleles Video-Interface besaß, welches zur Videoübertragung geeignet war, und als Breakout Board verfügbar war. Außerdem fanden sich zu diesem Sensor einige Projekte [2] [1], die einen Mikrocontroller zur Ansteuerung verwendeten.



**Abbildung 4.5:** Kamera Sensor OV7670 [14]

## 4.6 Mikrocontroller

Der Mikrocontroller sollte das WLAN Modul, die Kamera und die Aktoren steuern. Außerdem sollte es möglich sein, den Videostream möglichst effizient zu komprimieren, um die Datenrate über das WLAN-Modul zu minimieren. Der Stromverbrauch sowie der Arbeitsaufwand zum Entwickeln der Software sollten gering gehalten werden. Ein Möglicher Lösungsansatz war ein Single Board Computer (SBC) wie das Raspberry PI mit Linux Betriebssystem. Da diese Boards einen relativ hohen Stromverbrauch um die 500 mA und für die Größe des Roboters zu groß waren, schied diese Option aus. Die Verwendung eines Field Programmable Gate Array (FPGA) wäre vermutlich die effizienteste Varianten in Hinblick auf schnelles Videokodieren bei minimalem Stromverbrauch. Jedoch hätte die Implementierung der WLAN-Schnittstelle, des Kameratreibers sowie der Videokodierung den Rahmen der Arbeit gesprengt. Ein anderer Ansatz war mittels 8 Bit MCU. Hierbei wäre eine Videocodierung schon bei niedrigeren Bildraten nahezu unmöglich. Ich entschied mich schlussendlich für einen Mittelweg zwischen SBC und 8 Bit MCU. Ein 32 Bit Microcontroller wie der ARM-Cortex M4 mit genügend internen RAM und einem Stromverbrauch von circa 20 mA sollte allen Aufgaben gewachsen sein. Um den Implementierungs- und Entwicklungsaufwand gering zu halten, war ein Breakout Board mit fertigem Bootloader von Vorteil. Die in Frage kommenden Boards sind in Tabelle 4.5 mit ihren technischen Daten aufgelistet.

Name	Takt	SRAM	Pins	Größe	Programmierinterface	WLAN
Maple mini	72 MHz	20 KB	34	51x18 mm	USB,JTAG	Nein
Teensy 3.1	96 Mhz	64 KB	34	35x18 mm	USB	Nein
Spark Photon	120 Mhz	128 KB	18	37x20 mm	USB,Wlan,JTAG	Ja
mbed NXP LPC1768	100 MHz	64 KB	32	54x26 mm	USB,JTAG	Nein

**Tabelle 4.5:** Verschiedene Controllerboards und ihre Eigenschaften [11] [12] [15] [5]

Das Spark Photon war durch die integrierte WLAN-Schnittstelle und den großen Arbeitsspeicher am interessantesten. Jedoch war es zum Zeitpunkt des Baus des Roboters nicht verfügbar. Das Teensy 3.1 war die zweitbeste Option, da es einen großen SRAM hat, was die Entwicklungsarbeit der Videoübertragung wesentlich vereinfachte, und weil es von den Abmessungen das kleinste ist. Außerdem sind die meisten Chip-Funktionen wie Hardware Pulsweitenmodulation (PWM), Serial Peripheral Interface (SPI), Interrupts und Direct Memory Access Controller (DMA-Controller) in einer Arduino-kompatiblen Bibliothek vorhanden, wodurch sich die Entwicklungsarbeit der Software wesentlich verkürzen würde. Der Nachteil der fehlenden JTAG-Verbindung zeigte sich erst später bei der Fehlersuche und wurde zu diesem Zeitpunkt leider nicht erkannt.



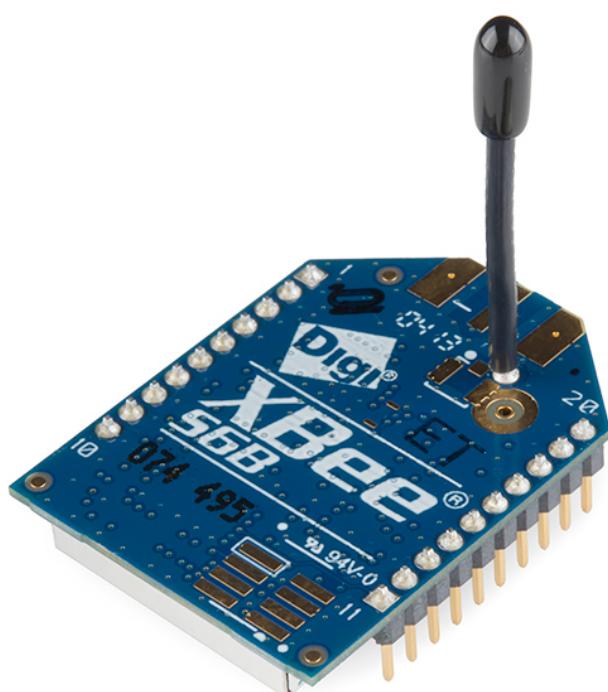
**Abbildung 4.6:** Teensy 3.1 [15]

## 4.7 WLAN-Modul

Für die WLAN Verbindung schieden alle Module, die nur über eine serielle UART-Schnittstelle angeschlossen werden, aufgrund ihrer mangelnden Geschwindigkeit von maximal 320 Kbit/s aus. Das XBee S6B überzeugte hier mit einem schnellen und einfach zu implementierenden SPI-Interface mit bis zu 1 Mbit/s [7]. Eine fertige Arduino-Bibliothek für das XBee-S6B [3] sowie die Tatsache, dass es ein Breakout Board für das Modul gibt, waren dabei ebenfalls ausschlaggebende Punkte. Da die 20-polige Pinbelegung des XBee standardisiert ist, lassen sich auch andere Funkmodule mit Bluetooth auf der Platine installieren. Dies wurde jedoch nicht weiter verfolgt, da die Übertragungsgeschwindigkeit des WLAN-Moduls für den Videostream notwendig war. In Tabelle 4.6 sind die wichtigsten Technischen Daten aus dem Datenblatt aufgelistet.

Datenrate	UART bis zu 320 Kbit/s, SPI bis zu 1 Mbit/s
Standard	802.11 b/g/n
Sende Strom	bis zu 309 mA
Empfangs Strom	100 mA

**Tabelle 4.6:** Technische Daten zum Xbee [7]



**Abbildung 4.7:** WLAN-Modul XBee S6B [7]

# 5 Hardware

## 5.1 3D Design

Da es keine passende Fahrbasis für den geforderten Roboter gab, wurde das Gehäuse mit Differenzialantrieb neu entworfen und mittels 3D-Druck gefertigt. Zum Zeichnen wurde Solid Edge verwendet, da es sich sehr intuitiv bedienen lässt und als Studentenversion verfügbar war. Zu Beginn waren die einzelnen Komponenten wie Motoren, Reifen, Akku und Schuss als fixe Bauteile nach den Bemaßungen aus den Datenblättern zu zeichnen. Um einen Überblick über das Gesamtgewicht des Roboters zu bekommen, wurde für jedes Bauteil das exakte Gewicht aus dem Datenblatt eingetragen. So konnte während des Zeichnens überprüft werden, ob der Roboter das Maximalgewicht von 500 g übersteigt.

### 5.1.1 Einschränkungen durch 3D-Druck



**Abbildung 5.1:** Ultimaker 2 beim Drucken

Gedruckt wurde mit dem Ultimaker 2, einem auf Schmelzschichtung basierendem 3D Drucker. Bei diesen Druckern wird Polylactid (PLA) in einer Düse erhitzt und in dünnen Streifen auf einer Glasplatte aufgetragen. Durch das Schichten mehrerer Lagen mit unterschiedlichen Umrissen lassen sich so dreidimensionale Objekte erstellen. Während der Entwicklungsarbeit wurden einige Optimierungen bezüglich Wandstärke, Geometrie und Bauteilgröße beim Drucken durchgeführt.

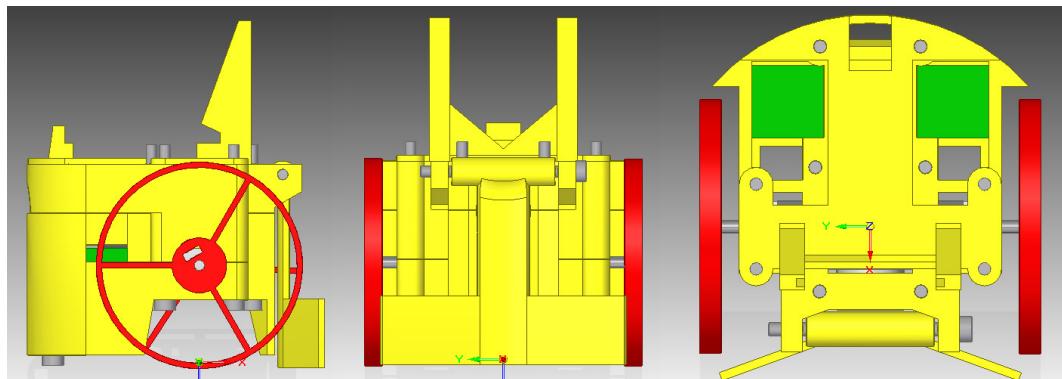
Zum einen hatte der Durchmesser der Düse exakt 0,4 mm, weshalb dieser Drucker nur Wandstärken mit einem Vielfachen von 0,4 mm drucken kann. Bei allen anderen Wandstärken entsteht

ein Spalt, den der Drucker mit einem Raster zu füllen versucht. Dies führt bei sehr dünnen Wänden von unter 10 mm zu langen Druckzeiten und oft sehr unregelmäßigen Oberflächen und instabilen Druckergebnissen. Erst bei dickeren Wänden wird das Raster stabil.

Ein weiterer wichtiger Punkt, der zu beachten war, war das Vermeiden von Support-Struktur. Support-Struktur wird benötigt, um Teile zu drucken, bei denen keine Seite des Bauteils komplett auf der Druckfläche aufliegt. Da diese Support-Struktur sowohl Druckdauer als auch Plastik verschwendet und sich erst nach intensiver Kalibrierung restlos entfernen lässt, galt es, sie möglichst zu vermeiden.

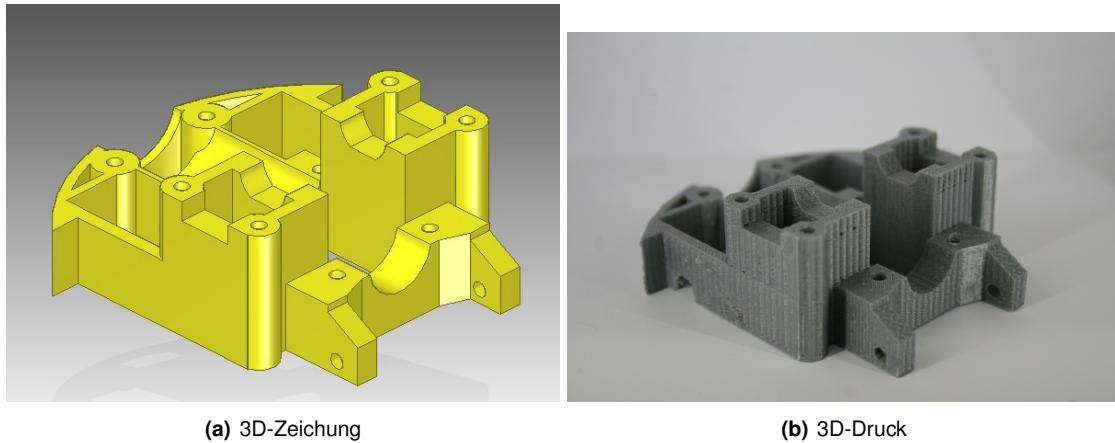
Der letzte Punkt, der das Prototyping mittels 3D-Drucker wesentlich vereinfacht, ist das vermeiden von großen Teilen. Auch wenn dadurch das Zusammenbauen des Roboters vereinfacht wird und möglicherweise zur Stabilität beigetragen wird, geht mit einer Druckdauer von über 20 Stunden pro Teil der Vorteil, schnell Änderungen vorzunehmen, verloren. Eine Druckdauer von maximal 4 Stunden war für die Arbeit am effizientesten. So konnte an einem Tag ein Teil neu gezeichnet, gedruckt, getestet und wenn nötig eine Fehlerkorrektur gedruckt werden.

### 5.1.2 Version 1

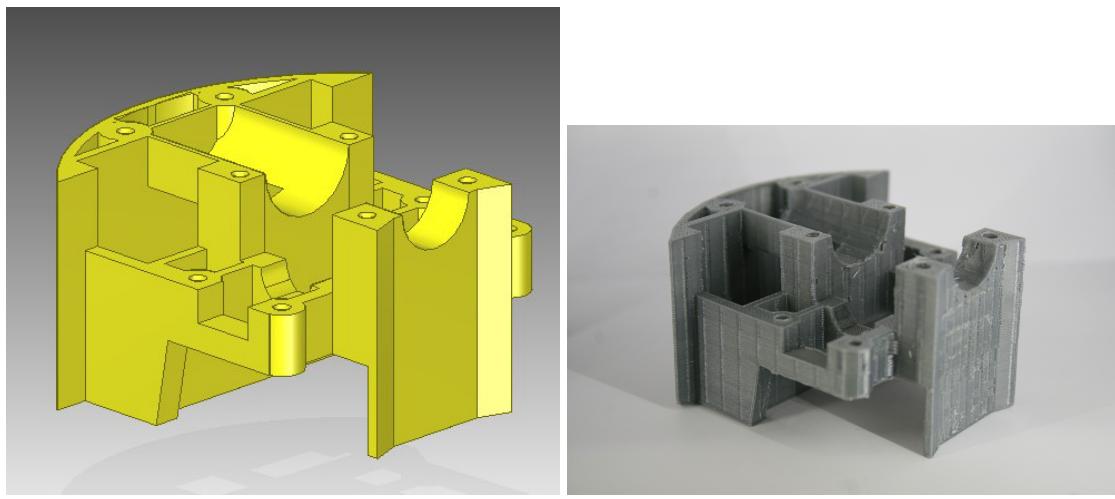


**Abbildung 5.2:** Gehäusedesign Version 1

Die erste Version des Roboters bestand aus 4 einzelnen Teilen. Dabei waren die zwei größten Teile für das Halten der Motoren, des Magnetschuss und des Akkus vorgesehen. Durch 12 senkrechte Schrauben wurden diese Komponenten in vorgesehenen Aussparungen eingeklemmt. Dieser Aufbau ermöglicht ein schnelles Zusammenbauen des Roboters. In Abbildung 5.3 und Abbildung 5.4 sind beide Bauteile zu sehen.



**Abbildung 5.3:** Gehäuse oben



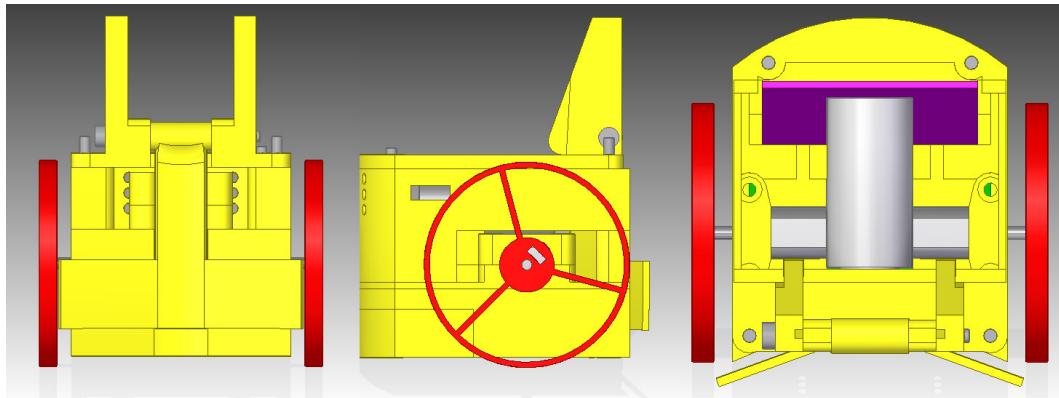
**Abbildung 5.4:** Gehäuse unten

Es zeigte sich, dass dieses Design für das Drucken der Teile ungeeignet war. Das untere Gehäuseteil aus Abbildung 5.4 ist nur mit einer großen Support-Struktur druckbar, was zu einer unnötigen Verlängerung der Druckzeit führt (siehe 5.1.1). Zusätzlich waren die Teile so groß, dass Änderungen am Design einen 20 Stunden langen Druck nach sich zogen. Außerdem war der Schwerpunkt des Roboters falsch positioniert, wodurch die Räder zu wenig Reibung auf dem Boden hatten und ein gerades Fahren unmöglich war.

### 5.1.3 Version 2

Um die Probleme aus Version 1 zu beheben, wurde der Roboter komplett neu modelliert. Diese Version bestand nun aus 15 Teilen, welche alle maximal 3 Stunden Druckdauer beanspruchten.

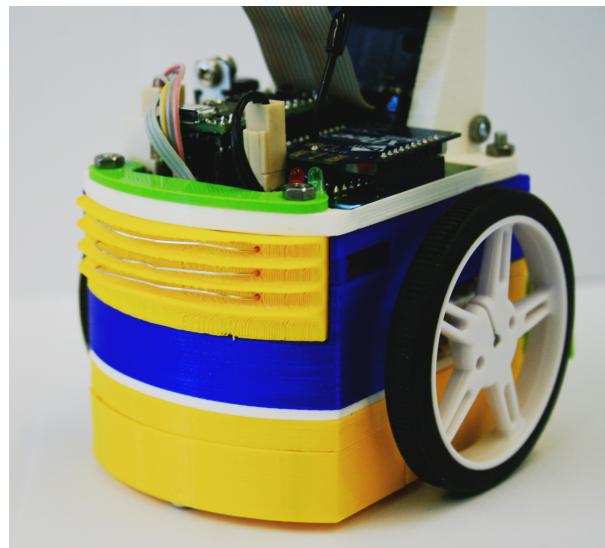
Dadurch ließen sich leicht Änderungen vornehmen. Außerdem wurde der Akku unter die Motoren verlegt, wodurch sich der Schwerpunkt nun fast vollständig auf den Rädern befindet. Dies löste das Problem der schlechten Haftung der Räder. Mehr Platz im Innenraum für Verkabelung, eine Aussparung für einen An-/Ausschalter sowie die Möglichkeit den Akku zu entnehmen ohne den Roboter zu zerlegen wurden zusätzlich eingeplant. Außerdem ist es nun möglich, die längeren Elektromotoren mit Radencoder einzubauen. Dies wurde jedoch nicht weiter verfolgt, da der Roboter mit dem verlagerten Schwerpunkt ausreichend gerade fährt.



**Abbildung 5.5:** Gehäusedesign Version 2

#### 5.1.4 Ladestation

Im späteren Verlauf der Arbeit wurde auch die Ladestation entwickelt. Durch den modularen Aufbau musste nur ein Teil des Roboters ersetzt werden, um die externen Ladekontakte anbringen zu können. In Abbildung 5.6 sind die Ladekontakte zu sehen. Das Laden des LiPos wird von einem externen LiPo-Ladegerät geregelt. Drei Kontakte ermöglichen das unabhängige Laden beider Zellen. Durch die vertikale Anordnung der Kontakte ist eine Verpolung des Akkus unmöglich.



**Abbildung 5.6:** Ladekontakte

## 5.2 Board Design

Da Kamerasensor, das WLAN-Modul sowie das Teensy 3.1 eine große Anzahl von Verbindungen untereinander haben, stand es nahe eine Trägerplatine für die Breakout Boards fertig zu lassen. Außerdem bot sich so die Möglichkeit den Treiber für den Magnetschuss, ein paar LEDs und den IR-Detektor mit auf der Platine zu integrieren. Zum Entwerfen der Platine wurde das Grundkonzept aus Abbildung 4.1 des Roboters durch einige Details erweitert, um auf Basis dieses Modells einen Schaltplan zu zeichnen. In Abbildung 5.7 sind die Anzahl der Pins für jede Schnittstelle zu sehen sowie einige Details der Stromversorgung.

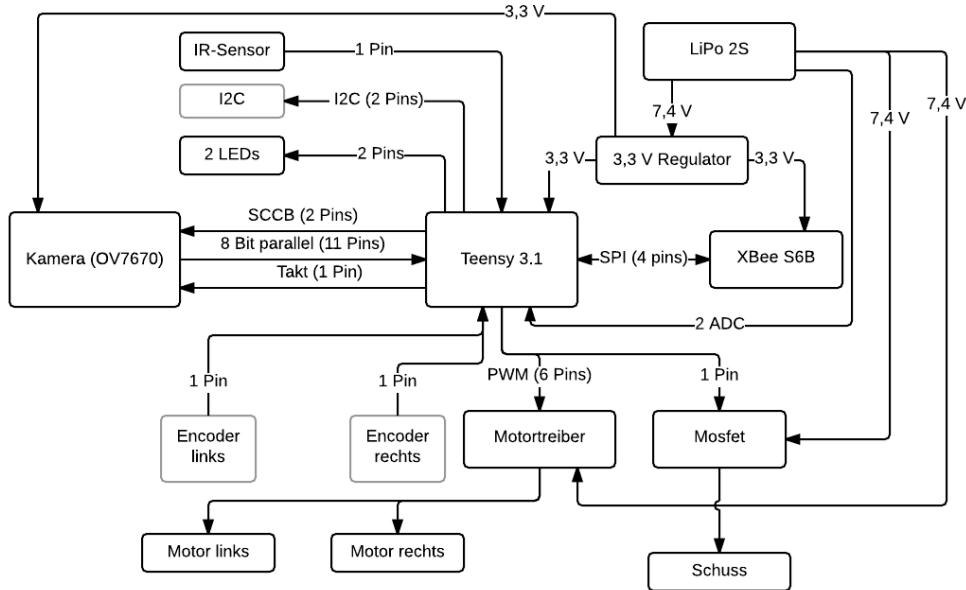


Abbildung 5.7: Schema der Platine

Um den Schaltplan mit Eagle zeichnen zu können, musste noch die Pinbelegung festgelegt werden. Wichtig war, dass der Motor über PWM-Ausgänge gesteuert wird, um Rechenzeit zu sparen. Außerdem sollten die Datenpins der Kamera auf einen kompletten Port fallen, um das schnelle Abfragen zu erleichtern. Um später verschiedene Interrupt Level nutzen zu können, liegen alle Pins, die als externer Interrupt funktionieren könnten, auf einem eigenen Port. So kann später die Priorität für die einzelnen Ports eingestellt werden und dadurch die Priorität der Interrupts verändert werden. Im Tabelle 5.1 ist eine Übersicht der Pinbelegung des Teensy zu sehen. Sowohl die Pinnummer der Arduino-Bibliothek als auch der Hardware PIN des ARM sind darin aufgelistet. Außerdem ist die Funktionsweise des Pins beschrieben.

ARM Pin	Teensy Pin	Belegung	Funktion
VDD	3,3V	3,3V	
GND	GND	GND	
AGND	AGND	GND	
PTB16	0	XBEE_TX	UART
PTB17	1	XBEE_RX	UART
PTC3	9	XBEE_NATTN	INTERRUPT
PTC4	10	XBEE_NSSEL	OUTPUT
PTC6	11	XBEE_MOSI	SPI
PTC7	12	XBEE_MISO	SPI
PTC5	13	XBEE_CLK	SPI
PTC1	14	XBEE_RESET	OUTPUT
PTD0	2	OV_D0	INPUT
PTD1	15	OV_D1	INPUT
PTD2	7	OV_D2	INPUT
PTD3	8	OV_D3	INPUT
PTD4	6	OV_D4	INPUT
PTD5	20	OV_D5	INPUT
PTD6	21	OV_D6	INPUT
PTD7	5	OV_D7	INPUT
PTE1	26	OV_PCLK	DMA_TRIGGER
PTB1	17	OV_VSYNC	INTERRUPT
PTA12	3	OV_XCLK	PWM
PTA13	4	OV_HREF	INTERRUPT
PTC10	29	OV_SCL	OUTPUT
PTC11	30	OV_SDA	IN/OUTPUT
PTB0	16	KICK	OUTPUT
PTB3	18	I2C_CLK	GPIO
PTB2	19	I2C_DAT	GPIO
PTA5	24	IR	INPUT
PTC2	23	MOTOR_A	PWM
PTC1	22	MOTOR_B	PWM
PTB18	32	MOTOR_C	PWM
PTB19	25	MOTOR_D	PWM
PTC9	27	BAT_C1	ANALOG IN
PTC8	28	BAT_C2	ANALOG IN
PTE0	31	LED_1	OUTPUT
PTA4	33	LED_2	OUTPUT

**Tabelle 5.1:** Pinbelegung des Teensy

Mit dieser Pinbelegung konnte nun der komplette Schaltplan gezeichnet werden. Dabei wurden die in den Datenblättern beschriebene Standardbelegung der Bauteile beachtet.

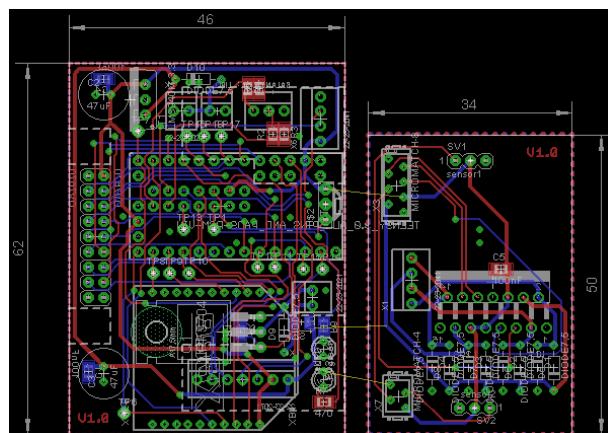
Die Maße des Roboters gaben die maximale Größe der Platine mit 40x60 mm vor. Leider passten die Bauteile nicht auf die geplante Platinengröße, weshalb der Motortreiber auf eine separate

Platine ausgelagert wurde. Diese sollte zuerst unter den Motoren, später hinten am Roboter positioniert werden.

Wie so oft bei ersten Versionen von neuen Projekten unterlaufen immer mehr oder minder schwere Fehler. Bei diesem Board war der Controller sowie das WLAN-Modul auf der Platine gespiegelt platziert, wodurch die gesamte Platine gespiegelt werden musste. Beim Vergleich von Abbildung 5.9 mit Abbildung 5.8, ist dies zu erkennen. Das Resultat sind einige verdrehte Bauteile und weniger Platz für diverse Stecker, wodurch die Funktion jedoch nicht eingeschränkt wurde. Ein anderes Problem war die Größe des XBee S6B, die die Standardbemaßung des XBee übersteigt. Dadurch lässt sich der Kamerastecker nur schwer einstecken. Außerdem wäre ein kleinerer Steckertyp für diverse Steckverbindungen von Vorteil.



**Abbildung 5.8:** fertige Platine



**Abbildung 5.9:** Layout der Platine

# 6 Software

Die Software soll eigenständig eine WLAN-Verbindung aufbauen. Sobald diese aufgebaut ist, steuern Pakete vom Server Motoren und Schuss. Nebenbei sendet der Roboter in regelmäßigen Abständen Statusnachrichten, in denen Sensordaten enthalten sind. Bei aktiver Videoübertragung wird das Videosignal codiert über die Funkverbindung an den Server gesendet. Eine USB-Schnittstelle zum Konfigurieren verschiedener Parameter wurde auch vorgesehen.

## 6.1 Übersicht

Die Software ist in mehrere Module aufgeteilt, welche in Abbildung 6.1 zu sehen sind. Im Zentrum ist das config-Modul, welches alle Einstellungen im EEPROM speichert. In einer späteren Version der Software wurde die Sensorabfrage ebenfalls über dieses Modul ausgeführt. Dadurch waren alle Informationen für das Erstellen der Statusnachrichten verfügbar. Die Module aktuators, sensors, Xbee und Ov7670 sind die Treiber für die Hardwarekomponenten. Wifi war als Schnittstelle für die XBee-Bibliothek zuständig. In diesem Modul werden eingehende Pakete verarbeitet und ausgehende Pakete direkt weitergeleitet.

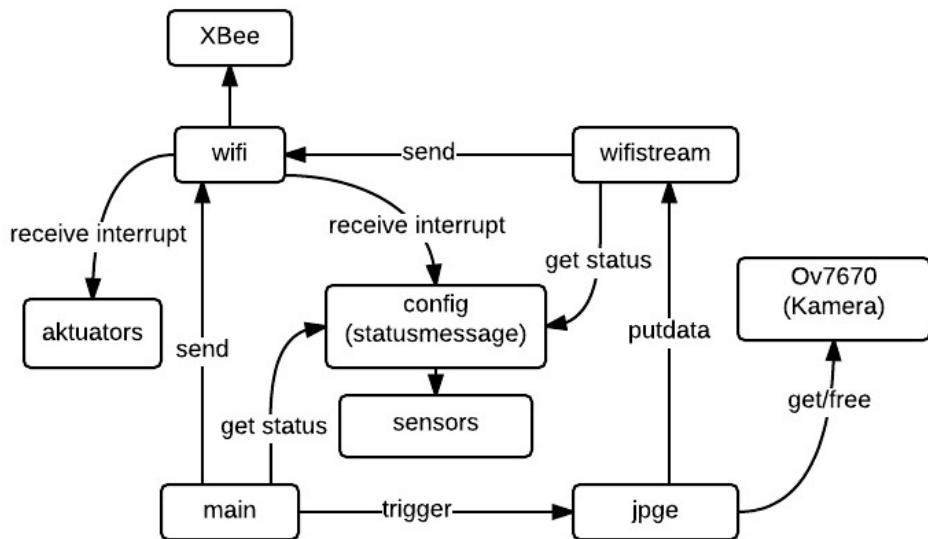
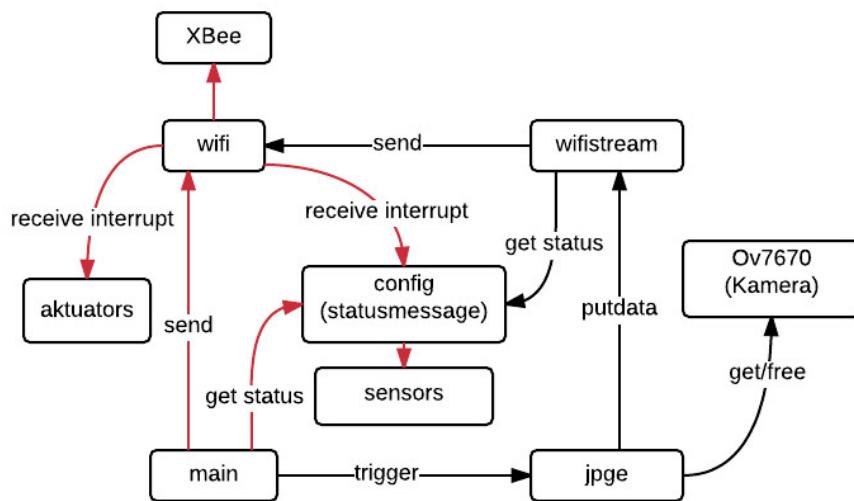


Abbildung 6.1: Softwarekonzept

Beim Start des Moduls initialisiert die main-Routine alle Module und versucht eine WLAN-Verbindung mit dem im EEPROM gespeicherten Accesspoint aufzubauen. Tritt ein Fehler auf, startet das Programm wieder von vorne. Für das Senden der Statusnachricht ohne Bild wird alle 100 Millisekunden ein Paket vom config-Modul geholt und direkt an das wifi-Modul weitergeleitet. In Abbildung 6.2 ist der Informationsfluss in diesem Modus zu sehen.



**Abbildung 6.2:** Softwarekonzept Kamera inaktiv

Ist die Kamera aktiv, wird das Senden der Statusnachrichten gestoppt. Nun startet das JPEG-Modul, welches zeilenweise Daten vom Kamerasensor holt. Nach je 8 Zeilen wird der erste Datenblock an den wifistream weitergeleitet. So kann noch während des Kompressionsvorgangs mit dem Senden des Bildes begonnen werden. Hat der wifistream ein Paket mit 1400 Byte voll, wird dieses Paket über das wifi-Modul an den Server gesendet. Abbildung 6.3 zeigt die Funktionsaufrufe, die bei diesem Modus aktiv sind.

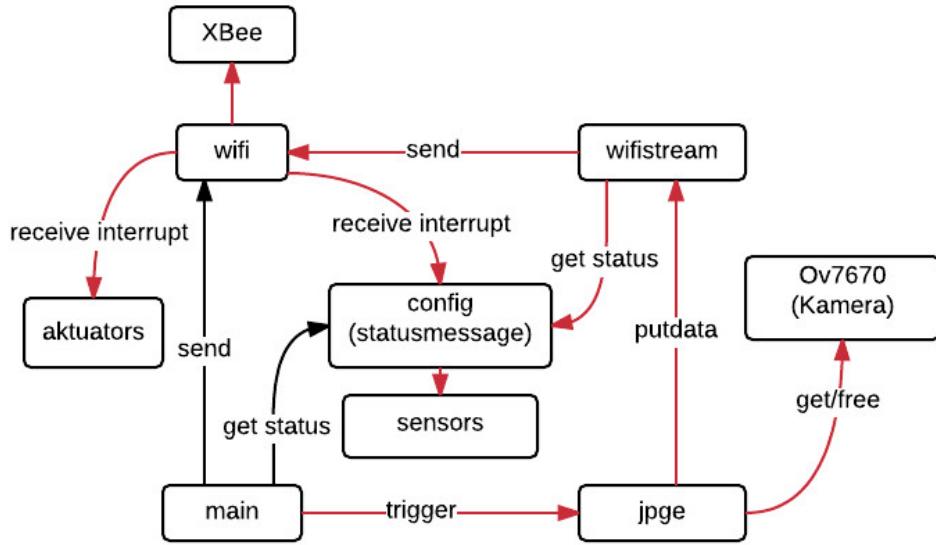


Abbildung 6.3: Softwarekonzept Kamera aktiv

## 6.2 Treiber Implementierung

Zu Beginn wurden alle Hardwarekomponenten wie Motoren, WLAN und Kamera implementiert. Dadurch bekam die Hauptroutine ein gewisses Level an Abstraktion gegenüber der Hardware. Außerdem ließen sich die einzelnen Komponenten unabhängig voneinander testen.

### 6.2.1 Kameratreiber

Der Kameratreiber stellte die größte Herausforderung dar, da das Interface eine sehr hohe Datenrate verarbeiten musste. Außerdem war es durch den begrenzten Arbeitsspeicher nicht möglich komplett Bilder bei hoher Auflösung zwischenzuspeichern, weshalb eine intelligente Pufferung nötig war.

#### Funktionsweise

Der OV7670 Kamerasensor besitzt ein paralleles Videointerface für den Videostream sowie eine serielle SCCB-Schnittstelle, um den Sensor zu konfigurieren. Außerdem muss ein externer Takt an XCLK angelegt werden, damit das Modul funktioniert. Das Videointerface besitzt 3 Kontrollpins

VREF, HREF und PSCLK und 8 Datenpins D0 bis D7. Alle Pins sind im Controller als Ausgang zu definieren. In Abbildung 6.4 und Abbildung 6.5 ist die Funktionsweise des parallelen Interface dargestellt. Wenn VREF einen positiven Puls sendet, startet der Sensor ein neues Bild. Bei jeder steigenden Flanke von HREF beginnt eine neue Zeile des Bilds. Diese wird bis zur fallenden Flanke von HREF übertragen. Während HREF positiv ist, toggelt PCLK und signalisiert so bei jeder fallenden Flanke dem Controller die 8 Datenbits zu lesen. Bei einer YUV Farbkodierung entsprechen 16 Bit je einem Pixel, wobei die ersten 8 Bit die Helligkeit (Y-Kanal) und die restlichen 8 Bit die Farbigkeit (U-/V-Kanal) darstellen.

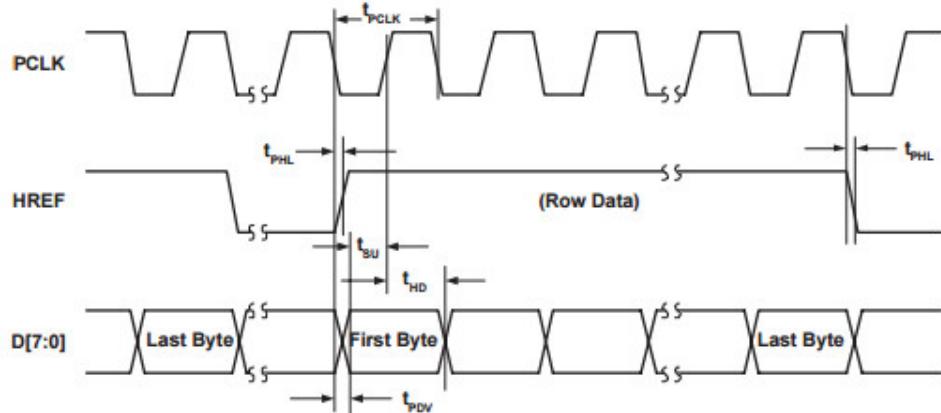


Abbildung 6.4: Videointerface [14]

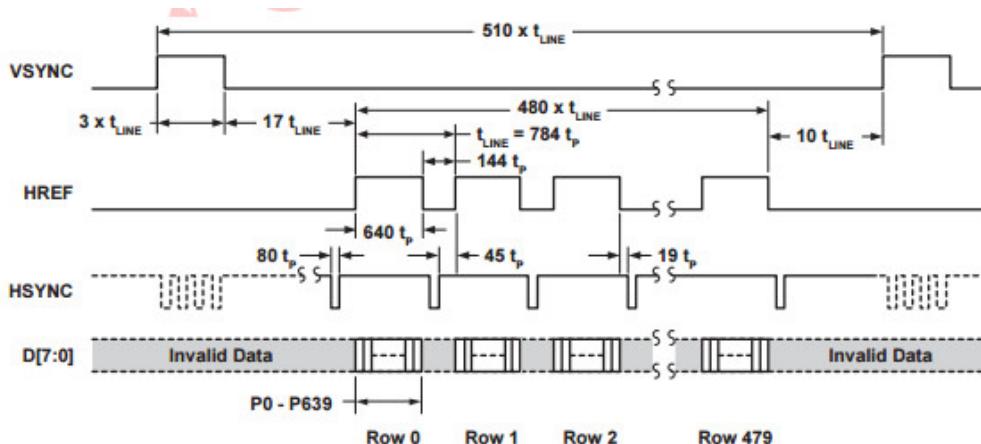


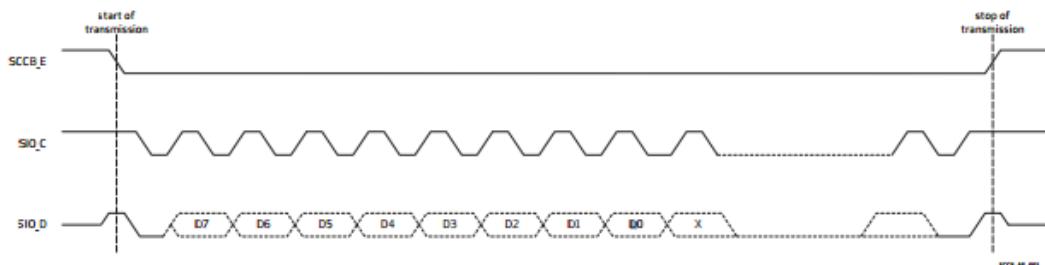
Abbildung 6.5: Videointerface [14]

Um die große Datenmenge ohne maximale Prozessorauslastung in den Arbeitsspeicher schreiben zu können, wurde der DMA-Controller des Teensy verwendet. Mit diesem ist es möglich auf ein bestimmtes Signal, wie einer steigenden Flanke an PCLK, Datentransfer zwischen unterschiedlicher Peripherie ohne Prozessorintervention auszuführen. In diesem Fall werden bei jedem Trigger von PCLK alle 8 Datenbits in einen definierten Speicherbereich geschrieben. Durch Interrupts an HREF und VSYNC wurde der DMA-Controller gesteuert, sodass er nur gültige Daten liest und diese zeilenweise in vorher reservierte Speicherbereiche schreibt.

## 6 Software

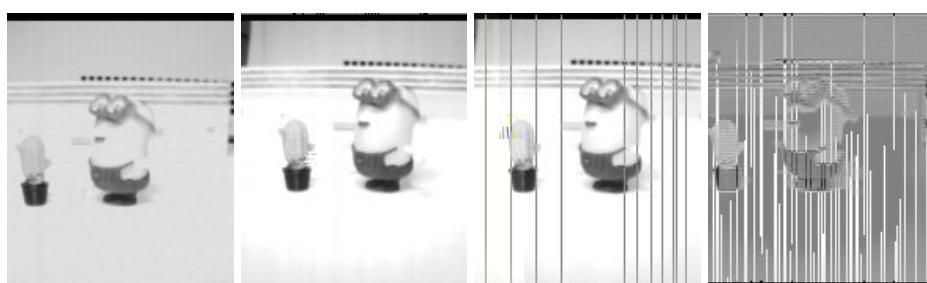
---

Da das Videointerface nicht bidirektional ist, benötigt die Kamera einen weiteren Bus zur Konfiguration. Dafür gibt es den Serial Camera Control Bus (SCCB), welcher vergleichbar mit einem I2C-Bus ist. In Abbildung 6.6 ist der Signalverlauf dargestellt. Er besitzt eine Taktleitung und eine Datenleitung. Da nur eine Kamera verwendet wird, ist der Enable Pin nicht nötig. Bei jeder steigenden Flanke sendet oder liest der Controller ein Bit aus der Datenleitung. Ein Schreibvorgang beginnt mit einem Startbit gefolgt von der Moduladresse 0x42. Daraufhin kann die Adresse des zu schreibenden Registers sowie der zu schreibende Wert gesendet werden. Ein Stopbit beendet die Übertragung. Der Lesevorgang beginnt ebenfalls mit einem Startbit und der Moduladresse 0x42, gefolgt von der zu lesenden Adresse. Jedoch wird nun mit einer Start-Stop-Sequenz fortgefahren, um daraufhin mit dem Senden der Moduladresse 0x43 den Lesebefehl zu geben. Der Kamerasensor übernimmt nun die Datenleitung und überträgt das gefragte Byte. Ein Nackbit gefolgt von einem Stopbit bestätigt den erfolgreichen Lesevorgang und beendet die Übertragung.



**Abbildung 6.6:** SCCB-Interface [13]

Um die Bildrate richtig einzustellen, musste die Frequenz der PCLK konfiguriert werden. Dazu wurden die Register des Prescalers und des Multipliers über den SCCB eingestellt. Der Prescaler teilt den externen Takt XCLK auf eine niedrigere Frequenz, wohingegen der Multiplier den Takt vervielfacht. Mit der richtigen Kombination sowie einer variablen externen Taktquelle ließen sich beliebig viele Frequenzen einstellen. Um herauszufinden, ab welcher Frequenz die Datenübertragung nicht mehr funktioniert, wurden die Frequenzen 0,7 Mhz, 2 Mhz, 4 Mhz und 6 Mhz getestet. Das Ergebnis ist in Abbildung 6.7 zu sehen. Es hat sich gezeigt, dass über den aktuellen Aufbau ab einem Takt von 4 Mhz schon signifikante Pixelfehler auftreten.



**Abbildung 6.7:** Pixelfehler bei 700 Khz, 2 Mhz, 4 Mhz und 6 Mhz

Die Auflösung lässt sich über das Common-Control-Register 7 einstellen. Es gibt die Einstellungsmöglichkeiten CIF mit 352x288 Pixel, QVGA mit 320x240 Pixel, QCIF mit 176x144 Pixel und VGA mit 640x480 Pixel. Für Testzwecke wurde ausschließlich CIF verwendet, da es das einzige Bildformat ist, das vollständig in den Arbeitsspeicher passt. Um auch die anderen Bildformate übertragen zu können, wurde eine Art Ringpuffer erstellt. Mit diesem wird überprüft, welche Zeile als Letztes gelesen wurde, um dann nur die darauf folgenden Zeilen in den Speicher zu schreiben. Da der Videostream von der Kamera nicht unterbrochen werden kann, werden anderen eingehende Zeilen ignoriert. So waren auch Auflösungen wie QVGA und QCIF möglich. Erst bei VGA-Auflösung kam der Encoder an die Grenzen des Arbeitsspeichers. In Abbildung 6.8 sind die verschiedenen Auflösungen zu sehen.



**Abbildung 6.8:** Auflösung CIF, QVGA und QCIF

Da der Ringpuffer des Kameratreibers oft zu Speicherüberläufen geführt hat, wurde ein alternativer Treiber mit gleichem Interface implementiert. Dieser überträgt ein Schachbrettmuster, ohne dabei große Mengen an Arbeitsspeicher zu verwenden. Dies wurde vor allem zum Testen des Videoencoders sowie des Netzwerkprotokolls verwendet.

## 6.2.2 WLAN Treiber

Als WLAN Treiber wurde die Arduino-kompatible Bibliothek "xbee-wifi-spi-arduino" von Chris Bearman verwendet [3]. Um diese nutzen zu können, musste das XBee zuerst richtig konfiguriert werden. Dies geschah über das Webinterface des XBee. Nachdem SPI aktiviert war, konnten die übrigen Parameter über dieses Interface verändert werden. Eine Übersicht über die wichtigsten Parameter des XBee ist in Tabelle 6.1 zu sehen.

Info	Name	Wert
API enable	AP	1 = API-enabled
Netzwerk Typ	AH	2 Infrastructure
IP Modus	MA	0 DHCP
SSID	ID	welcome
Verschlüsselung	EE	0 = keine
Firmware Version	VR	2021
Hardware Version	HV	2730

**Tabelle 6.1:** XBee Konfiguration

Damit die verwendete Bibliothek richtig funktioniert, musste die API des XBee aktiviert werden. Das Modul verbindet automatisch mit dem Accesspoint "welcome", sobald die SSID und die Verschlüsselung richtig eingestellt ist. Bei erfolgreichem Verbindungsauflaufbau ändert es seinen Status auf Verbunden, woraufhin der Controller anfängt, Pakete zu senden.

## Testroutinen

Um die maximale Datenrate des Xbee zu erreichen, wurden einige Testroutinen geschrieben. Getestet wurde zum einen das Senden von zufälligen Bytereihen mit unterschiedlichen Längen. Eine dieser Testreihen ist in Tabelle 6.2 zu sehen.

Paketgröße in Byte	Datenrate in Kbit/s
1	2
2	4
4	8
8	17
16	34
32	67
64	121
128	213
256	347
512	499
1024	635

**Tabelle 6.2:** Datenrate bei verschiedenen Paketgrößen

Man kann erkennen, dass die Datenrate mit der Größe der Pakete steigt. Sendet man die maximale Paketgröße (1400 Byte), die das XBee verarbeiten kann, so erreicht man eine Datenrate von 900 Kbit/s, was der maximalen Übertragungsgeschwindigkeit aus dem Datenblatt entspricht. Aus diesem Grund werden die Bilddaten in Paketen mit jeweils 1400 Byte gesendet.

### 6.2.3 Motortreiber

Der Chip L298 hat 4 digitale Eingänge zur Steuerung der H-Brücken. Je Motor werden 2 Pins benötigt. Soll der jeweilige Motor nach vorne fahren, wird Pin 1 angesteuert, bei der Gegenrichtung Pin 2. Sind beide Pins auf 0, dreht der Motor nicht. Durch Pulswidtemodulation lässt sich die Geschwindigkeit des Motors einstellen. Aus diesem Grund wurden die Pins des Motortreibers auf PWM-Ausgänge des Teensy gelegt. So kann das Signal ohne Prozessorintervention aktiv bleiben. Die Implementierung der Motorfunktionen bestand aus einer Reihe von Aufrufen einer Arduino-Bibliothek des Teensy. An Punkten wie diesen zeigte sich der Vorteil von Arduino. Um die Motorsteuerung etwas intelligenter zu gestalten, wurden außerdem einige Einstellungsmöglichkeiten wie Minimal- und Maximalgeschwindigkeit implementiert

## 6.3 Video Encoder

Die Wahl des Video-Encoding-Algorithmus fiel auf JPEG. Es ist klar, dass dies nicht die effizienteste Möglichkeit der Videocodierung darstellt, jedoch ist es einfach umzusetzen, benötigt nicht viel Arbeitsspeicher auf der Seite des Encoders und hat trotzdem eine ausreichend große Kompressionsrate. Eine MPEG-Implementierung mit Codierung über mehrere Frames ist aufgrund des kleinen Arbeitsspeichers nicht möglich. Als JPEG-Encoder wurde der "jpeg-compressor" von Rich Geldreich verwendet [9]. Dieser codiert rohe Pixel Arrays in JPEG-Dateien. Dadurch wird das Auswerten der Bilddaten wesentlich vereinfacht. Da die Kamera Pixelfarben mit YUV-Codierung überträgt, mussten die Bilddaten noch nachbearbeitet werden, bevor sie codiert werden konnten. Für Schwarzweiss musste nur der Y-Anteil betrachtet werden. Dies wurde erreicht, indem jedes zweite Byte im Bildpuffer ignoriert wird. Der so gefilterte Puffer konnte nun zeilenweise dem Encoder übergeben werden, der ihn dann codiert an die Netzwerkschnittstelle weiterreicht.

## 6.4 Netzwerkprotokoll

Wie schon zu Beginn erwähnt wurde, ist die Schnittstelle zum Roboter das Netzwerkprotokoll über ein WLAN-Netz. Das Protokoll ist bewusst simpel gehalten, um Missverständnisse zu vermeiden. Es besteht aus einer einfachen Verbindung über UDP an Port 44044. Dabei gibt es zwei Typen von Paketen. Zum einen ist dies die Statusmessage vom Roboter zum Server und zum anderen die Command-Message vom Server zum Roboter. Der Verbindungsauflauf wird durch den Roboter eingeleitet, welcher zu einer festen IP-Adresse sendet. Die Command-Message ist 6 Byte lang und ist wie in Tabelle 6.3 aufgebaut.

<b>name</b>	<b>Bytenummer</b>	<b>Bsp</b>	<b>Wertebereich</b>	<b>Beschreibung</b>
Startbyte	1	0xff	0xff	Start des Pakets
Kamera	2	0x01	0x00,0x01	Aktiviert den Kamerastream
Kick	3	0x00	0x00,0x01	Löst den Schuss aus
Motor links	4	0x32	-100..+100	Motorgeschwindigkeit in Prozent
Motor rechts	5	0x32	-100..+100	Vorzeichen gibt die Richtung an
Checksum	6	0x9A	0-0xff	s 0xff-(Byte2+Byte3+Byte4+Byte5)

**Tabelle 6.3:** Command-Message

Die Status-Message hat zwei verschiedene Größen. Bei inaktiver Kamera sind es 12 Byte, bei Aktiver Kamera 1500 Byte. Die 1500 kommen von der maximalen Paketlänge, die das Xbee senden kann. Die ersten 12 Byte haben bei beiden Paketen die gleiche Bedeutung. Die restlichen Bytes des 1500 Byte Paket sind Bilddaten. Die Bilddaten bestehen aus einem fortlaufenden Bytestrom von JPEG Dateien.

<b>name</b>	<b>Bytenummer</b>	<b>Bsp</b>	<b>Wertebereich</b>	<b>Beschreibung</b>
Roboter ID	1	0x01	1,2	feste ID des Roboters
Akkustand	2	0x09	0-100	Akkustand in Prozent
IR-Sensor	3	0x00	0,1	1 wenn er das Tor sieht, sonst 0
Kompass-Sensor	4,5	0x00	-	nicht implementiert
X-Koordinate	6,7	0x00	-	nicht implementiert
Y-Koordinate	8,9	0x00	-	nicht implementiert
Fehlercode	10	0x00	0,1,2	1 Kamerafehler, 2 Akkufehler
Bildlänge	11,12	1388	0-1388	immer 0 oder 1388
(optional) Bilddaten	13..1400		0x00 - 0xff	jedes neue Bild beginnt mit 0ffd8

**Tabelle 6.4:** Status-Message

## 6.5 Videotest

Leider konnte mit der Software die gewünschte Bildrate von 24 fps nicht erreicht werden. Um die Schwachstelle einzuschränken, war eine Routine für die Kalkulation der Datenrate und Bildrate notwendig. Nach jedem erfolgreich codierten und gesendeten Bild werden diese Werte von der Testfunktion neu kalkuliert. In Tabelle 6.5 ist die Datenrate vor und nach dem Codieren, sowie die Bildrate, mit der die Kamera sendet und die tatsächlich übertragenen Bildrate zu sehen. Getestet wurde unter Standardeinstellungen, mit deaktivierter WLAN-Verbindung, mit deaktiviertem JPEG-Encoder sowie mit angepasster PCLK-Frequenz.

Konfiguration	PCLK Frequenz	Gesendete FPS	Kamera FPS	Kamera Data	JPEG Data
Standard	3 Mhz	1,51 fps	15 fps	302,77 Kbit/s	49,83 Kbit/s
WLAN aus	3 Mhz	1,69 fps	15 fps	339,10 Kbit/s	53,71 Kbit/s
JPEG aus	3 Mhz	1,53 fps	15 fps	305,86 Kbit/s	301,13 Kbit/s
langsame PCLK	0,75 Mhz	2,26 fps	3,7 fps	452,22 Kbit/s	58,22 Kbit/s

**Tabelle 6.5:** Videostreamgeschwindigkeit unter verschiedenen Bedingungen

Aus der Kameradatenrate und der JPEG-Datenrate lässt sich die gewünschte Kompressionsrate von 1,2 Bit/Pixel herauslesen. Man erkennt auch, dass bei deaktiviertem WLAN die Datenrate um fast 10% steigt. Außerdem wird bei deaktiviertem JPEG-Encoder die maximale Datenrate von 900 Kbit/s nicht erreicht, was zeigt, dass der WLAN-Treiber nur bei maximaler Prozessorauslastung eine maximale Datenrate erreicht. Ein WLAN-Treiber, der die SPI-Schnittstelle über einen DMA-Controller liest und schreibt, würde dieses Problem beheben. Die Implementierung eines solchen Treibers stellte sich jedoch als zu aufwendig heraus. Interessant ist auch, dass durch eine niedrige PCLK-Frequenz und der daraus folgenden Entlastung des Kameratreibers die Bildrate steigt, was vermuten lässt, dass auch der Kameratreiber trotz DMA-Controller einen signifikanten Anteil der Prozessortakte in Anspruch nimmt. Der Kameratreiber lässt sich jedoch nur durch einen schnelleren Chip oder ein Chip mit integriertem Kamerainterface verbessern. Ein weiterer Punkt mit Optimierungsmöglichkeiten wäre der JPEG-Encoder. Durch feste Huffman-Tabellen und Optimieren der JPEG-Header ließen sich die Takte pro Bildkompression wesentlich verringern. Dies würde jedoch eine komplexere Schnittstelle sowie eine neue Implementierung der verwendeten JPEG-Bibliothek erfordern.

## 7 Diskussion

Die aktuelle Version des Roboters wird fast allen Kriterien der Aufgabenstellung gerecht. Jedoch ist durchaus noch Optimierungspotenzial vorhanden. Die Maximalgeschwindigkeit des Roboters ist so hoch, dass er nur gedrosselt sinnvoll steuerbar ist. Probleme zeigen sich bei niedrigen Geschwindigkeiten, da die Anfahrspannung beider Motoren unterschiedlich ist. Dieses Problem ließe sich durch die Regelung der Motoren mittels Radencoder lösen, welche im aktuellen Design schon vorgesehen sind. Der Schuss reicht aus, um einen Tischtennisball über das Spielfeld zu schießen. Im Vergleich zu der Geschwindigkeit des Roboters macht es jedoch wenig Sinn, ihn im Spiel zu verwenden. Eine höhere Spannung oder ein spezieller Magnet mit dickeren Wicklungen könnte die Schusskraft verstärken. Das Laden an einer Ladestation funktioniert theoretisch, jedoch wurde noch kein Langzeittest mit mehreren autonomen Ladezyklen durchgeführt. Die Akkulaufzeit bei stehendem Roboter beträgt ungefähr 4 Stunden. Fährt der Roboter durchgehend und schießt ab und an ist der Akku schon nach 30 Minuten leer. Geht man davon aus, dass ein durchschnittliches Spiel nicht länger als 10 Minuten dauert, ist das eine gute Zeit. Bei der Software wurden alle notwendigen Funktionen implementiert. Nur die Videoübertragung erreicht nicht die gewünschte Datenrate. Eine Optimierung der Software an verschiedenen Punkten wie WLAN-Treiber und Kameratreiber ist ein möglicher Ansatz, um eine bessere Latenz zu erreichen. Jedoch liegt am meisten Potenzial in einer effizienteren Videokodierung.

## 8 Anhang

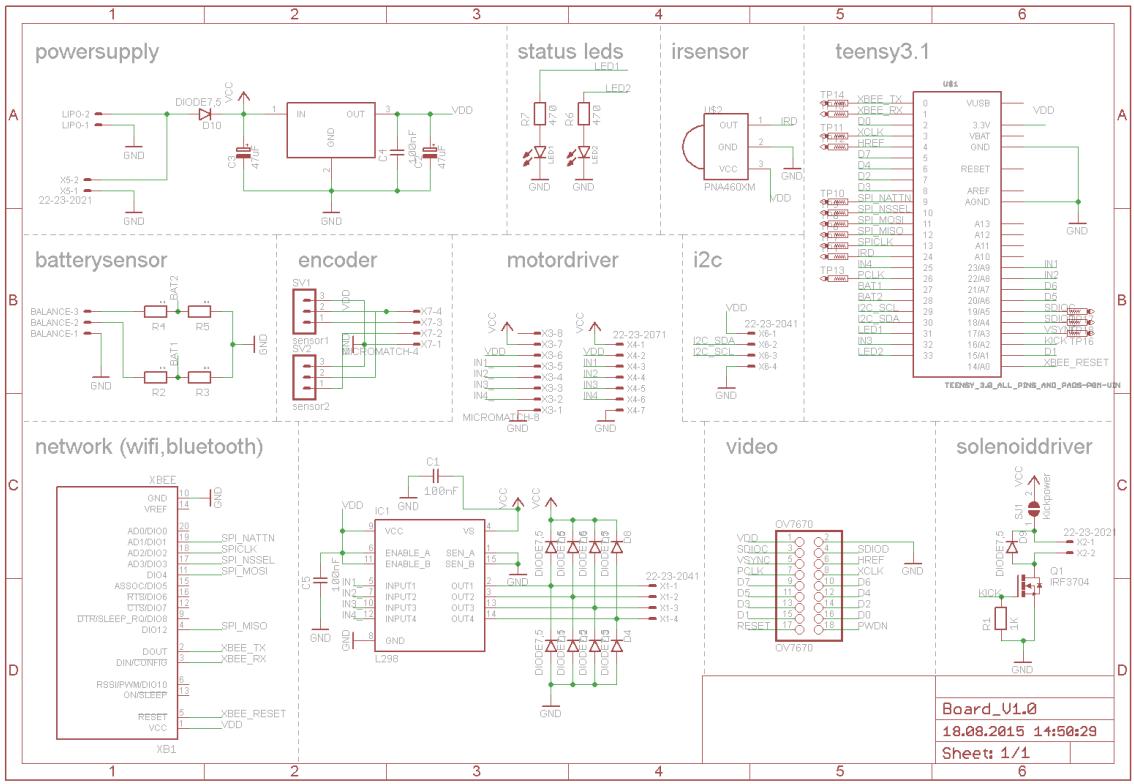


Abbildung 8.4: Schaltplan der Platine

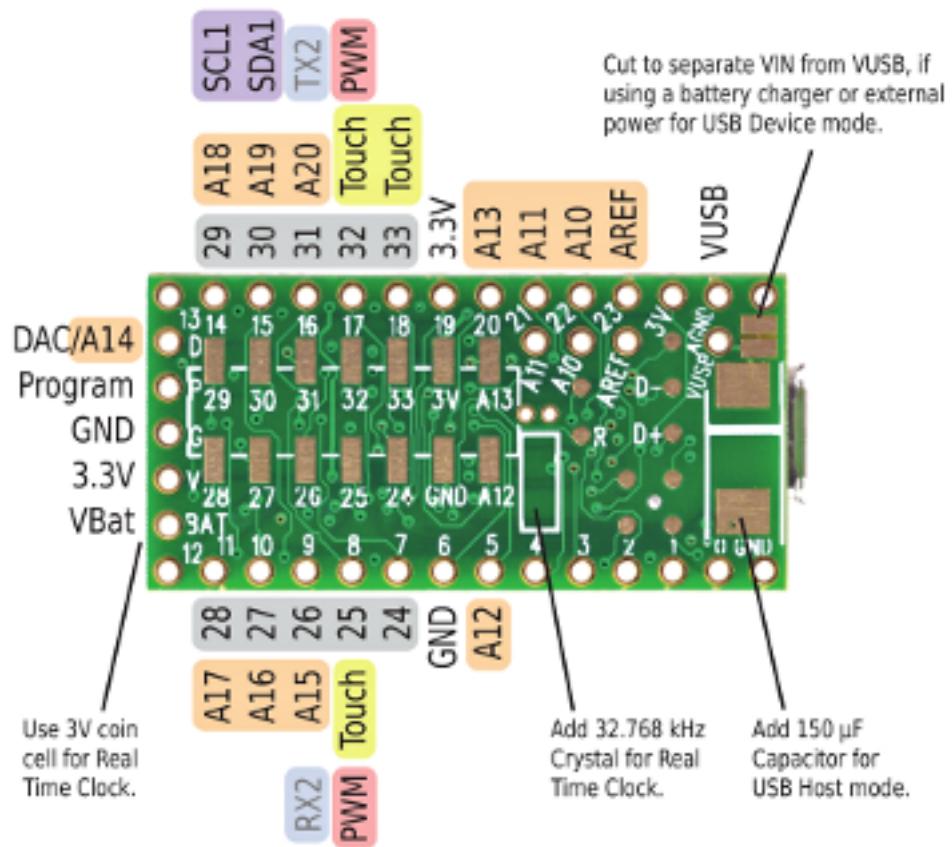


Abbildung 8.1: Teensy 3.1 Pinbelegung vorne [15]

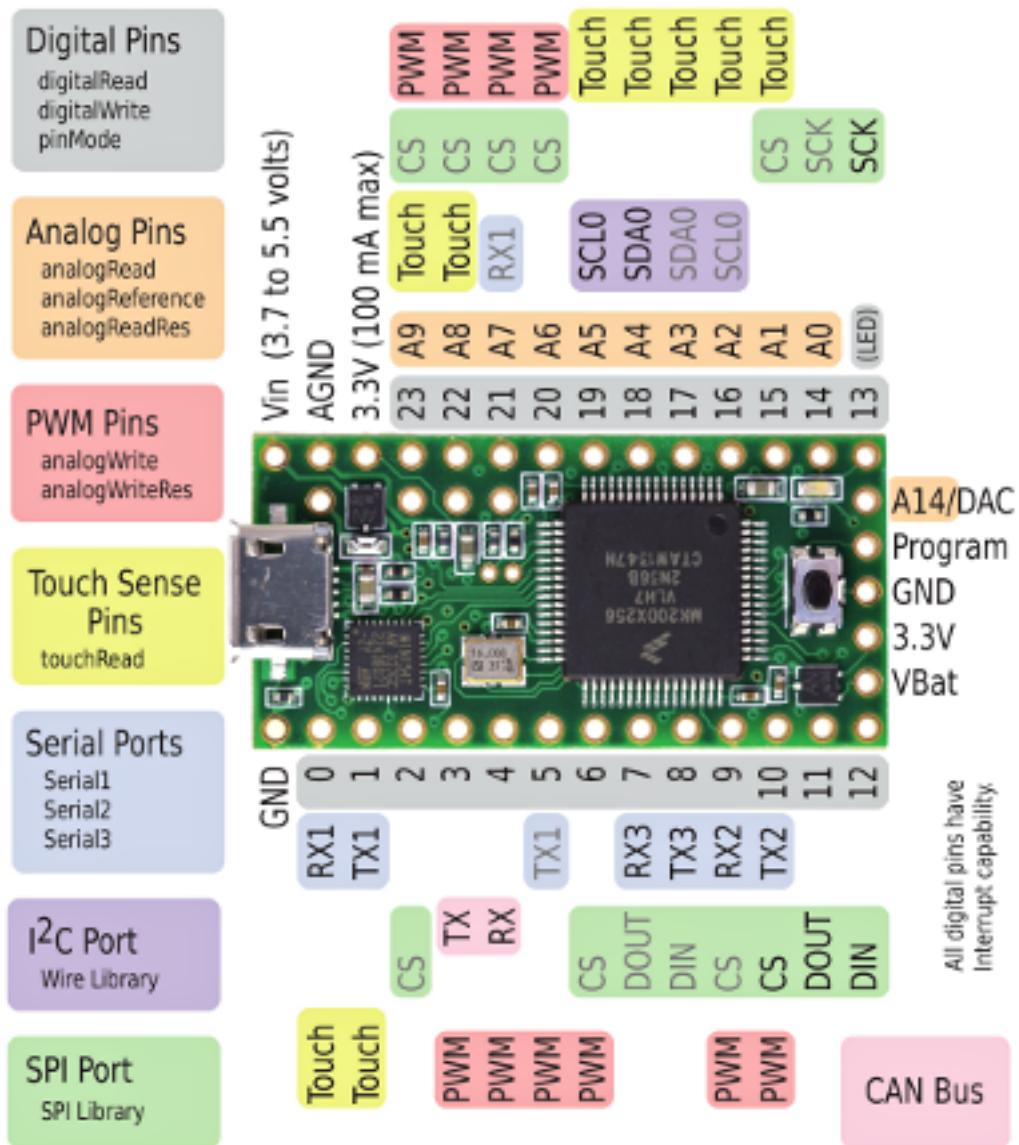
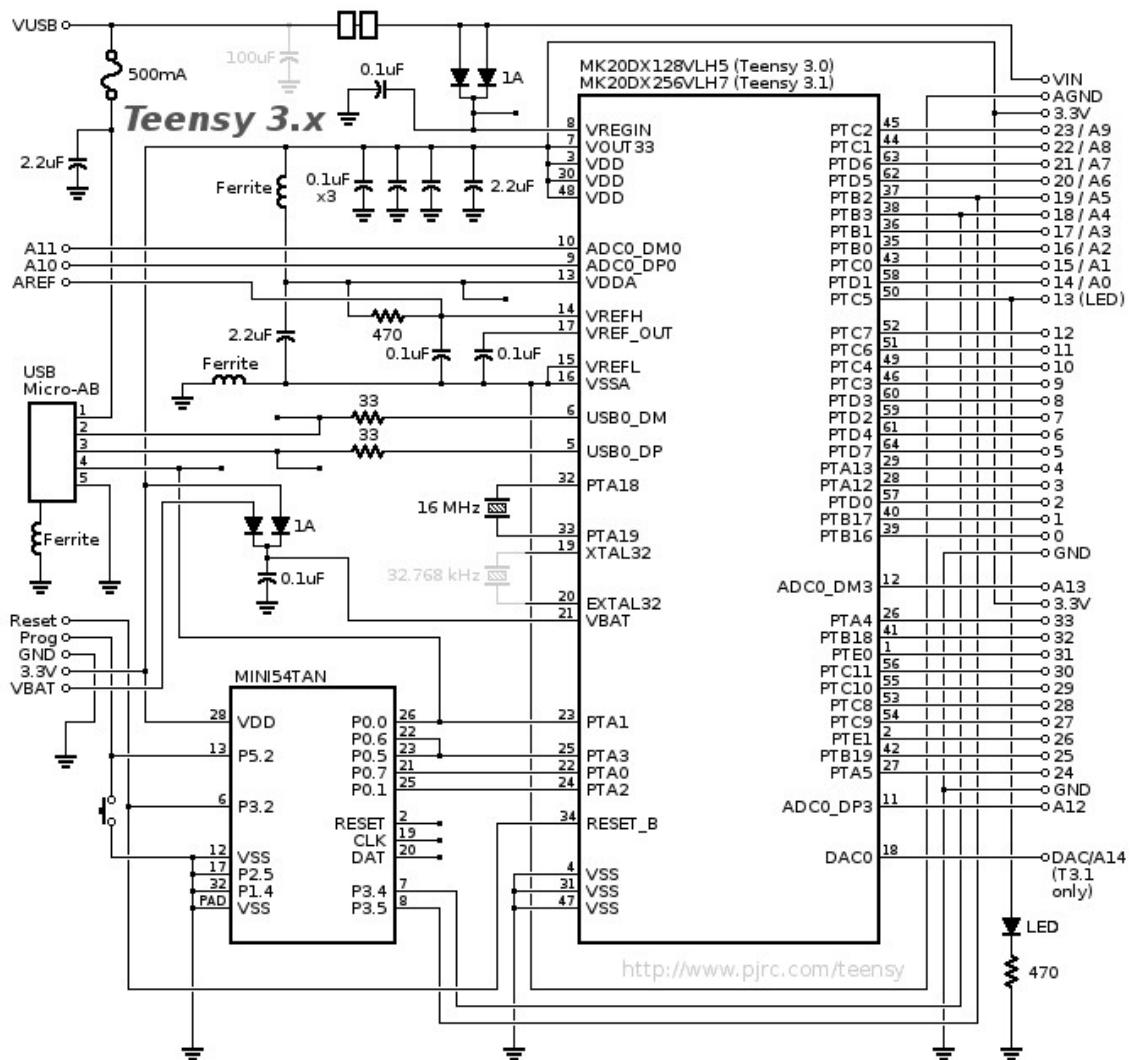


Abbildung 8.2: Teensy 3.1 Pinbelegung hinten [15]



**Abbildung 8.3:** Teensy 3.1 Schaltplan [15]



# Literaturverzeichnis

- [1] ov7670 camera sensor. <http://www.rpg.fi/desaster/blog/2012/05/07/ov7670-camera-sensor-success/>, Oktober 2015.
- [2] Jorge Aparicio. hacking the OV7670 camera module. <http://embeddedprogrammer.blogspot.de/2012/07/hacking-ov7670-camera-module-sccb-cheat.html>, Oktober 2015.
- [3] Chris Bearman. xbee-wifi-spi-arduino. <https://github.com/cjbearman/xbee-wifi-spi-arduino>, 2012.
- [4] BC118 Datasheet. Blue Creation, 2015.
- [5] BW. Photon Datasheet. <https://docs.particle.io/datasheets/photon-datasheet/>, April 2015.
- [6] Datenblatt LiPos energy 40C. Conrad Electronic, 2015.
- [7] XBee Manual. Digi International, 2015.
- [8] K20 Sub-Family Reference Manual. Freescale Semiconductor, 1.1 edition, 2012.
- [9] Rich Geldreich. jpeg-compressor. <https://code.google.com/p/jpeg-compressor>, Mai 2012.
- [10] Datenblatt ITS-LZ-1949. Intertec Components GmbH, 2015.
- [11] LeafLabs. Maple Mini. <http://leaflabs.com/docs/hardware/maple-mini.html>, Januar 2014.
- [12] mbed. mbed NXP LPC1768 prototyping board. <https://www.mbed.com/>, September 2009.
- [13] Serial Camera Control Bus (SCCB) Functional Specification. OmniVision Technologies, Inc., 2.2 edition, 2007.
- [14] Datenblatt OV7670. OmniVision Technologies, Inc., 1.01 edition, 2008.
- [15] Robin Coon Paul Stoffregen. Teensy 3.1. <http://www.pjrc.com>, Oktober 2015.
- [16] Datenblatt 50:1 Micro Metal Gearmotor. Pololu Corporation, 2015.
- [17] RN-42/RN-42-N Datasheet. Roving Networks, 3.1 edition, 2010.
- [18] Datenblatt L298. STMicroelectronics, 2000.
- [19] Dirk Schlender Tobias Schmidt. Technical report, Bergische Universität Wuppertal, 2003.



Name: Alexander Ulbrich

Matrikelnummer: 754343

---

**Erklärung**

Ich, Alexander Ulbrich, Matrikelnummer 754343, erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den ..... .

Alexander Ulbrich