Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Mikko Eberhardt

Line Tracking für autonomes Fahren von Modellfahrzeugen mittels eines 2D-Laserscanners

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Mikko Eberhardt

Line Tracking für autonomes Fahren von Modellfahrzeugen mittels eines 2D-Laserscanners

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Pareigis
Zweitgutachter: Prof. Dr. Lehmann

Eingereicht am: 24.10.2016

Mikko Eberhardt

Thema der Arbeit
Line Tracking für autonomes Fahren von Modellfahrzeugen mittels eines 2D-Laserscanners

Stichworte
Line Tracking, autonomes Fahren, Laserscanner, Pegelwandler, Logischer Schalter, Modell LKW

Kurzzusammenfassung
Dieses Dokument beschreibt den Arbeitsschritte an einem manuell gesteuerten Modellfahrzeug, hin zu einem Fahrzeug, das autonom auf einer Linie fährt, wobei ein 2D-Laserscanner eingesetzt wird. Eine Umschaltfunktion an der Fernbedienung gewährleistet, dass das Fahrzeug auch weiterhin manuell gesteuert werden kann.

Mikko Eberhardt

Title of the paper
Line tracking for autonomous driving of remote controlled vehicle by use of a 2D laserscanner

Keywords
Line Tracking, autonomous drive, laserscanner, logicleaver, remote controll

Abstract
This document discribes the workflow from an manual steering model vehicle to an autonomous drive on a line on the floor. With use of an 2D laser scanner to find the line. An leaver is used to change the mode from manual to autonomous driving.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Quellcode Auszüge

1 Einleitung

Modell LKWs sind ein beliebtes Hobby, werden aber auch des öfteren zur Simulation und zum Testen im kleinen Maßstab verwendet. Das Physiklabor der HAW in Bergedorf verwendet einen Modell LKW. Dort sollen Studenten im Rahmen eines Physikpraktikums mit diesem LKW eine Strecke abfahren wobei Sensoren auf dem Anhänger Daten während der Fahrt sammeln. Diese Daten müssen die Studenten im Nachhinein analysieren. Es ergeben sich bei jeder Fahrt andere Werte, da bei manueller Steuerung immer wieder andere Parameter, wie GEschwindigkeit und Strecke vorliegen. Um reproduzierbare Werte zu erhalten, soll der LKW autonom eine vorgegebene Strecke abfahren. Diese Arbeit beschäftigt sich mit der Entwicklung einer Steuerung hierfür. Zu der Entwicklung gehören sowohl Hardware als auch Software Aspekte. Die Entwicklung der Steuerungssoftware wird vom Erkennen der Strecke, über die Auswertung der Daten bis hin zur Steuerung von Geschwindigkeit und Lenkung detailliert beschrieben.

1.1 Motivation

Eine Teststrecke für den LKW kann manuell abgefahren werden um die Sensordaten zu sammeln. Die ermittelten Daten sollen von den Studenten des Physiklabors in Bergedorf mit vorangegangenen Messungen analysiert und verglichen werden. Die manuelle Steuerung bietet gravierende Nachteile:

- Die Strecke ist bei jeder Fahrt anders
- Die Geschwindigkeit ist nicht konstant
- Vergleich der Messwerte nur bedingt möglich

Diese Probleme, die eine aussagekräftige Analyse der Messdaten verhindern, sollen mit einer Fahrsteuerung gelöst werden. Indem der LKW einer vorgegebenen Strecke folgt, sollen die Daten reproduzierbar werden. Die Steuerungssoftware, welche den LKW autonom über die Strecke steuern soll, muss selbst auf Ereignisse reagieren können. Damit ist nicht nur das Lenken wenn der LKW sich von der Strecke entfernt, abgedeckt, sondern auch Fehlerfälle

abgefangen. Hält der LKW an, wenn er von der Strecke abkommt, oder stoppt er vor einem Hindernis?

1.2 Zielsetzung

Es soll eine Fahrsteuerung entwickelt werden. Diese Fahrsteuerung soll mit einem 2D-Laserscanner eine Linie erkennen und den LKW auf dieser Linie entlang fahren lassen. Zur Lösung muss das System vollständig analysiert werden und ein passender Entwurf entwickelt werden. Durch die zu entwickelnde Fahrsteuerung, sollte der LKW nach dem Start der Software selbständig eine vorgeschriebene Strecke abfahren können. Dann kann das Augenmerk während der Fahrt wie beabsichtigt auf der Sensorik auf dem Anhänger des LKW liegen und das Fahrzeug muss nicht manuell über die Strecke gesteuert werden.

1.3 Aufbau

Der Aufbau orientiert sich an den Schritten, die notwendig sind, um eine Fahrsteuerung zu gestalten, die aus einzelnen Komponenten zusammengestellt ist. Alle notwendigen Grundlagen, die notwendig sind, um ein Gesamtverständnis zur Steuerung zu ermöglichen, werden im Kapitel 2 Grundlagen erläutert. Nach einer Analyse in Kapitel 3 wird in Kapitel 4 ein Entwurf erstellt, der gleichzeitig die Veränderungen an der bestehenden Architektur des LKW aufzeigt. Danach erfolgt in Kapitel 5 der Aufbau der Software und Hardwarekomponenten. Die einzelnen Komponenten werden dann in Kapitel 6 getestet und überprüft. Im Kapitel 7 erfolgt zum Abschluss ein Fazit über die erreichte Funktionalität, mögliche Erweiterungen und Verbesserungen des Systems.

2 Grundlagen

Dieses Kapitel behandelt alle Grundlagen die zum Verständnis dieser Arbeit notwendig sind und für die Technologien, die eingesetzt werden. Gestartet wird mit einem Überblick über Modelle und deren Funktionen. Danach folgt ein kurzer Ausblick, wo konkret der LKW, der in dieser Arbeit verwendet wird, eingesetzt werden soll und wie die Umgebungsverhältnisse sind. Zum Ende werden alle technischen Grundlagen erläutert die im Rahmen dieser Arbeit Verwendung finden. Ein Einblick in die embeddet Linux Umgebung soll zusätzlich dem Verständnis dienen.

2.1 Modellfahrzeuge

Modellfahrzeuge gibt es in vielfachen Ausführungen, Modellen und Maßstäben. Ob als Spielzeug für kleine Kinder oder für den erfahrenen Bastler sind Autos ein sehr beliebtes Hobby. Die meisten Fahrzeuge laufen mit einem Elektromotor, es gibt aber auch welche mit Benzinmotor. Die Maßstäbe bewegen sich im Bereich von $1:6$ bis $1:24$, zusätzlich gibt es viele Sonderbauten die sich im Maßstab von zB. $1:87$ oder kleiner befinden. Der LKW, der in dieser Arbeit zur Anwendung kommt ist im Maßstab $1:14$.
Alle Fahrzeuge besitzen eine zentrale Steuereinheit, welche die Signale vom Empfänger auswertet und entsprechend die Aktoren steuert. Die Art der Steuerung unterscheidet sich ebenfalls. Analoge Fernsteuerungen senden mit einer Frequenz um $40MHz$. Diese Frequenz wird durch ein wechselbaren Quartz leicht in den Frequenzbändern geändert, damit zwei Fahrzeuge sich nicht gegenseitig auf einer Frequenz stören. Digitale Fernsteuerungen senden im $2.4GHz$ Bereich. Dort ist ein Verändern der Frequenz nicht notwendig, da die Frequenzen zwischen Bandbreiten hin und her springen. Dieses Springen zwischen den Frequenzbändern macht der Empfänger direkt mit. Durch einen Algorithmus haben Empfänger und Sender eine Sprungreihenfolge durch die Frequenzbänder. Durch ein anfängliches Pairing zwischen Sender und Empfänger synchronisieren die beiden sich und haben danach die gleiche Sprungreihenfolge, wodurch keine Störungen im Empfang auftreten. Selbst wenn zwei Sender auf einer Frequenz direkt zusammen senden, ab dem nächsten Sprung der Frequenzbänder ist wieder nur der eine Sender vorhanden, da andere Sender eine andere Abfolge von Sprüngen in den Frequenzbändern hat.

2.1.1 Stromversorgung

Modellfahrzeuge mit einem Elektromotor verfügen über einen Akku als Energiequelle. Bei dem in dieser Arbeit verwendeten LKW Modell kommt ein Akku der Marke LRP mit 5000 mAh zum Einsatz. Dieser ist notwendig um genügend Strom für alle Funktionen des LKW zu liefern. Für Modellfahrzeuge, die einen Benzinmotor besitzen reicht ein Akkupack, bestehend aus vier AA Batterien, um den Empfänger mit Strom zu versorgen und die Signale für den Gas- und Lenkservo zu liefern.

2.2 Bewegung & Sonderfunktionen

Bewegungen und Sonderfunktionen werden über PPM-Signale (siehe Abschnitt 2.5.5) an die Steuereinheit gesendet. Abbildung 2.1 zeigt eine Fernsteuerung und deren Steuerhebel. Die Pfeile markieren die möglichen Richtungen der Steuerhebel. Die hier zur Anwendung kommende Fernsteuerung ist eine Spectrum Dx18t. Diese besitzt 18 verwendbare Kanäle. Die Schalter, Knöpfe und Hebel auf den Wechselmodulen (Hier im Bild L2 und R3) sind frei über das Menü der Fernsteuerung programmierbar.

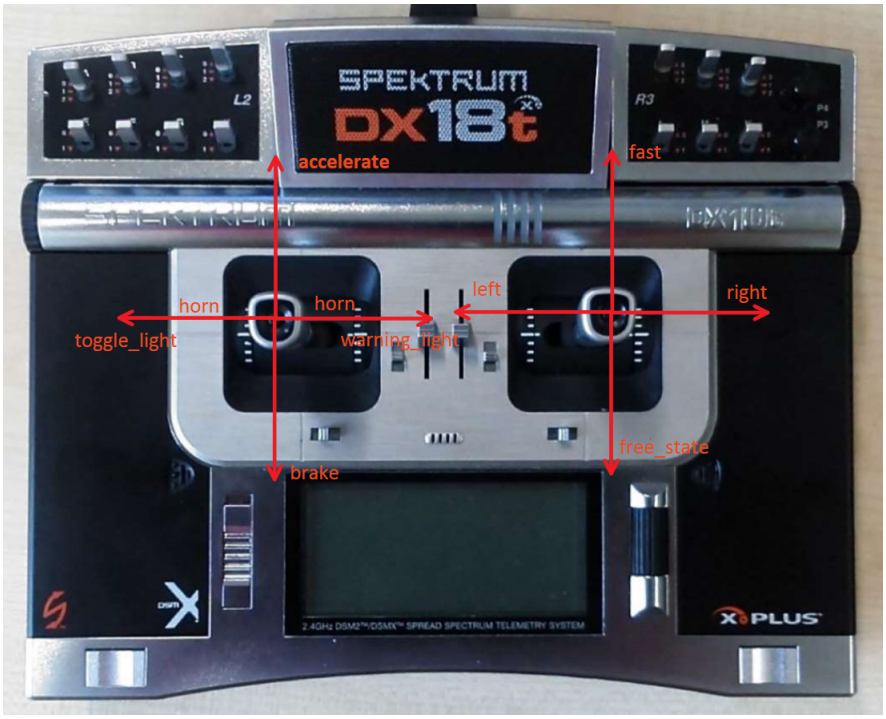

Abbildung 2.1: Eine Fernsteuerung mit den möglichen Bewegungen der Steuerhebel

2.2.1 Lenken

Gelenkt wird über die Fernsteuerung mit dem rechten Steuerhebel in der horizontalen Ebene. Dabei ist den Lenkeinschlag proportional zur Stellung des Steuerhebels. Bei halben Einschlag am Steuerhebel, ist der Lenkeinschlag beim LKW ebenfalls die Hälfte. Stellt man den Hebel auf maximalen Einschlag (bis auf Anschlag), so wird zusätzlich der Blinker für diese Richtung von der Steuereinheit im LKW dazugeschaltet. Umwelteinflüsse, wie ein glatter Boden lassen die Räder beim Einlenken leicht rutschen. Wichtig für die Steuerung ist zudem, dass der LKW bei einer Kurve auf der Strecke ebenfalls eine Kurve fahren kann.

2.2.2 Beschleunigen

Gas wird gegeben, indem der linke Steuerhebel nach oben gedrückt wird. Nach unten wird die Bremse aktiviert. Beim erneuten Drücken der Bremse wird der Rückwärtsgang aktiviert. Die Steuereinheit schaltet ein weißes Rückfahrlicht an. Abhängig vom Getriebe, welches sich vor dem Motor befindet, fährt der LKW im ersten oder im zweiten Gang. Auch ein Leerlauf kann über den rechten Steuerhebel durch Drücken nach unten eingestellt werden. In Ruhestellung des rechten Steuerhebels fährt der LKW im ersten Gang. Nach Oben wird in den zweiten Gang geschaltet. Für die Entwicklung der angestrebten Steuerungssoftware ist zu bedenken, dass der LKW, abhängig vom Gewicht der Zuladung, beim Beschleunigen durchdrehende Reifen haben kann. Auch eingestellte Geschwindigkeiten verhalten sich unterschiedlich mit der Beladung des LKW.

2.2.3 Beleuchtung

Beim linken Steuerhebel werden bei Vollausschlag nach links die drei Stufen der Beleuchtung durchgeschaltet. Stufe 1 schaltet vorne zwei kleine Lampen ein. Ein erneutes Drücken schaltet die zweite Stufe, ein wo zwei zusätzliche Lampen leuchten. Bei der dritten Stufe werden zwei weitere Lampen dazugeschaltet und der LKW hat volle Beleuchtung. Um alle Stufen durchzuschalten, muss der Steuerhebel zwischen den Stufen einmal wieder auf Ruhestellung in die Mitte gebracht werden. Bei allen drei Stufen leuchtet am LKW nur ein Rücklicht. Drückt man den Steuerhebel nach rechts, so kann man den Warnblinker ein- und aus-schalten. Zusätzlich besitzt der LKW oben auf der Fahrzeugkabine zwei gelbe Warnlichter. Diese können ebenfalls “Ein“ und “Aus“ geschaltet werden. Diese Warnlichter sind nicht an die Steuereinheit des LKW angeschlossen und werden über einen separaten Schalter geschaltet. Eine $9V$ Blockbatterie versorgt diese Warnlichter mit Strom.

2.2.4 Signalhorn

Bei der Hupe gibt es zwei unterschiedliche Varianten. Einmal eine kurzer Hupton und einen anhaltenden Hupton. Aktiviert wird die Hupe über den Steuerhebel, der auch die Lichter steuert, allerdings wird die Hupe bei halber Hebelstellung aktiviert. Wenn man den linken Steuerhebel zur Hälfte nach links drückt, so erklingt ein kurzer Ton. Drückt man den Steuerhebel nach rechts zur Hälfte, dann ertönt der anhaltende Ton wie beim dauerhaften Drücken einer LKW Hupe. So liegen zwei Funktionen auf einem Steuerhebel und die Auswahl, welche Aktion ausgeführt wird entscheidet dabei der Winkel, mit dem der Steuerhebel gedrückt wird. Die Auslenkung des Steuerhebels wird durch die Fernsteuerung an den Empfänger gesendet, die Steuereinheit hinter dem Empfänger wertet die PPM Signale (siehe Abschnitt 2.5.5)aus. Die Steuereinheit stellt dann die Aktoren nach den gesendeten Signalen ein.

2.3 Einsatzzweck

Der Abschnitt Einsatzzweck beschreibt wo der zur Anwendung kommende LKW eingesetzt wird und für welche Szenarien. Die Einsätze des LKW sind dabei unabhängig von autonomem und manuellem Betrieb. Der LKW hat zwei Betriebsmodi und kann sowohl autonom Fahren, als auch manuell von Hand über die Fernsteuerung gesteuert werden. In alles Szenarien können beide Betriebsmodi verwendet werden. Der Einsatzzweck bestimmt die Wahl des Betriebsmodi.

2.3.1 Die Messen

Der LKW ist gesponsert von der Firma BFSV Hamburg Institute of Packaging GmbH und soll auf Messen als Anschauungsobjekt dienen. Beladen mit unterschiedlichem Stückgut möchte die Firma ihre Arbeit präsentieren und veranschaulichen. Dazu soll eine Strecke mit einer Linie aus einem Klebestreifen frei gebaut werden können um den kleinen Platz auf Messen möglichst gut auszunutzen. Allerdings müssen bei den Kurven die maximalen und minimalen Radien eingehalten werden. Durch den hohen Radstand des LKW besitzt dieser einen großen Wendekreis was die Radien für Kurven sehr einschränkt.

2.3.2 Physik Labor

Für die Physikstudenten in Bergedorf, soll der LKW im Rahmen eines Praktikums eingesetzt werden. Die Studenten bestücken den Anhänger des LKW mit Sensoren und wollen nach einer Fahrt des LKW die Messdaten auswerten. Zum Beispiel sollen die Schwingungen der Ladung aus den Messwerten ausgelesen werden. Dafür soll der LKW einer Linie auf dem Boden nach

fahren können und dabei auch unterschiedlichen Untergrund überfahren. Es ist angedacht, diese Strecke in Modulen zu bauen, so dass sie aus einzelnen Puzzle-teilen zusammengesteckt werden kann. Dies verhindert, das die Radien unterschritten werden. Der Bau der Strecke ist nicht Teil dieser Arbeit.

2.4 Die Umwelt / das Umfeld

Durch die Einflüsse der Umwelt schränken sind die Einsatzgebiete und die Benutzung ein. Die Beschreibung der notwendigen Umgebungsverhältnisse, um einen autonomen Betrieb zu erzeugen, sind in diesem Abschnitt beschrieben. Dabei spielt der Untergrund und auch der Einsatzort eine Rolle. In beiden Betriebsmodi des LKW ist der Fahrer für die Fahrten des LKW verantwortlich. Er hat Sorge zu tragen, dass keine Schäden durch die Umgebung oder mit der Umgebung entstehen.

2.4.1 Untergrundverhältnisse

Der LKW soll auf unterschiedlichstem Untergrund fahren können. Allerdings muss immer die Markierung so leicht erkennbar sein, dass der Laserscanner sie komplett sehen kann. Erst durch Erkennung der Linie ist der LKW in der Lage, der Strecke zu folgen. Unterbrechungen oder verdeckte Teile der Linie können zu Fehlmessung führen und den LKW vom Kurs abbringen. Kann der LKW jedoch die Linie erkennen, ist der Untergrund abseits der Linie nicht von Belang. Sollte der Untergrund so beschaffen sein, dass der LKW durch Hügel oder ähnliches von der Strecke abkommt steuert dieser in Richtung der Linie gegen. Untergründe können beim abtasten in der Steuerungssoftware Fehlmessungen im Sinne von falscher Interpretation der Daten hervorrufen. Durch diese Fehler kann der LKW von der Strecke abkommen, wenn die Steuerungssoftware falsche Markierungen erkennt.

2.4.2 Einsatzorte

Der LKW ist für den Gebrauch in geschlossenen Räumen gedacht. Ausnahmen sollten nur in trockenen Bereichen unter freiem Himmel getätigt werden. Die geplanten Einsätze auf Messen und im Physiklabor erfüllen diesen Anforderungen. Trotzdem ist der LKW im manuellen Betrieb ferngesteuert und kann von Hand frei an beliebigen Orten betrieben werden. Im manuellen Betrieb ist der Fahrer für das Fahren verantwortlich. Im autonomen Betrieb steuert die Software den LKW. Der Fahrer an der Fernsteuerung ist trotzdem verantwortlich stets aufzupassen und sofort einzugreifen, sollte die Software den LKW von der Strecke abbringen.

2.5 Technische Grundlagen

Technische Grundlagen, die in dieser Arbeit zu Anwendung kommen, sind in den folgenden Abschnitten beschrieben und erläutert. Gestartet wird mit der Beschreibung des Barcode Scanners und die vom Barcode Scanner (2D-Laserscanner) verwendeten Teile. Danach kommen die elektrischen Bauteile, wie ein Pegelwandler, Step Up Konverter und ein logischer Schalter. Das Zusammenspiel der Komponenten wird in diesen Abschnitten noch nicht behandelt. Weiter werden dann die PPM Signale beschrieben, welche im Modellbau zur Datenübermittlung von Fernsteuerung zum Empfänger verwendet werden. Zum Schluß wird der Aufbau einer PRU, eines kleinen SubSystems auf dem BeagleBone Black (Einem Einplatinencomputer) beschrieben.

2.5.1 Laserscanner

Der Laserscanner stammt aus einem 2D-Handlaserscanner der Marke TaoTronics Modell TT-BS014. Dieser Laserscanner besteht, ohne sein Gehäuse, aus zwei Platinen, welche im Zusammenspiel das Erkennen und Auswerten des Barcodes ermöglichen. Die kleinere Platine (Abb: 2.2), im folgenden Laserplatine genannt, wird für diese Arbeit verwendet. Die Laserplatine ermittelt die Rohdaten und ermöglicht das Auslesen der Rohdaten auf einem der Ausgänge an der Laserplatine. Für eine Beschreibung der Ausgänge auf der Laserplatine siehe Tabelle 2.1. Die größere der beiden Platinen übernimmt die Auswertung der Rohdaten, im folgender Controller genannt. Der Controller decodiert die Informationen aus den Rohdaten, welche die Laserplatine liefert, um zu entscheiden ob die abgetasteten Rohdaten zu einem Barcode gehören. In Form einer einfachen Zeichenkette werden die Informationen dann über ein USB-Kabel an den Computer gesendet. Am Computer kann der Laserscanner als Eingabegerät, wie zB. eine Tastatur verwendet werden.
Die Laserplatine benötigt eine Versorgungsspannung von 5V. Auf der Laserplatine (Abb: 2.2) wird eine Spule mit einem $25Hz$ Rechtecksignal gepulst. Dadurch wird ein Magnet, der mit einem Spiegel verbunden ist, mit dem Wechsel des Rechtecksignals hin und her bewegt. Dieser Spiegel lenkt den Laser ab, wodurch es möglich ist, dass dieser über den Boden hin und her wandert. Innerhalb einer Periode der $25Hz$ wandert der Laser einmal hin und wieder zurück über den Untergrund. Dabei werden sowohl auf dem Hinweg als auch auf dem Rückweg Daten gesammelt. Die Belegung und Funktion der Anschlüsse ist in Abb 2.1 dargestellt. Die Pins ohne Namen konnten nicht weiter spezifiziert werden, da ein Auslesen der Daten keine Signale erzeugte. Die Belegung der Platine ist Betriebsgeheimnis der Firma TaoTronics, welche auf Anfrage keinerlei Informationen preisgab.

Pin	Funktion	Name
1	GND zum Controller	
2	5V	
3	5V	
4	GND für Laserplatine	GND
5	GND für Status LED	LED
6	5V Versorgungsspannung	VIN
7	Taster zum Starten einer Messung	START
8	Rohdaten des Lasers	DATA
8	25Hz Takt der Laserplatine	CLK

Tabelle 2.1: Pin-Out der Laserplatine

Im Auslieferungszustand wird die Versorgungsspannung für die Status-LED dafür genutzt eine laufende Messung anzuzeigen. Die Laserplatine liefert einen permanenten Strom von Rohdaten auf dem DATA Ausgang, es ist kein triggern oder eine Aktivierung von Hand notwendig. Zum Betrieb der Laserplatine werden lediglich die VIN und die Erdung (In Abbildung **??** GND genannt) angeschlossen. Auf dem CLK Pin kann das $25Hz$ Rechtecksignal abgegriffen werden und auf dem DATA Pin kommen die Rohdaten von der Laserplatine.

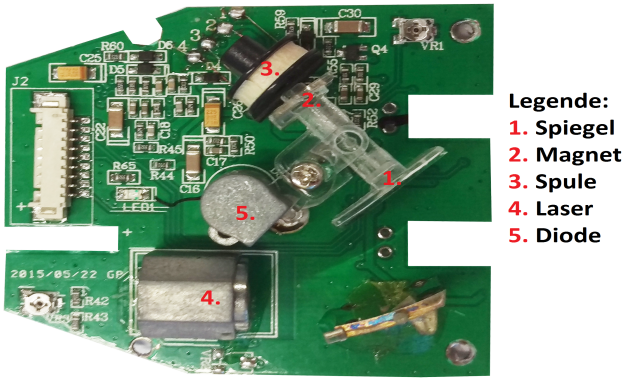

Abbildung 2.2: Laserplatine aus dem Barcodescanner

Der Controller besitzt einen Taster zum Starten einer Messung, einen kleinen Piezo Lautsprecher, ein $RJ45$ Stecker zum Übertragen der Daten an eine USB-Schnittstelle sowie einen kleinen Mikroprozessor, welcher für die Auswertung der Rohdaten der Laserplatine zuständig ist. Da der Controller für diese Arbeit nicht verwendet wird, ist auf eine detaillierte Funktionsbeschreibung verzichtet.

2.5.2 Pegelwandler

Pegelwandler sind kleine Bauteile, die eine Konvertierung von zwei unterschiedlichen Pegeln in den jeweilig Anderen ermöglichen. Wie in Abbildung 2.3 zu sehen ist, werden Eingangspegel auf den jeweils anderen Pegel konvertiert. Bei der Konvertierung bleiben die Verhältnisse der Pegel bezüglich der high-low Zeiten identisch. Die beiden mittleren Kontakte zu beiden Seiten der Platine sind einmal Erdung (GND) und die Referenzspannung. Dabei ist wichtig zu beachten das auf der Seite der Platine, wo LV steht, immer die kleinere der beiden Spannungen anliegt.

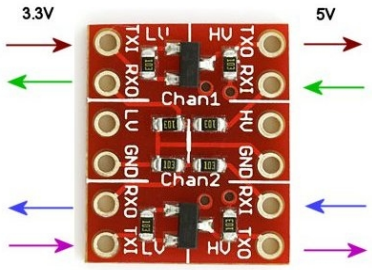

Abbildung 2.3: Platine des Pegelwandlers mit Beispielpfeilen für die Konvertierung der Signale

Der Pegelwandler leitet die Pegel ohne eine Messbare Zeitverzögerung weiter. Die Abbildung 2.4 zeigt, dass diese Pegel vom Eingang und Ausgang identisch sind und nur in Ihrer Pegelstärke verändert wurden. Durch diesen Vorteil kann der BeagleBone Black die Pegel auswerten ohne, dass eine Zeitverzögerung der Pegel durch den Pegelwandler auftritt. Ein weiterer Punkt ist, dass die Geschwindigkeit des Pegelwandlers ausreicht um selbst schnelle, kurze High Pegel unverfälscht zu konvertieren. Abbildung 2.5 zeigt das Signal der Rohdaten von der Laserplatine. Da die Signale vor dem Pegelwandler und nach dem Pegelwandler identisch sind, reicht die Geschwindigkeit aus, um das Signal von 5V auf 3V zu konvertieren und für den BeagleBone lesbar zu machen.

2.5.3 Step Up Konverter

Ein Step Up Konverter ist in der Lage aus einer niedrigen Spannung eine höhere Spannung zu modulieren. Diese Spannung kann über ein Potentiometer (siehe Abbildung 2.7 den einstellbaren Widerstand R1) stufenlos eingestellt werden. Der zum Einsatz kommende Step Up Spannungswandler Modell XL6009 kann eine Eingangsspannung von DC $3 - 32V$ auf eine Ausgangsspannung von DC $5 - 35V$ wandeln. Dabei darf der Eingangsstrom $1.5A$ nicht über-

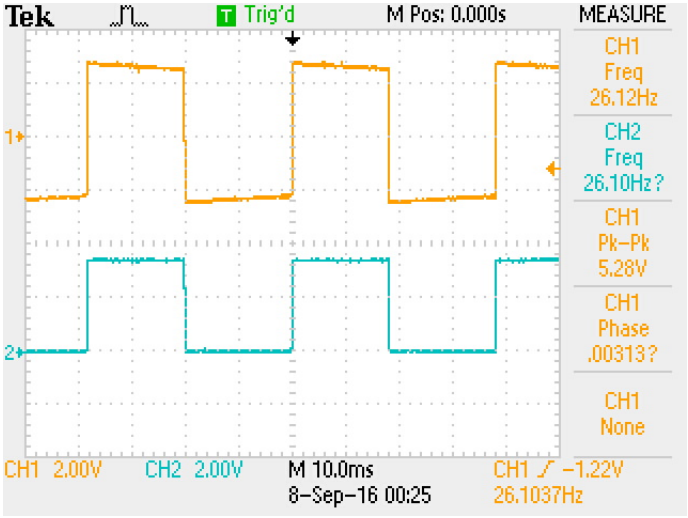

Abbildung 2.4: Vergleich von Eingangssignal zu Ausgangssignal des Pegelwandlers

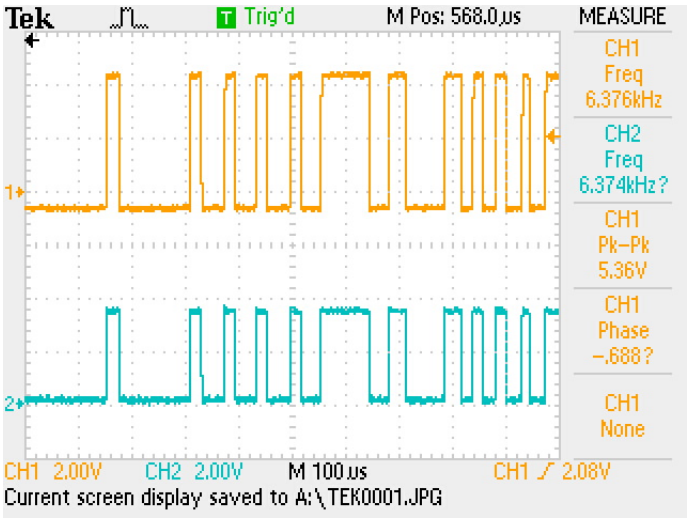

Abbildung 2.5: Vergleich von Eingangssignal zu Ausgangssignal des Pegelwandlers

schreiten. Sollte mehr Leistung als die $1.5A$ benötigt werden, ist eine zusätzliche Kühlung des Step Up Spannungswandlers von Nöten. Der Step Up Spannungswandler verbraucht ca. $20mA$ im Betrieb. Der intern verbrauchte Strom steht im Verhältnis zwischen Eingangs- und Ausgangsspannung. Bei Eingangsspannung gleich der Ausgangsspannung ergibt das am Ausgang einen messbaren Strom von $1.4A$. Für diese Arbeit wird der Step Up Spannungswandler bei $5V$ Eingangsspannung auf $12V$ Ausgangsspannung eingestellt. Durch seine kleine Bauweise ist dieses Bauteil ideal für unzählige Anwendungen im Elektronik-Bereich. Man findet dieses

Bauteil auch sehr oft unter dem Namen Step-Up Schaltregler oder DC-DC Konverter.

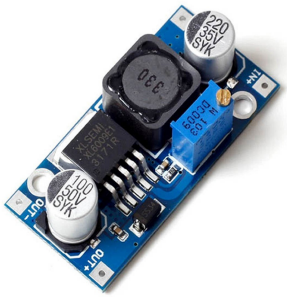

Abbildung 2.6: Der DC-DC Step Up Spannungswandler Modell XL6009

Tabelle 2.2 zeigt eine Auflistung der Ausgangswerte in Abhängigkeit von der Eingangsspannung. Bei steigender Differenz zwischen Eingangsspannung und Ausgangsspannung sinkt der mögliche Strom am Ausgang. Diese Werte beziehen sich auf einen Step Up Spannungswandler ohne separate Kühlung. Die Tabelle 2.2 wurde empirisch ermittelt, um den Zusammenhang zwischen der Eingangsspannung und der Ausgangsspannung, sowie dem Ausgangsstrom zu bilden. Gemessen wurde mit einem 100Ω Widerstand, um keinen Kurzschluss zu erzeugen. Unter Last kann der Strom schwanken. Die ermittelten Werte beziehen sich auf einen unbelasteten Step Up Spannungswandler.

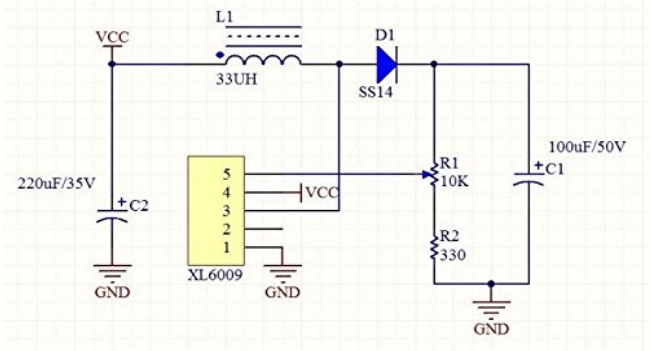

Abbildung 2.7: Schaltplan des DC-DC Step Up Spannungswandlers

Eingangsspannung	Ausgangsspannung	Ausgangsstrom
$5V$	$9V$	$0.7A$
$5V$	$12V$	$0.5A$
$5V$	$15V$	$0.4A$
$5V$	$24V$	$0.2A$
$12V$	$15V$	$1.0A$
$12V$	$19V$	$0.8A$
$12V$	$24V$	$0.6A$

Tabelle 2.2: Ausgangsspannung und Ausgangsstrom im Zusammenhang mit der Eingangsspannung

2.5.4 Logischer Schalter

Ein logischer Schalter trennt Signale physikalisch voneinander, dass diese nicht mehr Signale leiten können. Durch ein Steuersignal kann eine Leitung geöffnet und geschlossen werden. Systeme wie der BeagleBone Black können diese Schalter verwenden, um Signale zu steuern. Vom BeagleBone Black reicht ein GPIO Signal, um zwischen den beiden Zustanden des Schalters umzuschalten.
Zur Entkopplung der Signale wird ein DG403BDY-E3 Analoger SPDT-Schalter Dual verwendet. Dieses Bauteil ermöglicht über eine Steuerleitung Signale aus- und wieder anzuschalten. Abbildung 2.8 zeigt den Aufbau des Schalters.

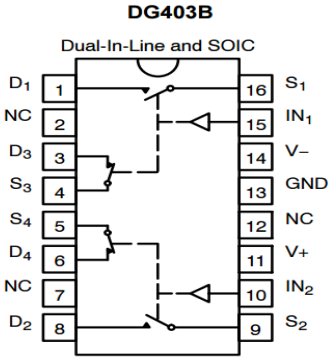

Abbildung 2.8: Aufbau des DG403B. Ein High-Speed CMOS Analog Schalter

Geschaltet werden kann über den IN_1 Eingang. Dieser Schaltet, intern invertiert zwischen den beiden Signalen in einer Wechselwirkung. Liegt am IN_1 Eingang eine logische 1, das sind Spannungen $\geq 2.4V$, wird das Signal, das am Eingang S_3 anliegt, auf den Ausgang D_3 weiter geleitet. Schaltet das Eingangsignal an IN_1 nun auf logisch 0 (Spannung $\leq 0.8V$), werden beide

Schalter umgelegt und das Signal, welches an S_1 anliegt, wird auf den Aufgang D_1 geleitet. Die Verbindung zwischen S_3 und D_3 wird dabei unterbrochen. Eine Wahrheitstabelle über die Zustände der Schalter SW_1 bis SW_4[1] ist in Abbildung 2.9 zu sehen.

TRUTH TABLE		
Logic	SW_1, SW_2	SW_3, SW_4
0	OFF	ON
1	ON	OFF

Abbildung 2.9: Wahrheitstabelle über die Schalter und Ihre Zustände abhängig vom Eingangssignal.

Der DG403BDY-E3 Analoger SPDT-Schalter Dual zeichnet sich durch eine schnelle Reaktionszeit die bei $100ns$ liegt aus. Zudem lassen sich Spannungen von $\pm 15V$ schalten. Da die PWM Signale vom BeagleBone und die PPM Signale vom RC Empfänger bei maximal $3.3V$ liegen, wird für die Signale keine weitere Hardware wie z.B. Pegelwandler benötigt. Durch den Schalter der zwei schaltbare Leitungen, besitzt können zweimal zwei Signale geschaltet werden. Für diese Arbeit werden die beiden Steuersignale auf ein Signal vom BeagleBone gelegt, da alle Signale zur gleichen Zeit vom BeagleBone umgeschaltet werden sollen.

2.5.5 PPM Signale

PPM Signale basieren auf der Puls-Pausen-Modulation (Englisch: puls-position modulation) und werden primär für die Steuerung von Modellfahrzeugen benutzt. Dabei ist es egal ob Flug-, Land- oder Wasser-Modelle. Bei allen digitalen RC-Fernsteuerungen im $2.4GHz$ Bereich werden die Daten der Steuerknüppel, Stellhebel und Schalter mit der Puls-Pausen-Modulation an das Fahrzeug gesendet.
Der Ablauf besteht aus mehreren Schritten. Eine Spannung die am Potentiometer, einem einstellbaren Widerstand des Steuerhebels, abfällt wird gemessen und in ein binäres Signal umgewandelt. Dieses Signal beinhaltet noch keine Informationen zu anderen Kanälen und kann nicht als einzelnes Signal übertragen werden.
Alle Kanäle zusammen werden von einem PPM-Codierer zu einem PPM Signalframe zusam-

[1] SW_1 ist dabei der Schalter der den Eingang S_1 mit dem Ausgang D_1 verbindet. SW_2 trennt S_2 mit D_2 und ebenso bei den Schaltern SW_3 und SW_4.

mengefasst. Dabei wird der Start zu Beginn angesetzt und dann folgen die Kanäle 1 bis 8. Diese Kanäle befinden sich immer in der gleichen Reihenfolge.

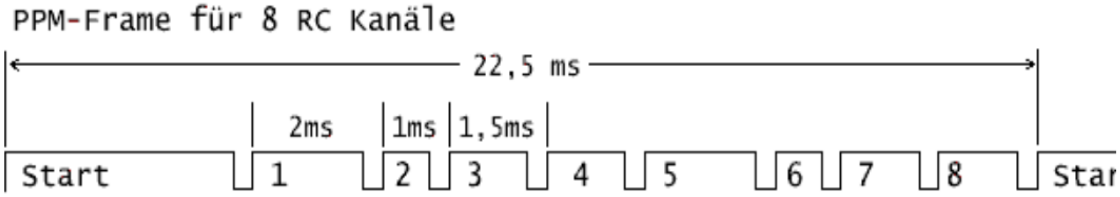

Abbildung 2.10: Aufbau eines PPM-frames

Wie die Abbildung 2.10 zeigt, besteht ein Frame des PPM-Signals aus einer langen Startinformation, sowie den einzelnen kodierten Kanälen. In diesem Fallbeispiel sind es 8-Kanäle. Die Rückgewinnung der einzelnen Kanäle geschieht über die Pausen zwischen den Pulsen, die immer $0.3ms$ lang sind. Die separaten Kanäle haben dann ein Zeitfenster von 1ms. Wobei $0ms$ ein Vollausschlag nach Links und $1ms$ einen Vollausschlag nach Rechts bedeuten. Zusätzlich hat jeder Kanal eine minimale Pulsdauer von $0.7ms$, dadurch erstreckt sich ein Kanal auf einen Puls von $0.7ms$ bis $1.7ms$. Zusammen mit der $0.3ms$ Pause zwischen den Kanälen, liegt dann ein Kanal bei einer Pulsdauer von 1ms bis $2ms$. Zusammen mit dem Start erzeugen alle 8-Kanäle eine PPM-Framelänge von $22.5ms$. Dieses Prinzip zur Kodierung der Daten ist Standard bei RC-Fernbedienungen. Einzelne PPM Kanäle können durch PWM Signale nachgebildet werden, indem die Periodendauer auf die $22.5ms$ gesetzt wird und der Impuls entsprechend der Kodierung erhöht oder verkleinert wird. Ganze PPM Frames lassen sich mit PWM nicht nachbilden, da man die Pausen nicht mit modulieren kann.

2.5.6 PRU Aufbau

Der BeagleBone Black besitzt zwei Programmable Realtime Units (PRU's). Da Linux nonpreemptiv ist, ist es schwer mit dem Linux System Echtzeitanwendungen zu entwickeln. Probleme hierbei entstehen überwiegend durch den Shedular, der bewirkt, dass man nie garantieren kann, dass der aktuelle Prozess innerhalb einer gewissen Zeit zur Terminierung kommt. Damit ist gemeint, dass dieser Prozess in der geforderten Zeit immer seine Aufgaben abarbeiten kann. Darunter fallen arbeiten, wie GPIO Signalgenerierung oder das Sampling von Signalen auf einem GPIO Pin. Für Aufgaben, welche eine Echtzeitkomponente fordern, besitzt der BeagleBone Black die beiden PRU's. Diese werden auch oft PRUSS (Programmable Realtime Unit Subsystem) genannt.
Jeder der beiden PRU Kerne besitzt einen eigenen 8KB Speicher, kann aber auch auf den

Speicher der jeweils anderen PRU über den internen 32-Bit Interconnect Bus zugreifen. Abbildung 2.11 zeigt das komplette Subsystem mit beiden PRU's und deren Hardware. Der gesamte Aufbau der beiden PRU's mit Ihrer Hardware heißt PRU-ICSS (Programmable Realtime Unit - Industrial Communication Subsystem).

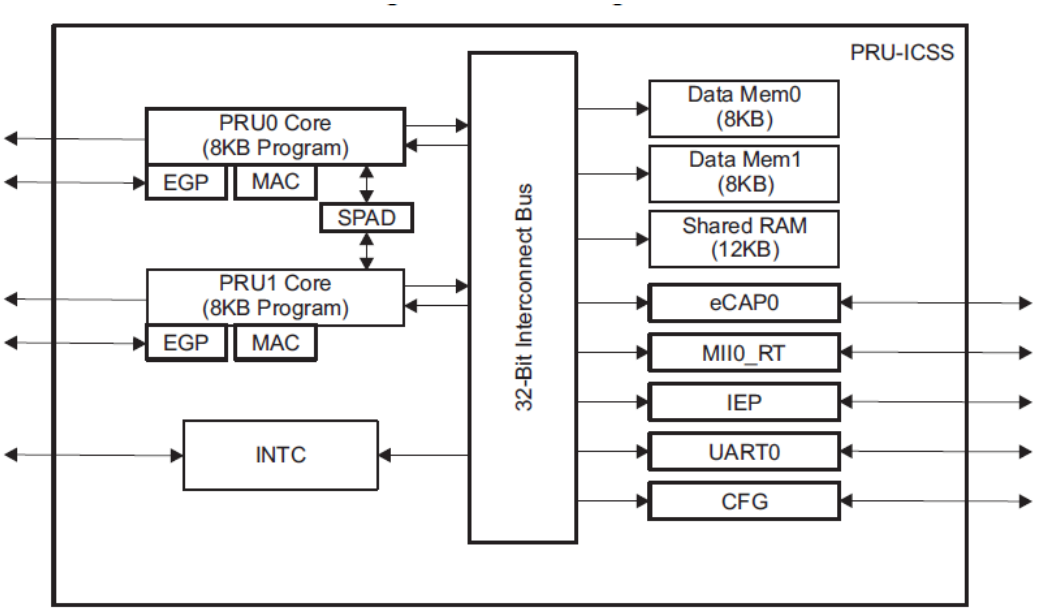

Abbildung 2.11: PRU Subsystem Aufbau im BeagleBone Black

2.6 Kernelmodul

Kernelmodule sind Programme, welche die Funktionalität des Kernels in einem Linux System erweitern. Diese Kernelmodule können zur Laufzeit des Linux in den Kernel geladen und wieder entladen werden, ohne das ein Neustart des Systems erforderlich ist. So kann ein Kernelmodul Zugriff auf Hardware bekommen, der aus einem Linuxprogramm im Userspace nicht möglich ist. Dies wäre zum Beispiel der Zugriff auf eine physikalische Adresse, um einen Wert auszulesen. Da ein Programm, das im Linux Userspace läuft, beim Start einen zugewiesenen Speicherbereich bekommt, hat dieses Programm keine Chance auf die physikalischen Adressen auf dem Speicher zuzugreifen. Diese Arbeit wird dann von einem Kernelmodul erledigt. Ohne diese Möglichkeit, Kernelmodule zur Laufzeit ein- und auszuladen wären wir gezwungen monolithische Kernel zu bauen. Bei monolithischen Kernels wird Funktionalität direkt mit in das Kernelimage eingebaut. Dies würde selbst bei großen Kernels bedeuten, dass bei jeder Funktionalität die hinzugefügt werden soll, der komplette Kernel neu gebaut und kompiliert werden muss.

2.7 Embedded Linux

Ein embedded Linux weist einige abgeänderte Funktionalitäten auf zu einem Linux, das auf einen Laptop oder einem PC läuft. Einige Funktionalitäten sind entfernt oder in kleineren Versionen enthalten, um den Betrieb auf leichtgewichtigen Einplatinensystemen wie dem BeagleBone zu gewährleisten. Da die Einplatinensysteme in der Regel sehr sparsam mit Leistung, Speicher und Daten umgehen müssen, ist das Embedded Linux auf solche kleinen Systeme zugeschnitten. Die Unterschiede, die für die Bearbeitung dieser Arbeit wichtig sind, werden in diesem Abschnitt erläutert.

2.7.1 Flattened Device Tree

Ein Embedded Linux besitzt kein BIOS zum Starten und Laden der Hardware. Dafür werden Dateien verwendet, in denen die Hardware beschrieben ist, damit der BeagleBone beim Systemstart die Hardware initialisieren kann. Diese Dateien sind statisch und können während der Laufzeit nicht mehr verändert werden. Diese statischen Dateien sind einzigartig für jedes System. Der Flattened Device Tree des BeagleBone funktioniert nur auf dem BeagleBone Black und seinem Bruder, dem BeagleBone White. Um nach Systemstart Hardware einzubinden, oder bei den GPIO Pins andere Funktionalitäten zu aktivieren, werden Device Tree Overlays benutzt.

2.7.2 Device Tree Overlays

Da auf vielen GPIO Pins mehrere Funktionalitäten erreichbar sind, muss definiert sein, welche der Funktionalitäten erreicht werden soll. Ansonsten liefert der Pin im besten Fall einen falschen Wert. Overlays sind variable Hardware Beschreibungen, welche in einem Embedded Linux zur Laufzeit eingebunden und wieder entfernt werden können. Overlays werden dann benötigt, wenn GPIO Pins in einem anderen Modus betrieben werden sollen. So ein Modus kann zum Beispiel eine Verwendung als GPIO Pin sein, aber auch für die Verwendung mit einem Bus genutzt werden, wie zum Beispiel UART. Der Modus muss für jeden Pin so definiert werden, dass nur eine Funktionalität pro Pin aktiviert ist. Für die benötigten Pins muss ein Overlay geschrieben werden, das genau die Verwendung der GPIO Pins, welche die PRU verwendet beschreibt.

2.7.3 Secure Mode vs. Fast Mode

Die GPIO Pins auf dem BeagleBone haben zwei Modi für den Betrieb als Input/Output. Im Rahmen dieser Arbeit wurde der Secure Mode verwendet. Der Secure Mode stellt sicher, dass keine ungültigen Werte auf den GPIO Pin geschrieben werden können. Der Fast Mode überprüft die Ausgabewerte nicht und ist deshalb deutlich schneller. Der GPIO Pin, der im Secure Mode betrieben wird, liefert 40 Werte pro Messung. Wird dieser Pin im Fast Mode betrieben, liefert er $140 - 170$ Werte bei den Messungen. Diese Werte schwanken stark,abhängig von der Auslastung des BeagleBone. In einem Programm wo nur Messungen durchgeführt werden, liefert der Pin ca. 170 Werte. Wird ein Thread dazu gestartet, sinkt die Anzahl der Messwerte bereits auf unter 160. Durch die deutlich erhöhte Anzahl der Werte aus einer Messung werden die Muster für die Erkennung der Linie ebenfalls komplexer. Steht der LKW am Rand des Wertebereichs, ist beim Fast Mode ein Rauschen auf den Daten, was eine exakte Bestimmung der Position deutlich aufwändiger macht. Dazu kommt, dass die Auswertung der Messdaten mehr Zeit in Anspruch nimmt, was in den Messungen zu Überschneidungen in der Erfassung von Messdaten im $25Hz$ Takt führte. Wegen seiner weniger aufwändigen Auswertung und kleinere Anzahl an Muster für das Schema einer Linie wurde im Rahmen dieser Arbeit der Secure Mode verwendet.

3 Analyse

Die Analyse der Fahrzeugplattform und der Aufgabe für diese Arbeit erfolgt in diesem Kapitel. Zuerst wird die Fahrzeugplattform analysiert und die technischen Daten werden vorgestellt. Danach erfolgt eine Analyse der Aufgabe, die im Rahmen dieser Arbeit erarbeitet werden soll.

3.1 Die Fahrzeugplattform

Die Fahrzeugplattform besteht aus einem Modell LKW im Maßstab $1:14$. Eine Analyse des LKW und die Auswertung der technischen Daten erfolgt in diesem Abschnitt.

3.1.1 Aufbau des LKW's

Der LKW besteht aus einem Fahrerhaus, in dem der überwiegende Teil der Technik verbaut ist. Der LKW ist ein Drei-Achser Triebfahrzeug für einen Sattelauflieger. Ein Auflieger ist ebenfalls vorhanden wird aber innerhalb dieser Bachelorarbeit nicht weiter verwendet. Die vordere Achse ist, wie im Original lenkbar. Die Lenkung erfolgt über einen Servomotor, der von der Steuereinheit angesteuert wird. Direkt unter dem Fahrerhaus sitzt der Motor, der über eine Spannung aus der Steuereinheit betrieben wird. Zusätzlich gibt es einen Servomotor, der ein Getriebe vor dem Motor ansteuern kann.

3.1.2 Technische Daten

Der LKW, der für diese Bachelorarbeit zur Verfügung steht, besitzt folgende Daten:

3.2 Die konkrete Aufgabe

Die Grundaufgabe besteht darin ein vorgegebenes System (Modell LKW) soweit zu erweitern, dass der LKW selbstständig einer Linie auf dem Boden folgen kann. Das System soll einfach, robust und nach Möglichkeit wartungsfrei sein, um einen langfristigen Einsatz zu gewährleisten.

Parameter	Wert
Maßstab	1 : 14
Breite	210mm
Höhe	290mm
Länge	510mm
Radstand	230mm
Lichter	3 Stufen Vorne / Rücklicht / Rückfahrlicht
Blinker	Vorne & Hinten jeweils Rechts/Links
Lautsprecher	Einmal für Sound und Hupe

Tabelle 3.1: Auflistung der technischen Daten des LKW

3.2.1 Steuerung

Über die Fernsteuerung kann im manuellen Betrieb gesteuert werden. Legt man auf der Fernsteuerung einen Hebel um, der für diese Anwendung Programmiert wurde (L2.E), so übernimmt der Mikrocontroller die Steuerung. Dies bewirkt, dass ein Nutzer jederzeit zwischen dem autonomen und dem manuellen Betrieb umschalten kann. Wenn der Mikrocontroller steuert, soll es nicht möglich sein mit der Fernsteuerung manuell in die Signale (Steuerung) einzugreifen. Umgekehrt soll der Mikrocontroller im manuellen Betrieb still sein und keinerlei Signale an die Steuereinheit des LKW senden.
Gesteuert wird von der Fernsteuerung standardmäßig mit dem PPM-Verfahren (siehe Abschnitt 2.5.5). Dieses Signal soll durch den Mikrocontroller mit PWM nachgebildet werden, um bei der Ursprungstechnologie zu bleiben. Dies bedeutet, dass in das bestehende System nur durch das Ändern der Signalquelle der Steuersignale in das System eingegriffen wird, da die Signale im autonomen Betrieb vom Mikrocontroller kommen anstatt von der Fernsteuerung über den Empfänger.

3.2.2 Linie

Der LKW soll einer Linie auf dem Boden folgen. Dabei wurde von vornherein nicht näher spezifiziert, wie diese Linie auszusehen hat. Die Vorgabe dabei ist allerdings, dass die verwendete Linie leicht verlegbar ist. Das bedeutet, die Linie muss so konzipiert sein, dass ohne großen Aufwand eine neue Strecke erzeugt werden kann. Eine Idee, die bei dem Kennenlerngespräch mit dem für den LKW Verantwortlichen aus dem Physiklabor in Bergedorf entstand, war die Verwendung von einer Art Klebestreifen als Linie. Diese ist so leicht neu zu platzieren und lässt gleichzeitig zu, dass Unebenheiten im Untergrund keine Probleme beim Markieren der Strecke verursachen.

3.2.3 Benötigte Funktionalitäten

Die Steuerung erfolgt, nach der Freischaltung über die Fernsteuerung, durch den BeagleBone Black. Dieser schaltet das analoge Signal der Fernsteuerung aus und übernimmt durch jeweils einen PWM Ausgang die Steuerung der Geschwindigkeit und der Lenkung. Die Steuerung der Lichter, der Hupe und der Blinker sind weiterhin über die Fernsteuerung möglich. Lediglich das Blinken, wenn der LKW stark einlenkt, funktioniert weiterhin im autonomen Betrieb, da dieses Blinken von der Steuereinheit selbst erzeugt wird. Der BeagleBone überprüft das Signal der Fernsteuerung dazu welche Geschwindigkeit der LKW fahren soll und wenn eine Linie gefunden wird, stellt der BeagleBone die Geschwindigkeit über das PWM Signal ein. Der BeagleBone wertet kontinuierlich die Daten des Laserscanners aus und sucht die Linie. Sollte über einige Zeit keine Linie gefunden werden, bleibt der LKW stehen. Das Umschalten zwischen manuellem und autonomem Betrieb soll jederzeit möglich sein.

3.2.4 Erwartete Schwierigkeiten

Es werden Schwierigkeiten bei der Erkennung der Linie erwartet. Es könnten sich Fehler dadurch ergeben, dass die Linie an einer falschen Position erkannt wird. Es könnte weiter dadurch Schwierigkeiten auftreten, da auf dem BeagleBone Black nur die beiden PRU's vorhanden sind um mit hoher Geschwindigkeit die GPIO Pins auszulesen.

4 Entwurf

In diesem Kapitel wird der Entwurf zur Erstellung der Steuerungssoftware für den LKW erstellt. Dabei wird zunächst die Wahl der Linie, welcher der LKW folgen soll, näher spezifiziert. Dabei wird auf die Linienform, das Material und den Untergrund näher eingegangen. Danach wird die Wahl des zur Verwendung kommenden Mikrocontrollers vorgestellt und folgend die Änderungen an der bestehenden Architektur des LKW. Zum Schluss wird dann auf die Entkopplung von Signalen und das Softwarekonzept eingegangen.

4.1 Wahl der Linie

Bei der Wahl der Linie sind mehrere Kriterien zu berücksichtigen, um eine gute Erkennung für ein fahrenden LKW, der dieser Linie folgen soll zu erreichen.

4.1.1 Linienform

Das wichtigste Kriterium für eine Erkennung ist die Form der Linie, besser gesagt die Form des Musters auf dieser Linie. Der verwendete Laserscanner (siehe Abschnitt 2.5.1) empfängt die Reflexionen des Lasers auf dem Boden. Linienformen mit Strichen waren dabei direkt auf dem Oszilloskop sichtbar. Die Unterschiede der Muster auf der Linie belaufen sich auf die Anzahl und die Strichstärke der Striche auf der Linie. Abbildung 4.1 zeigt eine Auswahl der Linien die zur Verfügung stehen.

Abbildung 4.1: Auswahl von Linien, welche für die Erkennung in Frage kommen.

4.1.2 Material des Klebebandes

Das Material aus dem das Klebeband besteht spielt eine große Rolle. Ein Isolierband mit mehreren Streifen ist ungeeignet, da hier die Beschichtung auf dem Klebeband zu glatt ist und der Laserscanner dort keine Farbübergänge erkennen kann. Auf einem Reisband (Washi Tape) erkennt der Laserscanner die Streifen genauer, da die Struktur des Reisbandes rau und uneben ist. Der Laserscanner erzeugt bei schwarzer Tinte auf weißem Papier die höchste Genauigkeit, da Papier eine gleichmäßige Oberfläche bietet auf der Farbkontraste am Besten erkannt werden.

4.1.3 Untergrund

Der Untergrund kann die Ergebnisse beeinflussen, indem der Laserscanner beim Abtasten unerwartete Werte liefert. Ein rauer Untergrund kann rechts und links von der Linie ein starkes Rauschen in den Abtastwerten hervorrufen und die genaue Bestimmung der Linienposition verhindern. Glatte Oberflächen sind leichter für die Erkennung der Linie, aber nicht realistisch, da in den Anwendungsgebieten nur selten wirklich glatte Oberflächen zur Verfügung stehen. Eine Störung durch den Untergrund ist nicht vermeidbar, da der LKW über unterschiedliche Untergründe fahren soll und dabei immer Werte vom Laserscanner eingelesen werden, die nicht für die Erkennung der Linie verwendbar sind.

4.1.4 Entscheidung über die gewählte Markierung

Für diese Arbeit wird ein weißes Washi Tape eingesetzt. Auf dem Tape befinden sich drei gleich breite, schwarze Striche über die gesamte Länge des Washi Tapes. Durch die drei Striche erzeugt das Washi Tape beim Laserscanner ein Muster, das durch den BeagleBone Black ausgewertet werden kann. Die Oberfläche des Washi Tapes besitzt eine papierähnliche Struktur. Durch das Muster der Striche auf dem Washi Tape werden Fehler durch Rauschen abseits der Markierung reduziert, weil durch die markante Markierung das Muster nicht so häufig vorkommt innerhalb von unkontrolliertem Rauschen. Durch den variablen Untergrund abseits der Markierung kann man ein Rauschen leichter filtern bei dieser Wahl der markanten Markierung.

4.2 Wahl des Mikrocontroller

Für diese Arbeit wurde der BeagleBone Black gewählt, da er das Programmable Realtime Unit Subsystem besitzt, das bestimmte Funktionalität besitzt mit der keine weitere, externe Hardware benötigt wird. Der BeagleBone Black mit dem PRUSS (siehe Abschnitt 2.5.6), beinhaltet die

beiden PRU's mit denen effizient GPIO Pins gesampelt werden können. Durch vier PWM Ausgänge liefert der BeagleBone genug Schnittstellen, um die Steuerung für den LKW zu übernehmen. Eine genaue Auflistung über die verfügbaren Pins auf dem BeagleBone liefern die Abbildungen 8.1 und 8.2 im Anhang. Zum Systemstart des BeagleBone, ohne dass Overlays aktiviert wurden, bieten die GPIO Pins nur begrenzte Funktionalität. Einige Funktionen werden erst nach dem Hinzufügen von Overlays verfügbar (siehe Abschnitt 2.7.2). In dieser Arbeit werden die beiden PRU's auf dem BeagleBone benutzt. Dafür muss für jede PRU ein Overlay erstellt werden, welches die Funktionalität der GPIO Pins überschreibt und verfügbar macht. Eine genaue Auflistung, welche GPIO Pins für die PRU verfügbar sind befindet sich in den Tabellen 8.1 und 8.2 im Anhang.

4.3 Hardware- und Softwarearchitektur

In diesem Abschnitt werden die Veränderungen, die für die Entwicklung der Fahrsteuerung vorgenommen werden, dargestellt und erläutert. Zunächst wird die ursprüngliche Architektur vorgestellt und im Bezug darauf dann aufgezeigt, welche Änderungen vorgenommen werden müssen um den Eingriff in die Architektur so gering wie möglich zu halten.

4.3.1 Ausgangsarchitektur

Die ursprüngliche Architektur besteht aus der Fernsteuerung, welche die Steuersignale an den Empfänger im LKW sendet. Die Signale vom Empfänger werden direkt in die Steuereinheit auf dem LKW geleitet. Die Steuereinheit regelt anhand des PPM Signals (siehe Abschnitt 2.5.5) aus dem Empfänger, den jeweiligen Aktor, wie zum Beispiel das Lenken, Gas geben, Bremsen, Hupen oder das Licht an und aus schalten. Alle Signale laufen über die Steuereinheit und diese leitet die Signale weiter und erzeugt, abhängig von der Geschwindigkeit, Geräusche welche einen fahrenden LKW simulieren sollen. Es ist möglich, dass ein Signal mehrere Aktionen in der Steuereinheit auslöst. So wird bei starkem Lenkeinschlag zusätzlich der Blinker für diese Richtung aktiviert.

4.3.2 Anpassung der Architektur

Die Signale, die der Empfänger von der Fernsteuerung an die Steuereinheit sendet, sollen im autonomen Betrieb durch ein vom BeagleBone erzeugtes PWM Signal ersetzt werden. Um die aktuelle Architektur möglichst unverändert zu lassen, soll der BeagleBone zwischen den Empfänger und die Steuereinheit gebaut werden. Der Eingriff in die bestehende Architektur

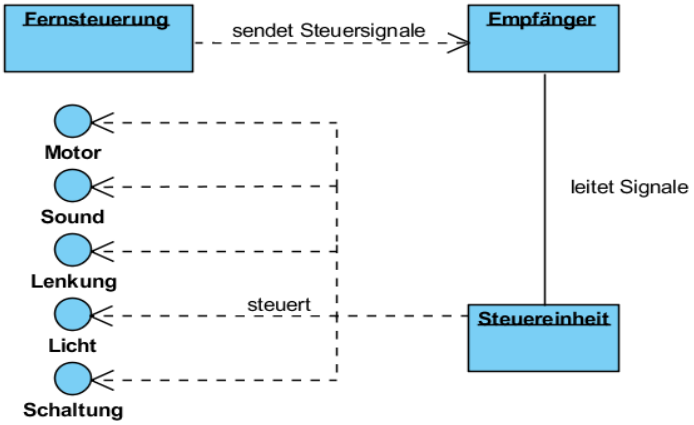

Abbildung 4.2: Architektur wie sie Ursprünglich vorhanden ist.

ist daher sehr gering. Hinzu kommt ein Schalter, der zwischen der Verbindung von Empfänger und der Steuereinheit sitzt. Aufgabe des Schalters is die Trennung der Signale. Ein Überlagern der Signale vom BeagleBone und von der Fernsteuerung führt zu Fehlern in der Steuereinheit des LKW. Dieser Schalter schaltet zwischen den beiden Signalen um. Abbildung 4.3 zeigt den schematischen Aufbau der Architektur nach Hinzufügen der neuen Komponenten. Auf der Fernsteuerung werden Hebel für die Steuerung der Geschwindigkeit und den An/Aus Schalter verwendet. Im Empfänger werden die Kanäle, auf denen die Signale für die Geschwindigkeit und den An/Aus Schalter ankommen, zum BeagleBone geführt, um sie dort durch die PRU (Abschnitt 5.2.3) auszuwerten. Der BeagleBone kann nach Auswertung des Signals eine Entscheidung treffen, ob im manuellen oder im autonomen Betrieb gesteuert werden soll. So stellt er den Schalter so ein, das nur ein Signale von der Fernsteuerung oder dem BeagleBone zur Steuereinheit geleitet wird.

4.4 Entkopplung der Signale

Um sicher zu stellen, dass die Signale vom RC Empfänger und vom BeagleBone sich nicht überschneiden und Störungen verursachen, müssen die Signale beim jeweiligen Betriebsmodus entkoppelt werden. Im manuellen Betrieb sollen die Signale vom BeagleBone abgeschaltet werden und die vom RC Empfänger durchgelassen werden. Im autonomen Betrieb wiederum sollen Signale, welche die Fernsteuerung sendet, Geschwindigkeit und Lenkung, abgeschaltet werden und der BeagleBone übernimmt die Steuerung. Wichtig dabei ist, dass die Signale für

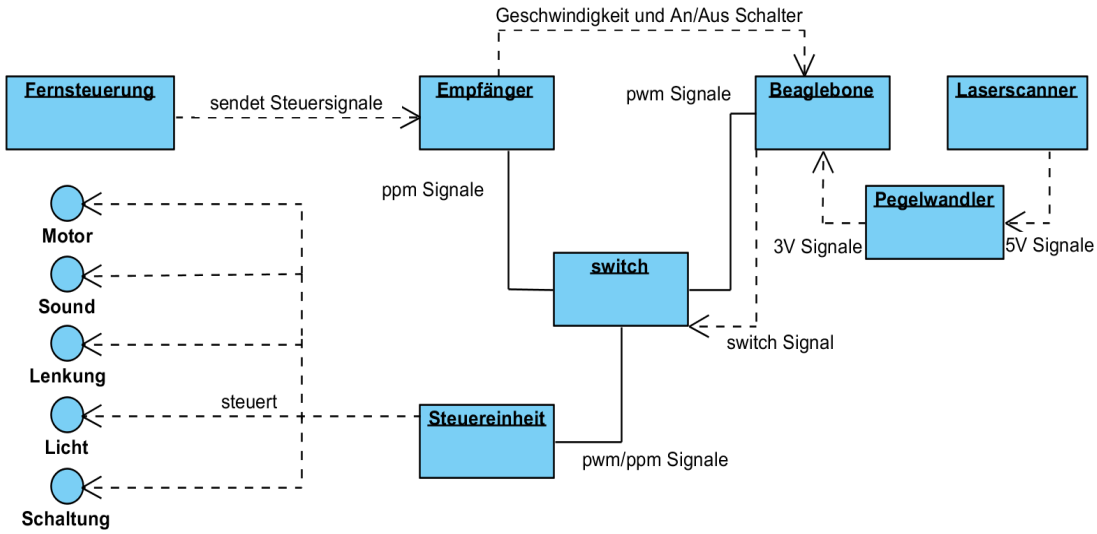

Abbildung 4.3: Architektur mit dem BeagleBone und der hin zugesetzten Hardware

die Geschwindigkeit und den AN/Aus Schalter separat zum BeagleBone geführt werden und nicht ausgeschaltet werden.

4.4.1 Logischer Schalter

Das DG403BDY-E3 Analoger SPDT-Schalter Bauteil (siehe Abbschnitt 2.5.4) besitzt zwei schaltbare Leitungen, was das Schalten von zwei Leitungen über eine Steuerleitung ermöglicht. Dadurch kann das PWM Signal vom BeagleBone ausgeschaltet werden, während das Signal von der RC Fernsteuerung eingeschaltet ist.

4.4.2 Step-Up Konverter

Der DG403B benötigt eine Versorgungsspannung von $12V$. Der BeagleBone besitzt nur eine Spannung von entweder $3.3V$ oder $5V$, was beides nicht ausreicht um den DG403 zu betreiben. Um die benötigte Versorgungsspannung von $12V$ zu erreichen, wird der in Abschnitt 2.5.3 vorgestellte DC-DC Step Up Spannungswandler Modell XL6009 verwendet. Dieser Step Up Spannungswandler erlaubt es die Eingangsspannung auf eine höhere Spannung herauf zu setzen. Über einen einstellbaren Widerstand (im Schaltplan $R1$ genannt, siehe Abbildung 2.7) kann die Ausgangsspannung stufenlos eingestellt werden.

4.5 Softwarekonzept

Der gesamte Aufbau und das Zusammenspiel der Komponenten in der Software, wird in diesem Abschnitt erläutert. Dabei wird zuerst die Steuerung des Fahrzeugs mit dem BeagleBone beschrieben. Zum Ende, wie der BeagleBone die Erkennung der Markierung auf dem Boden abarbeitet.

4.5.1 Steuerung des Fahrzeugs

Der LKW soll im autonomen Betrieb weiterhin über die eingebaute Steuereinheit die Aktoren steuern. Dadurch wird verhindert, dass direkte Signale an die Servos oder den Motor gesendet werden muss. Lediglich die Signale für Lenkung und Geschwindigkeit werden vom BeagleBone mit PWM Signalen nachgebildet. Das Signal für die Steuerung des Motors wird an die Signale aus dem Empfänger angepasst und nachgebildet. Abbildung 4.4 zeigt das Motorsignal in der Grundstellung, wenn der LKW still steht.

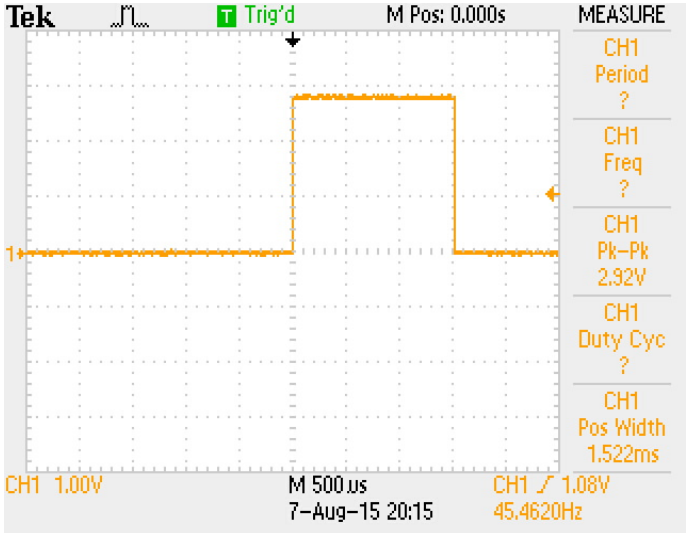

Abbildung 4.4: PPM Signal vom Empfänger bei dem der LKW still steht.

Durch Veränderung des PWM Signals wird der LKW gesteuert, wie im manuellen Betrieb durch die PPM Signale der Fernsteuerung, somit wurde nicht in die Signalübertragung eingegriffen.

4.5.2 Die Steuerungssoftware

Die Software ist in mehrere Module unterteilt. Ein Modul befindet sich innerhalb eines Threads, der die Aufgaben des Moduls abarbeitet. Das Hauptprogramm übernimmt die Initialisierung und das Starten der einzelnen Module (Threads). Zusätzlich übernimmt das Hauptprogramm die Erkennung der Linie und die Auswertung der Position des LKW's zur Linie. Das Lenkmodul erzeugt das PWM Signal für die Lenkung und passt den Wert an, sofern das Hauptprogramm eine Position ermittelt, die von der Mitte der Linie abweicht. Das Fahrmodul, das ein PWM Signal erzeugt, steuert die Geschwindigkeit. Diese wird durch die Werte, die die PRU ausgelesen hat verändert. Es stehen dabei drei Zustände zur Verfügung: Stillstand, langsame Geschwindigkeit und schnelle Geschwindigkeit. Ein weiteres OnOffmodul, das die erste PRU startet und ein Programm überspielt verwaltet die Zustände des Systems: autonomer oder manueller Betrieb. Mit diesem Programm kann die PRU einen Eingangspin auslesen und die Dauer des High Pegels ermitteln. Dieser Wert wird dann in den Speicher der PRU geschrieben, von wo ihn das OnOffmodul auslesen kann und abhängig von dem ausgelesenen Wert, den autonomen Modus des BeagleBone ein oder aus schalten kann. Das letzte Tempomodul überspielt ebenfalls ein Programm an die zweite PRU. Diese PRU ist zur Ermittlung der Geschwindigkeit zuständig, indem sie ebenfalls die Dauer des High Pegels ermittelt und in den Speicher schreibt, damit das Tempomodul den Wert auslesen kann. Besonders hierbei ist, dass es drei unterschiedliche Werte geben kann und nicht, wie bei dem anderen OnOffmodul, nur zwei zum An- und Ausschalten des autonomen Betriebs. Diese Werte werden dann an das Fahrmodul, das das PWM Signal für die Geschwindigkeit regelt, übertragen damit der LKW die Geschwindigkeit ändert. Die Zugriffsrechte auf den Speicher der PRU liefert dabei ein Kernelmodul, welches die Werte aus dem physikalischen Speicher der PRU an die Software liefert.

4.5.3 Erkennung der Linie

Der BeagleBone schreibt alle Daten aus dem DATA Signal der Laserplatine mit. Er startet dabei mit einer steigenden Taktflanke des CLK Signals aus der Laserplatine und beendet eine Messung mit der nächsten fallenden Taktflanke des CLK Signals. Die Beschreibung der Signale von der Laserplatine befindet sich in Abschnitt 2.5.1. Nach einem Durchlauf hat die Software ein Array, welches alle Werte einer Messung beinhaltet. Diese Messwerte bestehen aus einer 1 oder einer 0. Dieses Array wird durchsucht, um das Muster der Markierung zu lokalisieren. Ist eine Position auf dem Array gefunden, an dem sich ein Muster der Markierung befindet, kann die Steuerungssoftware die Position weitergeben zur Einstellung des PWM Signals für die Lenkung.

Gesucht wird im ersten Durchlauf von der ersten Position des Arrays bis zur ersten 1. Bei dieser 1 wird überprüft ob sie der Beginn eines Musters der Markierung ist. Dafür werden die folgenden Werte angeschaut. Bilden die erste 1 mit den darauf folgenden Werten ein Muster gehört diese 1 zum Beginn Musters und kann als Position der Linie auf der Markierung weiter verwendet werden. Ist die 1 nicht der Anfang des Musters wird das Array weiter durchsucht, bis zu nächsten 1. Sollte keine 1 gefunden werden, die den Anfang des Musters bildet, verwirft die Software das Array und wartet auf die nächste Messung. Ab dem ersten Durchlauf, nachdem eine Linie gefunden wurde und auch eine Position ermittelt werden konnte, startet die nächste Suche nach der Linie vor der Position, an der die letzte Linie gefunden wurde.

5 Realisierung

Das System für die Steuerung des LKW besteht aus mehreren Komponenten. Dabei gibt es die Unterscheidung zwischen Hardwarekomponenten und Softwarekomponenten. Zu Beginn wird der Laserscanner und seine Aufgabe für die Steuerung erläutert. Nachdem die PWM Signale, die im System vorkommen, aufgezeigt wurden kommt der Pegelwandler als Hardwarekomponente. Darauf wird die Platine präsentiert die im Rahmen dieser Arbeit entwickelt wurde. Zum Ende kommen dann die Softwarekomponenten, wie die PRU und ihre Programme, sowie Implementierungsdetails zu der Steuerungssoftware.

5.1 Hardwarekomponenten

Bei den Hardwarekomponenten wird der Laserscanner, die PWM Signale vorgestellt. Danach wird der Pegelwandler verwendet wird und zum Schluss mit der Platine die ganze Hardware dargestellt.

5.1.1 Laserscanner

Die Laserplatine wird für die Erkennung der Linie benötigt. Der BeagleBone liest dafür auf einem Pin die Rohdaten ein, welche die Laserplatine auf dem DATA Pin sendet. Da die Laserplatine mit $5V$ betrieben wird, der BeagleBone an den GPIO Pins aber nur $3.4V$ veträgt, muss das Signal vorher durch den Pegelwandler (sieht Abschnitt 2.5.2) herabgesetzt werden. Die $5V$ Versorgungsspannung und die GND (Erdung) liefert der BeagleBone für die Laserplatine. Auch das $25Hz$ CLK Signal aus der Laserplatine greift der BeagleBone über einen GPIO Pin ab.

5.1.2 PWM Steuer Signale

Die Steuereinheit des LKW (siehe Abschnitt 2.1) bekommt vom Empfänger PPM Steuersignale (Siehe Abschnitt: 2.5.5). Diese werden im autonomen Betrieb vom BeagleBone Black mit PWM Signalen nachgebildet. Die PWM Signale werden zur Initialisierung mit den default Werten gestartet. Dadurch befinden sich der Motor und die Servos in einem definiertem Zustand. Die default Werte (siehe Tabelle 5.1) bedeuten für den Motor, dass er still steht. Für die Lenkung

stehen die Räder gerade, aus wenn die default Werte geladen werden. Das Signal, des vom BeagleBone erzeugten PWM Signals, ist in Abbildung 5.1 zu sehen und gezeigt wird das Signal in seiner Grundstellung für die Lenkung nach der Initialisierung.

Folgende Signale werden vom BeagleBone Black erzeugt:

Signalname	Steuert	Min. Wert	Default Wert	Max. Wert
pwm_stear	Lenkung	0.7ms	1.2ms	1.7ms
pwm_engine	Motor	0.7ms	0.7ms	1.7ms

Tabelle 5.1: Auflistung der vom BeagleBone Black erzeugten PWM Signale

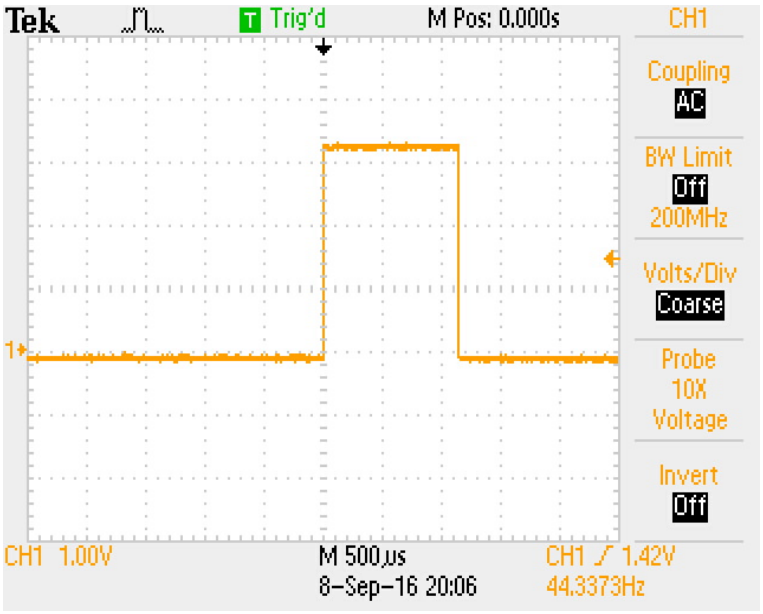

Abbildung 5.1: PPM Signal das der BeagleBone mit PWM für die Lenkung erzeugt.

5.1.3 Pegelwandler

Der BeagleBone Black besitzt einen Pin der $5V$ liefert, verwendet aber an den GPIO Pins, die keine Spannungsversorgung sind, $3.3V$. Ein Pinout der einzelnen Pins des BeagleBone ist in Abbildung 8.1 und 8.2 aufgeführt. Bei den GPIO Pins, die als Ausgang definiert werden, liefert der BeagleBone als high Pegel auch nur eine Spannung von $3.3V$. Die als Input definierten Pins vertragen nur $3.3V$. Höhere Spannungen können den BeagleBone beschädigen. Um die in Abschnitt 5.1.1 vorgestellte Laserplatine zu betreiben, werden die vom BeagleBone

bereitgestellten $5V$ benötigt. Die Signale von der Laserplatine haben eine maximale Spannung von $5V$ (Abschnitt 5.1.1). Damit der BeagleBone diese Signale verarbeiten kann, wird der Pegelwandler verwendet.
Das DATA und das CLK Signal von der Laserplatine werden nicht direkt in den BeagleBone geführt, sondern vorher durch den Pegelwandler geleitet. Im Pegelwandler werden die Signale auf 3.3V herab gesetzt. Die Funktionsweise des Pegelwandlers ist in Abschnitt 2.5.2 erläutert.

5.1.4 Platinenlayout

Für die Hardwarekomponenten, die in dieser Arbeit verwendet werden, wurde eine Platine entwickelt, um alle Hardwarekomponenten an einer Stelle zusammen zu koppeln. So kann auf der Platine durch direktes Aufstecken der jeweiligen Kabel Zugriff auf die Hardwarekomponenten bekommen. So ist der Pegelwandler auf zwei Sockelleisten aufgesteckt, um bei Bedarf einen defekten Pegelwandler zu tauschen. Die GND, HV und LV (siehe Abschnitt 2.5.2) sind auf einer separaten Steckleiste aufgelegt, da diese Steckleiste auch die anderen Hardwarekomponenten versorgt. Es wurden bei dem Pegelwandler zwei Signale durchgereicht die jeweils von rechts nach links auf Steckleisten liegen und direkt eingespeist oder abgegriffen werden können.
Ebenfalls von den $5V$ und dem GND wird der Step up Spannungswandler versorgt. Dieser wurde so eingestellt, dass er mit der $5V$ Spannung an seinem Eingang eine $12V$ Spannung am Ausgang erzeugt. Die $12V$ sind sowohl auf einer Steckleiste abgreifbar, als auch intern direkt als Spannungsversorgung für den Logischen Schalter verwendet. Die Wirkweise des Step Up Spannungswandler wird in Abschnitt 2.5.3 erläutert. Die $12V$ Versorgungsspannung betreiben den Logischen Schalter (Abschnitt 2.5.4). Dadurch benötigt der Schalter keine weiteren Spannungen, sondern nur Steuerleitungen und die Leitungen für die Signale die geschaltet werden sollen. Die Kontakte für die Steuerleitungen wurden zu einer Leitung zusammengefasst und liegen auf zwei Pins eines 10 Pin Steckers auf der Platine. Dieser Stecker hat folgende Belegung: Die ersten beiden Pins gehören der Steuerleitung zum Umschalten der Signale. Danach folgen der Reihe nach S_1 und der zugehörigen Ausgang D_1. Danach S_2, D_2, S_3, D_3, S_4, und D_4. Der Schaltplan der Platine ist in Abbildung 5.2 dargestellt.

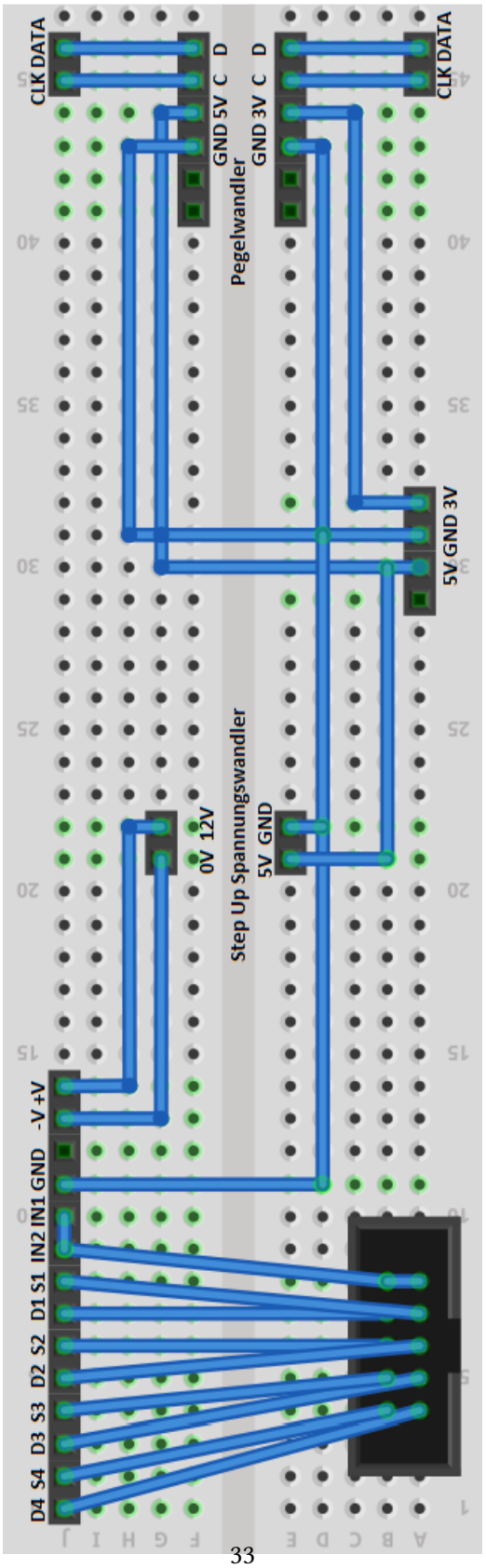

Abbildung 5.2: Platinenlayout der Hardwarekomponenten

5.2 PRU

In diesem Abschnitt wird dargestellt, welche Rahmenbedingungen erfüllt sein müssen, um die beiden PRU's, wie in der vorliegenden Arbeit, zu benutzen. Der Aufbau einer PRU wird im Kapitel Grundlagen 2.5.6 erläutert.

5.2.1 Overlays

Die Flattened Device Tree's(Abschnitt 2.7.1) können zur Laufzeit nicht mehr verändert werden. Dafür gibt es Device Tree Overlays, die in Abschnitt 2.7.2 beschrieben werden. Eine Übersicht zu allen Funktionalitäten der GPIO Pins im Beaglebone befindet sich in den Tabellen 8.1 und 8.2 im Anhang.

Für die Programmierung der beiden PRU's ist es notwendig, dass die verwendeten Pins überschrieben werden. Damit diese Pins nicht für andere Signale benutzt werden, werden sie durch das Device Tree Overlay definiert. So wie im Listing 5.1 zu sehen ist muss zuerst angegeben werden, welche Hardware benutzt wird. Danach werden die einzelnen Hardwarekomponenten den Gruppen zugeteilt, zB PRU oder GPIO. Der Name für solche Gruppen in der Hardware Datenstruktur lautet "fragment". Dort werden dann auch die Eigenschaften festgelegt, wie zum Beispiel Output oder Input. Zum Schluss muss die Hardware noch aktiviert werden. Damit ist ein Overlay vollständig und kann eingebunden werden. Da in einem Overlay nur eine PRU definitert wird, existieren zwei Overlays. Da beide PRUs ähnlich agieren, aber unterschiedliche Daten erzeugen und unterschiedliche Pins verwenden, sind sie auch separat in Overlays definiert.

```
1  / {
2     compatible = "ti,beaglebone", "ti,beaglebone-black";
3     part-number = "PRU-Overlay";
4     version = "00A0";
5     exclusive-use =
6           "P9.11", "P9.13", "P9.28", "pru0";
7
8     fragment@0 {
9        target = <&am33xx_pinmux>;
10       __overlay__ {
11          pru_pru_pins: pinmux_pru_pru_pins {   // The PRU pin modes
12             pinctrl-single,pins = <
```

```
                0x19c 0x26  // P9_28 pr1_pru0_pru_r31_3,
                            // MODE6 | INPUT | PRU
            >;
        };
    };
  };
  fragment@1 {         // Enable the PRUSS
    target = <&pruss>;
    __overlay__ {
      status = "okay";
      pinctrl-names = "default";
      pinctrl-0 = <&pru_pru_pins>;
    };
  };
};
```

Quellcode Auszug 5.1: Ausschnitt aus dem verwendeten Overlay zur Definition der PRU's.

5.2.2 Handhabung der PRU's

Da beide PRU's eigenständige Chips sind, wird für jede ein eigenes Hostprogramm benötigt, welche in den Modulen (Threads) untergebracht sind. Somit kennen nur die Module, die Werte von einer PRU benötigen, ihre zugehörige PRU. Der Rest der Software hat keinerlei Kontakt zu den PRU's und ihren Programmen. Ein weiterer Vorteil ist, dass der Quellcode für die PRU an einer zentralen Stelle liegt und nicht von mehreren Modulen verwendet wird. Ein Hostprogramm, um ein Assemblerprogramm in die PRU zu laden, findet sich in Listing 5.2.

```
#define PRU_NUM 0

void main(void) {
  printf("Starting export");
  if(getuid()!=0){
    printf("You must run this program as root. Exiting.\n");
    exit(EXIT_FAILURE);
  }
  // Initialize structure used by prussdrv_pruintc_intc
  // PRUSS_INTC_INITDATA is found in pruss_intc_mapping.h
  tpruss_intc_initdata pruss_intc_initdata = PRUSS_INTC_INITDATA;
  // Allocate and initialize memory
  prussdrv_init ();
```

```
14   prussdrv_open (PRU_EVTOUT_0);
15   // Map PRU's interrupts
16   prussdrv_pruintc_init(&pruss_intc_initdata);
17   prussdrv_exec_program (PRU_NUM, "./pru0_asm.bin");
18   printf("Export finished");
19 }
```

Quellcode Auszug 5.2: Hostprogramm, um ein Programm in die PRU zu laden

5.2.3 PRU Programm

Der BeagleBone hat zwei PRU's und jeder der beiden benötigt ein separates Programm. Die Gemeinsamkeit bei beiden Programme liegt darin, dass sie auf einem GPIO Pin das Eingangssignal abtasten. Ziel dabei ist es, die Dauer des jeweiligen High Pegels zu ermitteln.
Das Programm auf der PRU0 unterscheidet bei der Dauer des High Pegels in drei unterschiedlichen Werten. Diese drei Werte repräsentieren die drei Stufen von Geschwindigkeiten. Das Programm auf der PRU1 erfasst zwei Zustände. Diese spiegeln den Zustand des gesamten Systems wieder. Das bedeutet: beim ersten Zustand ist die manuelle Steuerung über die Fernsteuerung aktiv und im anderen Zustand wird im autonomen Betried vom BeagleBone Bacl gesteuert.

Da die PRU die Daten schneller bekommt, als die Werte vom Linux Hostprogramm abgefragt werden, schreiben beide PRU's ihre ermittelten Werte in ihren lokalen Speicher. Dies hat den Vorteil das der OCP_Port, der benötigt wird, damit die PRU auf den Speicher des Linux Systems zugreifen kann, nicht aktiviert zu werden braucht.
Die PRU startet mit Polling auf dem GPIO-Pin. Sollte dieser Pin einen High Pegel (Logisch "1") liefern, startet die PRU mit einer einfachen Zählschleife und zählt seine Takte für die Dauer des High-Pegels. Sobald der GPIO Pin einen Low-Pegel (Logisch "0") liefert, stoppt der Zähler und der ermittelte Wert wird in den lokalen Speicher der PRU geschrieben. Beide PRU's schreiben an ihre lokale Adresse 0x0000 0000. Wie aus Abbildung 5.3 ersichtlich ist, ist für PRU0 die Adresse 0x0000 0000 auf dem Data Ram0 und für die PRU1 auf dem Data Ram1. Dabei besitzt Data RAM0 im Linux System die Speicheradresse 0x4a30 0000. Data RAM1 hingegen mit dem Offset von 0x0000 2000 die Adresse 0x4a30 2000.

Start Address	PRU0	PRU1
0x0000_0000	Data 8KB RAM 0 (1)	Data 8KB RAM 1 (1)
0x0000_2000	Data 8KB RAM 1 (1)	Data 8KB RAM 0 (1)
0x0001_0000	Data 12KB RAM2 (Shared)	Data 12KB RAM2 (Shared)
0x0002_0000	INTC	INTC
0x0002_2000	PRU0 Control Registers	PRU0 Control Registers
0x0002_2400	Reserved	Reserved
0x0002_4000	PRU1 Control	PRU1 Control
0x0002_4400	Reserved	Reserved
0x0002_6000	CFG	CFG
0x0002_8000	UART 0	UART 0
0x0002_A000	Reserved	Reserved
0x0002_C000	Reserved	Reserved
0x0002_E000	IEP	IEP
0x0003_0000	eCAP 0	eCAP 0
0x0003_2000	MII_RT_CFG	MII_RT_CFG
0x0003_2400	MII_MDIO	MII_MDIO
0x0003_4000	Reserved	Reserved

Abbildung 5.3: Memory Mapping der PRU

5.3 Implementierungsdetails

5.3.1 Steuerungssoftware

Die Software ist in unterschiedliche Abschnitte unterteilt, welche jeweils einzelne Funktionalitäten kapseln. Der Hauptteil besteht primär aus dem Initialisieren der einzelnen Komponenten. Eine weitere Funktionalität im Hauptteil besteht darin, die Rohdaten der Laserplatine, die an dem GPIO Pin anliegen aufzunehmen und auszuwerten. Das Auslagern dieser Funktionalität in einen Thread hat zu Problemen durch das Scheduling[1] geführt, da der Thread unterschiedlich Rechenzeit in der CPU bekommt und so keine konstante Abtastung des GPIO Pins ermöglicht. Aus diesem Grund befindet sich diese Funktionalität weiterhin im Hauptteil der Software.

5.3.2 Positionsregelung

Für die Regelung werden die Daten der Messung nach dem Muster der Linie durchsucht. Sollte ein passender Kandidat gefunden werden, wird die Position an die Regelungsroutine weiter gegeben.
Innerhalb der Routine wird nun zunächst überprüft, ob die übergebene Position zu weit von der zuletzt gefundenen Position auf der Linie abweicht. Falls die Differenz zwischen den beiden Positionen zu groß ist, wird diese übergebene Position verworfen und die Untermethode wird beendet, um die Daten weiter nach einem Muster der Linie zu durchsuchen.

[1] Schedular ist ein Algorithmus zu Auswahl der laufenden Prozesse um Ihnen Rechenzeit in der CPU zu geben.

Ist die Differenz zwischen den Positionen innerhalb einer Toleranz, wird als nächstes der Wert ermittelt, den das PWM Signal annehmen muss, um den LKW wieder mittig auf die Linie zu lenken. Der maximale Wertebereich einer Messung liegt zwischen 0 und 40. Der Wertebereich definiert sich durch die Anzahl der Daten die eine Messung liefert. So liefert eine Abtastung des GPIO Pins immer 40 Werte vom Laserscanner. Innerhalb dieser Werte wird dann versucht eine Linie zu erkennen. Die Werte für die Toleranz der Positionen wurde empirisch ermittelt. Dazu werden unterschiedliche Fälle betrachtet:

Letzte Position	Aktuelle Position	Differenz der Positionen	Lenkeinschlag
20	20	0	0
20	23	<=3	50%
15	20	<=6	75%
30	23	>6	100%

Tabelle 5.2: Übersicht der Regelung für den Lenkeinschlag im LKW

Die neue Position wird nun aus der alten Position zusammen mit dem prozentualem Anteil des ermitteltes Lenkausschlags gebildet. Der LKW fährt mittig auf der Linie bei einer Position von 20 ± 3. Die Lenkrichtung richtet sich nach der letzten Position. Befindet sich diese Position in der unteren Hälfte des Wertebereiches, so wird der Wert für den Lenkausschlag auf die Position der letzten gefundenen Linie addiert. Befindet sich die Position in der oberen Hälfte wird der Wert subtrahiert.
Durch die Einteilung in Toleranzbereiche der Werte wird erreicht, das der LKW sich beim Fahren nicht aufschwingt[2]. Dabei ist der Vorteil, das erneute Messungen deutlicher gegensteuern können, wenn sich die Position auf der Linie wieder der Mitte nähert. Dadurch verhält sich die Lenkung deutlich weicher und nicht so abrupt.

[2] Aufschwingen beschreibt das Lenkverhalten, wenn bei Abweichungen ein starker Lenkeinschlag gesetzt wird. Dies führt zum Übersteuern, wodurch der LKW über die Linie hinaus fahren würde und sofort mit vollem Lenkeinschlag wieder gegensteuert

5.3.3 Geschwindigkeit des Fahrzeugs

Die drei Geschwindigkeiten des LKW wurden empirisch ermittelt. Als Vergleich zwischen den Geschwindigkeiten wurde eine $1m$ lange Strecke benutzt. Durch stoppen der Zeit die der LKW für die $1m$ lange Strecke benötigt, wurden drei unterschiedliche Geschwindigkeiten ermittelt. Die Ermittlung der Geschwindigkeiten ist in der Tabelle 5.3 dargestellt.

Geschwindigkeit	Dauer
Langsam	6,2s
Mittel	2.1s
Schnell	0,8s
2. Gang Schnell	0,5s

Tabelle 5.3: Übersicht zur Geschwindigkeit auf einer 1m Strecke

5.3.4 Lenkung des Fahrzeugs

Der LKW wird über einen Servo gelenkt. Dieser Servo bekommt von der Steuereinheit ein PPM Signal (Abschnitt 2.5.5), welches den Lenkeinschlag einstellt. Die Steuereinheit bekommt im manuellen Betrieb von der Fernsteuerung ebenfalls ein PPM Signal und im autonomen Betrieb ein aus einem PWM Signal erzeugtes PPM Signal. Da die Signale im manuellen und autonomen Betrieb gleich sind, lenkt der LKW bei einer Änderung des Signals in beiden Fällen gleich.

6 Verifikation

In diesem Kapitel geht es darum zu analysieren, wie können einzelne Komponenten im System getestet werden, wie sehen die Tests aus. Tests automatisieren sofern möglich. Zuerst werden die Tests dargestellt. Danach folgt die Auswertung der Steuerung und der PRU Programme.

6.1 Testerstellung

In diesem Abschnitt wird primär auf die Machbarkeit und Automatisierung von Tests eingegangen und welche Erfahrungen damit im Rahmen dieser Arbeit gemacht wurden.

6.1.1 Machbarkeit und Automatisierung

Als erstes wurden die kleinen Module, die nachher im System zusammenarbeiten sollen getestet.
Für die Erkennung der Linie wurde Polling benutzt, um die Werte aus der Laserplatine (Abschnitt 2.5.1) einzulesen. Der Algorithmus in Listing 6.1 zeigt die Auswertung der Werte aus der Laserplatine zum Abgleich mit den Mustern der Linie. Durch eine Erkennung eines Musters in den Werten der Laserplatine wird eine neue Position der Linie gesetzt. Das Testen ist durch die Position und den Vergleich der Position mit dem LKW auf der Linie möglich.
Die Funktionalität der PRU wird nur auf Terminierung einer Messung überprüft, da zur Auswertung der Werte, welche die PRU in den Speicher schreibt ein Kernelmodul notwendig ist. Dieses steht zum Zeitpunkt der Arbeit noch nicht zur Verfügung. Ein Test der Geschwindigkeit und der Lenkung ist technisch nicht umgesetzt. Die Überprüfung der Geschwindigkeit und Lenkung erfolgte durch visuelle Kontrolle der Reaktion.
Durch die Aufteilung der Komponenten in Module sind die meisten Module separat testbar. Für die testbaren Komponenten ist ein automatisierter Test entwickelt. Die PRU kann ohne ein, zum aktuellen Stand nicht betriebsbereites Kernelmodul, nicht auf Richtigkeit der Ausgabewerte überprüft werden.

```
/**
 *  Ausschnit aus dem Algorithmus welcher die Daten aussliest.
 **/
while (onOffSwitch.driving) {
                        if (interruptPin->getNumericValue() == 1 && interrupt == f
                                interrupt = false;
                                while (i < VALUES) {
                                        barcodeValues[i] = barcode.getNumericValue
                                        f << barcodeValues[i] << " ";
                                        i++;
                                }
                        } else {
       //Auswertung der Daten
     }
}
```

Quellcode Auszug 6.1: Auszug aus dem Algorithmus, der die Werte von der Laserplatine ausliest

6.1.2 Erfahrungen

Das Testen einzelner Komponenten funktioniert soweit, wie alle Informationen handhabbar sind. Im Fall der PRU klappt ein Test nicht auf Richtigkeit der Werte da ein Kernelmodul fehlt. Das Hostprogramm steuert die PRU und die von der PRU erzeugten Werte können mit Hilfe eines Kernelmoduls überprüft werden. Komponenten die direkt getestet sind, erfüllen immer ihre Funktionalität im Bereich der Testabdeckung. Sollten Fehler auftreten erkennt man meist sehr schnell die fehlenden Testfälle oder die Fehler. Bei den Komponenten die nicht selbst direkt testbar sind, ist die Fehlersuche deutlich aufwändiger, da man nicht sicher sein kann, ob der Fehler auch wirklich in der zu testenden Komponente besteht. Im Falle der PRU kann nicht genau gesagt werden ob die Daten Falsch eingelesen wurden, oder das Programm fehlerhaft ist und das Programm deshalb nicht Terminiert. Dieses Problem kostet Zeit bei der Fehlersuche, da zuerst die Ursache festgestellt werden muss.

6.1.3 Schema für die Erkennung der Linie

Zur Erkennung der Linie wurde ein kleines Testprogramm geschrieben, welches eine identische Messung durchführt, wie sie in der Steuerungssoftware durchgeführt wird. Dieses Testprogramm wertet die Anzahl der erfolgreichen Messungen aus. Aus den erfolgreichen Messungen

wurde eine in Listing 6.2 zu sehende Liste an möglichen Linienkombinationen erstellt. Um eine Vollständige Abdeckung des Erfassungsbereiches zu ermöglichen sehen die Muster für die Linien unterschiedlich aus. Die Position ist entscheidend für das Muster der Linie. Befindet sich der LKW mittig auf der Linie so erkennt der Laserscanner ein direktes Muster der Linie (zB. in Listing 6.2 Zeile 5 oder Zeile 7). Kommt der LKW weiter an den Rand des Erfassungsbereiches des Lasers verschwimmen die Linien leicht und die Gefahr eine falsche Linie zu erkennen steigt.

```
1  void initPatternArray() {
2          patternArray[0] = {1, 1, 0, 0, 1, 1, 0, 1}; //1
3          patternArray[1] = {1, 1, 0, 1, 1, 0, 1, 1}; //2
4          patternArray[2] = {1, 0, 1, 0, 0, 1, 1, 0}; //3
5          patternArray[3] = {1, 1, 1, 0, 0, 1, 1, 0}; //4
6          patternArray[4] = {1, 1, 1, 0, 0, 1, 0, 1}; //5
7          patternArray[5] = {1, 1, 1, 0, 0, 1, 0, 0}; //6
8          patternArray[6] = {1, 1, 1, 1, 0, 1, 0, 0}; //7
9          patternArray[7] = {1, 0, 1, 1, 1, 1, 0, 0}; //8
10         patternArray[8] = {1, 1, 1, 1, 0, 1, 0, 0}; //9
11         patternArray[9] = {1, 0, 1, 1, 0, 1, 0, 0}; //10
12 }
```

Quellcode Auszug 6.2: Grundmuster zur Erkennung der Linie aus den Messwerten

6.2 Test der Linienerkennung

Um einen präzisen Test der Linienerkennung, zu ermöglichen wurde ein Testprogramm geschrieben. Ein Auszug daraus ist in Listing 6.3 zu sehen. Das Testprogramm zeigt ein Verhältnis zur Anzahl der Messungen und dem Finden einer Linie aus den Messungen auf. Abbildung 6.4 zeigt eine Ausgabe eines Tests über die Anzahl von 1000 Messungen.

```
/**
 * Messtest der Linien aus den Daten des Laserscanners.
 **/
while (k <= 100) {
  if (interruptPin->getNumericValue() == 1) {
    while (interruptPin->getNumericValue() == 1) {
      barcodeValues[i] = barcode.getNumericValue();
      i++;
    }
  } else {
    if (i > 0) {
      //with lastLine
      find = lineDetect(lastLine - 10, i); //--> Anhang
      if (find >= 0) {
        for (int i = 0;i < barcodeValues.size(); ++i) {
          cout << barcodeValues[i] << " ";
        }
        cout << endl;
        lines++;
      }
      i = 0;
    }
  }
  k++;
}
cout << "Anzahl der Messungen: " << k << endl;
cout << "Anzahl der Gefundenen Linien: " << lines << endl;
cout << "Gefundene Linien: " << k/lines*100.0 << "%" << endl;
return 0;
```

Quellcode Auszug 6.3: Auszug aus dem Testprogramm für die Erkennung der Linie

```
1 //--Messung auf stehender Linie
2 root@beaglebone:~# /root/Beagle/test_line;exit
3 Anzahl der Messungen: 1000
4 Anzahl der Gefundenen Linien: 197
5 Gefundene Linien: 19.7%
6 logout
7
8 //--Messung auf langsam bewegter Linie
9 root@beaglebone:~# /root/Beagle/test_line;exit
10 Anzahl der Messungen: 1000
11 Anzahl der Gefundenen Linien: 117
12 Gefundene Linien: 11.7%
13 logout
14
15
16 //-- Messung auf bewegter Linie
17 root@beaglebone:~# /root/Beagle/test_line;exit
18 Anzahl der Messungen: 1000
19 Anzahl der Gefundenen Linien: 204
20 Gefundene Linien: 20.4%
21 logout
```

Quellcode Auszug 6.4: Ausgabe der Messung für den Test der Linienerkennung

6.3 Test der PRU Programme

Die PRU Programm wurden nur auf erfolgreiche Terminierung getestet. Durch den Umstand das ein Kernelmodul notwendig ist, um die Werte aus den Physikalischen Speicher auszulesen weil ein Linux Programm im Userspace keinen Zugriff auf den physikalischen Speicher hat. Da der Speicher der PRU auf den Hauptspeicher gemapped ist, kommt man nicht um ein Kernelmodul herum um mit einem Programm die Werte auszulesen. Listing 6.5 zeigt das PRU Programm, welches einmalig die Signale auf dem GPIO Pin auf die Dauer des High Pegels abtastet und den Wert in den Speicher schreibt. Nach dem Schreiben des Wertes sendet das PRU Programm ein Event an das Hostprogramm zurück und terminiert dann. Bekommt das Hostprogramm kein Event von der PRU zurück, ist ein Fehler aufgetreten.

```
1  .origin 0                    //start of program in pru memory
2  .entrypoint START            //program entry point
3
4  START:
5    MOV     r0, 0x00000000
6    MOV     r2, 0              //Counter set to zero
7
8  WAIT:                        //wait until signal rises up
9    QBBC    WAIT, r6.t31       //if GPIO high? if not WAIT
10
11 COUNT:                       //count time the signal is high
12   ADD     r2, r2, 1          //counter++
13   QBBS    COUNT, r6.t31      //if GPIO low? if not COUNT
14
15 MEMORY:
16   //writing to Memory for reading in c program
17   SBCO    r2, r0, 0, 4       // write r2, offset 0, bytes 4
18
19 END:
20   MOV    r31.b0, 35          //signal host program
21   halt                         //Halt the pru program
```

Quellcode Auszug 6.5: PRU Testprogramm zum Erkennen einer erfolgreichen Terminierung

7 Ergebnisse & Fazit

Die erreichte Funktionalität und ein Fazit zu der vorliegenden Arbeit ist in diesem Kapitel beschrieben. Zuerst wird die Machbarkeit der Arbeit näher aufgeschlüsselt, gefolgt von der erreichten Funktionalität. Danach wird erläutert was funktioniert, was gar nicht funktioniert oder welche Verbesserungen mögliche sind.

7.1 Machbarkeit

Der hohe Aufwand der nötig ist, um dieses Thema im Rahmen einer Bachelorarbeit zu bearbeiten scheint mir ein wenig zu groß, als dass es in den 6 Monaten zu schaffen ist. Die unterschiedlichen Teilbereiche in die ich mich einarbeiten muss, benötigen einen sehr großen Teil der Zeit. Da sind technische Fragen und Probleme, wie die Auswahl des Mikrocontrollers, das Finden eines Pegelwandlers, um die Spannungen für den BeagleBone tauglich zu machen oder die Wechselschaltung zwischen den Signalen vom BeagleBone und der vom Empfänger im manuellen Betrieb. Wobei diese Wechselschaltung auch noch einen eigenen Step-Up Konverter benötigt, da das Bauteil eine Versorgungsspannung von 12V benötigt, der BeagleBone aber ja nur maximal 5V liefert. Dazu kommt die Analyse des Barcodescanners und das Betreiben der Laserplatine zum Abtasten einer Linie. Alles in allem ist es möglich das Projekt so zu realisieren, allerdings sind dafür die 6 Monate sehr knapp bemessen. Zudem weist das einfache Abtasten der Linie mit einem Laserscanner Grenzen auf, welche durch die Beschaffenheit des Untergrundes zu verfälschten Ergebnisse führen kann. Diese sind wiederum nur mit sehr hohem Aufwand abzufangen.

7.2 Erreichte Funktionalität

Die Funktionalität, die erreicht wurde wird in den folgenden Abschnitten näher beleuchtet.

7.2.1 Fahren

Das Fahren des LKW wurde durch einfaches Nachbilden es PPM Signals erreicht, das die Fernsteuerung im manuellen Betrieb liefert. Über die Verschiebung des PPM Signals kann stufenlos über den PWM Ausgang vom BeagleBone die Geschwindigkeit reguliert werden. Es sind drei aufsteigende Geschwindigkeiten fest eingestellt. Das Zusammenspiel der Einzelnen Komponenten ist nicht vollständig implementiert. So kann zu derzeitigem Stand der Software die Geschwindigkeit über die Startparameter "1", "2" und "3" eingestellt und bei laufender Software nicht mehr verändert werden.

7.2.2 Linien Erkennung

Nach langer Analyse des Barcode Scanners wird über Polling eine gute Erkennung der Linie erreicht. Die Auswertung der Rohdaten verändert sich mit der Position des LKW auf der Linie. Befindet sich der LKW sehr weit am Rand des Erfassungsbereiches des Laserscanners, erzeugt der Laserscanner ein starkes Rauschen in den Daten, was zu einer Verfälschung der Linienposition führt.

7.2.3 PRU Programme

Die PRU läuft erfolgreich durch und schreibt die ermittelten Werte in den Speicher. Das Kernelmodul ist nicht funktionsfähig. Das bedeutet, das Hostprogramm kann die Werte aus dem Speicher der PRU nicht auslesen. Eine Überprüfung auf Richtigkeit der Werte ist zu diesem Zeitpunkt nicht möglich.

7.2.4 Regelung

Regelung ist ein wichtiges Thema, da ohne ein paar Regelungen die Software direkt auf ihre Messwerte regiert. Dies kann dazu führen, dass zum Beispiel die Lenkung sehr hart, schnell und abrupt regiert. Mit der Regelung steuert die Steuerungssoftware sichtbar leichter und die Lenkung schwingt sich nicht mehr auf. Hier musste eine Lösung gefunden werden, die sowohl schnell abzuarbeiten ist und dennoch gute Ergebnisse liefert. Es wurde eine Regelung eingeführt, die nach dem erfolgreichen Finden einer Linie bei der nächsten Messung dann wieder zuerst an der gleichen Stelle nach der Linie sucht und nicht bei jeder Messung von vorne in den Messdaten beginnt. Ebenfalls wurde eine kleine Hysterese, eingebaut welche erst eine Lenkung verändert, wenn sich der LKW um einen bestimmten Wert von der vorher gefundenen Linie entfernt hat. Dadurch wurde verhindert, das die Lenkung zu zittern beginnt, wenn die Erkennung der Linie um kleine Werte schwankt, was durch das Polling der Messdaten hervorgerufen werden kann.

Zusätzlich musste eine Regelung eingeführt werden, bei der erkannt wird, wann eine in den Messdaten gefundenen Linie zu weit von der zuletzt gefundenen abweicht. Die Ermittlung von passenden Werten wurde empirisch ermittelt. Sie verhindert, dass gefundene Linien in den Messdaten, welche aber keine Linie auf der Strecke abbilden, ignoriert werden.

7.2.5 Zusammenspiel komplettes System

Der Laserscanner (Abschnitt 2.5.1)liefert die Daten an den BeagleBone. Dieser ermittelt aus den Messwerten die Position auf der Linie und stellt abhängig von der Position das PWM Signal so ein, dass der LKW wieder in die Mitte auf die Linie lenkt. Eine Geschwindigkeit kann über Startparameter in der Software eingestellt werden, jedoch ändert sich nichts bei Betätigung der Hebel an der Fernsteuerung. Grund dafür ist die fehlende Kommunikation zwischen der PRU und dem Hostprogramm. Das benötigte Kernelmodul damit das Hostprogramm die Werte aus dem Speicher der PRU auslesen kann und an die Steuerungssoftware weiter reicht ist in dieser Arbeit nicht Funktionstüchtig.

7.2.6 Grenzen

Durch die Messung mit 25Hz und der in Kapitel 6.2 beschriebenen Genauigkeit verschlechtert das Fahrverhalten bei steigender Geschwindigkeit. Bei steigender Geschwindigkeit verlängert sich die Distanz zwischen zwei erfolgreichen Messungen, was zu einer starken Lenkung führt. Der LKW fährt Schlangenlinien auf einen geraden Strecke. Die Fehler treten bei hoher Geschwindigkeit auf, wodurch der LKW leichter von der Spur abkommt.

7.2.7 Polling statt Interrupts

Da sich das Programm im Linux Userspace befindet, kann es mit der Hardware interagieren, aber keine von Hardware erzeugte Interrupts empfangen. In dieser Arbeit wurde das Polling benutzt, da durch dieses Verfahren genaue Ergebnisse erzeugt wurden, ohne den Einsatz weiterer Softwarekomponenten, die eine erweiterte Funktionalität bieten, beim GPIO Pin auszulesen. Polling bringt den BeagleBone auf eine hohe Auslastung, da er mit voller Geschwindigkeit immer wieder die Abtastung des GPIO Pins abarbeitet. Eine aufwändigere Methode, wäre ein Kernelmodul, welches die Hardware Interrupts vom GPIO Pin an die Steuerungssoftware weiter leitet. Die Steuerungssoftware muss die Auswertung der Abstände zwischen den Interrupts und die Zeiten der High Pegel übernehmen.
Eine weitere Möglichkeit ist das Erfassen der Daten auf einem GPIO Pin mittels einer der beiden PRU's auf dem BeagleBone (siehe Abschnitt 2.5.6). Dabei würde die PRU eine lange

Zeichenkette aus den Werten am GPIO Pin erstellen, welche die Steuerungssoftware dann direkt verwenden könnte, um die Postion der Linie zu suchen. Da die beiden verfügbaren PRU's auf dem BeagleBone verwendet werden, um die Geschwindigkeit und den Systemstatus (autonom oder manuell) zu erfassen und diese zudem genau sein müssen, werden die Daten für die Linie mit Polling ermittelt. Durch das Polling werden die Rohdaten von der Laserplatine aus Abschnitt 5.1.1 erfasst und können ausgewertet werden.

7.2.8 Verbesserungen

Obwohl die Steuerungssoftware nicht vollständig ist, da ein Kernelmodul für die Kommunikation zwischen der PRU und dem Linux Programm fehlt, kann man doch einige Verbesserungen an der Software planen. Die erste Verbesserung, die für den LKW von Vorteil wäre ist der Einbau einer Bremse. Dabei muss beim Bremsen das PWM Signal aus dem BeagleBone nicht auf Defaultwerte (siehe Abschnitt 5.1.2) sondern auf den Rückwärtsgang stellen, wodurch die Bremse Aktiviert wird. Wichtig dabei ist noch, sollte die Bremse zweimal hintereinander aktiviert werden, so befindet sich der LKW im Rückwärtsgang und fährt unter Umständen rückwärts weiter.
Eine weitere Verbesserung, wäre eine Hinderniserkennung. Dabei könnte ein Abstandssensor nach Vorn den LKW vor Hindernissen stoppen lassen. Diese Entfernungsmessung muss eine große Reichweite haben, da der LKW ein bisschen Strecke braucht um zum Stehen zu kommen. Dieser Abstandssensor könnte entscheiden, ob sich ein Hindernis vor dem LKW befindet oder nicht. Eine Aussage wie breit das Hindernis ist, oder ob das Hindernis nur rechts oder links steht und der LKW durch ein wenig Lenken daran vorbei kommen würde kann durch einen einzelnen Abstandssensor nicht erreicht werden.
Die Linienerkennung variabel über ein kurzes Initialisierungsprogramm zu ermöglichen, könnte helfen die Fehlmessungen bei unterschiedlichen Untergründen zu reduzieren. Dabei wird der LKW auf die Linie gestellt und das Programm ermittelt dann selbstständig die Muster der Linie, auf welcher der LKW steht. Dadurch wird das Problem für eine Erkennung der Linie aufwändiger, da die unterschiedlichen Muster eventuell anders abgearbeitet werden müssen.

7.2.9 Was funktioniert nicht

Durch den Laserscanner, der den Untergrund nur 2D abtasten kann, ist eine Auswertung über den Abstand zum Boden nicht möglich. Dadurch kann bei unebenen Untergrund nicht entschieden werden, ob die Linie auf einer glatten Ebene ist oder rechts und links der Linie Unebenheiten sind. Ein Erkennen von Löchern ist nicht möglich.

Darüber hinaus kann auch nicht unterschieden werden, auf welchem Untergrund der LKW fährt. Der Laserscanner liefert nur die Reflexionen des Untergrundes. Dadurch werden nur die Farb- und Strukturübergänge erfasst.

7.3 Abschluss

Eine Fahrsteuerung für einen Modell LKW, der einer vorgegebenen Strecke auf dem Boden folgen soll, um Messergebnisse reproduzierbar zu machen, konnte durch das Fehlen eines Kernelmoduls nicht erreicht werden. Die Abtastung einer Strecke auf dem Boden wurde jedoch erreicht. Dabei ist die Abtastung der Strecke durch einen Laserscanner, der aus einem handelsüblichen Barcodescanner genommen wurde, nur der erste Schritt. Die Auswertung der Daten aus dem Laserscanner spielt genauso eine Rolle sowie das Erkennen der Linie aus den abgetasteten Werten des Laserscanners. Durch den Aufbau einer Platine konnten Probleme, wie die Pegelunterschiede von Signalen (5V und 3V), das Umschalten zwischen zwei unterschiedlichen Steuerleitungen und das Erzeugen einer 12V Spannung aus der 5V Versorgungsspannung des BeagleBone gelöst werden. Die zwei PRU Programme erlauben es durch Hebel, die über die Fernsteuerung programmiert wurden, zu betätigen und die Pegeländerung durch den BeagleBone mitzuschreiben. Diese Werte können mit einem entsprechenden Kernelmodul später von der Steuerungssoftware verwendet werden, um die Geschwindigkeit zu regeln oder zwischen dem manuellen und autonomen Betrieb umzuschalten.
Zusammenfassend kann man sagen, dass die Entwicklung einer Steuerungssoftware für einen LKW sehr interessant ist, jedoch vom Aufwand und Umfang der unterschiedlichen Themengebiete schwierig in der veranschlagten Zeit, von 6 Monaten zu bewerkstelligen ist. Diese Arbeit hat sehr viele Themengebiete die alle erarbeitet wurden und dadurch einen großen Wissenswert erzeugen.

8 Anhang

8.1 Scania LKW Manual

Eine genaue Bedienungsanleitung zu dem in dieser Arbeit zur Anwendung kommenden Scania LKW Truck 1:14 befindet sind unter:
http://www.tamiya.com/english/rc/rcmanual/56318.pdf
Das gesamte Manual ist eine Aufbauanleitung des LKW worin Schritt für Schritt der LKW zusammengebaut wird. Dort befinden sich Explosionszeichnungen des LKW und die Auflistung der Ersatzteile für diesen LKW.

8.2 Spektrum DX18t

Eine umfangreiche Beschreibung der Fernsteuerung befindet sich auf der Website:
http://www.horizonhobby.de/spektrum-dx18t.html
Dort befindet sich eine komplette Beschreibung der Fernsteuerung, die Auflistung der Spezifikationen und im Downloadbereich lassen sich bereift Basisprogramme für diverse Flugmodelle downloaden. Diese Programme erleichtern den Einstieg in die Fernsteuerung, da viele Einstellungen bereits getätigt sind und nichts vergessen werden kann.

8.3 BeagleBone Black Pinout Header

Beaglebone Black P8 Header

Head_pin	$PINS	ADDR/OFFSET	GPIO NO.	Name	Mode7	Mode6	Mode5	Mode4	Mode3	Mode2	Mode1	Mode0	PIN	Notes
P8_01				DGND										Ground
P8_02				DGND										Ground
P8_03	6	0x818/018	38	GPIO1_6	gpio1[6]						mmc1_dat6	gpmc_ad6	R9	Used on Board (Group: pinmux_emmc2_pins)
P8_04	7	0x81c/01c	39	GPIO1_7	gpio1[7]						mmc1_dat7	gpmc_ad7	T9	Used on Board (Group: pinmux_emmc2_pins)
P8_05	2	0x808/008	34	GPIO1_2	gpio1[2]						mmc1_dat2	gpmc_ad2	R8	Used on Board (Group: pinmux_emmc2_pins)
P8_06	3	0x80c/00c	35	GPIO1_3	gpio1[3]						mmc1_dat3	gpmc_ad3	T8	Used on Board (Group: pinmux_emmc2_pins)
P8_07	36	0x890/090	66	TIMER4	gpio2[2]					timer4		gpmc_advn_ale	R7	
P8_08	37	0x894/094	67	TIMER7	gpio2[3]					timer7		gpmc_oen_ren	T7	
P8_09	39	0x89c/09c	69	TIMER5	gpio2[5]					timer5		gpmc_be0n_cle	T6	
P8_10	38	0x898/098	68	TIMER6	gpio2[4]					timer6		gpmc_wen	U6	
P8_11	13	0x834/034	45	GPIO1_13	gpio1[13]	pr1_pru0_pru_r30_15		eQEP2B_in	mmc2_dat1	mmc1_dat5	lcd_data18	gpmc_ad13	R12	
P8_12	12	0x830/030	44	GPIO1_12	gpio1[12]	pr1_pru0_pru_r30_14		EQEP2A_IN	MMC2_DAT0	MMC1_DAT4	LCD_DATA19	GPMC_AD12	T12	
P8_13	9	0x824/024	23	EHRPWM2B	gpio0[23]			ehrpwm2B	mmc2_dat5	mmc1_dat1	lcd_data22	gpmc_ad9	T10	
P8_14	10	0x828/028	26	GPIO0_26	gpio0[26]			ehrpwm2_tripzone_in	mmc2_dat6	mmc1_dat2	lcd_data21	gpmc_ad10	T11	
P8_15	15	0x83c/03c	47	GPIO1_15	gpio1[15]	pr1_pru0_pru_r31_15		eQEP2_strobe	mmc2_dat3	mmc1_dat7	lcd_data16	gpmc_ad15	U13	
P8_16	14	0x838/038	46	GPIO1_14	gpio1[14]	pr1_pru0_pru_r31_14		eQEP2_index	mmc2_dat2	mmc1_dat6	lcd_data17	gpmc_ad14	V13	
P8_17	11	0x82c/02c	27	GPIO0_27	gpio0[27]			ehrpwm0_synco	mmc2_dat7	mmc1_dat3	lcd_data20	gpmc_ad11	U12	
P8_18	35	0x88c/08c	65	GPIO2_1	gpio2[1]	mcasp0_fsr			mmc2_clk	gpmc_wait1	lcd_memory_clk	gpmc_clk_mux0	V12	
P8_19	8	0x820/020	22	EHRPWM2A	gpio0[22]			ehrpwm2A	mmc2_dat4	mmc1_dat0	lcd_data23	gpmc_ad8	U10	
P8_20	33	0x884/084	63	GPIO1_31	gpio1[31]	pr1_pru1_pru_r31_13	pr1_pru1_pru_r30_13			mmc1_cmd	gpmc_be1n	gpmc_csn2	V9	Used on Board (Group: pinmux_emmc2_pins)
P8_21	32	0x880/080	62	GPIO1_30	gpio1[30]	pr1_pru1_pru_r31_12	pr1_pru1_pru_r30_12			mmc1_clk	gpmc_clk	gpmc_csn1	U9	Used on Board (Group: pinmux_emmc2_pins)
P8_22	5	0x814/014	37	GPIO1_5	gpio1[5]						mmc1_dat5	gpmc_ad5	V8	Used on Board (Group: pinmux_emmc2_pins)
P8_23	4	0x810/010	36	GPIO1_4	gpio1[4]						mmc1_dat4	gpmc_ad4	U8	Used on Board (Group: pinmux_emmc2_pins)
P8_24	1	0x804/004	33	GPIO1_1	gpio1[1]						mmc1_dat1	gpmc_ad1	V7	Used on Board (Group: pinmux_emmc2_pins)
P8_25	0	0x800/000	32	GPIO1_0	gpio1[0]						mmc1_dat0	gpmc_ad0	U7	Used on Board (Group: pinmux_emmc2_pins)
P8_26	31	0x87c/07c	61	GPIO1_29	gpio1[29]							gpmc_csn0	V6	
P8_27	56	0x8e0/0e0	86	GPIO2_22	gpio2[22]	pr1_pru1_pru_r31_8	pr1_pru1_pru_r30_8				gpmc_a8	lcd_vsync	U5	Allocated (Group: nxp_hdmi_bonelt_pins)
P8_28	58	0x8e8/0e8	88	GPIO2_24	gpio2[24]	pr1_pru1_pru_r31_10	pr1_pru1_pru_r30_10				gpmc_a10	lcd_pclk	V5	Allocated (Group: nxp_hdmi_bonelt_pins)
P8_29	57	0x8e4/0e4	87	GPIO2_23	gpio2[23]	pr1_pru1_pru_r31_9	pr1_pru1_pru_r30_9				gpmc_a9	lcd_hsync	R5	Allocated (Group: nxp_hdmi_bonelt_pins)
P8_30	59	0x8ec/0ec	89	GPIO2_25	gpio2[25]						gpmc_a11	lcd_ac_bias_en	R6	Allocated (Group: nxp_hdmi_bonelt_pins)
P8_31	54	0x8d8/0d8	10	UART5_CTSN	gpio0[10]	uart5_ctsn		uart5_rxd	mcasp0_axr1	eQEP1_index	gpmc_a18	lcd_data14	V4	Allocated (Group: nxp_hdmi_bonelt_pins)
P8_32	55	0x8dc/0dc	11	UART5_RTSN	gpio0[11]	uart5_rtsn		mcasp0_axr3	mcasp0_ahclkx	eQEP1_strobe	gpmc_a19	lcd_data15	T5	Allocated (Group: nxp_hdmi_bonelt_pins)
P8_33	53	0x8d4/0d4	9	UART4_RTSN	gpio0[9]	uart4_rtsn		mcasp0_axr3	mcasp0_fsr	eQEP1B_in	gpmc_a17	lcd_data13	V3	Allocated (Group: nxp_hdmi_bonelt_pins)
P8_34	51	0x8cc/0cc	81	UART3_RTSN	gpio2[17]	uart3_rtsn		mcasp0_axr2	mcasp0_ahclkr	ehrpwm1B	gpmc_a15	lcd_data11	U4	Allocated (Group: nxp_hdmi_bonelt_pins)
P8_35	52	0x8d0/0d0	8	UART4_CTSN	gpio0[8]	uart4_ctsn		mcasp0_axr2	mcasp0_aclkr	eQEP1A_in	gpmc_a16	lcd_data12	V2	Allocated (Group: nxp_hdmi_bonelt_pins)
P8_36	50	0x8c8/0c8	80	UART3_CTSN	gpio2[16]	uart3_ctsn			mcasp0_axr0	ehrpwm1A	gpmc_a14	lcd_data10	U3	Allocated (Group: nxp_hdmi_bonelt_pins)
P8_37	48	0x8c0/0c0	78	UART5_TXD	gpio2[14]	uart2_ctsn		uart5_txd	mcasp0_aclkx	ehrpwm1_tripzone_in	gpmc_a12	lcd_data8	U1	Allocated (Group: nxp_hdmi_bonelt_pins)
P8_38	49	0x8c4/0c4	79	UART5_RXD	gpio2[15]	uart2_rtsn		uart5_rxd	mcasp0_fsx	ehrpwm0_synco	gpmc_a13	lcd_data9	U2	Allocated (Group: nxp_hdmi_bonelt_pins)
P8_39	46	0x8b8/0b8	76	GPIO2_12	gpio2[12]	pr1_pru1_pru_r31_6	pr1_pru1_pru_r30_6		eQEP2_index		gpmc_a6	lcd_data6	T3	Allocated (Group: nxp_hdmi_bonelt_pins)
P8_40	47	0x8bc/0bc	77	GPIO2_13	gpio2[13]	pr1_pru1_pru_r31_7	pr1_pru1_pru_r30_7	pr1_edio_data_out7	eQEP2_strobe		gpmc_a7	lcd_data7	T4	Allocated (Group: nxp_hdmi_bonelt_pins)
P8_41	44	0x8b0/0b0	74	GPIO2_10	gpio2[10]	pr1_pru1_pru_r31_4	pr1_pru1_pru_r30_4		eQEP2A_in		gpmc_a4	lcd_data4	T1	Allocated (Group: nxp_hdmi_bonelt_pins)
P8_42	45	0x8b4/0b4	75	GPIO2_11	gpio2[11]	pr1_pru1_pru_r31_5	pr1_pru1_pru_r30_5		eQEP2B_in		gpmc_a5	lcd_data5	T2	Allocated (Group: nxp_hdmi_bonelt_pins)
P8_43	42	0x8a8/0a8	72	GPIO2_8	gpio2[8]	pr1_pru1_pru_r31_2	pr1_pru1_pru_r30_2		ehrpwm2_tripzone_in		gpmc_a2	lcd_data2	R3	Allocated (Group: nxp_hdmi_bonelt_pins)
P8_44	43	0x8ac/0ac	73	GPIO2_9	gpio2[9]	pr1_pru1_pru_r31_3	pr1_pru1_pru_r30_3		ehrpwm0_synco		gpmc_a3	lcd_data3	R4	Allocated (Group: nxp_hdmi_bonelt_pins)
P8_45	40	0x8a0/0a0	70	GPIO2_6	gpio2[6]	pr1_pru1_pru_r31_0	pr1_pru1_pru_r30_0		ehrpwm2A		gpmc_a0	lcd_data0	R1	Allocated (Group: nxp_hdmi_bonelt_pins)
P8_46	41	0x8a4/0a4	71	GPIO2_7	gpio2[7]	pr1_pru1_pru_r31_1	pr1_pru1_pru_r30_1		ehrpwm2B		gpmc_a1	lcd_data1	R2	Allocated (Group: nxp_hdmi_bonelt_pins)
P9 Header	cat $PINS	ADDR +	GPIO NO.	Name	Mode 7	Mode 6	Mode 5	Mode 4	Mode 3	Mode 2	Mode 1	Mode 0	CPU	Updates Available at www.derekmolloy.ie
		44e10000	(Mode 7)										PIN	
		Offset from:												
		44e10800												

User LEDs				
USR0	21	0x854/054	53	GPIO1_21
USR1	22	0x858/058	86	GPIO1_22
USR2	23	0x85c/05c	87	GPIO1_23
USR3	24	0x860/060	88	GPIO1_24

GPIO Settings				
Bit 6	Bit 5	Bit 4	Bit 3	Bit 2,1,0
Slew Control	Receiver Active	Pullup/Pulldown	Enable Pullup/down	Mux Mode
0 Fast	0 Disable	0 Pulldown select	0 Enabled	000 Mode 0 to
1 Slow	1 Enable	1 Pullup select	1 Disabled	111 Mode 7

e.g. OUTPUT GPIO(mode7) 0x07 pulldown, 0x17 pullup, 0x?f no pullup/down

e.g. INPUT GPIO(mode7) 0x27 pulldown, 0x37 pullup, 0x?f no pullup/down

Abbildung 8.1: Übersicht über die GPIO Pins und deren Modi auf dem BeagleBone Header P8

Beaglebone Black P9 Header

Head_pin	$PINS	ADDR/OFFSET	Name	GPIO NO.	Mode7	Mode6	Mode5	Mode4	Mode3	Mode2	Mode1	Mode0	PIN	Notes
P9_01			GND											Ground
P9_02			GND											Ground
P9_03			DC_3.3V											250mA Max Current
P9_04			DC_3.3V											250mA Max Current
P9_05			VDD_5V											1A Max Current (only if DC jack powered)
P9_06			VDD_5V											1A Max Current (only if DC jack powered)
P9_07			SYS_5V											250mA Max Current
P9_08			SYS_5V											250mA Max Current
P9_09			PWR_BUT											Has a 5V Level (pulled up by TPS65217C)
P9_10			SYS_RESETn									RESET_OUT	A10	
P9_11	28	0x870/070	UART4_RXD	30	gpio0[30]	uart4_rxd_mux2		mmc1_sdcd	rmii2_crs_dv	gpmc_csn4	mii2_crs	gpmc_wait0	T17	NB: GPIOs limit current to 4-6mA output
P9_12	30	0x878/078	GPIO1_28	60	gpio1[28]	mcasp0_aclkr_mux3		gpmc_dir	mmc2_dat3	gpmc_csn6	mii2_col	gpmc_be1n	U18	and approx. 8mA on input.
P9_13	29	0x874/074	UART4_TXD	31	gpio0[31]	uart4_txd_mux2		mmc2_sdcd	rmii2_rxerr	gpmc_csn5	mii2_rxerr	gpmc_wpn	U17	
P9_14	18	0x848/048	EHRPWM1A	50	gpio1[18]	ehrpwm1A_mux1		gpmc_a18	mmc2_dat1	rgmii2_td3	mii2_txd3	gpmc_a2	U14	
P9_15	16	0x840/040	GPIO1_16	48	gpio1[16]	ehrpwm1_tripzone_input		gpmc_a16	mii2_txen	rmii2_tctl	gmii2_txen	gpmc_a0	R13	
P9_16	19	0x84c/04c	EHRPWM1B	51	gpio1[19]	ehrpwm1B_mux1		gpmc_a19	mmc2_dat2	rgmii2_td2	mii2_txd2	gpmc_a3	T14	
P9_17	87	0x95c/15c	I2C1_SCL	5	gpio0[5]			pr1_uart0_txd	ehrpwm0_synci	I2C1_SCL	mmc2_sdwp	spi0_cs0	A16	
P9_18	86	0x958/158	I2C1_SDA	4	gpio0[4]			pr1_uart0_rxd	ehrpwm0_tripzone	I2C1_SDA	mmc1_sdwp	spi0_d1	B16	
P9_19	95	0x97c/17c	I2C2_SCL	13	gpio0[13]		pr1_uart0_rts_n	spi1_cs1	I2C2_SCL	dcan0_rx	timer5	uart1_rtsn	D17	Allocated (Group: pinmux_i2c2_pins)
P9_20	94	0x978/178	I2C2_SDA	12	gpio0[12]		pr1_uart0_cts_n	spi1_cs0	I2C2_SDA	dcan0_tx	timer6	uart1_ctsn	D18	Allocated (Group: pinmux_i2c2_pins)
P9_21	85	0x954/154	UART2_TXD	3	gpio0[3]	EMU3_mux1		pr1_uart0_rts_n	ehrpwm0B	I2C2_SCL	uart2_txd	spi0_d0	B17	
P9_22	84	0x950/150	UART2_RXD	2	gpio0[2]	EMU2_mux1		pr1_uart0_cts_n	ehrpwm0A	I2C2_SDA	uart2_rxd	spi0_sclk	A17	
P9_23	17	0x844/044	GPIO1_17	49	gpio1[17]	ehrpwm0_synco		gpmc_a17	mmc2_dat0	rgmii2_rxdv	gmii2_rxdv	gpmc_a1	V14	
P9_24	97	0x984/184	UART1_TXD	15	gpio0[15]	pr1_pru0_pru_r31_16	pr1_uart0_txd		I2C1_SCL	dcan1_rx	mmc2_sdwp	uart1_txd	D15	
P9_25	107	0x9ac/1ac	GPIO3_21	117	gpio3[21]	pr1_pru0_pru_r31_7	pr1_pru0_pru_r30_7	EMU4_mux2	mcasp1_axr1	mcasp0_axr3	eQEP0_strobe	mcasp0_ahclkx	A14	Allocated (Group: mcasp0_pins)
P9_26	96	0x980/180	UART1_RXD	14	gpio0[14]	pr1_pru1_pru_r31_16	pr1_uart0_rxd		I2C1_SDA	dcan1_tx	mmc1_sdwp	uart1_rxd	D16	
P9_27	105	0x9a4/1a4	GPIO3_19	115	gpio3[19]	pr1_pru0_pru_r31_5	pr1_pru0_pru_r30_5	EMU2_mux2	mcasp1_fsx	mcasp0_axr3	eQEP0B_in	mcasp0_fsr	C13	
P9_28	103	0x99c/19c	SPI1_CS0	113	gpio3[17]	pr1_pru0_pru_r31_3	pr1_pru0_pru_r30_3	eCAP2_in_PWM2_out	spi1_cs0	mcasp0_axr2	ehrpwm0_synci	mcasp0_ahclkr	C12	Allocated (Group: mcasp0_pins)
P9_29	101	0x994/194	SPI1_D0	111	gpio3[15]	pr1_pru0_pru_r31_1	pr1_pru0_pru_r30_1	mmc1_sdcd_mux1	spi1_d0		ehrpwm0B	mcasp0_fsx	B13	Allocated (Group: mcasp0_pins)
P9_30	102	0x998/198	SPI1_D1	112	gpio3[16]	pr1_pru0_pru_r31_2	pr1_pru0_pru_r30_2	mmc2_sdcd_mux1	spi1_d1		ehrpwm0_tripzone	mcasp0_axr0	D12	Allocated? Mcasp0_pins? Check...
P9_31	100	0x990/190	SPI1_SCLK	110	gpio3[14]	pr1_pru0_pru_r31_0	pr1_pru0_pru_r30_0	mmc0_sdcd_mux1	spi1_sclk		ehrpwm0A	mcasp0_aclkx	A13	Allocated (Group: mcasp0_pins)
P9_32			VADC											Voltage Reference for ADC (NB: 1.8V)
P9_33			AIN4										C8	NB: 1.8V tolerant
P9_34			AGND											Ground for ADC
P9_35			AIN6										A8	NB: 1.8V tolerant
P9_36			AIN5										B8	NB: 1.8V tolerant
P9_37			AIN2										B7	NB: 1.8V tolerant
P9_38			AIN3										A7	NB: 1.8V tolerant
P9_39			AIN0										B6	NB: 1.8V tolerant
P9_40			AIN1										C7	NB: 1.8V tolerant
P9_41A	109	0x9b4/1b4	CLKOUT2	20	gpio0[20]	EMU3_mux0	pr1_pru0_pru_r31_16	timer7_mux1	clkout2	tclkin		xdma_event_intr1	D14	Both signals are connected to P21 of P11
P9_41B		0x9a8/1a8	GPIO3_20	116	gpio3[20]	pr1_pru0_pru_r31_6	pr1_pru0_pru_r30_6	emu3	Mcasp1_axr0		eQEP0_index	mcasp0_axr1	D13	Both signals are connected to P21 of P11
P9_42A	89	0x964/164	GPIO0_7	7	gpio0[7]	xdma_event_intr2	mmc0_sdwp	spi1_sclk	pr1_ecap0_ecap_capin_apwm_o	spi1_cs1	uart3_txd	eCAP0_in_PWM0_out	C18	Both signals are connected to P22 of P11
P9_42B		0x9a0/1a0	GPIO3_18	114	gpio3[18]	pr1_pru0_pru_r31_4	pr1_pru0_pru_r30_4		Mcasp1_aclkx	Mcaspo_axr2	eQEP0A_in	Mcasp0_aclkr	B12	Both signals are connected to P22 of P11 Allocated (Group:mcasp0_pins)
P9_43			GND											- See Pg.50 of the SRM
P9_44			GND											Ground
P9_45			GND											Ground
P9_46			GND											Ground
														For updates see: www.derekmolloy.ie
P9 Header	cat $PINS	ADDR +	Name	GPIO NO.	Mode 7						Mode 1	Mode 0	CPU	Notes
	Allocated	44e10000		(Mode 7)										Please e-mail me directly at:
		Offset from:												derek@derekmolloy.ie
		44e10800												if you notice a mistake
														Thanks Frank for the PRU work!

GPIO Settings				
Bit 6	Bit 5	Bit 4	Bit 3	Bit 2,1,0
Slew Control	Receiver Active	Pullup/Pulldown	Enable Pullup/Pulldown	Mux Mode
0 Fast	0 Disable	0 Pulldown select	0 Enabled	000 Mode 0 to
1 Slow	1 Enable	1 Pullup select	1 Disabled	111 Mode 7

e.g. OUTPUT GPIO(mode7) 0x07 pulldown, 0x17 pullup, 0x?f no pullup/down

e.g. INPUT GPIO(mode7) 0x27 pulldown, 0x37 pullup, 0x?f no pullup/down

Abbildung 8.2: Übersicht über die GPIO Pins und deren Modi auf dem BeagleBone Header P9

Literaturverzeichnis

[1] : *BeagleBoard.org - BlackLib C++ Library.* – URL http://beagleboard.org/project/blacklib/. – Zugriffsdatum: 2016-10-12

[2] : *BeagleBoard.org - libpruio-0.2 (fast and easy D/A - I/O).* – URL http://beagleboard.org/project/libpruio/. – Zugriffsdatum: 2016-10-12

[3] : *BeagleBone Black PRU: Hello World Âń Small Golden Sceptre.* – URL http://mythopoeic.org/bbb-pru-minimal/. – Zugriffsdatum: 2016-10-12

[4] : *deeplyembedded/ReSyLib.* – URL https://github.com/deeplyembedded/ReSyLib. – Zugriffsdatum: 2016-08-31

[5] : *Derek Molloy Electronics.* – URL http://derekmolloy.ie/. – Zugriffsdatum: 2016-10-12

[6] : *Exploring BeagleBone.* – URL http://exploringbeaglebone.com/. – Zugriffsdatum: 2016-10-12

[7] : *How to use GPIO signals - RidgeRun Developer Connection.* – URL http://developer.ridgerun.com/wiki/index.php?title=How_to_use_GPIO_signals. – Zugriffsdatum: 2016-10-12

[8] : *INIBUD 2 x DC-DC Step Up Spannungswandler in 3V-32V zu out 5V-35V XL6009 400KHz 4A Max: Amazon.de: Gewerbe, Industrie & Wissenschaft.* – URL https://www.amazon.de/gp/product/B01E086QZ0/ref=oh_aui_detailpage_o03_s00?ie=UTF8&psc=1. – Zugriffsdatum: 2016-10-12

[9] : *An Introduction To The BeagleBone PRU | Hackaday.* – URL http://hackaday.com/2014/06/22/an-introduction-to-the-beaglebone-pru/. – Zugriffsdatum: 2016-08-31

[10] : *The Linux Kernel Module Programming Guide.* – URL http://tldp.org/LDP/lkmpg/2.6/html/. – Zugriffsdatum: 2016-10-12

[11] : *Pegelwandler - Watterott electronic.* – URL http://www.watterott.com/de/Pegelwandler. – Zugriffsdatum: 2016-10-12

[12] : *Pegelwandler - Watterott electronic.* – URL http://www.watterott.com/de/Pegelwandler. – Zugriffsdatum: 2016-10-21

[13] : *PRU Assembly Instructions - Texas Instruments Wiki.* – URL http://processors.wiki.ti.com/index.php/PRU_Assembly_Instructions. – Zugriffsdatum: 2016-10-12

[14] : *PRU Training: Hands-on Labs - Texas Instruments Wiki.* – URL http://processors.wiki.ti.com/index.php/PRU_Training:_Hands-on_Labs. – Zugriffsdatum: 2016-10-12

[15] : *RC PPM Signal - MFTech.* – URL http://www.mftech.de/ppm.htm. – Zugriffsdatum: 2016-08-31

[16] : *TaoTronics USB Laser-Barcodescanner: Amazon.de: BÃ¼robedarf & Schreibwaren.* – URL https://www.amazon.de/gp/product/B00FEALRWG/ref=s9_top_hd_bw_b156bH_g229_i3?pf_rd_m=A3JWKAKR8XB7XF&pf_rd_s=merchandised-search-3&pf_rd_r=1PNSR7E8EG3HWHRNDV3N&pf_rd_t=101&pf_rd_p=baac8a60-9fc7-55a6-8604-194b7bbd1ab2&pf_rd_i=15993351. – Zugriffsdatum: 2016-10-12

[17] : *Ti AM33XX PRUSSv2 - eLinux.org.* – URL http://elinux.org/Ti_AM33XX_PRUSSv2#Beaglebone_PRU_connections_and_modes. – Zugriffsdatum: 2016-10-12

[18] : *BeagleBone Black: How to Get Interrupts Through Linux GPIO.* März 2014. – URL https://www.linux.com/learn/beaglebone-black-how-get-interrupts-through-linux-gpio. – Zugriffsdatum: 2016-10-12

Glossar

BIOS	Englisch: Basic input/output system. Wird zum starten von x86-Computer Architekturen benötigt.
GND	Ground - Erdung. Ist der elektrische Nullpunkt zu dem Spannungen gemessen werden..
GPIO	General-purpose input/output - Beschreibt einen Pin eines Mikrokontrollers den der User zur Laufzeit lesen oder mit Werten beschreiben kann.
HDMI	High Definition Multimedia Interface - Eine Schnittstelle die digitale Bild- und Tonübertragungen handhabt.
PPM	Englisch: pulse-position-modulation Deutsch: Puls-Pausen-Modulation.
PRU	Programmable Realtime Unit.
PRUSS	Programmable Realtime Unit Subsystem.
PWM	Pulsweitenmodulation - Ermöglicht stufenloses einstellen der Ausgangsspannung.
Washi Tape	Reisband - Klebeband aus einem Reisstärkegemisch, ähnelt von der Struktur Papier.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 24.10.2016 Mikko Eberhardt