Bachelorarbeit

Sebastian Eickhoff

Ein modulares Sensor-Aktor-System
für mobile Robotik

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Sebastian Eickhoff

Ein modulares Sensor-Aktor-System für mobile Robotik

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

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

Betreuender Prüfer: Prof. Dr. rer. nat. Stephan Pareigis
Zweitgutachter: Prof. Dr.-Ing Franz Korf

Abgegeben am 28. Februar 2011

Sebastian Eickhoff

Thema der Bachelorarbeit

Ein modulares Sensor-Aktor-System für mobile Robotik

Stichworte

Eingebettete Systeme, Planung, Scheduling, Time-Triggered-System, Framework, Aktor, Sensor, Modularisierung

Kurzzusammenfassung

In dieser Arbeit wird ein Sensor-Aktor-System entwickelt, welches auf mobilen Robotern zum Einsatz kommen kann. Es werden die Grundlagen für dieses Projekt analysiert und daraus die entsprechenden Anforderungen an das System erstellt. Der Katalog aus Anforderungen stellt die Grundlage für das folgende Design der Hardware und Software dar. Der Schwerpunkt liegt im Bereich Softwarearchitektur, welche auf dem Mikrocontroller implementiert wird.

Title of the paper

A modular Sensor-Actor-System for mobile Robotics

Keywords

Embedded System, System Design, Scheduling, Time-Triggered-System, Framework, Actor, Sensor, Modular Design

Abstract

This thesis deals with the development of an Sensor-Actor-System, which will be used in mobile robotics. The initial situation is analyzed and according to this the requirements are generated. The list of requirements is the basis for the following design proccess for hardware and software. The main focus is on the software architecture, which is implemented on the microcontroller

Inhaltsverzeichnis

Kapitel 1

Einleitung

Autonome mobile Roboter haben in der jüngeren Vergangenheit einen hohen Stellenwert erlangt. Dabei handelt es sich um Systeme, welche die ihnen zugedachten Aufgaben und Probleme selbstständig lösen können. Das Entwicklungspotential in dem Bereich dieser Systeme ist deshalb so hoch, weil sie ihre Aufgaben meist deutlich besser erfüllen können als ihre menschlichen Konkurrenten.

Der Mensch hat in vielen Fällen das Nachsehen, wenn es um die Verarbeitung von Informationen seines Umfeldes geht. Er hat z. B. Probleme mit dem gleichzeitigen Erfassen von verschiedenen Informationen. Extreme Umwelteinflüsse, wie sehr hohe oder niedrige Temperaturen, können sein Handeln beeinflussen und auch seine Reaktionszeiten bei plötzlich auftretenden Ereignissen können stark schwanken - und dies sind nur einige wenige Punkte.

Die Beeinträchtigungen des Menschen führen zu einem weiteren wichtigen Themenbereich - der Sicherheit. Viele Aufgaben - auch im alltäglichen Leben - sind mit einem nicht zu unterschätzenden Risiko behaftet. Durch den Einsatz von Robotern und Assistenzsystemen besitzen wir die Möglichkeit, Menschen vor gefährlichen Situationen zu schützen oder, im Falle von Robotern, sie erst garnicht in solche geraten zu lassen.

Bei all den Vorteilen darf man jedoch nicht in ungebrochene Euphorie verfallen. Eine Fähigkeit des Menschen, welche nur spärlich in Technologie abzubilden ist, ist seine Erfahrung aus welcher er schöpfen kann. Hier muss die Entwicklung noch einige Hürden nehmen, bis Roboter ihr Umfeld mit der gleichen Sicherheit einschätzen können wie ein Mensch.

Diese Arbeit soll die Entwicklung solcher Systeme und Roboter, wie sie an der Hochschule für Angewandte Wissenschaften Hamburg erforscht werden, unterstützen.

1.1 Motivation und Ziel der Arbeit

Diese Arbeit hat die Entwicklung eines Prototyps zum Ziel. Dabei handelt es sich um eine modulare Sensor-Aktor-Plattform, welche auf mobilen Robotern zum Einsatz kommen soll. Eine solche Plattform stellt eine erweiterbare Hardware-Abstraktions-Schicht (Hardware Abstraction Layer - HAL) dar. Diese Schicht verbindet gewissermaßen Computer, welche höhere bzw. komplexere Rechenoperationen ausführen, mit der darunterliegenden Hardware bzw. dem mobilen Roboter.

An der Hochschule für Angewandte Wissenschaften Hamburg wird in unterschiedlichen Forschungsprojekten an der Entwicklung von autonomen Fahrzeugen und Fahrerassistenzsystemen gearbeitet.

Bisher wurde im Rahmen eines jeden Projektes eine individuelle Plattform entwickelt und betrieben. Diese sind zumeist recht unterschiedlich, von der Bedienung bis hin zur eigentlichen Architektur. Durch die Verwendung von ungeeigneten Materialien sowie dem bekannten „Flickwerk“, sind die Vorgängermodelle den aktuellen Anforderungen nicht immer gewachsen. Zuletzt darf man auch die kurze Zeit, die ein Student an einer solchen Plattform arbeitet nicht vernachlässigen. Wird dieser kurze Zeitraum noch für verschiedenste Fehleranalysen benötigt, so schwindet die Ausbeute der Arbeiten.

Das Ziel dieser Arbeit ist der Prototyp eines konfigurierbaren Systems, welches mit geringen Anpassungen auf allen Plattformen gleichermaßen eingesetzt werden soll. Diese Entwicklung soll den oben genannten Problemen entgegenwirken, ein einfaches und robustes Produkt entwerfen und den künftigen Nutzern mehr Zeit für die Forschung geben.

1.2 Gliederung dieser Arbeit

Dieser Abschnitt soll den Aufbau dieser Arbeit erläutern.

Kapitel 1 gibt eine Einleitung für die Arbeit. Es legt den Rahmen dieser Arbeit dar und bietet einen kleinen Ausblick über aktuelle Projekte anderer Forscher.

Kapitel 2 behandelt einige grundlegende Themenbereiche für diese Arbeit und schafft ein Verständnis für die wesentlichen Punkte der Entwicklung eines solchen Prototypen.

Kapitel 3 beschreibt die Analyse. Dies ist der erste Schritt bei der Entwicklung. Es wird die Ausgangslage beschrieben, sowie ein Katalog aus Anforderungen an das Produkt entwickelt.

Kapitel 4 beinhaltet das Hardware-Design, welches sich aus dem Anforderungskatalog aus der Analysephase ergibt, sowie das Software-Design. Kernpunkte hier sind die Entwicklung eines Schedulers und eines einfachen Frameworks.

Kapitel 5 gibt den Einblick in die Umsetzung des Software-Designs. Hierzu zählen die Konfiguration des Schedulers, sowie das Anlegen des Framework.

Kapitel 6 legt dar, welchen abschließenden Tests der Prototyp unterzogen wurde.

Kapitel 7 enthält das Fazit und einen Ausblick auf weitere Entwicklungsmöglichkeiten.

1.3 Aktueller Stand der Forschung

Im Folgenden werden einige der größten Veranstaltungen und erfolgreichsten Projekte zu mobilen autonomen Robotern vorgestellt:

DARPA Challenges

Das Verteidigungsministerium der USA hat seine Technologieabteilung (Defense Advanced Research Projects Agency - DARPA) mit der Ausschreibung von Wettbewerben für fahrerlose Landfahrzeuge beauftragt. Ziel dieser Wettbewerbe ist die Entwicklung von autonomen Fahrzeugen, welche einen vorgegebenen Kurs durch offenes Gelände (Grand Challenges 2004 / 2005), sowie einen urbanen Kurs mit mehreren Aufgaben (Urban Challange 2007) bewältigen können. Konnte bei der ersten Veranstaltung im Jahr 2004 noch keiner der Kandidaten das Ziel erreichen, so lässt sich an dem Verlauf und Ergebnis der folgenden Wettkämpfe die schnelle Entwicklung im Bereich der autonomen Fahrzeuge erkennen.

Abbildung 1.1: Gewinner der Urban Challange 2007: Tartan Racing Team

ELROB - The European Robot Trial

„ELROB is a trial! It allows to demonstrate and compare the capabilities of unmanned systems in realistic scenarios and terrains. Terefore it is as close as possible to the typical deployment scenario for today. ELROB is designed to assess current technology to solve problems at hand, using whatever strategy to achieve it.“ (Schneider, 2011)

Abbildung 1.2: Teilnehmerfahrzeug „Hanna“ des ISE RTS (Gottfried Wilhelm Leibniz Universität Hannover, 2011)

Wie aus der Einleitung der offiziellen ELROB Internetpräsenz zu entnehmen ist, handelt es sich bei der ELROB (im Vergleich zur DARPA) nicht um einen Wettbewerb. Vielmehr ist es eine Art Experiment (Trail (engl.): Belastung, Erprobung, Experiment). Die ELROB ist demnach eine Veranstaltung, zu welcher erprobt werden soll, was die aktuelle Robotortechnologie leisten kann. Die Veranstaltungen werden in zwei unterschiedlichen Ausrichtungen abgehalten.

M-ELROB Die militärische ELROB orientiert sich bei den Aufgaben und Anforderungen klar an den Szenarien der deutschen Bundeswehr. Die Teilnehmer sind zum großen Teil Rüstungsunternehmen, welche ihre Produkte in realistischen Szenarien präsentieren können. Aktuelle Aufgaben der M-ELROB 2012:

- Collaborative Urban Search and Rescue
- Reconnaissance and surveillance
- Transport - Mule
- Transport - Movements

C-ELROB Die zivile ELROB richtet sich an Teilnehmer, welche nicht an dem militärichen Einsatz interessiert sind. Hierzu zählen hauptsächlich Teams von Universitäten und anderen Forschungseinrichtungen, welche ihre Roboter in realitätsnahen (nicht militärischen) Szenarien testen wollen. Aktuellen Aufgaben der C-ELROB 2011:

- Reconnaissance and surveillance
- Transport - Mule
- Camp security
- Autonomous navigation

VIAC Challenge

Die VIAC (VisLab Intercontinental Autonomous Challenge) ist ein Langzeittest für eine Gruppe von autonomen Fahrzeugen, welche von der Universität in Parma entwickelt wurden. Bei diesem Wettbewerb sollen vier autonome Minibusse über eine Strecke von Italien bis nach China getestet werden - eine Strecke von 13000km.

Abbildung 1.3: Fahrzeug des VIAC Projekts

Die Fahrzeuge agieren jeweils zu zweit. Ein Fahrzeug übernimmt die Führung, fast autonom, und das zweite Fahrzeug folgt dem Führungsfahrzeug, in diesem Fall völlig autonom. Das Führungsfahrzeug erfordert nur zu wenigen Zeitpunkten menschliches Eingreifen - etwa wenn zu dem zu durchquerenden Gebiet schlicht kein Kartenmaterial zur autonomen Navigation existiert.

Weitere Informationen zu dem erhält man auf der Internetseite des Projektes (VisLab Universität Parma, 2010).

Unbemannte Luftfahrzeuge

Bei unbemannten Luftfahrzeugen (auch als „Drohnen“ bekannt) handelt es sich um kleinere Luftfahrzeuge, welche ohne einen Piloten an Bord diverse Aufgaben im militärischen, wie auch zivilen Bereich erfüllen können. Es existieren sowohl autonome als auch ferngelenkte Varianten.

Ein praktisches Beispiel, ist der Einsatz von verschiedenen Drohnen durch die Polizei in Deutschland. Sie befinden sich teils noch in der Erprobung, konnten jedoch auch bei diversen Einsätzen schon erfolgreich eingebunden werden. Darunter fallen die Luftaufklärung, die Überwachung von Fußballspielen oder auch demonstrierenden Menschenmengen.

Abbildung 1.4: Der FanCopter von EMT-Penzberg

Kapitel 2

Grundlagen eingebetteter Systeme

Im folgenden Kapitel werden einige grundlegende Begriffe erläutert, welche zur Entwicklung dieser Arbeit und dem besseren Verständnis von eingebetteten Systemen nötig sind.

2.1 Eingebettete Systeme

„Eingebettete Systeme sind informationsverarbeitende Systeme, die in ein größeres Produkt integriert sind, und die normalerweise nicht direkt vom Benutzer wahrgenommen werden.“ (S.1 Marwedel, 2007)

Mit dieser Definition als Einstieg, lassen sich eine Reihe von Beispielen anführen. Von aktuellen Kaffeeautomaten der Oberklasse, über Assistenzsysteme in Fahrzeugen bis hin zur Toilette im Zug - beinahe überall im alltäglichen Leben begegnen wir eingebetteten Systemen.

Obwohl eingebettete Systeme eine große Bandbreite an unterschiedlichen Aufgaben abdecken, so gibt es doch eine Reihe von Merkmalen, die (beinahe) alle Systeme teilen.

Verlässlichkeit Eingebettete Systeme arbeiten häufig in einem sicherheitskritischen Umfeld. Anlagen und Systeme, welche bei einem fehlerhaften Verhalten große Schäden an Mensch und/oder Umgebung anrichten können, sind auf ein Maximum an Verlässlichkeit der einzelnen Komponenten angewiesen.

Effizienz Wenn man in der Informatik von einer „effizienten“ Lösung eines Problems spricht, so ist damit die Minimierung des Aufwandes gemeint, welcher für diese Lösung benötigt wurde. Im Bezug auf ein eingebettetes System gilt es also einen

Algorithmus mit möglichst geringer Laufzeit und wenig Speicherverbrauch zu finden.

Einsatzzweck Häufig werden eingebettete Systeme für eine genau festgelegte Aufgabe entwickelt. Beispielsweise ziehen Schweißroboter in der Automobilindustrie immer die gleichen Schweißnähte und lassen sich in der Regel nicht für andere Arbeiten nutzen. Dies unterscheidet eingebettete System von den PC-Systemen, welche eine Reihe unterschiedlicher Aufgaben erfüllen können.

Echtzeit Der Echtzeitbetrieb eines Systems ist in der Norm DIN 44300 beschrieben *„Ein Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen.“*

Derart betriebene Systeme teilen sich in der Regel in zwei Subsysteme. Ein externes System, welche die Daten erzeugt (z.B. Hardwareplattform mit Sensorik, welche Werte liefert) und damit Anforderungen an das Zeitverhalten der Weiterverarbeitung stellt, sowie ein internes System, welches diesen zeitlichen Anforderungen genügen muss und die Daten verarbeitet. Die Norm lässt an dieser Stelle offen, wie mit den Anforderungen des externen Systems umgegangen werden soll. Durchgesetzt haben sich die Bezeichnungen harte und weiche Echtzeitanforderungen.

Von harten Echtzeitanforderungen spricht man, wenn bei deren Verletzung schwerwiegende Fehler, Schäden an anderen Systemen oder gar Menschen entstehen können. Unter weichen Echtzeitanforderungen versteht man jene, bei deren Verletzung zwar ein Berechnungsfehler entsteht, hierbei aber kein gravierender Schaden entstehen kann.

2.2 Modularisierung

In der Informatik ist die Modularisierung ein Vorgehen aus dem Bereich des Software Engineering. Hierbei wird eine Aufgabe mit hoher Komplexität in verschiedene, einfachere Teilaufgaben zerlegt. Die wesentlichen Ziele der Modularisierung sind eine Verbesserung der Softwarequalität, insbesondere der Zuverlässigkeit, Wartbarkeit, Portabilität und Verständlichkeit (vgl. Gabler, 2001, S. 2154).

Die Aufteilung eines Problems muss sich hierbei aus Sicht der Hardware nicht auf ein einzelnes System beschränken. Im Bereich der Robotik können Teilaufgaben sehr un-

terschiedliche Anforderungen aufweisen. Bedingt durch dieses breite Spektrum der Anforderungen, müssen Teilaufgaben gegebenenfalls auf unterschiedliche Systeme verteilt werden. Teilaufgaben, welche eine Bildverarbeitung erfordern, werden wegen ihres hohen Rechenaufwandes beispielsweise auf einem schnelleren PC-System ausgeführt. Die Ansteuerung von Sensoren und Aktoren hingegen , wird wegen der benötigten, schnellen Antwortzeiten häufig auf Systemen mit Mikrocontrollern ausgeführt werden.

2.3 Autonome mobile Roboter

In diesem Punkt gibt es leider keine einheitliche Definition. Man kann lediglich die einzelnen Begriffe deuten und in einen Zusammenhang bringen.

Autonom Der Begriff stammt von dem griechischen Wort *autonomos* und bedeutet soviel wie selbstständig oder auch eigengesetzlich. Der Wissenschaftler Markus Christen hat in einem Gastvortrag aus dem Jahre 2003 über die *Maschinenautonomie* referiert und sich in diesem Zuge an einer Definition versucht:

„Maschinenautonomie - eine spezifischere Definition:

- *Das System entscheidet aufgrund sensorischem Input, Planen, Schlussfolgern und Abschätzen von Konsequenzen.*
- *Das System handelt aufgrund externer Zielvorgaben selbstständig durch die Kombination von Planungs- und Überwachungsschritten.*
- *Das System kann Lernen und Fehler beseitigen.*
- *Das System kann mit anderen autonomen Systemen kommunizieren.“*

(S. 20 Christen, 2003)

Mobil Dieser Aspekt erklärt sich fast schon von selbst. Der Roboter darf nicht fest montiert sein, sondern muss eine Art Antrieb besitzen, um sich in seiner Umgebung selbstständig fortbewegen zu können.

Roboter In der VDI Richtlinie 2860 ist ein Roboter definiert als: *„Ein Roboter ist ein universell einsetzbarer Bewegungsautomat mit mehreren Achsen, dessen Bewegungen hinsichtlich Folge und Wegen beziehungsweise Winkeln frei programmierbar und gegebenenfalls sensorgeführt sind“* (S. 215 VDI-Fachbereich Produktionstechnik und Fertigungsverfahren, 1990)

Zusammenfassend kann man also sagen, dass ein autonomer mobiler Roboter, ein frei programmierbarer Bewegungsautomat ist, welcher sich aufgrund von Sensorinformationen frei in seinem Umfeld bewegen kann.

Kapitel 3

Ausgangslage und Entwicklung der Anforderung

Das Kapitel beschreibt die Ausgangssituation, welche dieser Arbeit zugrunde liegt. Aus dieser Situation werden die Anforderungen für den Prototypen der FAUSTmosap Plattform entwickelt.

3.1 Beschreibung der Ausgangssituation

An der Hochschule für Angewandte Wissenschaften Hamburg wird seit einiger Zeit verstärkt an der Entwicklung von autonomen Fahrzeugen geforscht. Die Entwicklung der ersten Systeme zeigte, dass die einzelnen Plattformen eine Reihe von Übereinstimmungen aufweisen. In dieser Arbeit wird aufgrund dieser Übereinstimmungen eine einheitliche Plattform für alle Forschungsprojekte der Hochschule für Angewandte Wissenschaften Hamburg entwickelt. Der nächste Abschnitt wird erläutern, um welche Projekte es sich handelt.

3.2 FAUST Projekte

Mit den FAUST-Projekten wurde an der Hochschule für Angewandte Wissenschaften Hamburg ein Forschungszweig geschaffen, welcher sich mit Fahrerassistenz- und Autonomen Systemen beschäftigt. Die Schwerpunkte liegen hier auf den folgenden Punkten:

- Sensorik, Telemetrie und Bildverarbeitung

- Echtzeitsysteme und Bussysteme
- Software- und Hardwarearchitekturen
- Algorithmik und Steuerung

Auf den verschiedenen Fahrzeug- und Roboterplattformen werden hier Projekte von Master- und Bachelorstudenten realisiert. (vgl. Hochschule für angewandte Wissenschaften Hamburg, 2010)

Allgemeiner Plattformaufbau

Eine FAUST-Plattform besteht aus drei Schichten. Die Architektur wurde gewählt, um ein hohes Maß an Modularität zu erzielen. Der Funktionsumfang jeder Schicht, wie in Abbildung 3.1 dargestellt, wird im Folgenden kurz erläutert:

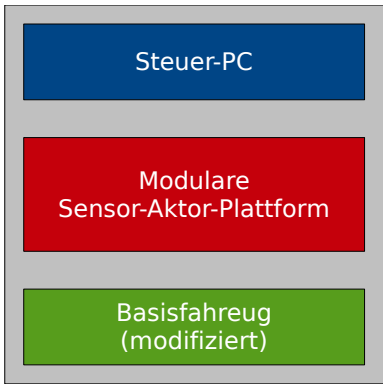

Abbildung 3.1: Schichtmodell

Steuer-PC Es handelt sich hierbei um einen handelsüblichen x86 PC in kombination mit einer Kamera. Als Betriebssystem kommt ein echtzeitfähiges Debian-Linux zum Einsatz. Die Hauptaufgaben des Steuer-PCs sind die Bahnplanung, die Kartographierung und die Orientierung im Raum. Um diesen Aufgaben gerecht zu werden, wurde das FAUST-Framework entwickelt - eine modulare Software, welche aus einem kompakten Softwarekern (FAUSTcore) mit einem Pluginsystem (FAUSTplugins) besteht. Die Hardware kann hier variieren. Derzeit werden Subnotebooks vom Typ ACER Aspire One, ein IBM Laptop X31 und ein mini-itx PC verwendet.

Modulare Sensor-Aktor-Plattform Ein eingebettetes System, welches eine Abstraktionsschicht zum Roboter darstellt. Diese wird ab Kapitel 4 in dieser Arbeit beschrieben.

Basisfahrzeug Als Basisfahrzeug dienen verschiedene Roboter, sowie Modellfahrzeuge. Da der Ursprungszustand der Fahrzeuge einen Einsatz als autonomes Fahrzeug nicht ohne weiteres ermöglicht, müssen (teils weitreichende) Änderungen vorgenommen werden.

3.2.1 FAUST CaroloCup

Der CaroloCup ist ein Wettbewerb für autonome Fahrzeuge. Die jährliche Veranstaltung hat das Ziel, verschiedenen Studententeams die Möglichkeit zu bieten, sich mit der Entwicklung und Umsetzung von autonomen Modellfahrzeugen zu beschäftigen. Dabei steht der Wettbewerb im Vordergrund und motiviert die Teams, das bestmögliche Fahrzeug zu entwickeln. (vgl. Technische Universität Braunschweig, 2010)

Die Hochschule für Angewandte Wissenschaften Hamburg nimmt mit einem Team aus Bachelor- und Masterstudenten an diesem Wettbewerb teil.

Abbildung 3.2: Tamiya Ford F350

Basisfahrzeug: Tamyia Ford F350

Der *Ford F350* von Tamiya ist ein elektrisch getriebenes Modellauto. Das Fahrzeug ist ein Pickup im Maßstab 1:10. Für die Teilnahme am CaroloCup wurde das Fahrzeug von dem FAUST-Team weitgehend modifiziert.

Tabelle 3.1: Leistungsdaten des Tamiya Ford F350

Motor(en)	1x Elekrisch
Leistung	1x 155W
Maße	492 x 216 x 220 mm
Vmax	ca. 15km/h
Antrieb	4WD
Lenkung	Ackermann

3.2.2 FAUST IntelliTruck

Der IntelliTruck ist ein Forschungsprojekt der HAW Hamburg. Das langfristige Ziel ist die Bewältigung eines Kurses durch ein autonomes Fahrzeug. Den maßgeblichen Unterschied zu dem FAUST CaroloCup (3.2.1) stellt das Umfeld dar. Im Rahmen des CaroloCup wird in einer Halle auf einer nahezu idealen Wettkampfbahn gefahren. Der IntelliTruck kommt außerhalb, auf dem Gelände der Hochschule für Angewandte Wissenschaften Hamburg zum Einsatz. Diese Unterschiedlichen Einsatzgebiete erfordern, teils erhebliche, konzeptionelle Unterschiede im Aufbau der Fahrzeuge.

Abbildung 3.3: Carson Comanche

Basisfahrzeug: Carson Comanche

Die Grundlage des Fahrzeugs bildet der *Comanche* des Herstellers Carson. Dabei handelt es sich um ein ferngesteuertes Modellauto im Maßstab 1:6. Im Rahmen des Umbaus zu einem autonomen Fahrzeug, wurden an dem Modell weitreichende Änderungen vorgenommen.

Tabelle 3.2: Leistungsdaten des Carson Comanche

Motor(en)	Verbrenner (27ccm)
Leistung	2kW / 2.6PS
Maße	750 x 490 x 205 mm
Vmax	65 km/h
Antrieb	2WD
Lenkung	Ackermann

3.2.3 FAUST VolksBot

Der VolksBot ist ein noch junges Projekt der Hochschule für Angewandte Wissenschaften Hamburg. Angestrebt wird hier die Entwicklung eines Info-Roboters, welcher sich autonom über das Gelände der Hochschule für Angewandte Wissenschaften Hamburg bewegen kann und Informationen für Studenten und Gäste anbieten soll.

Abbildung 3.4: Volksbot XT

Basisfahrzeug: Fraunhofer VolksBot XT

Das Fraunhofer Institut für intelligente Analyse- und Informationssysteme hat mit dem VolksBot ein Baukastensystem für die mobile Robotik entwickelt. Es kommt in erster Linie in der professionellen Forschung und Entwicklung, sowie im Ausbildungsbereich zum Einsatz. Von verschiedenen Chassis und Modulen, bis hin zur Software können mit diesem modularen System viele Ansprüche abgedeckt werden.

Tabelle 3.3: Leistungsdaten des VolksBot XT

Motor(en)	2x Elekrisch
Leistung	2x 150W
Maße	700 x 620 x 330 mm
Vmax	0.5 m/s
Antrieb	6WD (Panzerlenkung)
Lenkung	Panzerlenkung

Das Modell, welches die Hochschule für Angewandte Wissenschaften Hamburg nutzt, besteht aus dem Grundchassis, sowie einem VolksBot Motor Controller (VMC). Der VMC ist über eine serielle Schnittstelle ansprechbar. Eine rudimentäre CAN-Bus Schnittstelle steht ebenfalls zur Verfügung.

3.3 Entwicklung der Anforderungen

In diesem Abschnitt werden die Anforderungen entwickelt, welche die unterschiedlichen Basisfahrzeuge sowie der Steuer-PC an die FAUSTmosap Plattform stellen. Im Rahmen des Requirement Engineering von Embedded Systems lassen sich zwei Gruppen von Anforderungen unterscheiden:

Funktionale Anforderungen *Was soll das Produkt machen?* In diese Gruppe der Anforderungen fallen Beschreibungen der verschiedenen Funktionen eines Systems und seiner physikalischen Eigenschaften.

Nichtfunktionale Anforderungen *Welche Eigenschaften sollen das Produkt haben?* In diese Gruppe fallen Anforderungen, welche die Qualität des Produktes beschreibt.

Die Einteilung lässt sich noch weiter verfeinern. Unter der Abkürzung *FURPS* versteht man fünf Kategorien, welche die Anforderungen betreffen können. *FURPS* steht demnach für:

- **F**unctionality (Funktionalität)
- **U**sability (Benutzbarkeit)
- **R**eliability (Zuverlässlichkeit)
- **P**erformance (Effizienz)
- **S**upportability (Änderbarkeit)

Die Begriffe wurden zuerst bei Hewlett-Packard entwickelt. Hierbei entspricht das „F“ den funktionalen Anforderungen, während die übrigen Anfangsbuchstaben Merkmale von nicht-funktionalen Anforderungen beschreiben. Die deutschen Übersetzungen sind so auch in der Norm ISO 9126 festgehalten.

3.3.1 Erstellen der Anforderungen

Nach der Analyse der drei Basisfahrzeuge sowie des Steuer-PC lassen sich folgende Anforderungen definieren. Hierbei folgt jeder formalen Anforderung eine detailierte Begründung. Diese legt die wesentlichen Eigenschaften dar, aus welcher die entsprechende Anforderung erstellt wurde.

1. **Die FAUSTmosap Plattform soll gleichermaßen in geschlossenen Räumen wie auch im Außenbereich betrieben werden können.**
   Diese Anforderung leitet sich aus den verschiedenen Einsatzgebieten der projekte

ab. Der IntelliTruck fährt, aufgrund seines Verbrennungsmotors, nur im freien Gelände. Die Fahrzeuge des CaroloCup werden ausschließlich in geschlossenen Räumen betrieben und der VolksBot kann in beiden Umgebungen zum Einsatz kommen.

2. **Die Einbaumaße des Systems dürfen 20cm x 20cm x 6cm (ohne Anschlüsse) nicht überschreiten.**
   Damit die FAUSTmosap Plattform auf allen Fahrzeugen verbaut werden kann, müssen die maximalen Einbaumaße des Fahrzeugs mit dem geringsten Platzangebot berücksichtigt werden. Bei den FAUST Projekten sind dies die Fahrzeuge für den CaroloCup.

3. **Die Verkabelung zwischen den Komponenten soll eine EMV-Schirmung besitzen**
   Die Kabel, welche auf den Fahrzeugen verlegt werden, führen zum Teil sehr dicht an Elektromotoren oder den Zündkerzen entlang. Zur Vermeidung von Störungen während der Signalübertragung müssen alle Kabel geschirmt sein.

4. **Das System soll (mindestens) zwei Aktoren ansprechen können.**
   Da in den FAUST-Projekten autonome Fahrzeuge entwickelt werden, soll das System mindestens einen Aktor für eine Richtungsänderung (Lenkung), sowie einen Aktor für die Geschwindigkeitsänderung ansprechen können.

5. **Das System soll (mindestens) zwölf Sensoren abfragen können.**
   Die Sensoren zur Wahrnehmung des Umfelds sollen wie folgt aufgeteilt sein:

   - 4x Inkrementalgeber
   - 4x Entfernungsmesser mit breitem Kegel
   - 4x Entfernungsmesser mit präziser Punktmessung

   Die Anforderung muss aufgrund der bisherigen (benötigten) Ausstattung der Fahrzeuge des CaroloCup berücksichtigt werden. Die erforderliche Ausstattung des IntelliTruck geht darin auf. Das Projekt um den VolksBot verfügt derzeit noch über keine Anforderungen bezüglich der Sensorik.

6. **Das System soll über einen manuellen Modus verfügen**
   Für die Entwicklung und die damit verbundenen Testreihen ist es von Bedeutung manuell eingreifen zu können, wenn das System vom vorgesehenen Verhalten abweicht. Ferner ist ein solcher Modus für den Einsatz im CaroloCup vorgeschrieben.

7. **Das System soll über einen sicheren Zustand verfügen**
   Kommt es bei autonomen Fahrzeugen zu schwerwiegenden Fehlern, dann muss es einen sicheren Zustand geben, in welchen das System wechseln kann, um Schäden zu vermeiden und den Fehler zu signalisieren.

8. **Zur Kommunikation soll das System über mehrere Interfaces verfügen (USB, CAN, Ethernet, RS232)**
   Bisher wurden die verschiedenen Plattformen über diverse Schnittstellen angesprochen. Die sollen aus Gründen der Kompatibilität weiterhin erhalten bleiben.
   - USB - veraltete Schnittstelle zur Kommunikation mit dem Steuer-PC
   - CAN - Kommunikation zwischen Controllern, oder mehreren Teilnehmern vom Typ der FAUSTmosap Plattform
   - Ethernet - aktuelle Schnittstelle zur Kommunikation mit dem Steuer-PC
   - RS232/1 - Kommunikation zu dem Motor-Controller des Volksbot
   - RS232/2 - Programmierung des Systems / Debug-Schnittstelle
9. **Das System soll harten Echtzeitanforderungen genügen**
   Diese Anforderung ergibt sich aus den beschriebenen Grundlagen. Da das System in einem sicherheitskritischen Umfeld zum Einsatz kommt, muss ein Verhalten absolut vorhersagbar sein.
10. **Das System soll in Zyklen von 25ms mit dem Steuer-PC kommunizieren**
    Der Scheduler des Steuer-PCs arbeitet in festen Zyklen von 25ms. Dieser Wert ist durch die Rechenleistung und Auslastung des Steuer-PCs durch dessen Algorithmen bedingt.
11. **Die Aktoren sollen innerhalb eines Zyklus gesetzt werden**
    Die Ansteuerung der Aktoren mit den aktuellen Werten des Steuer-PC soll innerhalb eines Zyklus erfolgen, um die Verlässlichkeit zu steigern. Würden in diesem Punkt Verzögerungen auftreten, würde dies die Berechnungen des Steuer-PCs erschweren, bzw. unmöglich machen.
12. **Die Sensoren sollen innerhalb eines Zyklus ausgelesen werden**
    Das Auslesen der Sensoren soll in einem Zyklus erfolgen, damit die Berechnungen des Steuer-PCs auf aktuellen Werten basieren. Würden hier Verzögerungen auftreten, könnte der Steuer-PC im Extremfall nicht rechtzeitig auf Ereignisse reagieren.
13. **Das System soll über eine extern zugängliche Programmierschnittstelle verfügen**
    Eine Programmierschnittstelle ist bei allen Systemen sinnvoll, welche während ihres Produktlebenszyklus in kürzeren Abständen Aktualisierungen erfahren. Das (physikalische) Herausführen über einen Anschluss hat zwei wesentliche Gründe:
    - Die vereinfachte Handhabung - kurzfristig erforderliche Änderungen können im Feld umgesetzt werden.

- Die Schonung der mechanischen Elemente - das System muss nicht zerlegt werden um an entsprechende Schnittstellen zu gelangen.

14. **Der Aufbau der Software soll einen modulare Struktur besitzen**
    Die Modularisierung der Software wirkt sich auf viele Aspekte aus. Einer der Hauptgründe ist die schnelle Anpassbarkeit. Da das System auf verschiedenen Plattformen zum Einsatz kommen soll, muss auch die Software entsprechend (schnell) anpassbar sein. Durch das (De-)Aktivieren verschiedener Module sollen unterschiedliche Konfigurationen für jede Plattform erstellt werden können.

15. **Das System soll mit mehreren (Steuer-)Rechnern kommunizieren können**
    Auf größeren Plattformen ist ein einzelner Steuer-Rechner in der Regel nicht mehr ausreichend. Für diesen Fall, welcher auf der VolksBot-Plattform angestrebt wird, sollen mehrere Steuer-Rechner die Daten des Systems nutzen können.

16. **Das System soll über eine Debug-Schnittstelle verfügen**
    Zur Verbesserung der Wartbarkeit, soll das System über eine Debug-Schnittstelle verfügen. Hierüber sollen zur Laufzeit wichtige Informationen über den Zustand des Systems ausgegeben werden.

17. **Das System soll mögliche Mechanismen zum Einsparen von Energie nutzen**
    Auf den mobilen Plattformen ist es wichtig, den Energieverbrauch so gering wie möglich zu halten. Um dieses Ziel zu erreichen, soll das System mögliche Energiesparmechanismen nutzen - sofern dies unter Einhaltung aller Anforderungen möglich ist.

18. **Das System muss mit einer variablen Versorgungsspannung betrieben werden können**
    Bei einem Betrieb der Batterien muss beachtet werden, dass diese keine konstante Spannungsversorgung zur Verfügung stellen können, bedingt durch die kontinuierliche Entladung. Diese Anforderung ist typisch für mobile, batteriebetriebene Systeme.

Kapitel 4

Design von Hard- und Software

Die Ausarbeitung des Anforderungskatalogs macht deutlich, dass der Entwurf einer vollständigen eingebetteten Plattform, eine extrem vielschichtige Aufgabe ist. Da die Entwicklung einer Plattform mit zugehöriger Peripherie und Installation auf verschiedenen Plattformen den Rahmen einer einzelnen Bachelorarbeit überschreiten würde, beschränkt sich die folgende Entwicklung auf die Kernkomponenten der FAUSTmosap Plattform.

Eine Frage die bei den unterschiedlichsten Punkten wiederkehrt, ist die Grundsatzentscheidung, ob eine Komponente eingekauft werden kann, oder selbst entwickelt wird. Da die FAUSTmosap Plattform im Rahmen des Studiums der Technischen Informatik entwickelt wird, liegt der Schwerpunkt der Eigenentwicklungen im Bereich der Softwareentwicklung. Die Eigenentwicklung der Hardware, speziell der nicht-elektronischen, ist in diesem Fachbereich nicht möglich, weshalb diese Komponenten eingekauft werden müssen.

4.1 Überblick

Die Entwicklung beginnt mit der groben Einteilung der FAUSTmosap Plattform. Es ergeben sich zwei Kernbereiche des Systemdesigns: Hardware und Software. Da das Design der Software abhängig von der verwendeten Hardwarelösung ist, wird dieser Schritt als zweites bearbeitet. Es folgt demnach das Design für die Hardware.

4.2 Hardwaredesign

Die Designentscheidungen im Bereich der Hardware teilen sich grob in die Bereiche Aufbau (Gehäuse und Verkabelung) und Mikroelektronik.

4.2.1 Gehäuse

Das Gehäuse beinhaltet alle mikroelektronischen Komponenten und bildet damit die Kernkomponente der Plattform. Über eine Reihe von Anschlüssen werden die restlichen Komponenten mit der FAUSTmosap Plattform verbunden.

Wie Erfahrungen aus den FAUST-Projekten gezeigt haben, ist die Konstruktion eines solchen Gehäuses nicht zu empfehlen. Das Gehäuse wird somit eingekauft. Die Suche und Auswahl wird durch einige Anforderungen (siehe 3.3.1) bestimmt.

Maße Die Abmessungen des Gehäuses werden durch Anforderung Nr.2 vorgegeben. Mit 6cm maximaler Höhe fällt die Entscheidung auf ein einzelnes Gehäuse.

Material Es kommen unterschiedliche Materialien für das Gehäuse in Frage. Grundsätzlich lässt sich zwischen metallischen Gehäusen und jenen aus Kunststoff unterscheiden. Die Ausführungen bringen Eigenschaften mit sich, welche für das angestrebte Produkt verschiedene Vor- und Nachteile bieten. Aus den Bewertungen der

Tabelle 4.1: Gehäuse - Material

	Elektrisch leitend	Gewicht	Widerstandsfähigkeit	Kosten
Kunststoff	+	+	–	++
Metall	–	–	+	–

Legende			
Negativ	Neutral	Positiv	Sehr Positiv
–	o	+	++

Materialtabelle lässt sich erkennen, dass ein Kunststoffgehäuse die richtige Wahl darstellt. Gerade bei den für eingebettete mobile Systemen wichtigen Eigenschaften wie Gewicht und Preis, sind die Varianten aus Kunststoff im Vorteil.

Schutzart Die Anforderung Nr.4 stellt Bedingungen an den Aufbau des Gehäuses. Für den Betrieb von elektrischen Systemen unter verschiedenen Umweltbedingungen wurden durch das Deutsche Institut für Normung in den Normen DIN EN 60529

und DIN 40050 Teil 9 die IP-Schutzarten beschrieben (IP: International Protection). Anhand dieser Schutzarten lässt sich die Eignung eines Gerätes für ein bestimmtes Umfeld ermitteln.

Für den Einsatz in den FAUST-Projekten hat sich die Schutzart IP65 als ausreichend erwiesen. Diese Schutzart beinhaltet den vollständigen Schutz gegen Berührung (Staubdicht) sowie gegen das Eindringen von Strahlwasser (aus allen Richtungen).

Mit den Rahmenbedingungen aus Maßen, Material und Schutzart wurde nach intensiver Suche ein Gehäuse der Firma Richard Woehr GmbH gewählt.

Bezeichnung	GH02KS022/235
Maße	190 x 190 x 55 mm
Schutzart	IP65
Material	ABS Kunststoff: • Rohmaterial nach UL94-HB • Hochfest • äußerst Robust • Schlagstabil
Temperaturbereich	-20°C bis +90°C

Tabelle 4.2: Daten GH02KS022/235

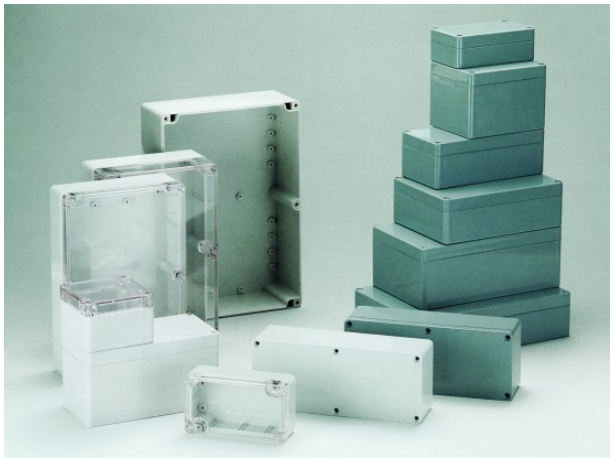

Abbildung 4.1: Beispielabbildung von IP65 Gehäusen der Firma Woehr

4.2.2 Verkabelung

Die Verkabelung auf der Plattform unterliegt im Grunde den gleichen Anforderungen wie das Gehäuse. Zusätzlich muss jedoch die Anforderung Nr.3 beachtet werden. Bei den unterschiedlichen Angeboten ließ sich feststellen, dass geschirmte Varianten einen erheblichen Aufpreis bedeuten. Anschließende Tests auf den maßgeblichen Plattformen zeigten, dass eine Schirmung nicht zwingend erforderlich ist, wenn die Verkabelung in einem ausreichend großen Abstand zu den Störquellen verlegt wird. Dieses Vorgehen ist ausdrücklich *keine* Lösung für eine später gefertigte Plattform und verletzt die zuvor genannte Anforderung.

Für die Verkabelung des Prototyps kommt die Serie 707 des Herstellers Binder zum Einsatz. Die wesentlichen Eigenschaften sind:

- Kabelsteckverbinder
- Rundsteckverbinder M5 mit Schraubverriegelung
- Gewindering mit Rüttelsicherung
- Steckverbinder umspritzt am Kabel
- Schutzart IP 67

Abbildung 4.2: Binder Serie 707

Die Anschlüsse eignen sich besonders für den Einsatz auf den Fahrzeugen, da sie mit der Schraubverriegelung und der Rüttelsicherung unanfällig gegen die Vibrationen der Fahrzeuge sind. Wackelkontakte und lose Stecker waren in der vergangenheit häufige Fehlerquellen, die wertvolle Zeit kosteten.

4.2.3 Mikroelektronik

Nach Bereitstellung des Gehäuses liegt der nächse Schritt in der Auswahl der signalverarbeitenden Mikroelektronik. Da es sich bei dem Produkt um eine Plattform handelt, wird die Integration der Signalverarbeitung in die Steuerrechner - in Form von Steck- oder Erweiterungskarten - als Lösungsansatz ausgeschlossen. Dies würde einer hohen Modularität des Systems entgegenwirken.

Es verbleiben demnach die folgenden Ansätze, die Signaldaten zu erfassen und für die Steuerungsrechner bereitzustellen:

Diskrete Schaltungstechnik In der diskreten Schaltungstechnik werden konventionelle elektronische Bauteile verwendet. Eine derart aufgebaute Schaltung wäre nur schwer an eventuelle Änderungen anpassbar.

FPGA Bei den FPGAs (Field Programmable Gate Array) handelt es sich um programmierbare integrierte Schaltkreise. FPGAs zeichnen sich besonders durch die Geschwindigkeit bei der parallelen Verarbeitung von Signalen aus, da sie, im Gegensatz zu (den meisten) Mikroprozessoren, echte Parallelität bieten können. Nachteilig wirkt sich das Prinzip der FPGAs dann aus, wenn beispielsweise komplexe sequentielle Steuerungsabläufe realisiert werden sollen. Hier wird die „echte“ Parallelität zum Problem und es muss ein gewisser Aufwand getrieben werden, um ein sequentielles Verhalten abzubilden.

Mikrocontroller Mikrocontroller stellen Prozessoren dar, welche in der Regel mitsamt Peripherie und Speicher auf einem Chip gefertigt werden. Man darf sie als vollständiges Computersystem auf engsten Raum betrachten. Die Bandbreite an Mikrocontrollern wächst stetig und bietet für nahezu jeden Anspruch eine Lösung. Herausstechenste Merkmale sind hier die einfache Handhabung und Entwicklung, sowie der geringe Stückpreis der Chips.

Auf den Fahrzeugen der FAUST-Projekte soll die FAUSTmosap Plattform, wie schon zuvor beschrieben, die Ansteuerung von Sensorik und Aktorik übernehmen. Um eine Entscheidung für einen der Ansätze zu fällen, muss man diese im Kontext der FAUST-Projekte betrachten.

Die einfachste Bewertung ist hier für die diskrete Schaltung zu treffen. Die diskreten Schaltungen widersprechen dem Prinzip des modularen Aufbaus und würden die Plattform damit für spätere Weiterentwicklungen zu stark einschränken. Ebenso darf man den erheblichen Aufwand einer solchen Schaltung nicht vernachlässigen. Diese Variante scheidet somit aus.

Die Entscheidung zwischen einem FPGA und einem Mikrocontroller muss jedoch etwas genauer betrachtet werden. Zum einen bietet der FPGA die besten Vorraussetzungen für die parallele Erfassung der unterschiedlichen Signale, während der Mikrocontroller eine deutlich einfachere Entwicklung der Abläufe erlaubt.

Die Entscheidung fiel nach den Recherchen zu den aktuell verfügbaren Modellen und Technologien. Mikrocontroller haben in den letzten Jahren deutlich an Performance gewonnen, wie zum Beispiel ein Geschwindigkeitsvergleich eines etablierten Echtzeitbetriebssystems wie FreeRTOS (vgl. FreeRTOS, 2011) über verschiedene Mikrocontrollerarchitekturen der letzten Jahre zeigen kann. Sieht man die Performance as ausreichend an, so bleiben noch die Punkte der Handhabung in der Entwicklung sowie die fortlaufende Anpassbarkeit offen. Unter dem Aspekt der hohen Modularität, welche von diesem Projekt gefordert wird, und der eventuellen Weiterentwicklung ist absehbar, dass die schnelle und einfache Anpassbarkeit bei dieser Plattform gewährleistet werden muss.

Zieht man die einzelnen Entscheidungen zusammen, so wird erkennbar, dass ein Entwurf auf Basis eines Mikrocontrollers hier die beste Wahl darstellt.

4.2.4 Auswahl eines Mikrocontrollers

An die Entscheidung für einen Mikrocontroller knüpft sich selbstverständlich die Auswahl eines spezifischen Produktes. Auf dem Gebiet der Mikrocontroller herrschen hier diverse Architekturen von unterschiedlichen Herstellern vor:

PIC 8-, 16- oder 32Bit RISC-Prozessoren des Herstellers Microchip Technology Inc.

AVR 8Bit RISC-Prozessoren mit modifizierter Havard-Architektur des Herstellers Atmel.

ARM / CORTEX Kerndesign für 32Bit Mikroprozessoren dem RISC Prinzip folgend. Die Designs werden von ARM Limited entworfen und von diversen Herstellern lizensiert. Derzeit wird die bekannte ARM Architektur (ARMv4 bis ARMv6) durch einen vollständig neu entwickelten Nachfolger ersetzt - die CORTEX Architektur (ARMv7).

Die Auswahl unter den Mikrocontrollern fällt vergleichsweise einfach, da die älteren Architekturen PIC und AVR schlicht nicht genug Peripherieanschlüsse zur Verfügung stellen können - sie sollten jedoch der Vollständigkeit halber erwähnt werden, da sie für andere (kleinere) Projekte stets eine Alternative darstellen.

Die CORTEX Architektur (ARMv7) bietet als Nachfolger der ARM Architektur(en) eine Reihe an Vorteilen. Diese Vorteile betreffen nahezu alle Eigenschaften der Mikrocontroller, von höherer Geschwindigkeit, über einen gesenkten Stromverbrauch bis hin zu einem deutlich vereinfachten Programmiermodell. Da die CORTEX Architektur trotz ihrer Neuerungen weitestgehend mit ihrem Vorgänger kompatibel ist, stehen für die Entwicklung bereits etablierte Werkzeuge zur Verfügung. Die einzelnen Unterschiede sollen hier nicht im Detail erläutert werden, da sich die Beschreibungen zu weit vom eigentlichen Thema dieser Arbeit entfernen würden.

Informationen für den Einstieg sind auf der Internetseite von ARM Limited (ARM Ltd., 2010) zu erhalten. Für einen tieferen Einstieg empfehle ich das Buch „The Definitive Guide to the ARM Cortex-M3“ von Joseph Yiu (Yiu, 2009).

4.2.5 Basisplatine & Development-Board

Die vollständige Eigenentwicklung einer Platine, ausgestattet mit einer CORTEX-M3 MCU sowie erforderlicher Peripherie hätte im Rahmen dieser Arbeit zuviel Zeit in Anspruch ge-

nommen. Aus diesem Grund wurde der Anteil an Eigenentwicklung auf eine Basisplatine beschränkt und ein passendes Development-Board des Herstellers KEIL eingekauft.

Neben dem Aspekt der Entwicklungszeit, bringt diese Lösung viele weitere Vorteile mit sich.

Fehleranfälligkeit Auf der Basisplatine sind lediglich Schaltungen geringerer Komplexität untergebracht. Im Falle eines Fehlers kann dieser, soweit er die Basisplatine betrifft, leicht lokalisiert und behoben werden. Durch den Einkauf des (deutlich) komplexer aufgebauten Development-Boards wird somit den schwer zu bestimmenden Fehlerfällen ausgewichen, da es sich um ein etabliertes und getestetes Produkt handelt.

Modularität Durch den einfachen Aufbau lässt sich die Plattform mittels Weiterentwicklung der Basisplatine oder Austausch des Development-Boards kurzfristig an neue Herausforderungen und Aufgaben anpassen.

4.2.6 Basisplatine im Detail

Bei der sogenannten Basisplatine, handelt es sich um eine Leiterplatte, welche verschiedene Funktionen der FAUSTmosap Plattform erfüllt. Sie ist in Abbildung 4.3 zu erkennen (grüne Leiterplatte).

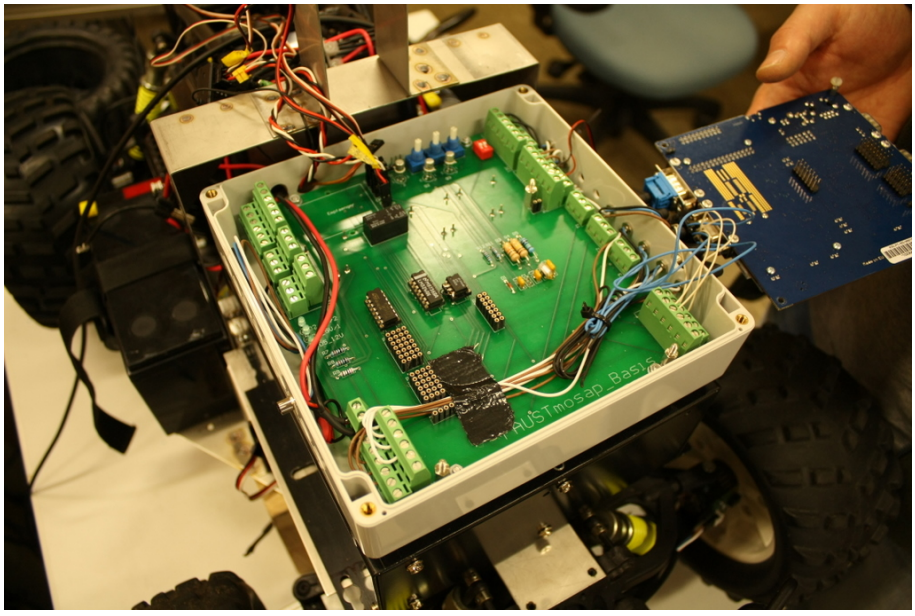

Abbildung 4.3: Die Basisplatine während der Endmontage

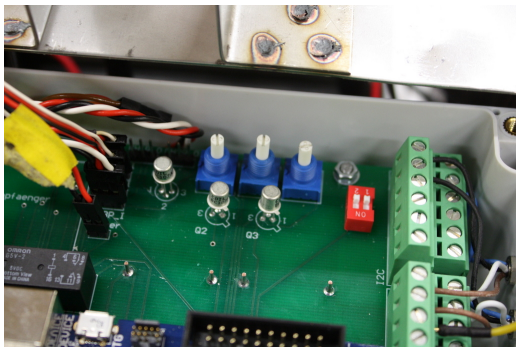

Abbildung 4.4: Begrenzungsschaltung auf der Basisplatine

Abbildung 4.5: Montage der Anschlüsse noch unvollständig

Routing Es werden die Ein- und Ausgänge des Development-Boards mit den Anschlussterminals verbunden.

Spannungsversorgung Die Platine besitzt verschiedene Gleichspannungswandler, welche die variierende Eingangsspannung aus den Akkus in die benötigten, stabilen Versorgungsspannungen umwandeln. Die Bausteine erfüllen die Anforderung Nr.18 und können in einem Bereich von 9-36V betrieben werden.

Signalvorverarbeitung Den Signalen einiger Sensoren müssen teilweise diskrete Schaltungselemente vorgeschaltet werden, damit sie von dem Mikrocontoller weiter verarbeitet werden können.

Modusschalter Bei dem Modusschalter handelt es sich um eine diskret aufgebaute Schaltung, welche das Signal für die Aktoren zwischen einem Fernbedienungssignal und dem des autonomen Systems umschalten kann. Diese Schaltung erfüllt die Anforderung Nr.6.

Begrenzerschaltung Hierbei handelt es sich um eine Zusatzfunktion der Platine. Sie soll ungeübten Nutzern der Fahrzuge Sicherheit bieten, indem sie bei einer bestimmten Höchstgeschwindigkeit abriegelt. Sie ist auf der Abbildung 4.4 zu erkennen.

Die Platine wurde im Rahmen der FAUST-Projekte mit der Unterstützung von Daniel Arnold entwickelt, welcher zum Zeitpunkt dieser Arbeit eingeschriebener Student der Elektrotechnik ist. Die Platine ist demnach kein direkter Bestandteil dieser Arbeit, sollte für das bessere Verständnis jedoch vorgestellt werden.

4.2.7 KEIL MCB1760 im Detail

Das KEIL MCB1760 ist ein Development-Board, wie sie im Bereich der Entwicklung eingebetteter Systeme häufig zu finden sind. Diese Art von Produkt erlaubt es Entwicklern von Endgeräten, den in Frage kommenden Mikrocontroller schon in einer frühen Phase der Entwicklung zu testen und auszuwählen.

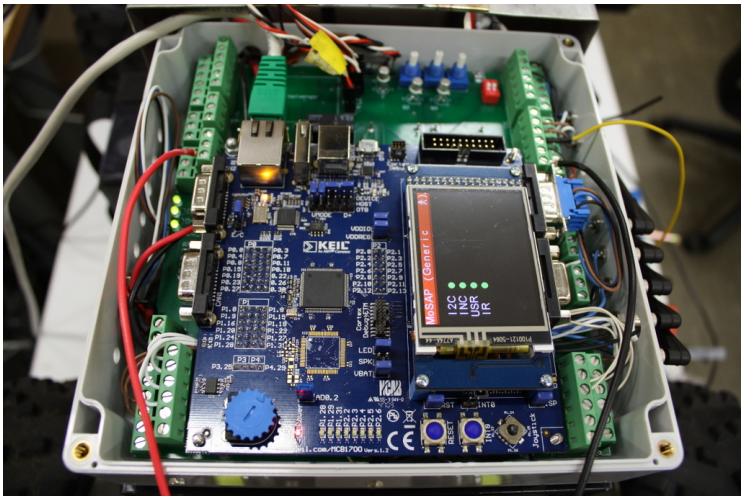

Abbildung 4.6: Die FAUSTmosap Plattform während der Endmontage.

In der Regel sind diese Produkte nicht für einen späteren Einsatz gedacht. Im Rahmen dieser Arbeit eignet sich das KEIL MCB1760 jedoch sehr für den Aufbau eines Prototyps. Es bringt alle in Anforderung Nr. 8 beschriebenen Schnittstellen in ausreichender Zahl mit sich und eignet sich wegen seiner kompakten Ausführung für den Einbau im Gehäuse. Folgend seien die wichtigsten technischen Merkmale des KEIL MCB1760 genannt.

- 100MHz ARM Cortex-M3 processor-based MCU
- On-Chip Memory: 512KB Flash and 64KB RAM
- Color QVGA TFT LCD
- 10/100 Ethernet Port
- USB 2.0 Full Speed - USB, USB-OTG, and USB Host
- 2 CAN interfaces
- 2 Serial Ports
- SD/MMC Card Interface
- 5-position Joystick and push-button
- Analog Voltage Control for ACD Input
- Amplifier and Speaker
- Up to 70 GPIO
- Debug Interface Connectors:
  20-pin JTAG
  10-pin Cortex debug
  20-pin Cortex debug + ETM Trace

In Abbildung 4.6 ist das Keil MCB1760 (hier: blaue Platine) zu erkennen, wie es auf der

BasisPlatine montiert wurde. Die vollständige Dokumentation des Produktes kann von der Internetseite des Herstellers KEIL bezogen werden (KEIL, 2011).

4.3 Softwaredesign

Dieser Abschnitt befasst sich mit dem Aufbau und dem Design der zum Einsatz kommenden Softwarearchitektur. Hierzu sind zunächst einige grundsätzliche Betrachtungen nötig, welche später bei der Umsetzung genauer ausgeführt werden. Für die Entwicklung der Software-Architektur der FAUSTmosap Plattform muss zunächst das gesamte System betrachtet werden. Wie dem Katalog der Anforderungen zu entnehmen ist, sollen mehrere unterschiedliche Aufgaben durch die Plattform bearbeitet werden. Da durch die Auswahl der CORTEX-M3 MCU nur ein Prozessor für diese Aufgaben zur Verfügung steht, müssen diese folglich nacheinander bearbeitet werden. Eine Aufgabe wird im Folgenden als Task bezeichnet.

Zur Bearbeitung von mehreren Tasks müssen diese in eine Ausführungsreihenfolge gebracht werden. Das Erzeugen und Verwalten einer solchen Reihenfolge ist in der Informatik als Scheduling bekannt.

4.3.1 Scheduling Verfahren

Der erste wesentliche Unterschied bei den verschiedenen Verfahren zum Scheduling betrifft die Unterbrechbarkeit von Tasks. Die Verfahren sind als *preemptive Scheduling* und *non-preemptive Scheduling* bekannt.

non-preemptive Hierbei werden alle Tasks nacheinander ausgeführt. Ein Task wird vollständig ausgeführt und gibt erst nach seiner Beendigung die Kontrolle an den Scheduler zurück. Bei langen Ausführungszeiten von Tasks kann es hier zu Problemen bei der Reaktion auf externe Ereignisse kommen.

preemptive Tasks können bei diesem Verfahren vom Scheduler unterbrochen werden, wenn andere Tasks oder externe Ereignisse dies erfordern. Präemptive Scheduler können daher schneller auf nicht geplante Ereignisse reagieren.

Es ist zu erkennen, dass das Zeitverhalten eines Systems mit präemptiven Scheduling-Verfahren erheblich schwerer zu berechnen ist. Da unser System ein vorhersagbares Zeitverhalten erfordert, wird der nicht-präemptive Ansatz bevorzugt.

Michael J. Pont beschreibt in seinem Buch „Patterns for Time-Triggered Embedded Systems“(Pont, 2001) ein Konzept für einen nicht-präemptiven Scheduler für Systeme mit harten Echtzeitanforderungen. Dieser Entwurf soll die Grundlage für den Scheduler in dieser Arbeit sein.

4.3.2 Anpassung des Schedulers

Der Entwurf von Pont sieht vor, dass das System einen Timer besitzt, welcher sogenannte *System Ticks* erzeugt. Diese Ticks stellen die Zeitschritte des Systems dar, innerhalb derer die Tasks ausgeführt werden. Um ein vorhersagbares Verhalten zu erlangen, soll nach Pont der *SystemTick Timer* die einzige Interruptquelle des Systems sein.

Aus technischen Gründen müssen im Falle der FAUSTmosap Plattform jedoch einige Schritte der Signal- und Kommunikationsverarbeitung mit Interrupts realisiert werden. Um die korrekte Funktion des Schedulers sicher zu stellen, darf folgende Bedingung nicht verletzt werden:

$$(T_{Tasks} + T_{Sched}) < T_{SystemTick} \tag{4.1}$$

Die Ausführungszeit aller Tasks (T_{Tasks}), welche innerhalb eines Ticks($T_{SystemTick}$) bearbeitet werden sollen, dürfen zusammen mit der Ausführungszeit des Schedulers(T_{Sched}) nicht die Länge eines *SystemTicks* überschreiten.

Bei genauer Betrachtung beinhaltet die Ausführungszeit des Schedulers unter anderem die Zeit des Interrupts des *SystemTick Timers*. Besitzt das System nun mehrere Interruptquellen, so müssen deren Bearbeitungszeiten in der obigen Formel berücksichtigt werden. Es ergibt sich:

$$(T_{Tasks} + T_{Sched} + T_{Interrupts}) < T_{SystemTick} \tag{4.2}$$

Damit diese Formel Gültigkeit besitzen kann, muss es sich bei den zusätzlich auftretenden Interrupts um periodisch auftretende handeln. Sporadische Interrupts, zum Beispiel durch eine Schnittstelle zur menschlichen Interaktion, müssen ausgeschlossen werden und würden die Vorhersagbarkeit des Systems stören. Bei Interruptquellen mit einer schwankenden Frequenz der Interrupts muss zur Berechnung stets die höchste Frequenz herangezogen werden.

4.3.3 Framework

Aus den Erfahrungen in der Entwicklung vorheriger Systeme für die FAUST-Projekte hat sich gezeigt, dass es für eine schnelle und trotzdem sichere Anpassung an zukünftige Aufgaben von Vorteil ist, wenn das System im Sinne eines modularen Frameworks abgebildet

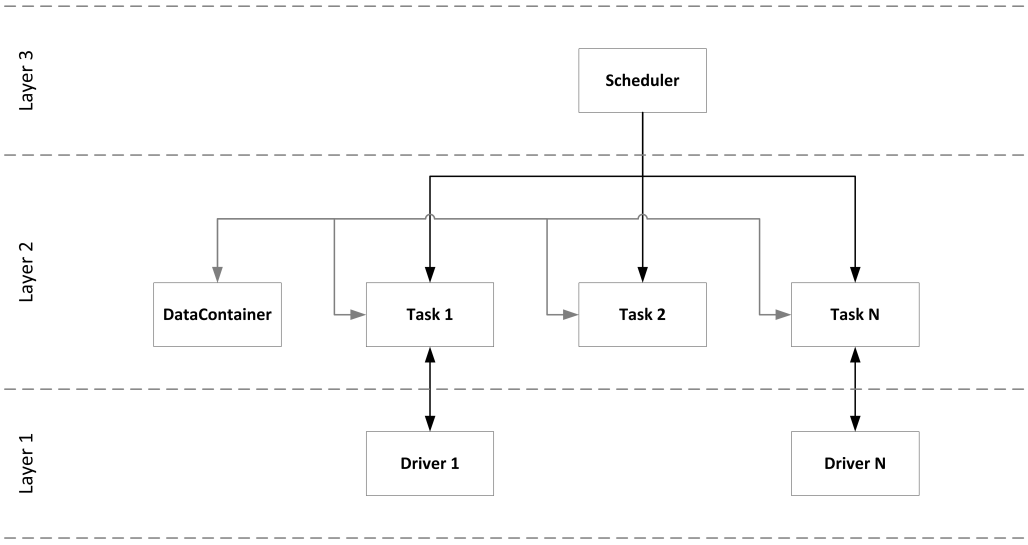

Abbildung 4.7: Framework der Plattform

wird. Dies bedeutet, dass möglichst alle Teilaufgaben, die sich aus dem Anforderungskatalog ergeben, in seperaten Modulen realisiert werden, mit einem Minimum an Abhängigkeiten zu weiteren Modulen. Die Auswahl des Schedulers aus Kapitel 4.3.2 unterstützt diesen Ansatz optimal, da jedes Modul in einem eigenen Task realisiert werden kann.

Abbildung 4.7 zeigt den Entwurf des Frameworks. Es besteht aus drei Schichten, die im Folgenden kurz erläutert werden:

Layer 3 In dieser Schicht des Frameworks befindet sich lediglich der Scheduler. Der Scheduler ruft die Tasks auf und bearbeitet diese. Die Tasks sollen keinen Zugriff auf den Scheduler haben. Der Entwurf von Pont sieht diverse Funktionen des Schedulers vor, um Tasks entfernen oder hinzufügen zu können. Um diese Funktionen sicher nutzen zu können, müsste eine vierte Schicht für die Kontrolle des Schedulers eingeführt werden. Damit wäre ein dynamisches Scheduling möglich, welches wegen seines Aufwandes einer gesonderten Betrachtung bedarf. In dieser Arbeit soll der Ansatz des statischen Schedulings ausreichen.

Layer 2 Alle Tasks des Systems befinden sich in dieser Schicht. Die Tasks dürfen lediglich nach unten mit ihren Treibern kommunizieren, oder aber auf gleicher Ebene mit dem Datencontainer. Generell wurde im Design eine Unabhängigkeit zwischen den Tasks gefordert. Der hier aufgeführte Datencontainer ist kein Task im eigentlichen Sinne. Er bildet eine globale Struktur für den Datenaustausch zwischen den Tasks ab, um Abhängigkeiten der Tasks untereinander zu vermeiden. Er benötigt *keine* Ausführungszeit.

Layer 1 Auf dieser Ebene sind alle Treiber angesiedelt. Die Treiber des Frameworks dienen der Abstraktion und sollen einen einfachen Zugriff der Tasks auf die Hardware erlauben. Einige solcher Treiber werden bereits durch die Hersteller der Mikrocontroller angeboten.

Kapitel 5

Implementierung der Architektur

In diesem Abschnitt wird die Implementierung des Designs aus den Kapiteln 4.3.2 und 4.3.3 auf dem CORTEX-M3 Mikrocontroller beschrieben.

5.1 Framework und Scheduler

In der Umsetzung sind das Framework und der Scheduler dicht verzahnt. Einer der angedeuteten Vorteile aus Kapitel 4.2.4 ist hier von nutzen. Die CORTEX-M3 MCU bietet mit dem *SysTim-Timer* einen seperaten Timer an, welcher speziell zur Erzeugung von Systemzeiten geschaffen wurde.

Die Abbildung 5.1 zeigt den Programmfluss des Schedulers. Ein Zyklus beginnt mit dem Auslösen des Interrupts des *SysTim-Timers*. Die zugehörige ISR `SysTick_Handler()` ruft die Funktion `SCH_Update()` auf. In dieser Funktion des Schedulers, werden alle Task-Daten durchlaufen und gegebenenfalls aktualisiert. Mit dem Verlassen der ISR wird der bis zu diesem Zeitpunkt angehaltene Programmablauf fortgesetzt. Die Funktion `SCH_Dispatch_Tasks()` wird aufgerufen. Diese durchläuft die (zuvor aktualisierten) Task-Daten. Ist ein Task bereit wird dieser sofort ausgeführt. Sind die Task-Daten durchlaufen wechselt der CORTEX-M3 durch das Makro `__WFI()` wieder in der Ruhezustand und wartet auf den nächsten Interrupt des *SysTim Timers*. Der Wechsel in den Ruhezustand entspricht der Anforderung Nr.17.

Zur Konfiguration des Schedulers kommt die Formel 4.2 aus Kapitel 4.3.2 zum Einsatz. Für einen sauberen Ablauf, sind einige Berechnungen notwendig. Diese sollen beispielhaft am Modell des Intelli-Truck aus Kapitel 3.2.2 durchgeführt werden.

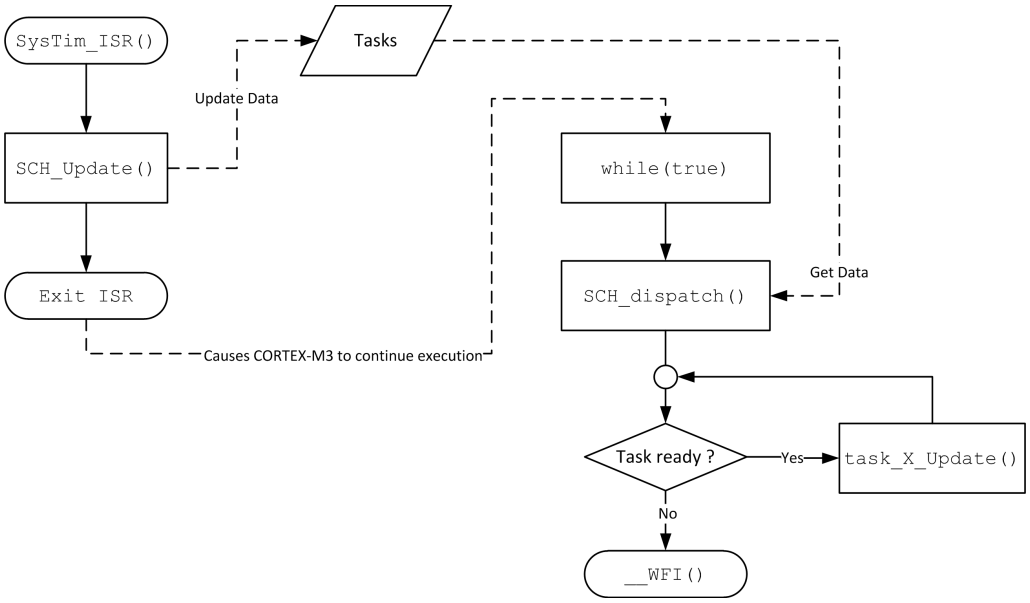

Abbildung 5.1: Ablauf des Schedulers

5.2 Beispielkonfiguration: Intelli-Truck

Grundlage für die Konfiguration des Schedulers ist die Ausstattung des Fahrzeugs und die damit verbundenen Aufgaben, welche die FAUSTmosap Plattform übernehmen muss. Zu der, für die Konfiguration relevanten, Ausstattung gehören:

4x Hallsensoren Zur Detektion der Radrotation wurde der Truck mit Sensoren des Typs KMI15/1 von Philips/NXP ausgestattet.

1x AD-Wandler Die Auswertung des AD-Wandlers erfolgt zur Kontrolle der Spannungsversorgung. Angepasst über einen Spannungsteiler liegt hier die Versorgungsspannung aus den beiden Akkus an. Der AD-Wandler des NXP LPC1768 verfügt über sechs Kanäle. Für spätere Erweiterungen der Sensorik und für Testzwecke, werden schon jetzt alle Kanäle ausgewertet.

2x Stellservo Auf dem Fahrzeug sind zwei Servos verbaut. Diese dienen der Kontrolle von Geschwindigkeit (Gas/Bremse) sowie der Lenkung.

SteuerPC Wie auf allen bereits vorgestellten FAUST-Plattformen ist auch der Intelli-Truck mit einem SteuerPC ausgestattet (vgl. Kapitel 3.2).

Aus den genannten Komponenten ergeben sich nun eine Reihe von bereits angesprochenen Aufgaben. Diese werden nach dem Ansatz des Frameworks in einzelnen Tasks des Schedulers abgebildet. Die einzelnen Tasks und deren Aufgabe werden nun kurz erläutert.

5.2.1 Tasks

Ein Task ist bewusst einfach aufgebaut. Er besteht aus einer `task_XY_Init(void)` Routine, sowie einer `task_XY_Update(void)` Routine. Die Init-Routinen aller Tasks werden nach der Initialisierung des Schedulers durchlaufen. Sie bieten die Möglichkeit, initiale Aufgaben zu erledigen oder gegebenenfalls lokale Datenstrukturen zu initialisieren. Die Update-Routine wird von dem Scheduler aufgerufen, wenn dieser feststellt, dass der jeweilige Task mit der Ausführung an der Reihe ist. In dieser Routine laufen fortlaufende Berechnungen und Algorithmen ab.

`task_DataContainer` Bereitstellung einer globalen Datenstruktur, in welcher alle Module Informationen ablegen können. Dieser Task ist in jeder Konfiguration vorhanden.

`task_Distance` Der Task wertet die Daten der Hallsensoren aus. Da die Informationen der Hallsensoren zu den wichtigsten in der mobilen Robotik gehören, werden diese per Interrupt erfasst. Es werden die *Capture* Funktionalitäten der internen `TIMER0` und `TIMER2` genutzt. Bei diesem Verfahren werden je zwei Hallsensoren von einem Timer überwacht.

Bei Erkennung einer steigenden Flanke am Eingang wird die ISR des jeweiligen Timers aufgerufen und der zugehörige Wert im Datencontainer inkrementiert. Sollten beide Kanäle eines Timers zeitgleich ausgelöst haben, so werden diese in einem ISR-Durchlauf bearbeitet und deren Interrupt zurückgesetzt. Neben der Erfassung der Daten per Interrupt, besitzt der Task die übliche Update-Routine. In dieser werden Berechnungen anhand der Daten der Hallsensoren angestellt, welche in diesem Fall lediglich die Geschwindigkeit liefern.

`task_Display` Dieser Task steuert die LCD-Anzeige des KEIL MCB1760 an. Das Display soll späteren Debugzwecken dienen und dem Anwender Informationen über den Zustand der FAUSTmosap Plattform liefern.

`task_ComETH` Kommunikation mit dem SteuerPC per Ethernet. Auch dieser Task muss mit einem gemischten Ansatz aus Interrupts und der normalen Update-Routine implementiert werden. Im Anforderungskatalog ist unter der Nummer 10 eine Zykluszeit von 25ms gefordert. Diese ist in der Update-Routine des Tasks implementiert. Im Gegensatz zu den geforderten Zyklen des Sendens der Sensorinformationen ist für das Empfangen der Stellinformationen des Steuer-PCs keine Zeitangabe definiert worden. Dieser Umstand erzwingt eine Implementierung in der ISR `ENET_IRQHandler` des Tasks, da die Stellinformationen zu jeder Zeit entgegengenommen werden müssen.

task_IrRangers Auswertung des Analog-Digital-Wandlers. Die Benennung des Tasks mag im ersten Moment irreführend erscheinen, ist jedoch an die Auswertung der Infrarotsensoren angelehnt, welche zumeist an dem AD-Wandler angeschlossen sind. Dieser Tasks besitzt keine Interrupts, da der AD-Wandler des NXP LPC1768 im *Burst-Mode* betrieben wird. Hierbei werden die Eingänge fortlaufend überwacht und das jeweils letzte Ergebnis einer Umwandlung in den jeweiligen Registern des Analog-Digital-Wandlers vorgehalten. Es ist also ausreichend, wenn der Task zyklisch die Ergebnisse in den Datencontainer einträgt.

Um eine schnellst mögliche Reaktion auf das Abfallen der Versorgungsspannung garantieren zu können, soll der AD-Wandler in möglichst kurzen Zyklen abgefragt werden und gegebenenfalls das System in den von Anforderung Nr. 7 geforderten sicheren Zustand versetzen.

task_ActorsPWM In diesem Task werden die Stellwerte für die Aktoren gesetzt. Da die Stellwerte zyklisch vom SteuerPC eintreffen, würde es ausreichen, diese an das Eintreffen der Informationen angepasst zu setzen. Durch den Akkubetrieb des Systems kann jedoch kein gleichbleibendes Ansprechverhalten der Aktoren erwartet werden. Damit eine später zu entwickelnde Regelung leichter implementiert werden kann, wird dieser Task schon jetzt mit kürzeren Zyklen angelegt.

5.3 Konfiguration des Schedulers

Nach der Einrichtung der zurvor beschrieben Tasks können die Berechnungen für die Konfiguration des Schedulers vorgenommen werden. Diese basieren zum einen auf Laufzeitmessungen der einzelnen Tasks und Interrupts, zum anderen auf dem Wissen über die Struktur und den Aufbau des Mikrocontrollers. Durch die Analyse lässt sich bestimmen, ob zu einer Konfiguration ein Scheduling möglich ist.

Die genauen Ergebnisse (Diagramme) der Laufzeitmessungen und der dazugehörige Testaufbau sind im Anhang A zu finden. Unter der Anforderungen Nr. 10 ist die Vorgabe von einem 25ms Zyklus für die Kommunikation mit dem SteuerPC zu finden. Dies stellt den längsten Zyklus des Systems dar, da bis zu dem Zeitpunkt der Datenübermittlung alle Berechnungen abgeschlossen sein müssen. Da einige der Tasks häufiger ausgeführt werden sollen (vgl. 5.2.1), muss dieses Intervall noch unterteilt werden. Hierfür bieten sich nur ganzzahlige Teiler von 25 an, womit die Wahl zwischen SystemTicks von 1ms oder 5ms getroffen werden kann. Da spätere Aufgaben wie eine Regelung möglichst fein arbeiten sollen, wird hier eine Auflösung von 1ms / SystemTick gewählt. Diese Entscheidung gilt es durch eine Berechnung zu prüfen.

5.3.1 Berechnung Scheduler

Der Formel 4.2 nach müssen zunächst die Laufzeit des Schedulers sowie aller Interrupts betrachtet werden. Das Diagramm A.1 zeigt die Messung der Scheduler-Routine. Wie in Kapitel 5.1 beschrieben, handelt es sich um eine ISR. Zu der gemessenen Zeit in der Routine müssen also Zeiten für das Betreten und das Verlassen der ISR addiert werden. Dies sind insgesamt 24 CPU-Zyklen, wie man den Unterlagen zum technischen Design des CORTEX-M3 entnehmen kann (siehe ARM Ltd., 2010). Daraus ergeben sich folgende Berechnungen für die Dauer eines CPU-Zyklus ($T_{CoreCycle}$) und die Gesamtlaufzeit des Schedulers (T_{Sched}):

$$T_{CoreCycle} = \frac{1}{100MHz} = 10ns \tag{5.1}$$

$$T_{Sched} = (N_{CoreCycles} \times T_{CoreCycle}) + T_{SchedISR} \tag{5.2}$$

Die Messung A.1 gibt als maximale Dauer 3,024μs an. Mit diesem Wert ergibt sich für die gesamte Laufzeit des Schedulers:

$$T_{Sched} = (24 \times 10ns) + 3,024\mu s \approx 3,264\mu s \tag{5.3}$$

Da bei Messungen immer mit Toleranzen gerechnet werden muss, wird dieser Wert (großzügig) zu 4μs aufgerundet. Die Routine lastet den Mikrcontroller bei einer Schrittweite von 1ms / SystemTick also zu 0,4% aus.

5.3.2 Berechnung Interrupts

Die unter 5.2.1 beschriebenen Tasks liefern für die Berechnung zwei Interruptquellen: zum einen die Kommunikation über Ethernet und zum anderen die Erfassung der Hallsensoren.

Hallsensor Interrupt

Die Interrupts der Hallsensoren treten (während der Fahrt) zyklisch auf. Die Frequenz ist direkt abhängig von der Geschwindigkeit. Um ein zuverlässiges Scheduling erstellen zu können sollte in solchen Fällen mit der maximalen Frequenz und einem zusätzlichen Zeitpolster gerechnet werden. Um sich in diesem Fall nicht auf Messungen verlassen zu müssen, wird für die Berechnung die technisch maximal erfassbare Frequenz von 20kHz

des Sensors herangezogen. Für die Periodenlänge zwischen den Interrupts eines Sensors ($T_{Hall-Freq}$), sowie die Anzahl möglicher Interrupts durch einen Hallsensor (N_{Hall}) während eines SystemTicks ergeben sich folgende Rechnungen:

$$T_{Hall-Freq} = \frac{1}{20kHz} = 50\mu s \tag{5.4}$$

$$N_{Hall} = \frac{T_{SystemTick}}{T_{Hall}} = \frac{1000\mu s}{50\mu s} = 20\ Interrupts \tag{5.5}$$

Während eines SystemTicks können nach Gleichung 5.5 bis zu 20 Interrupts auftreten - durch einen Kanal. Bei der Erfassung von vier Sensoren, welche im schlechtesten Fall alle nacheinander auslösen, kann es also bis zu 80 Interrupts kommen.

Die Messung A.9 zeigt die Laufzeit von zwei aufeinander folgenden Interrupts. Es ist zu erkennen, dass die Behandlung einer ISR relativ genau 2μs in Anspruch nimmt ($T_{HallISR}$). Mit diesem Wert lässt sich die Gesamtlast der Hallsensoren während eines SystemTicks berechnen.

$$T_{Hall} = (N_{Hall} \times N_{ISR}) \times ((N_{CoreCycles} \times T_{CoreCycle}) + T_{HallISR}) \tag{5.6}$$

$$T_{Hall} = (20 \times 4) \times (24 \times 10ns) + 2\mu s) \approx 179,2\mu s \tag{5.7}$$

Die Prozessorzeit, welche durch die Hallsensoren in einem SystemTick verbraucht wird, wird als „ca. Wert“ angegeben, da die Berechnungen auf einem Messergebnis beruhen. Um spätere Berechnungen zu erleichtern, wird dieser Wert auf 180μs aufgerundet.

Ethernet Interrupt

Die Berechnung des Interrupts für die Ethernet-Kommunikation stellt in diesem Teil das größte Problem dar. Es muss hier zwischen dem Senden und Empfangen von Nachrichten unterschieden werden.

Senden Die eigentliche Leistung, das Vorbereiten und Kopieren der Nachricht, wird in der Routine des Tasks abgewickelt. Ist das Senden jedoch abgeschlossen, wird durch den Ethernetbaustein ein Interrupt ausgelöst, welcher über den Erfolg oder Misserfolg des Versuchs Auskunft gibt. Dieser Interrupt, in dem Diagram A.3 abgebildet, tritt *ausschließlich* und *unmittelbar* nach dem Versenden auf. Er muss deshalb nicht

für die Berechnung der dauerhaften Last berücksichtigt werden und kann der Laufzeit der Task-Routine zugeschrieben werden.

Empfangen Das Empfangen von Nachrichten ist vollständig Interrupt gesteuert. Da über das Laufzeitverhalten des Steuer-PCs keine verlässliche Aussage getroffen werden kann, muss angenommen werden, dass der zugehörige Interrupt während der gesamten Laufzeit auftreten kann.

Der Interrupt durch den Datenempfang tritt während des 25ms Zyklus des Steuer-PCs nur einmal auf, was die Berechnung vereinfacht. Der Versuch einer Messung des Empfangsinterrupts gestaltet sich in der Praxis sehr schwierig. Das Diagram A.2 zeigt diesen Versuch, wobei der erste, kurze Interrupt durch das Senden verursacht wird und der zweite durch das Empfangen. Mit einer (großzügigen) Abschätzung wird dem zweiten Interrupt eine Dauer von 50μs zugeschrieben.

$$T_{Eth} = (N_{CoreCycles} \times T_{CoreCycle}) + T_{EthISR} \tag{5.8}$$

$$T_{Eth} = (24 \times 10ns) + 50\mu s \approx 50,24\mu s \tag{5.9}$$

5.3.3 Berechnung der gesamten Last durch Scheduler und Interrupts

Nach der Berechnung der einzelnen Laufzeiten kann nun eine Aussage über die Zeit ($T_{reserved}$) getroffen werden, welche in jedem SystemTick für Interruptbehandlungen und den Scheduler reserviert werden muss.

$$T_{reserved} = T_{Sched} + T_{Hall} + T_{Eth} \tag{5.10}$$

$$T_{reserved} = 4\mu s + 180\mu s + 50,24\mu s = 234,24\mu s \tag{5.11}$$

Der Mikrocontoller wird demnach im ungünstigsten Fall $\approx 235\mu$s pro SystemTick für die Interruptbehandlung und den Scheduler benötigen. Dies entspricht einer Auslastung von 23,5%. Für die Bearbeitung von Tasks stehen somit noch 765μs pro SystemTick zur Verfügung.

$$T_{TaskMax} = 1ms - 235\mu s = 765\mu s \tag{5.12}$$

5.3.4 Verteilung der Tasks

Die Berechnung der statischen Last aus dem vorherigen Kapitel bildet die Grundlage für die Verteilung der Tasks innerhalb der Zeitslots. Die ermittelten Laufzeiten, deren Messergebnisse im Anhang unter A zu finden sind, seien hier zusammengefasst dargestellt:

Task	ActorPWM	Ethernet	IrRangers	Distance	Display
Laufzeit (μs)	1,6	158,3	1,707	2,133	1070

Tabelle 5.1: Laufzeiten der Task Routinen

Es ist zu erkennen, dass bis auf den Task für die Anzeige alle Laufzeiten weit unter der geforderten Grenze liegen. An diesem Punkt gibt es zwei unterschiedliche Lösungsansätze. Es kann die Auflösung der SystemTicks verändert und an die höchste Laufzeit angepasst werden, oder es wird versucht, durch geschickte Plazierung der Tasks im Scheduling, einen "*leeren* SystemTick hinter dem Task mit der Zeitverletzung zu platzieren.

Befindet sich hinter einem SystemTick mit einer Zeitverletzung ein weiterer (teil-)belegter SystemTick, so würde dessen Beginn um den Anteil der Überschreitung verzögert. Dies ist durch das Funktionsprinzip eines kooperativen Schedulers gegeben. Das Zeitverhalten würde mit einem Jitter belegt und das Verhalten des Systems wären nicht mehr verlässlich.

Um die hohe Reaktionsgeschwindigkeit des Systems beizubehalten, wird hier der zweite Ansatz verfolgt. Aus den bisher beschriebenen Bedingungen und Eigenschaften der Tasks ergibt sich damit folgende Verteilung über die Zeitschritte des Schedulers:

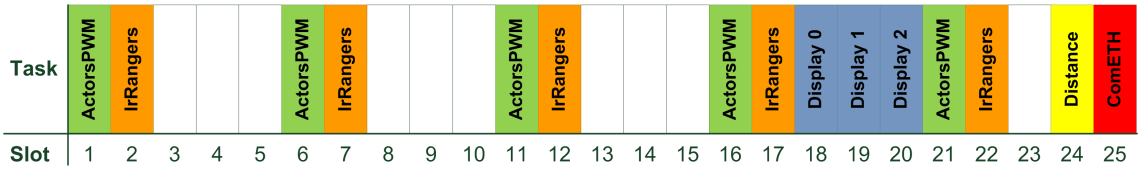

Abbildung 5.2: Task-Belegung des Schedulers

Durch eine entsprechende Verteilung konnte ein gültiges Scheduling erstellt werden, da die Ausführungszeiten der einzelnen Tasks zum einen kurz genug waren, bzw. im Fall einer Verletzung der maximalen Ausführungszeit eine geeignete Anzahl an Slots zum Abfangen der Überschreitung gefunden werden konnte.

Die Laufzeiten der einzelnen Tasks sind an dieser Stelle *nicht* proportional gekennzeichnet, da die Darstellung sonst unkenntlich wäre. Dem Display-Task werden drei Slots zugeordnet. Die Ausführung des Tasks beginnt in Slot 18. Da es sich um einen Task mit

Zeitverletzung handelt, werden hier (großzügig) die beiden folgenden (sonst freien) Slots zusätzlich belegt. Weiße Slots haben keine Belegung und sind frei.

Einen tieferen Einblick in die Planungsverfahren gibt Dieter Zöbel in seinem Buch „Echtzeitsysteme - Grundlagen der Planung“ in den Kapiteln 3 und 4 (Zöbel, 2008).

Kapitel 6

Testszenarien

Der bisher dargestellten Entwicklung des Sensor-Aktor-Systems folgen einige statische Tests. Als Plattform für den Test kommt ein autonomes Fahrzeug aus dem FAUST-Projekt der Hochschule für Angewandte Wissenschaften Hamburg zum Einsatz - der IntelliTruck. Wie im Verlaufe dieser Arbeit dargestellt wurde, leistet die FAUSTmosap Plattform zwei wesentliche Dienste. Es werden Sensordaten vom Fahrzeug an den Steuer-PC übertragen und im Gegenzug die Stellwerte von dem Steuer-PC an die Aktoren weitergeleitet. Diese beiden Funktionen sollen in diesem Kapitel getestet werden.

6.1 FAUSTmosap ServiceTool

Für die Verwendung in Tests wurd eine kleine Software entwickelt, welche die durch die Plattform gelieferten Sensordaten anzeigen kann. Ferner ist es mit diesem Programm möglich, Nachrichten mit Stellgrößen an die Plattform zu senden. Abbildung 6.1 zeigt diese Anwendung.

6.2 Test: Sensorik

6.2.1 Aufbau

Da es sich um die ersten Funktionstests eines Prototypen handelt, wäre es unverantwortlich, diese direkt unter realen Bedingungen durchzuführen. Für den Test der Sensorik wur-

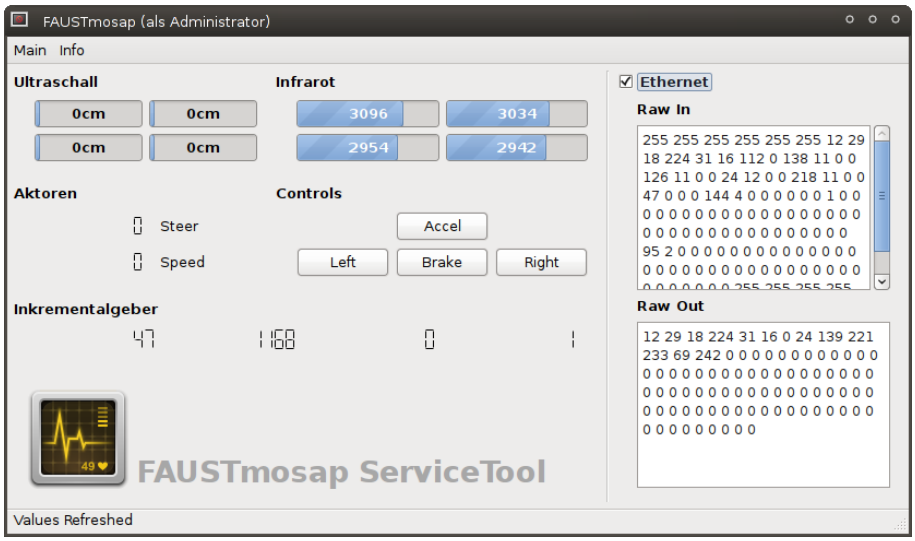

Abbildung 6.1: ServiceTool für die Plattform

de der Intelli-Truck im Labor aufgebockt und die Eingänge für die Radsensorik mit einem Funktionsgenerator verbunden. Diese Art von Aufbau bietet zwei Vorteile:

- Es kann ohne Gefahr für das Fahrzeug eine sehr hohe Geschwindigkeit simuliert werden.
- Bei dem Anschluss an einen einzigen Funktionsgenerator werden alle vier Eingänge gleichzeitig ausgelöst.

6.2.2 Durchführung

Die Durchführung des Tests muss in zwei Teile aufgespalten werden, damit der Verlauf nachvollziehbar ist. Das von den Hallsensoren erzeugte Signal wird zunächst von der Plattform erfasst (Test: Sensor zu Plattform), und dann zum PC übermittelt (Test: Plattform zu PC).

Test: Sensor zu Plattform

Um über die Signalerfassung eine möglichst sichere Aussage treffen zu können, soll für diesen Test die maximal mögliche Frequenz der Hallsensoren erkennbar überschritten werden. Statt der in Kapitel 5.3.2 beschriebenen 20kHz, werden für diesen Test 30kHz über den Funktionsgenerator erzeugt.

Die Last durch die Interrupts der Hallsensoren, welche in der Formel 5.7 mit 179,2μs berechnet wurde, steigt bei einer Frequenz von 30kHz auf 295,68μs. Das unter Kapitel 5.3.4 beschriebene Scheduling bietet für diesen Anstieg genügend Reserven und wird nicht verletzt.

Um die Signalverarbeitung messbar zu machen, wird in den Interrupt-Routinen zur Erfassung der Hallsensoren ein sonst ungenutzer Pin des Mikrocontrollers geschaltet.

Das Ergebnis der Messung kann im Anhang A.8 betrachtet werden. Es ist zu erkennen, dass die Interruptbehandlung (Kanal B / rot) im direkten Vergleich mit dem Signal des Frequenzgenerators (Kanal A / blau) mit sinkendem Pegel beginnt. Dieses Verhalten ist durch einen invertierenden Schmitt-Trigger auf der Basisplatine bedingt, welcher hier zur Signalaufbereitung zwischengeschaltet ist. Das Diagramm zeigt aus technischen Gründen lediglich die Bearbeitung einer Interrupt-Routine.

Der erste (Teil-)Test kann mit dieser Messung als erfolgreich angesehen werden, da jeder übermittelten Signalflanke die entsprechende Bearbeitung/Erfassung durch den Mikrocontroller folgt. Die Daten wären somit im Datencontainer der Plattform angekommen.

Test: Plattform zu PC

Für den zweiten Testschritt wurde das Vorgehen geändert, um auch die Hallsensoren auf ihre Funktion prüfen, und das Ergebnis besser darstellen zu können. In diesem Fall wird der Funktionsgenerator aus dem Test entfernt und die Hallsensoren wieder angeschlossen. Zur Signalerzeugung wird ein Rad mit der Hand bedient.

Zur Überprüfung der Kommunikation zwischen dem Mikrocontroller und dem SteuerPC wird diese mit dem Programm *WireShark* aufgezeichnet. Zusätzlich wurde eine Software entwickelt, welche die Daten des Mikrocontrollers darstellen kann.

Als Ergebnis des Tests sind die Log-Daten des Programms *WireShark* zu betrachten. Wie in Abbildung 6.2 zu erkennen ist, werden die geforderten Zyklen von 25ms für den Versand der Daten eingehalten - dies ist aus den Zeitangaben im linken Teil des Fensters zu erkennen (Zeitangaben des Programms in Sekunden).

Zusätzlich sind auf dem Bildschirmfoto zwei einzelne Nachrichten geöffnet. In der Darstellung des Datenteils, ist jeweils ein Wert rot markiert. Diese stellen Werte der Hallsensoren dar, welche fortlaufend ansteigen, wenn das Rad bewegt wird.

Der Test der Kommunikation kann also ebenfalls als erfolgreich bezeichnet werden. Es werden im Datencontainer auflaufende Werte in der geforderten Zeit an den PC übermit-

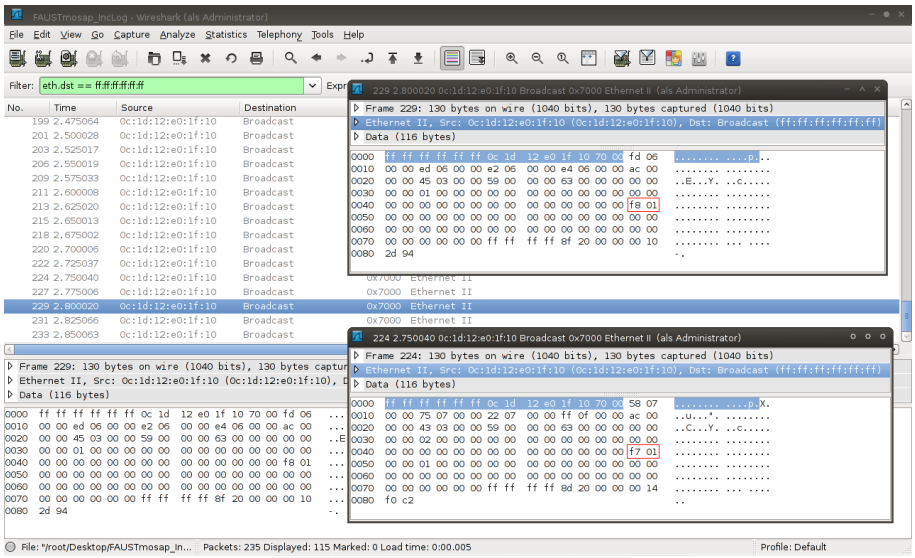

Abbildung 6.2: Das Programm *WireShark* nach dem Test

telt. Im Rahmen dieses zweiten (Teil-)Tests wurde durch den geänderten Testaufbau auch das Zusammenspiel mit den Hallsensoren positiv getestet.

6.2.3 Ergebnis

Somit ist der Test für die Sensorik abgeschlossen. Im ersten Schritt, wurde erfolgreich bewiesen, dass der Mikrocontroller die geforderten Datenmengen, welche die Hallsensoren produzieren, verarbeiten kann. Der zweite Schritt hat das Zusammenspiel der Plattform mit den echten Hallsensoren, sowie die korrekte Kommunikation mit dem SteuerPC bewiesen.

6.3 Test: Aktorik

Der Test soll die zweite Funktion der Plattform, das Übermitteln von Stellwerten an die Sensorik, prüfen. Auch dieser Test wurde aus Sicherheitsgründen mit einem aufgebockten Fahrzeug durchgeführt.

6.3.1 Aufbau

Das *FAUSTmosap ServiceTool*, welches im vorrausgegangenen Test erwähnt wurde, bietet neben der Darstellung der Sensordaten ebenfalls die Möglichkeit Stellwerte an die Plattform zu senden. Für den Test der Aktorik, wird ein LinuxPC mit der FAUSTmosap Plattform per Ethernet verbunden.

6.3.2 Durchführung

Sobald der PC mit dem Fahrzeug verbunden ist, wird über das FAUSTmosap ServiceTool die Veränderung einer Stellgröße in zwei Schritten durchgeführt. Durch das zweimalige Drücken des *Accel*-Buttons wird die Inkrementierung der Stellgröße für den Stellmotor um einen Wert von jeweils „+10“ veranlasst. Der angesprochene Stellmotor kontrolliert Gas und Bremse auf dem Intelli-Truck.

Die Werte, welche vom Steuer-PC an die Plattform geschickt werden reichen von -100 bis +100. Diese Abstraktion ist durch eine fehlende Geschwindigkeitsregelung auf den bisherigen Plattformen begründet, welche reale Geschwindigkeitswerte in entsprechende Stellgrößen umsetzen würde. Im Modellbaubereich lässt sich dieses Intervall linear auf die Pulsbreite von PWM-Signalen verteilen. Die Berechnung der Pulsbreite in μs zur Stellgröße N_{Stell} lautet:

$$T_{Pulsbreite} = 1500 + (5 * N_{Stell}) \tag{6.1}$$

Daraus ergibt sich die in Tabelle 6.1 dargestellt Verteilung.

Stellgröße PC	-100	...	0	...	+100
Pulsbreite PWM (μs)	1000	...	1500	...	2000

Tabelle 6.1: Verteilung der Stellgrößen auf PWM-Pulsbreiten

Zur Auswertung des Tests werden einmal die Werte betrachtet, welche durch den Steuer-PC verschickt werden und zum anderen die Stellgröße gemessen, welche die Plattform

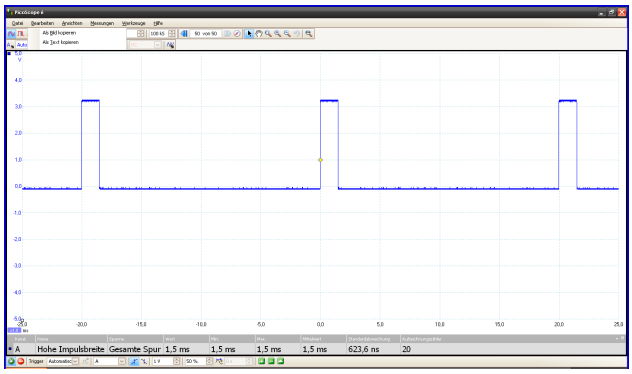

Abbildung 6.3: PWM-Signal Test Anfang

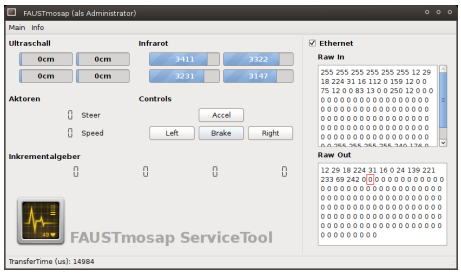

Abbildung 6.4: FAUSTmosap ServiceTool Test Anfang

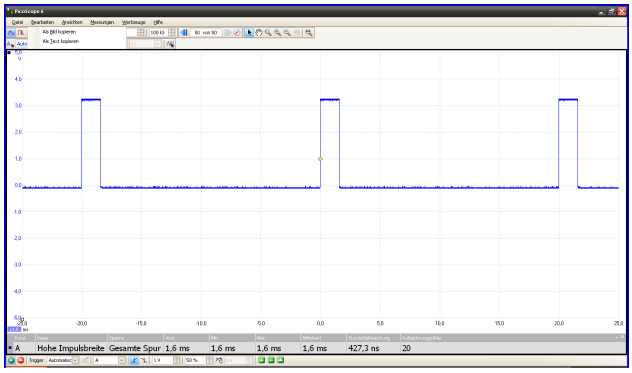

Abbildung 6.5: PWM-Signal Test Ende

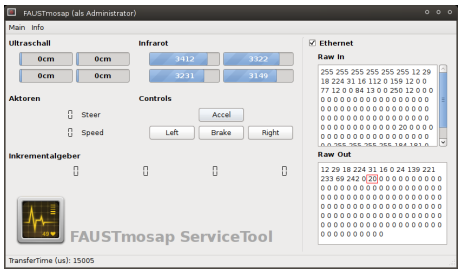

Abbildung 6.6: FAUSTmosap ServiceTool Test Ende

daraufhin ausgibt. Um die einzelnen Schritte darstellen zu können, wurde die Kommunikation zwischen Steuer-PC und Plattform wieder mit dem Programm WireShark protokolliert.

6.3.3 Ergebnis

In den Abbildungen 6.3 und 6.4 sind die Werte zu Beginn des Tests festgehalten. Das Bildschirmfoto zeigt das FAUSTmosap ServiceTool, in welchem die Stellgröße mit rot als 0 markiert ist. Das entsprechende Diagramm zeigt eine Pulsbreite von 1,5ms , was laut Tabelle 6.1 der Stellgröße entspricht.

Die Abbildungen 6.5 und 6.6 zeigen die Werte zum Ende des Tests. Das Digramm zeigt eine Pulsbreite von 1,6ms , was dem auf dem Bildschirmfoto zu sehenden Stellwert von 20 entspricht.

In Abbildung 6.7 sind Informationen über die Kommunikation während des Tests aufbereitet. Zu sehen sind hier, in der Grafik links unten, eine Nachricht zu Beginn des Tests, mit

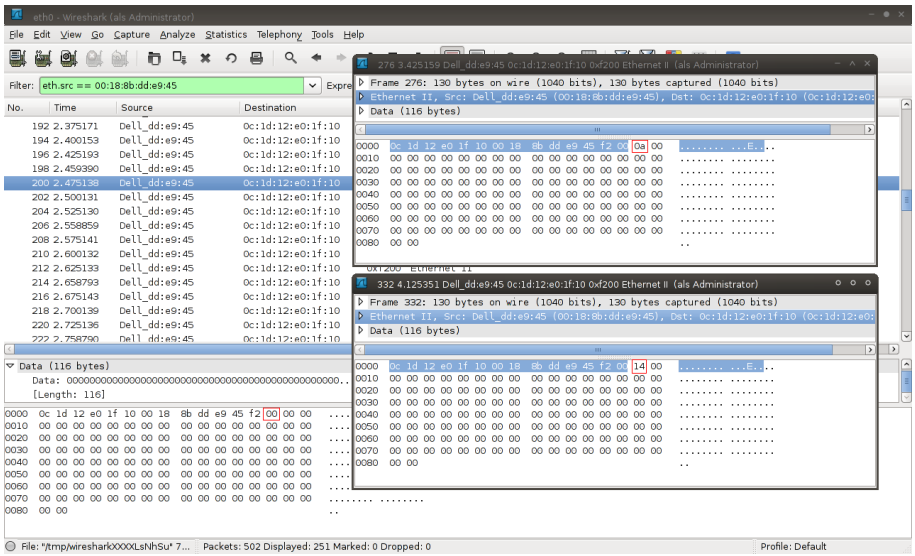

Abbildung 6.7: Log-Daten von WireShark zum Test

dem rot markierten Stellwert „0“. Zusätzlich zum Beginn und dem Ende des Tests, lässt sich in dem kleinen Fenster oben rechts, auch eine Nachricht des Testverlaufs betrachten. In dieser Nachricht ist die Stellgröße mit dem Wert „0x0A“ rot gekennzeichnet, welcher nach dem ersten Drücken des *Accel*-Buttons übertragen wurde. Das kleine Fenster unten rechts zeigt abschließend den schon bekannten Endstand des Tests mit dem rot markierten Stellwert von „0x14“.

Kapitel 7

Ausblick und Fazit

Das letzte Kapitel dieser Arbeit soll einen Ausblick auf die Möglichkeiten zukünftiger Entwicklungen bieten und ein kurzes Fazit geben.

7.1 Ausblick

In nur sechs Monaten kann selbstverständlich kein vollständiger Produktentwicklungszyklus durchlaufen werden. Weitreichendere Tests sind nötig und die vielseitigen Möglichkeiten der Erweiterung werden erst durch weitere Arbeiten und Forschung genutzt werden können. Einige Möglichkeiten, aber auch dringend erforderliche Änderungen sollen hier dargestellt werden.

Erforderliche Änderungen:

Montage Die Endmontage des Beispielfahrzeugs konnte noch nicht abgeschlossen werden, da bis zuletzt kleinere Änderungen an der Mechanik nicht abgeschlossen waren. Auch fehlen noch einige Kleinteile die bisher nicht bestellt werden konnten.

Basisplatine Die jetzige erste Version der Platine kann grundsätzlich eingesetzt werden. Wie bereits im Kapitel Hardwaredesign beschrieben, hat sich die Lösung des PWM-Schalters als diskrete Schaltung jedoch nicht bewährt. Die Schaltung sollte durch einen kleinen Mikrocontroller (bsp. AVR oder PIC) ersetzt werden. Auch ein Einsatz der von Enrico Hensel entwickelten Sicherheitsschaltung kann hier eine Lösung sein (Hensel, 2077). Neben dem PWM-Schalter, haben sich in der ersten Version kleinere Fehler in der Zuordnung der Ausgänge ergeben. Diese behindern keine Funktionen der Plattform, führen aber zu einem erhöhten Verkabelungsaufwand bei der Montage.

Möglichkeiten und Empfehlungen zur Weiterentwicklung:

Kommunikation Die Kommunikation zwischen der Plattform und dem SteuerPC wurde lediglich rudimentär, auf Basis von einfachen Ethernet-Nachrichten implementiert. Besonders im Bezug auf eine mögliche Erweiterung des Aufbaus, hin zu einem Betrieb mit mehreren SteuerPC's, empfiehlt sich hier die Entwicklung von einem gesonderten Nachrichtenformat.

Regelung Die Plattform sollte, mit einer Regelung für die Geschwindigkeit ausgestattet werden. Diese Funktionalität wurde bei vergangenen Projekten bisher nicht implementiert, wird aber von dieser Plattform durch das verhältnismäßig fein aufgelöste Scheduling besonders unterstützt.

Persistente Konfigurationen sobald die Plattform auf mehreren Fahrzeugen zum Einsatz kommt, wird es nötig sein, eine Art Konfiguration abzulegen. Dies sollte bei der oben genannten Entwicklung eines Nachrichtenformats berücksichtigt werden, da der Mikrocontroller eine Programmierung zur Laufzeit (IAP - In Application Programming) unterstützt. Es ergibt sich also die Möglichkeit die Plattform nach dem Einschalten zu konfigurieren, ohne sie demontieren zu müssen.

7.2 Fazit

Im Verlaufe dieser Arbeit, wurde ein Protoyp einer modularen Sensor-Aktor-Plattform entwickelt, umgesetzt und getestet. Begonnen wurde mit der Analyse der Fahrzeuge, auf welchen die Plattform zum Einsatz kommen sollte. Die Analyse hatte einen hohen Anteil an dieser Arbeit und stützte sich auf Erfahrungen, die in den letzten Jahren der Projekte gesammelt wurden. Der Analyse folgte die Erstellung der Anforderungen. Diese halten formal fest, welche Eigenschaften erfüllt, oder zumindest realisierbar sein müssen. Die angeschlossene Umsetzung beschränkte sich auf die Kernbereiche der Anforderungen mit dem klaren Ziel eine funktionsfähiges System zu erhalten. In die entwickelte Softwarestruktur lassen sich weitere Aufgaben in einfacher Weise einbinden.

Das Ergebnis ist ein, nach abgeschlossener Montage, einsatzbereiter Prototyp eines modularen Sensor-Aktor-Systems. Es beweißt, dass ein solches System mit ausreichender Planung realisierbar ist. Durch die Entwicklung dieses Prototypen wurden einige bekannte Schwachstellen der alten Plattformen beseitigt, in der Hoffnung, dass die FAUSTmosap Plattform damit eine stabile Grundlage für die zukünftigen Forschungsarbeiten darstellt.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Gabler 2001] *Gabler Wirtschaftslexikon, 8 Bde.* 15. Gabler, 9 2001. – ISBN 9783409303880

[ARM Ltd. 2010] ARM LTD.: *CORTEX-M3 Processor*. 2010. – URL http://www.arm.com/products/processors/cortex-m/cortex-m3.php. – Abruf: 2011-01-16

[Christen 2003] CHRISTEN, Markus: *Maschinenautonomie - zum „Selbst“ eines Roboters*. 2003. – URL http://www.encyclog.com/_upl/files/Maschinenautonomie_03.pdf. – Abruf: 2010-01-10

[FreeRTOS 2011] FREERTOS: *Embedded System Performance Comparisons*. 2011. – URL http://www.freertos.org/PC/. – Abruf: 2011-02-03

[Gottfried Wilhelm Leibniz Universität Hannover 2011] GOTTFRIED WILHELM LEIBNIZ UNIVERSITÄT HANNOVER: *Institut für Systems Engineering - Fachgebiet Echtzeitsysteme*. 2011. – URL http://www.rts.uni-hannover.de. – Abruf: 2011-01-08

[Hensel 2077] HENSEL, Enrico: *Design und Implementation eines Sicherheitskonzepts für den Betrieb eines autonomen Fahrzeuges*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2077

[Hochschule für angewandte Wissenschaften Hamburg 2010] HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG: *FAUST-Projekt*. 2010. – URL http://www.informatik.haw-hamburg.de/faust.html. – Abruf: 2010-10-23

[KEIL 2011] KEIL: *MCB1700 Evaluation Board*. 2011. – URL http://www.keil.com/mcb1700/. – Abruf: 2011-01-16

[Marwedel 2007] MARWEDEL, Peter: *Eingebettete Systeme*. 1., Aufl. 2007. Korr. Nachdruck. Springer, Berlin, 2 2007. – URL http://amazon.de/o/ASIN/3540340483/. – ISBN 9783540340485

[Pico Technology 2011] PICO TECHNOLOGY: *PicoScope 3000 Series*. 2011. – URL http://www.picotech.com/picoscope3000.html. – Abruf: 2011-02-7

[Pont 2001] PONT, Michael J.: *Patterns for Time-Triggered Embedded Systems: Building Reliable Applications with the 8051 Family of Microcontrollers (with CD-ROM)*. 1st. Addison-Wesley Professional, 7 2001. – URL http://amazon.com/o/ASIN/0201331381/. – ISBN 9780201331387

[Schneider 2011] SCHNEIDER, Frank E.: *ELROB - The European Robot Trial*. 2011. – URL http://www.elrob.org. – Abruf: 2011-01-10

[Technische Universität Braunschweig 2010] TECHNISCHE UNIVERSITÄT BRAUNSCHWEIG: *CaroloCup*. 2010. – URL http://www.carolo-cup.de/. – Abruf: 2010-10-22

[VDI-Fachbereich Produktionstechnik und Fertigungsverfahren 1990] VDI-FACHBEREICH PRODUKTIONSTECHNIK UND FERTIGUNGSVERFAHREN: *Montage- und Handhabungstechnik; Handhabungsfunktionen, Handhabungseinrichtungen; Begriffe, Definitionen, Symbole*. VDI-Gesellschaft Produktion und Logistik, 1990

[VisLab Universität Parma 2010] VISLAB UNIVERSITÄT PARMA: *VIAC Hompage*. 2010. – URL http://viac.vislab.it/. – Abruf: 2011-01-10

[Yiu 2009] YIU, Joseph: *The Definitive Guide to the ARM Cortex-M3, Second Edition*. Newton, MA, USA : Newnes, 2009. – ISBN 185617963X, 9781856179638

[Zöbel 2008] ZÖBEL, Dieter: *Echtzeitsysteme: Grundlagen der Planung*. 1. Springer, 3 2008. – URL http://amazon.com/o/ASIN/3540763953/. – ISBN 9783540763956

Anhang A

Laufzeitanalysen

In diesem Teil des Anhangs sind die Messergebnisse zur Laufzeit der verschiedenen Tasks aufgeführt. Die Diagramme wurden mit einem Oszilloskop des Typs 3206 der Firma Pico Technology durchgeführt (Pico Technology, 2011).

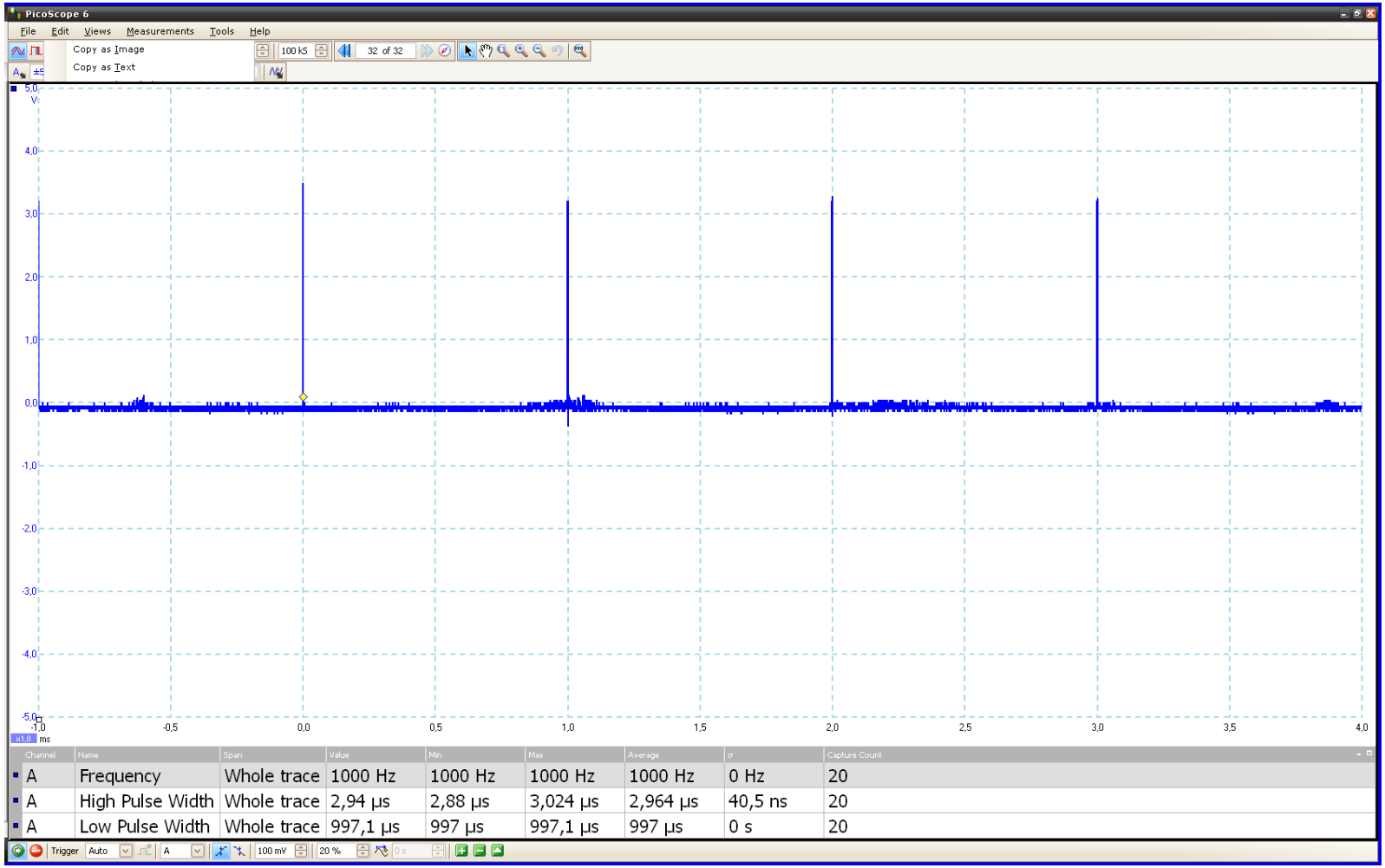

Abbildung A.1: Die Laufzeit des Schedulers (Interrupt)

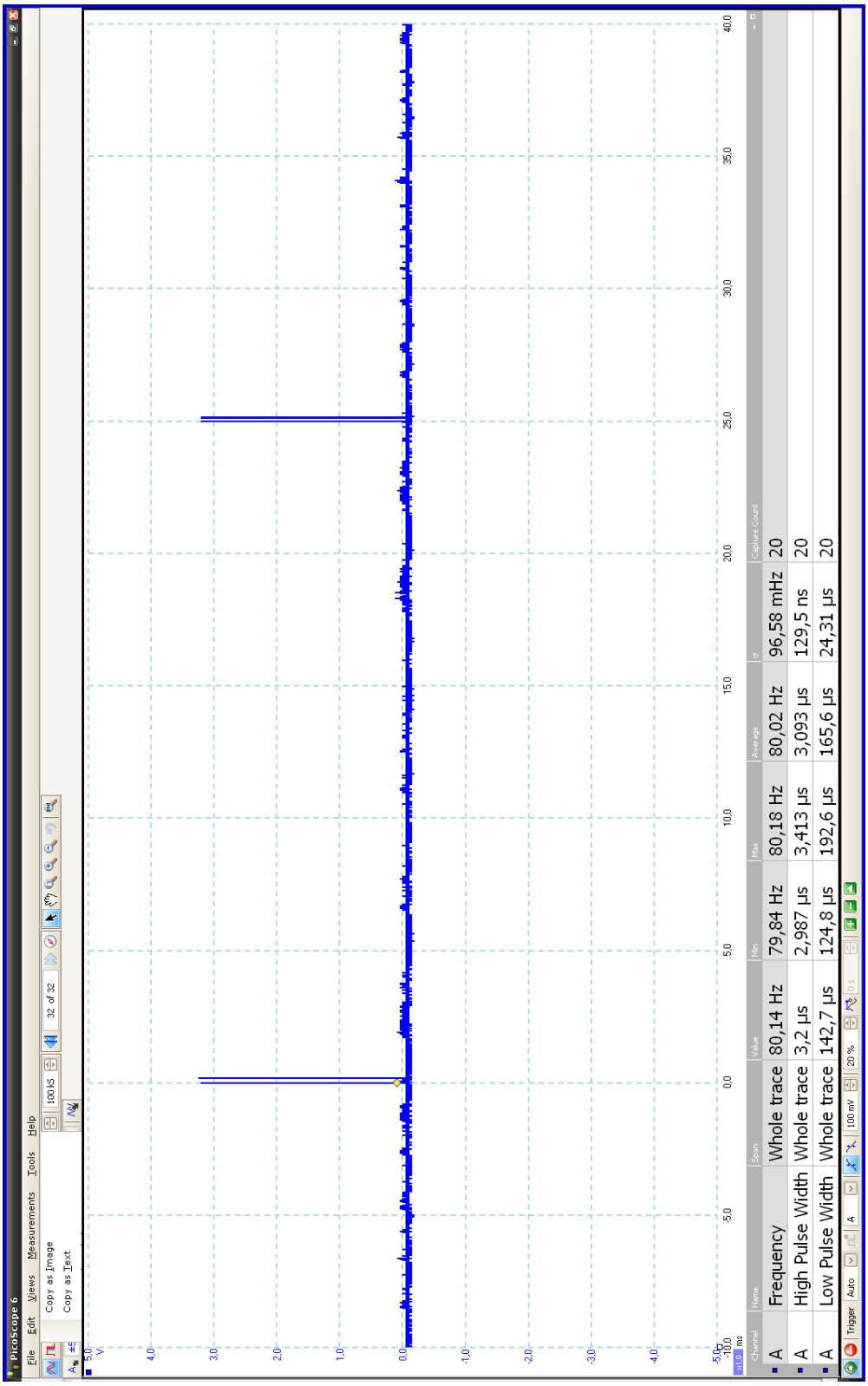

Abbildung A.2: Empfang und Senden per Ethernet (Interrupt)

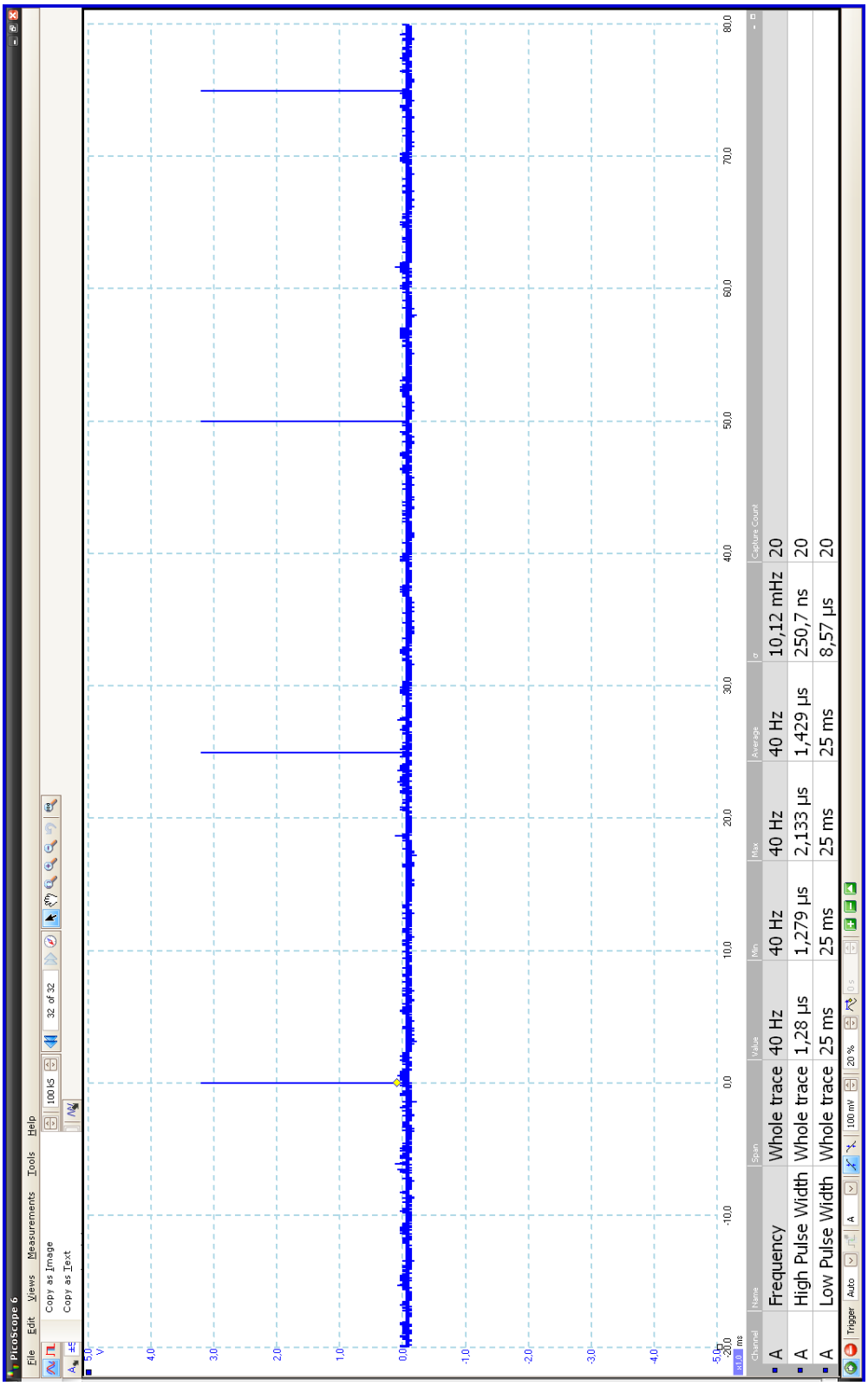

Abbildung A.3: Senden per Ethernet (Interrupt)

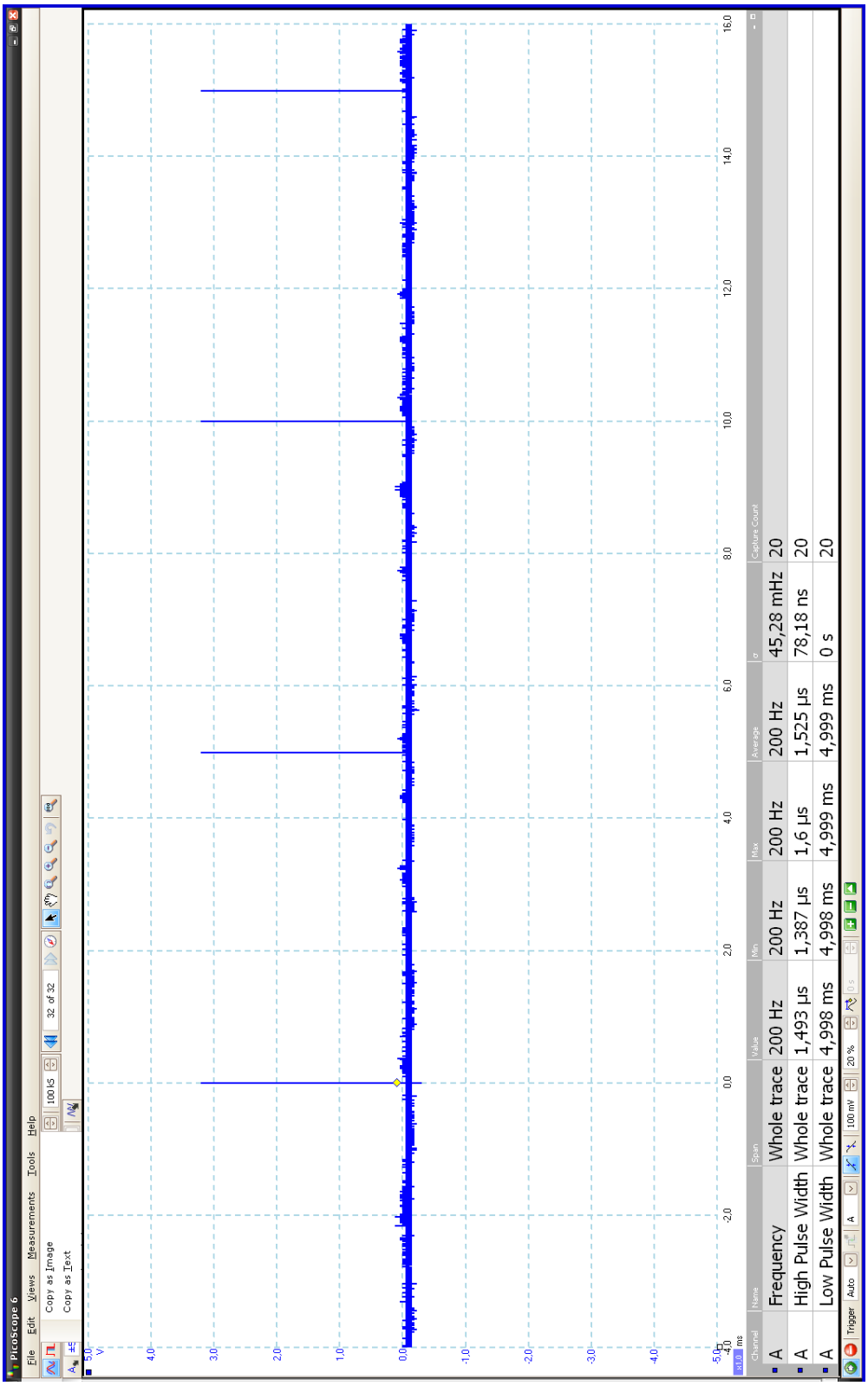

Abbildung A.4: Laufzeit des `task_ActorsPWM`

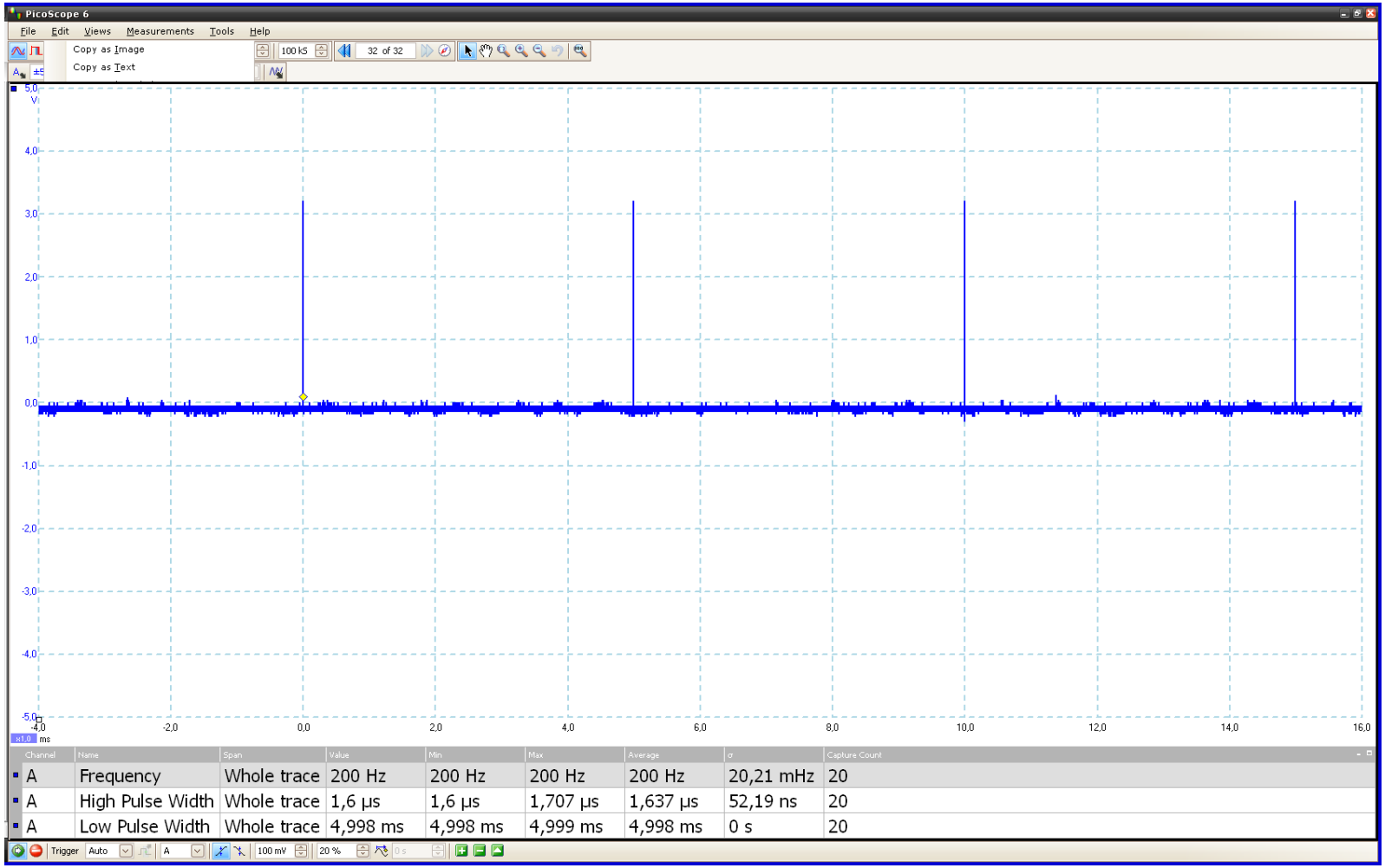

Abbildung A.5: Laufzeit des `task_IrRangers`

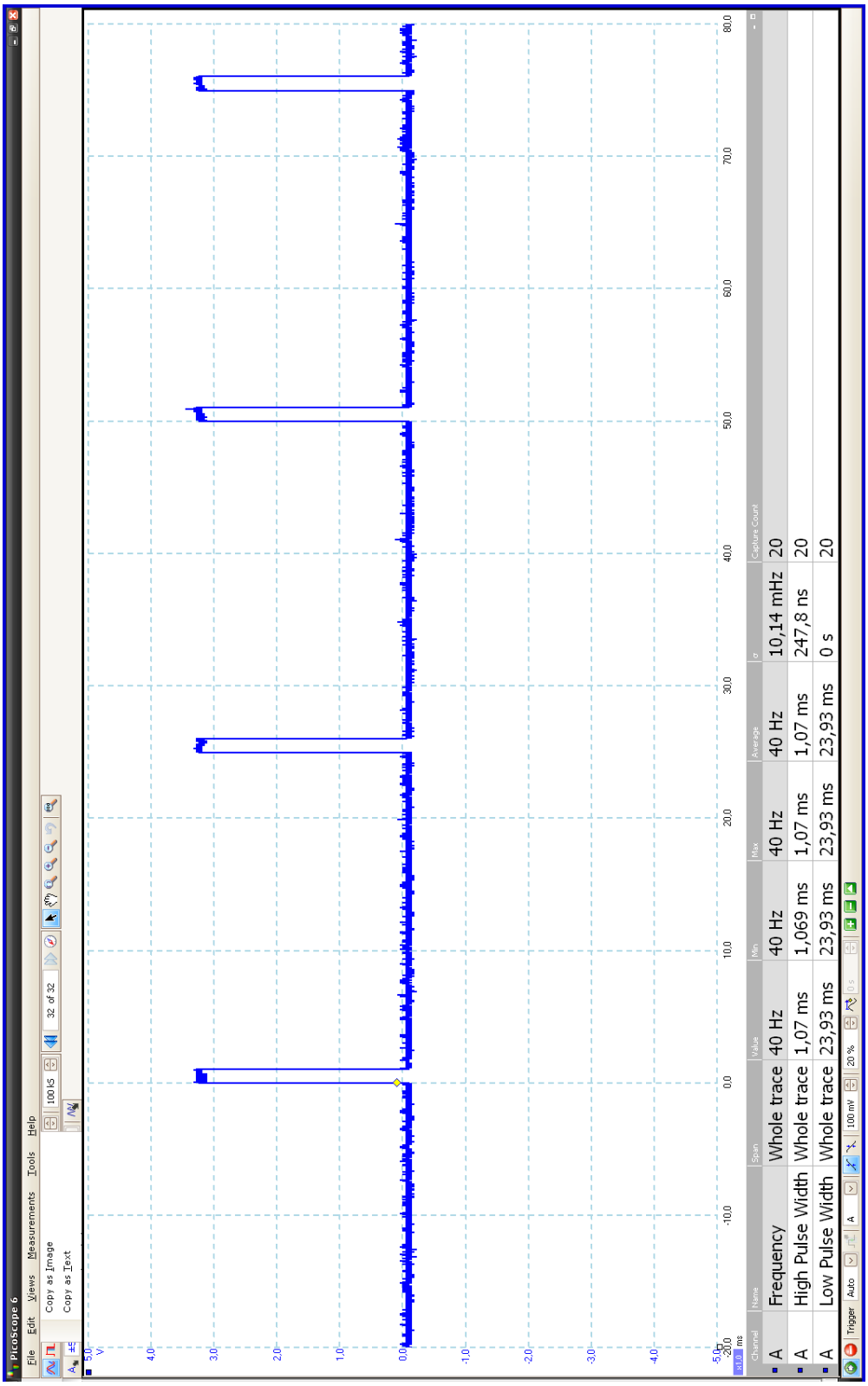

Abbildung A.6: Laufzeit des `task_Display`

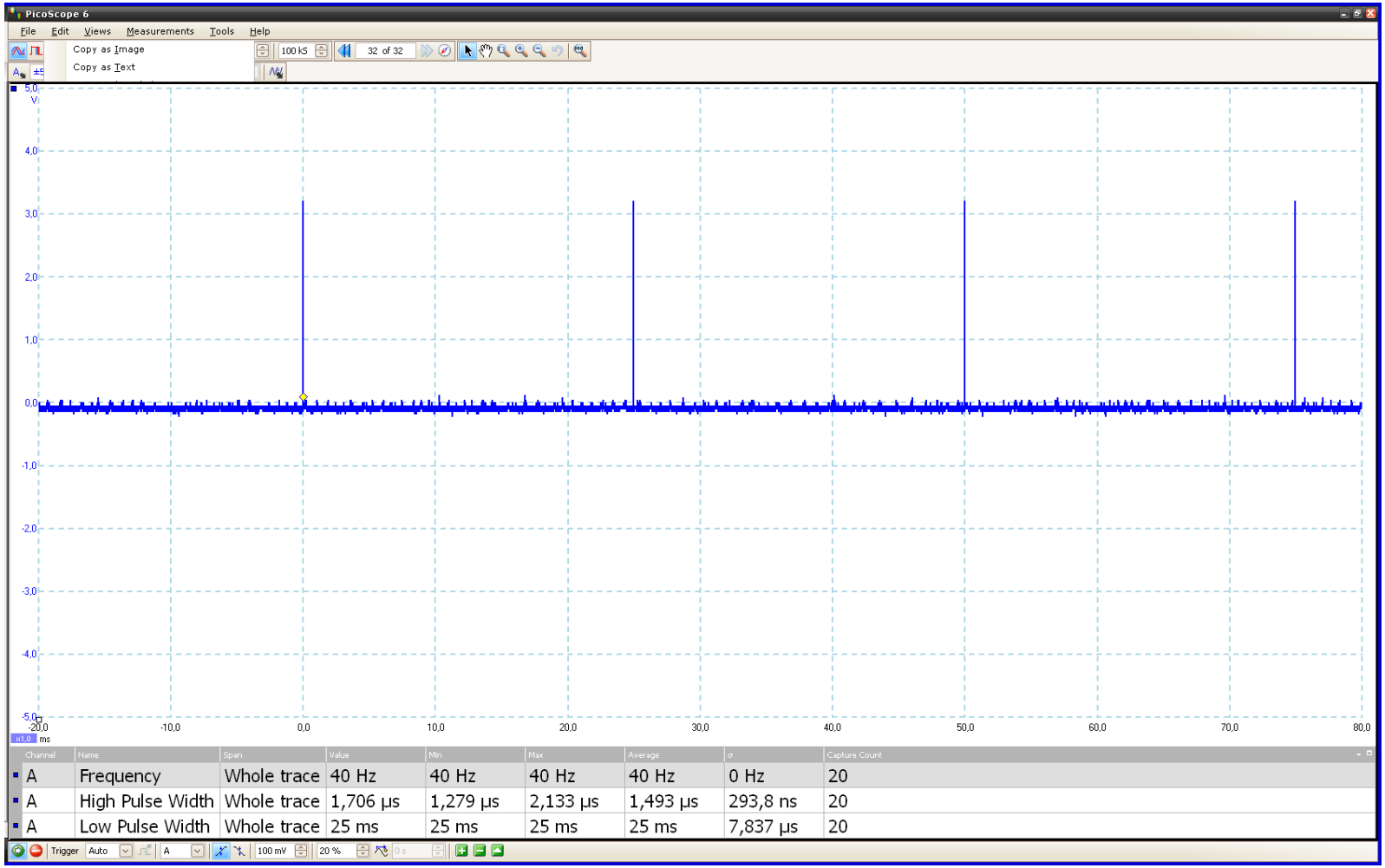

Abbildung A.7: Laufzeit des `task_Distance`

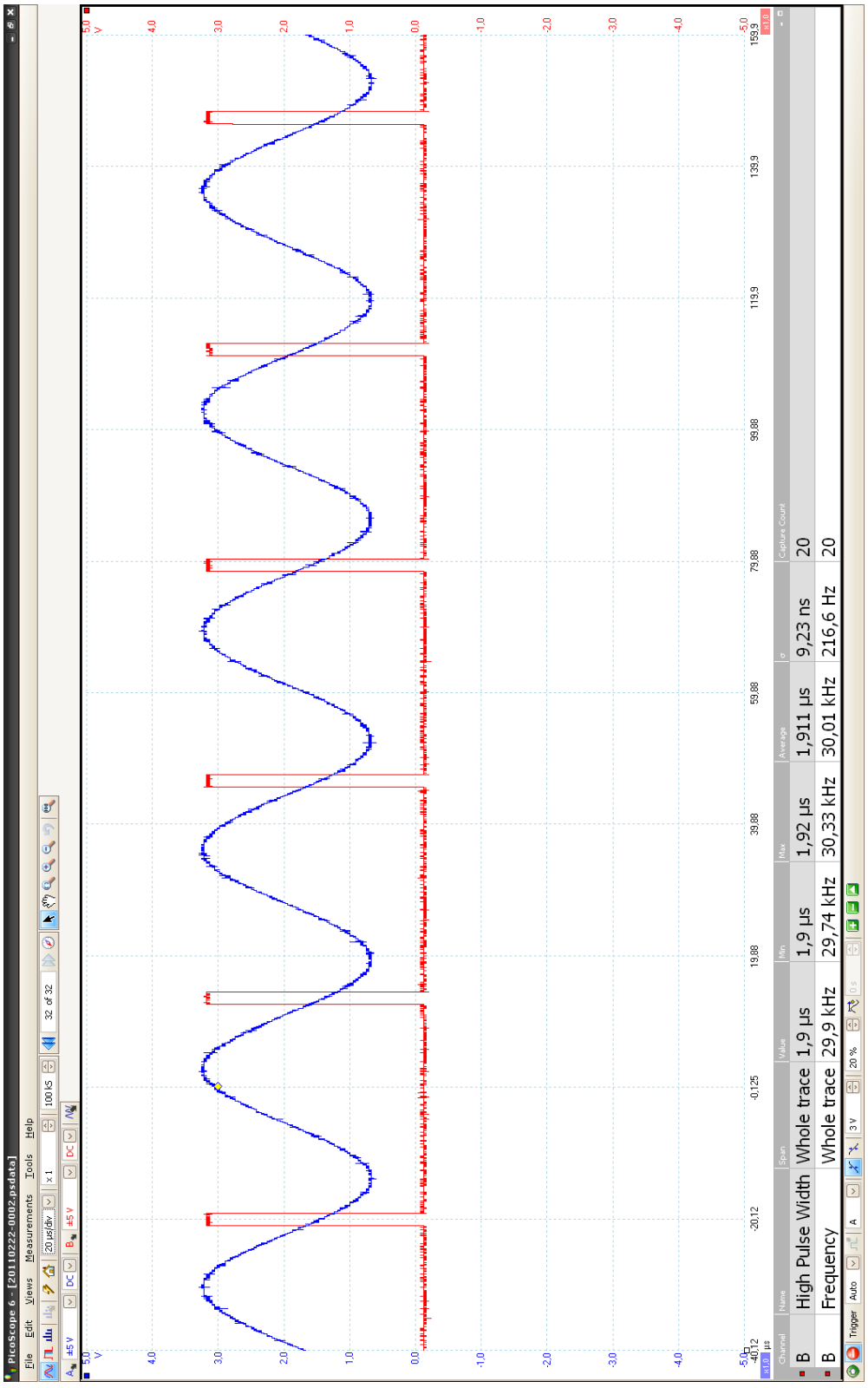

Abbildung A.8: Laufzeit des Softwaretests

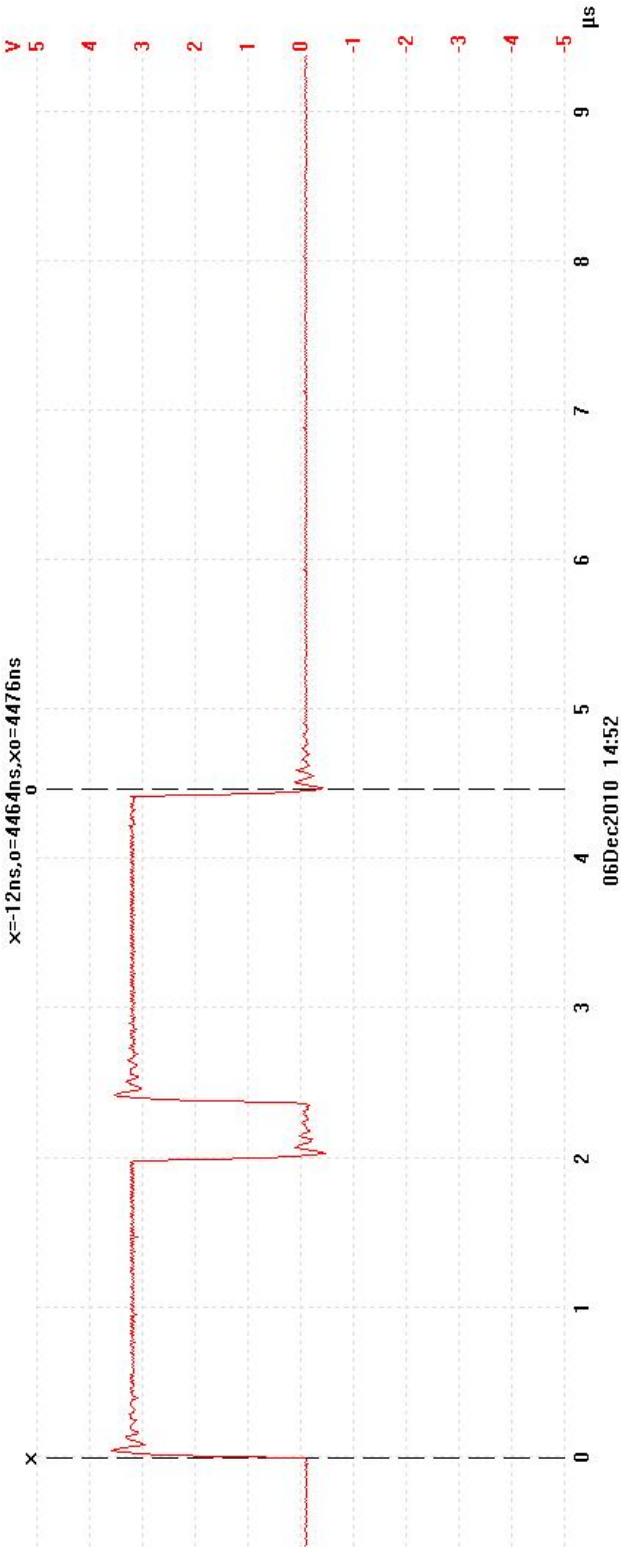

Abbildung A.9: Laufzeit der Tick Interrupts

Anhang B

Basisplatine

Der Anhang B enthält verschiedene technische Unterlagen zur Basisplatine. Diese wurden von Daniel Arnold erstellt und mit seiner Genehmigung hier aufgeführt.

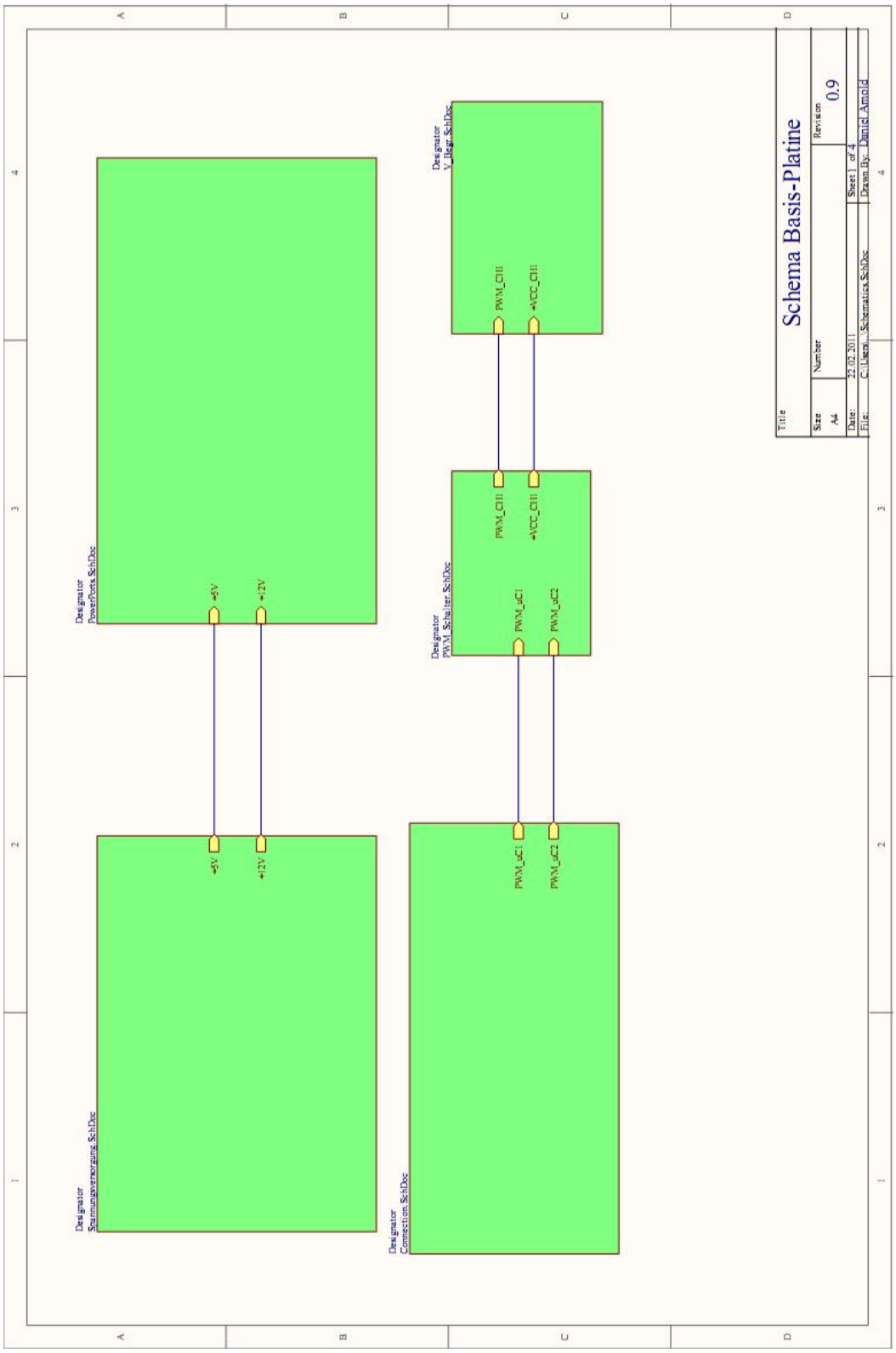

Abbildung B.1: Schema der Basisplatine

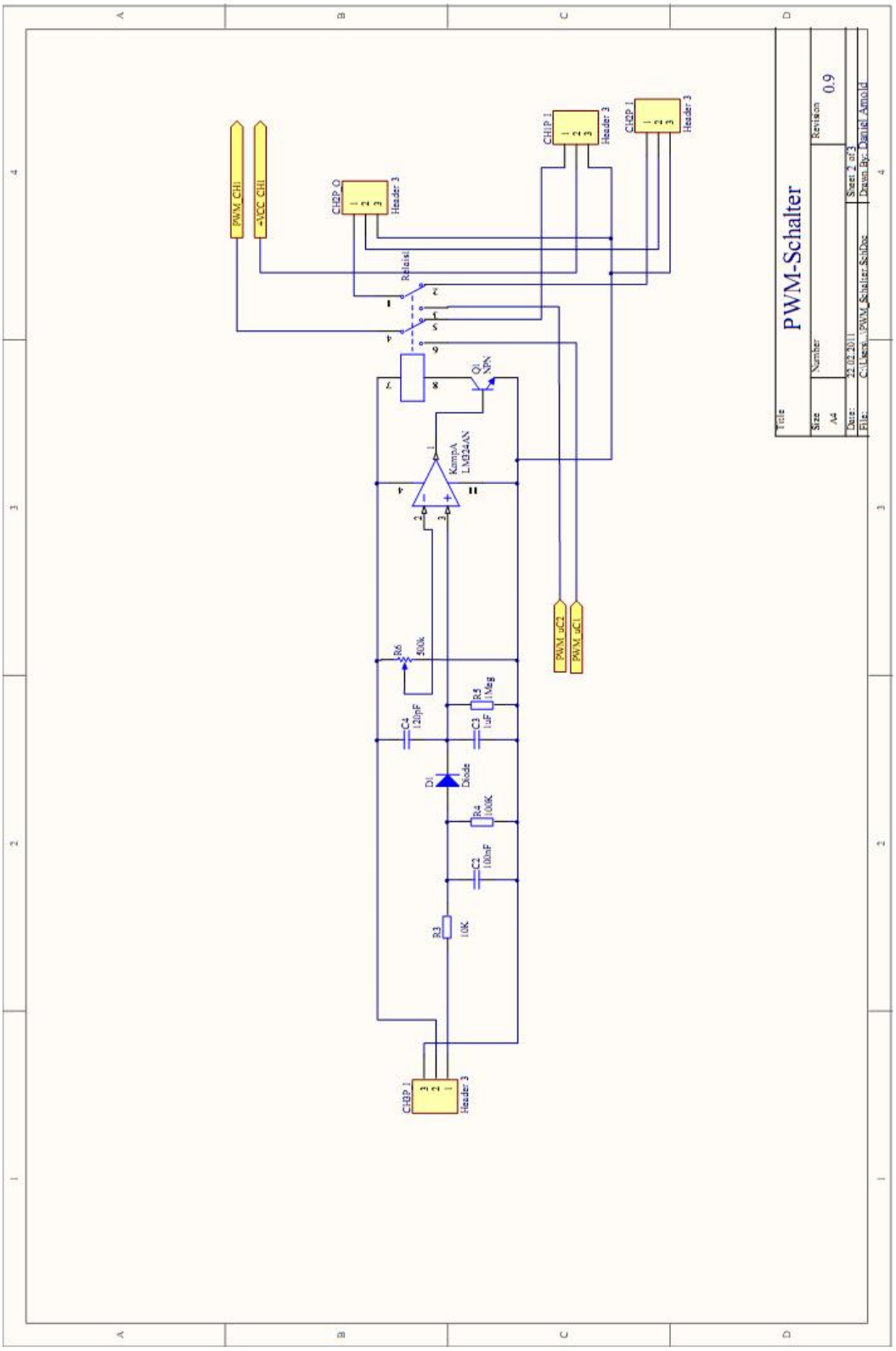

Abbildung B.2: Schaltung des PWM Schalter

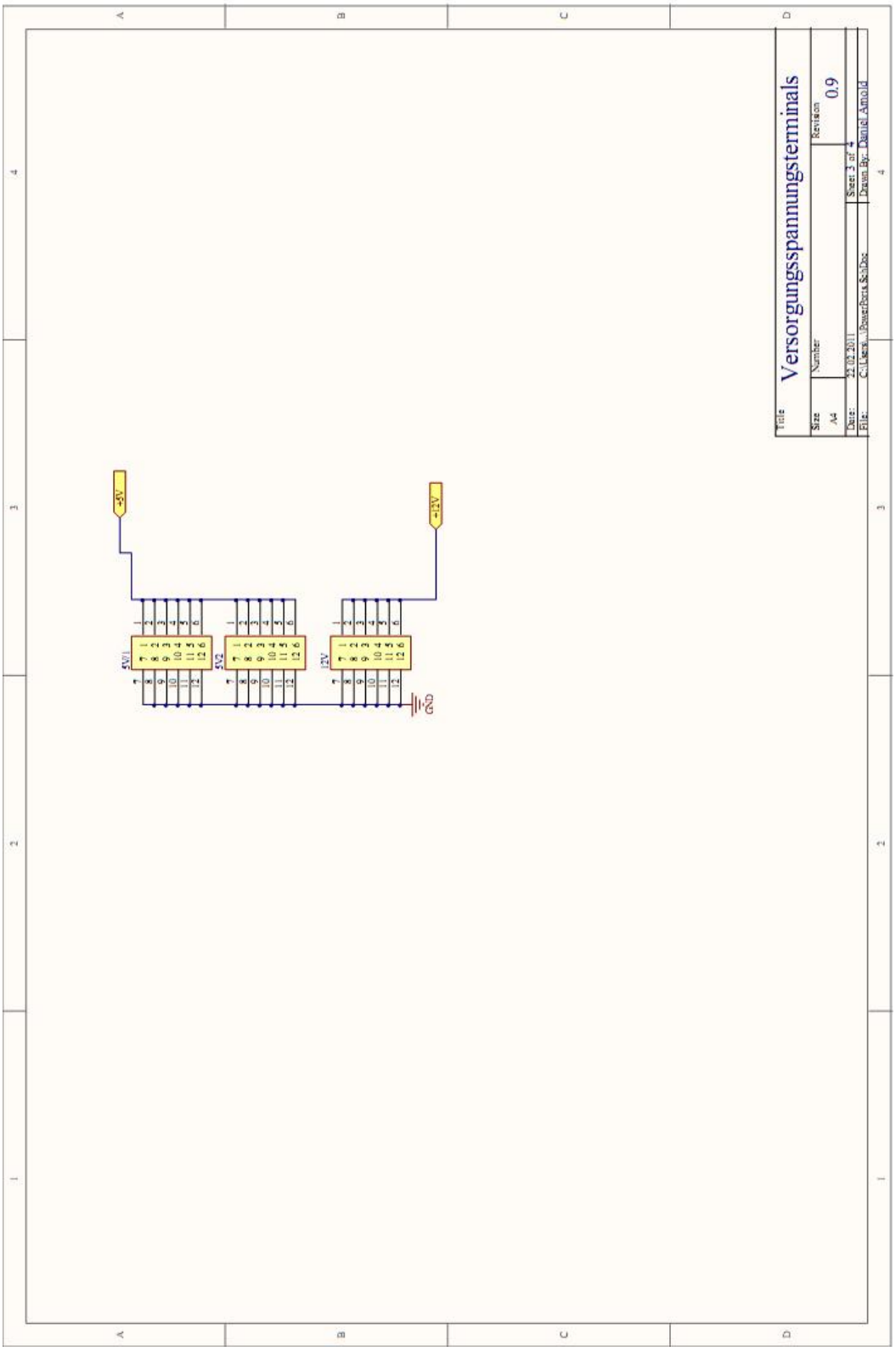

Abbildung B.3: Schaltung der Versorgungsterminals

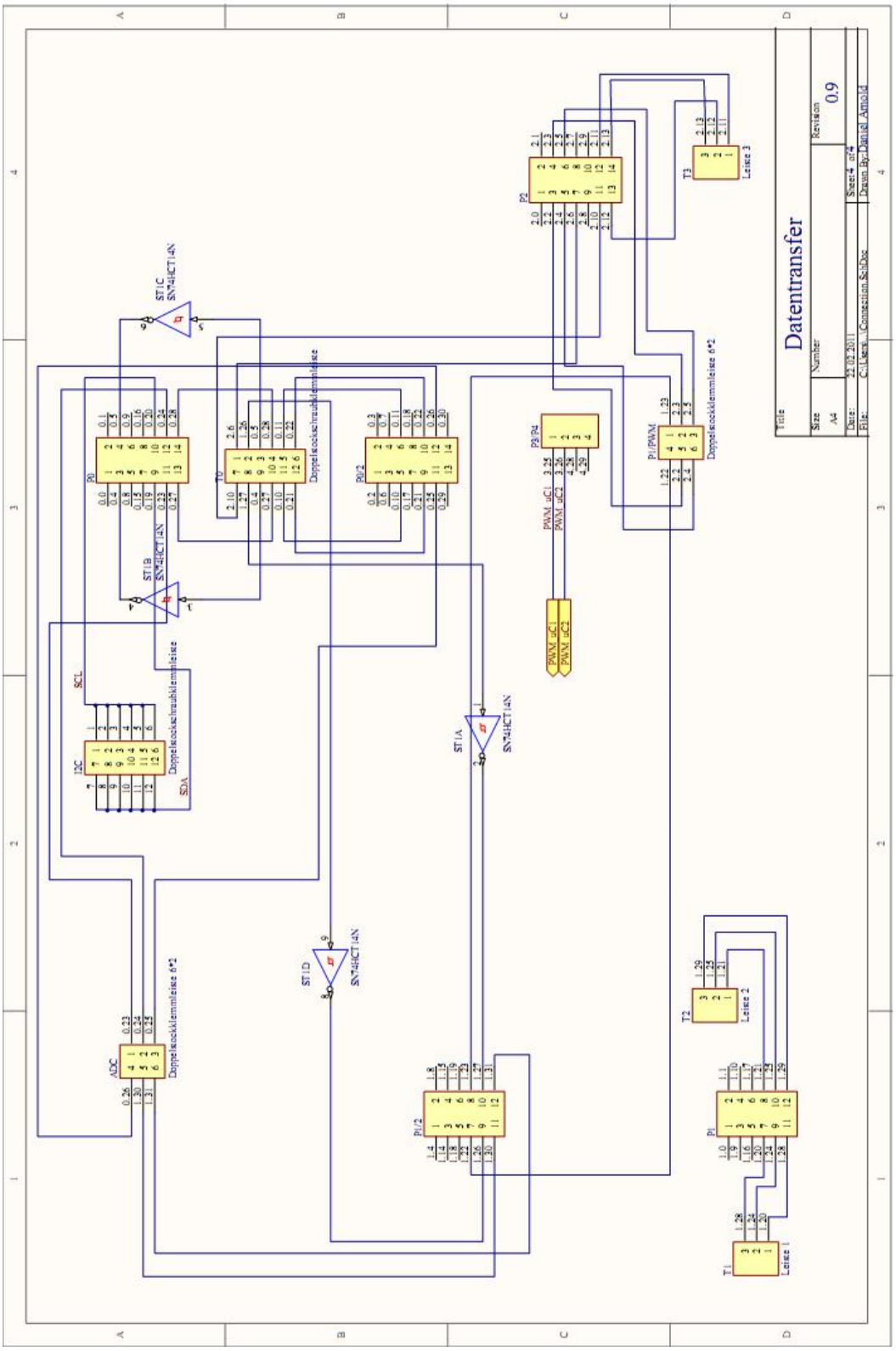

Abbildung B.4: Schaltung der Signalleitungen

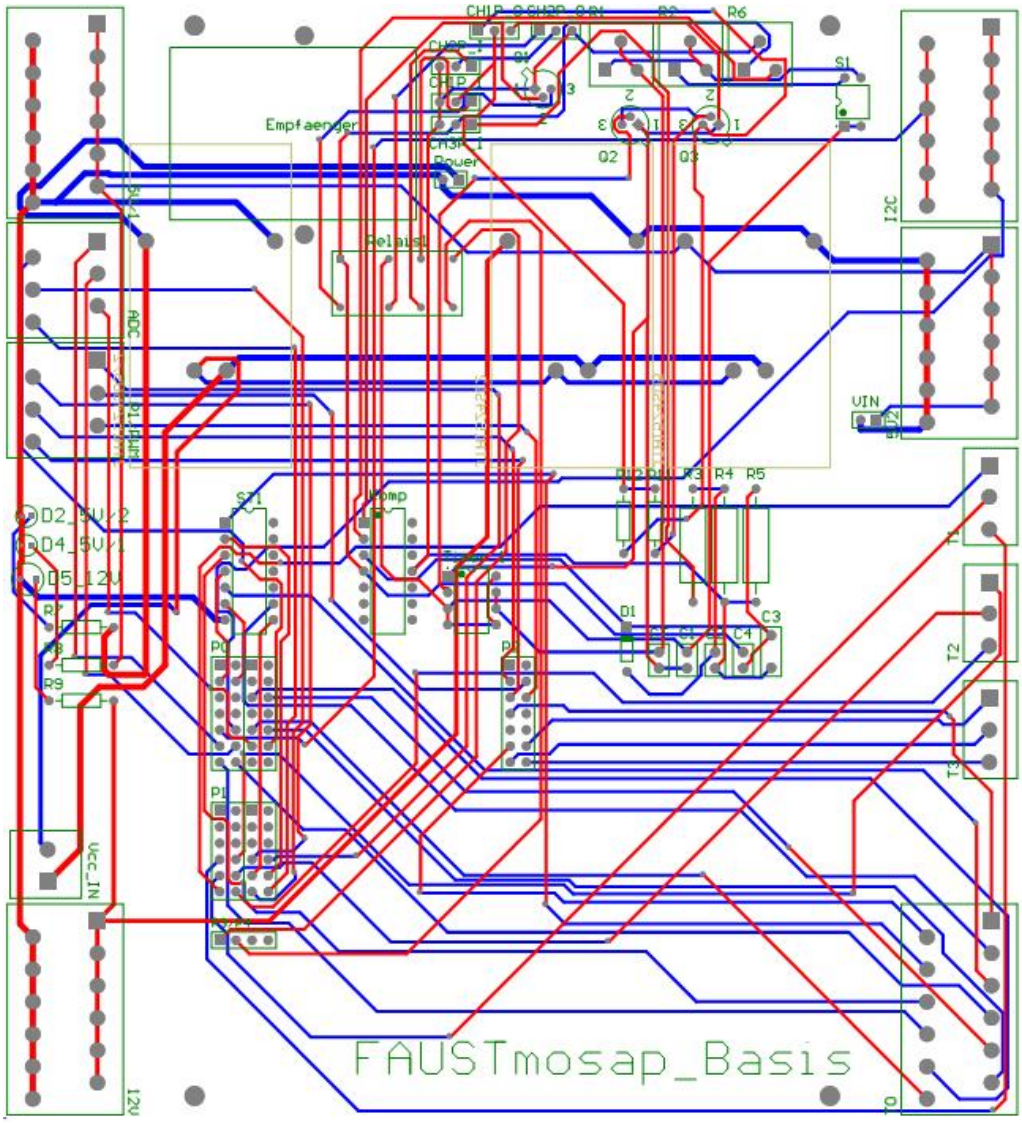

Abbildung B.5: Layout der Basisplatine

Bill of Materials

Bill of Materials For Project [Faust1.PrjPcb] (No PCB Document Selected)

Source Data From: Faust1.PrjPcb
Project: Faust1.PrjPcb
Variant: None

Creation Date: 22.02.2011 11:24:08
Print Date: 40596 40596.47519

Footprint	Comment	LibRef	Designator	Description	Quantity
Schraubklemmschiene 2*6 Pin	Doppelstockschraubklemmleiste	Doppelstockschraubklemmleiste	5V2, 5V/1, 12V, I2C, T0		5
Schraubklemmschiene 2*3 Pin	Doppelstockklemmleiste 6*2	Doppelstockklemmleiste 6*2	ADC, P1/PWM		2
CAPR2.54-5.1x3.2	Cap	Cap	C1, C2, C4, CT	Capacitor	4
CAPR5.08-7.8x3.2	Cap	Cap	C3	Capacitor	1
HDR1X3	Header 3	Header 3	CH1P_I, CH1P_O, CH2P_I, CH2P_O, CH3P_I	Header, 3-Pin	5
DIO7.1-3.9x1.9	Diode	Diode	D1	Default Diode	1
LED_3mm	LED0	LED0	D2_5V/2, D4_5V/1	Typical INFRARED GaAs LED	2
LED_5mm	LED0	LED0	D5_12V	Typical INFRARED GaAs LED	1
FP-DC/DC JTH-Series	DC/DC JTH-Series	DC/DC JTH-Series	JTH1524S05, JTH1524S05/2, JTH1524S12		3
N14A	LM324AN	LM324AN	Komp	Low Power Quad Operational Amplifier	1
HDR2X7	Header 7X2	Header 7X2	P0, P0/2, P2	Header, 7-Pin, Dual row	3
HDR2X6	Header 6X2	Header 6X2	P1, P1/2	Header, 6-Pin, Dual row	2
HDR1X4	Header 4	Header 4	P3/P4	Header, 4-Pin	1
HDR1X2	Header 2	Header 2	Power, VIN	Header, 2-Pin	2
CAN-3/D5.9	NPN	NPN	Q1	NPN Bipolar Transistor	1
CAN-3/D5.9	BC107	2N3904	Q2, Q3	NPN General Purpose Amplifier	2
Bourns_Poti_3310Y	RPot	RPot	R1, R2, R8	Potentiometer	3
AXIAL-0.7	Res2	Res2	R3, R4, R5	Resistor	3
AXIAL-0.4	Res2	Res2	R7, R8, R9, R11, R12	Resistor	5
Relais(OMRON)	Relay-DPDT	Relay-DPDT	Relais1	Dual-Pole Dual-Throw Relay	1
DIP-4	SW DIP-2	SW DIP-2	S1	DIP Switch, 2 Position, SPST	1
N014	SN74HCT14N	SN74HCT14N	ST1	Hex Schmitt-Trigger Inverter	1
Schraubleiste_einfach	Leiste 1	Header 3	T1	Header, 3-Pin	1
Schraubleiste_einfach	Leiste 2	Header 3	T2	Header, 3-Pin	1
Schraubleiste_einfach	Leiste 3	Header 3	T3	Header, 3-Pin	1
N08E	LMC555CN	LMC555CN	Timer 1	CMOS Timer	1
Schraubklemme 2_Pol	Header 2	Header 2	Vcc_IN	Header, 2-Pin	1
					55

Approved	Notes	

Abbildung B.6: Bestückungsliste der Basisplatine

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 28. Februar 2011	
Ort, Datum	Unterschrift