TECHNISCHE UNIVERSITÄT DORTMUND
FACHBEREICH INFORMATIK

PG 524

RAPTOR - Room Adaptive Precisely Topologically Orienting Robot

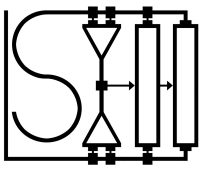
Endbericht

Sommersemester 2008

INTERNE BERICHTE
INTERNAL REPORTS

Lehrstuhl XII (Embedded System Design)
Fakultät für Informatik
Technische Universität Dortmund

Betreuer:
Olivera Jovanovic
Robert Pyka



GERMANY · D-44221 DORTMUND

Endbericht

Projektgruppe 524

Room Adaptive Precisely Topologically Orienting Robot – RAPTOR

Sommersemester 2008

Teilnehmer:
Ender Ayalp, Sariette Bille, Omar Bousbiba, Daniel Etges, Igor Ionov, Selma Jabour, Oliver Jokiel, Dominik Kopczynski, Markus Künne, Yi Lin, Samir Rachidi, Daniel Richter

Betreuer:
Olivera Jovanovic, Robert Pyka

Technische Universität Dortmund
Fachbereich Informatik
LSXII
Prof. Dr. Peter Marwedel

Inhaltsverzeichnis

1 Einleitung

Die Projektgruppe 524, auch RAPTOR (Room Adaptive Precisely Topologically Orienting Robot) genannt, entwickelt mit dem c't Bot als Grundlage, einen Staubsaugerroboter ohne Saugmechanismus. Motiviert von den Unzulänglichkeiten bestehender Roboter strebt die Projektgruppe einige Verbesserungen, siehe Abschnitt 1.1 auf dieser Seite, an. Auf das Ziel und die Vorgehensweise wird in den Abschnitten 1.2 und 1.3 näher eingegangen.

1.1 Motivation

Schon immer war es ein Traum der Menschen künstliche Wesen zu erzeugen. Diese sollten ihnen mühsame, beziehungsweise gefährliche Arbeiten abnehmen oder zur Unterhaltung der Menschen dienen. Inzwischen gibt es eine Vielzahl von Robotern für die unterschiedlichsten Einsatzbereiche. Die bekanntesten Roboter sind die Industrieroboter. Diese übernehmen unterschiedliche Aufgaben, wie beispielsweise schweißen und montieren, in Anwendungsbereichen wie zum Beispiel Schiffsbau und Elektrotechnik. In der Forschung und Unterhaltungsindustrie werden humanoide Roboter entwickelt, dessen Herausforderung vor allem darin liegt, dem Roboter das Gehen auf zwei Beinen beizubringen. Desweiteren gibt es unterschiedliche mobile Roboter, die für den Einsatz im Weltraum oder als Überwachungs- und Serviceroboter dienen. In der Computer-Fachzeitschrift c't erschien 2006 das Projekt c't-Bot [8]. In diesem Projekt wurde ein Roboter vorgestellt, der sich einfach nachbauen läßt. Mit Hilfe des Simulators `c't sim` läßt sich sein Verhalten virtuell testen. Er besitzt die Fähigkeit seinen Standort zu bestimmen und erstellt eine Karte seiner Umgebung. Mit seinem Maussensor wertet er seine Ausrichtung aus. Durch seinen Radencoder lässt sich die zurückgelegte Entfernung bestimmen. Mit Hilfe seiner Infrarotsensoren erkennt er Hindernisse und erstellt eine metrische Rasterkarte seiner Umgebung. Zusätzlich besitzt der Roboter eine Abgrunddetektion. Der Funktionsumfang des Projektes c't-Bot umfasst das eigenständige Erkunden der Umgebung, Slalom fahren und das Einsammeln von Golfbällen, die an eine bestimmte Position gebracht werden. Mit der entsprechenden Programmierung ist der Roboter allerdings auch in der Lage komplexere Tätigkeiten auszuführen.

Zu den beliebtesten Serviceroboter gehören die Staubsaugerroboter. Je nach Ausstattung und Saugleistung gibt es sie in unterschiedlichen Preisklassen zu kaufen. Die Roboter werden immer leistungsfähiger und mittlerweile statten die Hersteller diese auch mit einem Treppensensorsystem aus, das verhindert, dass der Roboter Treppen herunterfällt. Weiterhin fahren die meisten Roboter ihre Ladestation eigenständig an, wenn der Akku leer ist. Einige Firmen bieten ferner eine Absaugstation zur Entleerung des Filters an. Zur Hinderniserkennung dienen entweder Drucksensoren am Gehäuse oder Ultraschallsensoren.

Ein Ersatz für herkömmliche Staubsauger sind diese Roboter noch nicht. Deshalb gibt es einige Ansätze, die heutigen Staubsaugerroboter um wichtige Eigenschaften zu erweitern:

- Die Analyse des Verschmutzungsgrades
- Das Erkennen von Menschen und Haustieren
- Das Säubern von Ecken und Rändern

Um die Erweiterungen zu realisieren, können die Mechanik, bestimmte Sensoren und Algorithmen vom c't-Bot übernommen werden. Für eine genügende Rechenleistung sollte ein Embedded Linux Board benutzt werden. Dieses ist eine vorgefertigte standardisierte Steuereinheit auf der ein eingebettetes Linux oder RTLinux zum Einsatz kommt.

1.2 Ziel

Die Projektgruppe soll in zwei Semestern einen autonomen, mobilen Roboter entwickeln, der als Staubsaugerroboter in einer Wohnung eingesetzt werden könnte. Allerdings braucht der Roboter keinen Saugmechanismus zu enthalten, da die heutige Technologie in derartig kleiner Bauform noch keine ausreichende Reinigungsleistung erreicht. Auf Grund dessen soll die mechanische Konstruktion auf das zur Fortbewegung Notwendige reduziert werden. Zugleich kann damit mehr Zeit in die Softwareentwicklung und Strategieplanung investiert werden.
Der Staubsaugerroboter RAPTOR hat folgende Besonderheiten:

- Der Verschmutzungsgrad des Bodens soll mit Hilfe eines Kameramoduls erkannt werden. Da größtenteils immer dieselben Stellen in einem Raum besonders verschmutzt sind, soll der Roboter immer direkt dorthin fahren. Mit Hilfe dieser Strategie kann so schnell ein gepflegtes Gesamtbild der Wohnung erreicht werden. Der Roboter soll einen Ausschnitt des Bodens fotografieren, anschließend staubsaugen und noch einmal fotografieren um dann durch eine Analyse der Bilder den Verschmutzungsgrad in diesem Abschnitt zu bestimmen. Da der RAPTOR keinen Saugmechanismus enthält, müssen die Teilnehmer der Projektgruppe das Staubsaugen mit anderen Hilfsmitteln übernehmen.

- Menschen und Haustiere sollen mittels Sensoren erkannt werden, damit diese rechtzeitig umfahren werden können und das Staubsaugen nicht als Störung empfunden wird.

- Der Roboter soll eine gründliche Reinigung der Ecken und Ränder von Möbel und Tischen in einem Raum gewährleisten. Dies setzt eine Software voraus, die in der Lage ist Ecken und Ränder zu erkennen. Ferner wird die Hardware benötigt, die in der Lage ist ein nahes Heranfahren an Hindernisse zu ermöglichen, damit auch dort der Schmutz entfernt werden kann.

Die mechanische Realisierung kann vom c't-Bot weitestgehend übernommen werden. Da der Durchmesser des c't-Bots nur 12cm beträgt, müssen seine Komponenten für den RAPTOR angemessen skaliert werden. Anstelle des Mikrocontrollers wird ein Embedded Linux Board benutzt um eine ausreichende Rechenleistung für die vorgestellten Erweiterungen sicherzustellen. Damit steht den Teilnehmern der Projektgruppe eine fertige Entwicklungsumgebung zur Verfügung. Um neben den Infrarotsensoren auch Ultraschallsensoren, Navigations- und Ortungssensoren anschließen zu können, besitzt das Board vielfältige I/O-Anschlüsse. Der Roboter sollte selbstständig, nach einer von der Projektgruppe ausgewählten Strategie, den Raum (beziehungsweise auch mehrere Räume wenn möglich) erkunden und eine Karte erstellen. Außerdem sollte er in der Lage sein sich selbst zu lokalisieren. Zusätzlich soll eine graphische Kontroll- und Steuerungsschnittstelle implementiert werden. Mit Hilfe dieser Schnittstelle werden Parameter, wie beispielsweise der Akkuladestand, die Sensorwerte und das aktuelle Kamerabild, des Roboters überwacht und die graphisch aufbereitete Darstellung des Raummodells zugänglich gemacht. Die Kommunikation mit dem Roboter erfolgt funkbasiert. Das ausgewählte Embedded Linux Board stellt eine WLAN-kompatible Schnittstelle zur Verfügung.

1.2.1 Minimalziele

Für den Erwerb des Leistungsnachweises sind die folgenden Minimalziele zwingend erforderlich:

- Hardwareentwurf des Roboters inklusive Auswahl der zu verwendenden Bauelemente. Beim Entwurf ist zu gewährleisten, dass sich der Roboter, unter Nutzung der Navigationsvorrichtung, zielsicher in einem Raum bewegen kann.

- Implementierung einer Low-Level Steuerungssoftware zur Bewegungssteuerung des Roboters und Positions- beziehungsweise Distanzmessung. Der Roboter soll damit vordefinierte Punkte im Raum anfahren können.

- Implementierung einer High-Level Steuerungssoftware zur Navigation. Dabei soll ein Modell des Raums zur Verfügung stehen und eine festgelegte Mission implementiert sein. Eine Mission könnte beispielsweise sein, den Raum möglichst vollständig abzufahren.

- Demonstration der Funktionsfähigkeit der erstellten Hard- und Software im praktischen Einsatz.

1.2.2 Erweiterungsmöglichkeiten

Der Wettbewerb `Patente Studierende` [23] prämiert, im September 2008 in Düsseldorf, Haushaltsroboter die dem Menschen sowohl behilflich, als auch individuell einsetzbar sind. Dabei sollte er nicht allein die Funktion eines Staubsaugers ausüben. Diese Tatsache

hat die Projektgruppe RAPTOR bereits animiert über Erweiterungsmöglichkeiten nachzudenken: Der Abschnitt 4.9.1 beschreibt die Idee und Realisierung.
Die Bewertungskriterien der Jury, die sich im Wesentlichen aus Vertretern von Unternehmen und Universitäten zusammensetzt, sind:

- Originalität in der Problemlösung
- Neuheit gegenüber dem Stand der Technik
- Anwendernutzen und Verwertbarkeit im Markt
- Effizienz (Praktischer Nutzen zum technischen Aufwand)
- Sorgfalt in der inhaltlichen und formalen Ausführung
- Präsentation

1.3 Vorgehensweise

Die Projektgruppe 524 findet im Wintersemester 2007/2008 und im Sommersemester 2008, mit zwölf studentischen Teilnehmern am Lehrstuhl 12 der Fakultät Informatik der Technischen Universität Dortmund, statt. Betreut wird das Projekt von den wissenschaftlichen Mitarbeitern Olivera Jovanovic und Robert Pyka des Lehrstuhls 12 für Technische Informatik und Eingebettete Systeme.
Unter der URL `http://ls12-www.cs.tu-dortmund.de/raptor` ist der Internetauftritt der Projektgruppe zu finden.
Das Projekt befasst sich sowohl mit Hard- als auch mit Software. Kapitel 2.1 beschreibt die Auswahl der Hardwareelemente, die die Projektgruppe in den Anfangswochen traf. Nachdem die Grundlagen geklärt waren, konnte die Realisierung des Prototyps eines autonomen Staubsaugerroboters beginnen. Nachzulesen ist dies im Kapitel 3.
Da sich die Projektgruppe bei der Herstellung ihres Staubsaugerroboters nah an dem Projekt des c't Bots orientiert, gibt es einiges an vorgefertigter Software. In den ersten Wochen verschaffte sich die Projektgruppe einen Überblick über vorhandene Software zu diesem Themengebiet. Das Kapitel 2.2 beschreibt die Vorüberlegungen. Die letztendliche Realisierung der Software wird in dem Kapitel 4 geklärt.
Die ausbleibenden Programmierarbeiten und Hardwareergänzungen werden im Kapitel 5 festgehalten.

2 Grundlagen

2.1 Hardware

Zu Beginn der Projektgruppe wurde die entsprechende Hardware für den autonomen Roboter ausgesucht. Dazu gehört die Auswahl des passenden Linux Board, der Ultraschallsensoren, der Kontaktsensoren, der Lichtsensoren, des Maussensors, des Kompasssensors, der Getriebemotoren, des Mikrocontrollers und der Kamera. Die folgenden Abschnitte befassen sich mit diesem Thema.

2.1.1 Linux Board

Der Staubsaugerroboter soll in der fertigen Version eine hohe Anzahl an teilweise sehr komplexen Funktionalitäten, wie z.B. Stauberkennung, Navigation, Objekterkennung etc., ausführen können. Die benötigten Daten aus der Umgebung werden durch verschiedene Sensoren bereitgestellt, anhand derer der Roboter dann seine nächsten Aktionen planen kann. Eine Bearbeitung von Sensordaten erfordert ein hohes Maß an Rechenleistung, da Methoden aus den Bereichen der Bild- und Signalverarbeitung benötigt werden. Zudem soll der Roboter einen Großteil der Berechnungen in Echtzeit durchführen können, da er z.B. möglichst rechtzeitig Objekte erkennen und ggf. ausweichen soll. Hierfür ist neben einer effektiven Programmierung auch leistungsfähige Hardware erforderlich. Zusätzlich muss eine Vielzahl an verschiedensten Ein- und Ausgabegeräten, wie z.B. Sensoren, Kameras usw., unterstützt werden können.

Im Gegenzug zur benötigten Rechenleistung und Flexibilität stehen jedoch diverse Einschränkungen, die beim Roboter berücksichtigt werden müssen. Diese bestehen vor allem in der begrenzten Platz- und Energiekapazität, welche nicht jede beliebige Hardware, wie sie bei normalen Desktop-PCs verwendet werden, zulässt.

Aus Sicht der Software wird die Programmierung des Roboters extern auf Desktop-PCs durchgeführt, da diese immer noch eine erhöhte Rechenleistung im Vergleich zum Roboter aufweisen und der Programmierkomfort wegen der Vielzahl verschiedener Software höher einzuschätzen ist. Es wäre aufwendiger, direkt auf dem Roboter zu programmieren, zumal mehrere Personen für verschiedene Softwarebereiche nicht mehr gleichzeitig an der Programmierung arbeiten können. Der fertig kompilierte Programmcode wird auf dem Roboter ausgelagert und lokal auf dem Gerät ausgeführt. Ohne ein Betriebssystem, welches die passende Programmierumgebung unterstützt, ist dies jedoch nicht ohne weiteres möglich. Ein geeignetes Betriebssystem für den Roboter und eingebettete Systeme allgemein wäre Linux: Es ist als OpenSource-Software verfügbar, beansprucht wenig Speicherplatz, läuft

bei korrekter Konfiguration sehr stabil und schnell, erlaubt viele Freiheiten in der Treiberprogrammierung für verschiedene Peripherie und stellt viele, bereits vorgefertigte Tools und Treiber zur Verfügung.

Die geeignete Rechnerhardware für eingebettete Systeme, zu denen der Roboter sicherlich zu zählen ist, wurde in den sog. Gumstix-Boards der amerikanischen Firma gumstix inc gefunden. Diese zeichnen sich durch eine hohe Rechenleistung bei kompakter Größe (die einheitliche Größe der Rechner-Boards beträgt 80mm x 20mm), und geringer Betriebsspannung (3.3V - 5V) aus. Um eine Vielzahl unterschiedlicher Peripherie zu unterstützen, können die Rechner-Boards mit einer breiten Palette an verschiedenen Erweiterungs-Boards (welche eine ähnliche Größe wie die Hauptplatinen aufweisen) kombiniert werden. Insgesamt gibt es drei verschiedene Gruppen von gumstix-Rechner-Boards (basix, connex und verdex). Diese unterscheiden sich hauptsächlich in der Rechenleistung, der Speichergröße und der Anzahl an kompatiblen Erweiterungs-Boards.

Rechner-Board: gumstix verdex XL6P

Die Projektgruppe entschied sich für das Modell **gumstix verdex XL6P**, welches bis zu zwei Erweiterungs-Boards unterstützt und über die höchste Rechenleistung aller bestellbaren Rechner-Boards verfügt. Zwar ist das gumstix verdex XL6P das teuerste, verfügbare Board (um 70 Dollar teurer im Vergleich zum billigsten Modell der Produktgruppe basix), dafür ist es in der Lage, allen vom Roboter gestellten Anforderungen gerecht zu werden. Wichtig ist insbesondere die Möglichkeit, bis zu zwei Erweiterungs-Boards an das gumstix verdex XL6 anzuschließen, wodurch die Anzahl möglicher Ein-/Ausgabegeräte deutlich erhöht wird. Zudem gibt es einige spezielle Erweiterungs-Boards (u.a. das console-vx und das netwifimicroSD), welche nur mit den Rechner-Boards der verdex-Klasse kompatibel sind.

Abbildung 2.1: gumstix verdex XL6P [12]

Technische Details:

- Prozessor: Marvell PXA270 XScale-Prozessor mit 600 MHZ
- Speicher: 128MB RAM, 32MB Flash-Speicher
- Features: USB-host Signale, CCD-camera Signale
- Anschlüsse für Erweiterungs-Boards:
  - 60-pin Hirose connector
  - 120-pin MOLEX connector
  - 24-pin Flex ribbon
- Größe: 80mm x 20mm
- Gewicht: 8g

Das Board enthält bei Bestellung bereits einen vorgefertigten Linux-Kernel 2.6 auf dem Flash-Speicher, welcher nach Belieben verändert oder neu beschrieben werden kann. Dieses gilt jedoch nicht für den Bootloader. Sollte dieser beschädigt oder fehlerhaft eingestellt werden, so muss das Produkt zurück zum Hersteller gesendet werden, damit der Speicher (gegen Aufpreis) neu geflasht werden kann.

Mit diesem Board ist der Bedarf nach einer hohe Rechenleistung inkl. genügendem Arbeitspeicher erfüllt. Nun muss sichergestellt werden, dass eine ausreichende Menge an Anschlüssen für die verschiedenen Ein-/Ausgabegeräte vorhanden ist. Erreicht wird dies durch die folgenden zwei Erweiterungs-Boards:

Erweiterungs-Board: Console-vx

Das Erweiterungs-Board console-vx wird an den 60-pin Hirose connector des gumstix verdex XL6P angeschlossen.

Abbildung 2.2: console-vx [12]

Technische Details:

- 3 x RS-232 Ports mit miniDIN8 Verbindungssteckplätzen
- USB mini-B Anschluß mit USB-Host Signalen
- 18-bit LCD Anschluß
- Serielle Funktionsports (alternative GPIO Verbindungen) auf 0.100 Zoll kleinen Löchern
- Verbindungsanschluß : 60-pin I/O header
- Größe: 80mm x 34mm
- Betriebsspannung: 3.5V - 5V

Ursprünglich wurde es bestellt, damit die PG-Teilnehmer erste Arbeitserfahrungen mit dem Gumstix sammeln können. Hierfür wurden drei Pins eines RS-232 Ports verkabelt, mit einer Stromquelle verbunden und mit Hilfe eines seriellen Anschlusses an den Hauptrechner der Projektgruppe im Hardware-Labor angeschlossen. Bei ausreichender Versorgungsspannung wird das Gumstix automatisch vom Rechner erkannt und kann nun mit Hilfe des Programms Minicom konfiguriert werden. Die ersten Versuche, Daten vom Hauptrechner auf das Gumstix zu flashen, verliefen ohne größere Komplikationen. Leider erweist sich die Geschwindigkeit des seriellen Anschlusses als sehr langsam, weshalb im Laufe der Zeit nach anderen, schnelleren Möglichkeiten zur Datenübertragung auf das Gumstix gesucht wurden. Das Erweiterungs-Board verfügt zusätzlich über GPIO-Schnittstellen, welche den Anschluß weiterer, einfacher Peripherie wie z.B. Sensoren unterstützen. Erwähnenswert ist die auch Unterstützung von USB-Host Signalen, welche dem console-vx erlauben, selber als Host z.B. für Netzwerke zu agieren. Diese Möglichkeit wurde für den Roboter allerdings nicht in Anspruch genommen.

Erweiterungs-Board: netwifimicroSD EU

Das Erweiterungs-Board netwifimicroSD-EU wird an den 120-pin MOLEX connector des gumstix verdex XL6P angeschlossen.

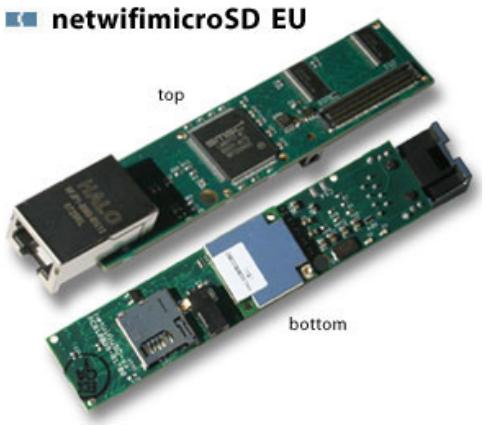


Abbildung 2.3: netwifimicroSD EU [12]

Technische Details:

- 10/100baseT ethernet
- microSD-Slot
- 802.11(b) und 802.11(g) WLAN-Verbindungsprotokoll
- Verbindungsanschluß : 120-pin Busheader
- Betriebsspannung: 3.5V - 6V
- Größe : 93mm x 20mm

Dieses Board wird primär zur drahtlosen Kommunikation zwischen dem Roboter und den Hauptrechnern eingesetzt. Wichtige Eigenschaften sind die Ethernetfähigkeit und die Unterstützung von WLAN (802.11(b) und 802.11(g) Kommunikationsnormen). Es gibt insgesamt zwei Versionen dieses Board, wobei eine Version einen EU-inkompatiblen WLAN-Standard unterstützt. Zusätzlich ist auf dem Board ein Slot für microSD-Karten vorzufinden. Zwar besitzt das Gumstix bereits 32 MB Flash und 128 MB Arbeitsspeicher, doch gerade bei speicherintensiven Prozessen, wie in der Bildbearbeitung, stößt man hier schnell an mögliche Kapazitätsgrenzen. Deshalb erweist sich eine Aufrüstung des (Platz-)Speichers als sinnvoll. Ein weiterer Vorteil bei der Benutzung einer Speicherkarte besteht darin, dass die Datenübertragung im Gegensatz zum konventionellen Flashen mittels serieller Schnittstelle wesentlich schneller erfolgen kann.

Erfahrungen mit dem Gumstix

Aufgrund seiner Empfindlichkeit ist Vorsicht im Umgang mit dem Gumstix geboten. Die PG-Teilnehmer dürfen z.B. das Board nur mit angelegtem Erdungsband berühren, damit keine statische Entladung innerhalb des Boards stattfindet und somit evtl. Beschädigungen in der Hardware hervorgerufen werden.

Während der Inbetriebnahme des Gumstix traten diverse Probleme auf:

- Falls eine zu geringe Versorgungspannung angelegt wird (i.d.R. kleiner als 4-5V), wird der Linux-Kernel des Gumstix nicht vollständig gebootet, weshalb immer eine ausreichend hohe Versorgungsspannung anzulegen ist. Jedoch darf sie aus Sicherheitsgründen wiederum nicht zu hoch ausfallen.

- Die WLAN-Funktionalität vom netwifimicroSD-EU kann nicht korrekt benutzt werden. Vermutet werden neben Treiberproblemen auch evtl. Hardware-Fehler. Problematisch ist hierbei auch die Tatsache, dass Support und Erfahrungsberichte seitens Hersteller und Privatanwendern im Internet nicht vorzufinden waren.

- Ein I^2C-Anschluß war auf allen drei Boards nicht direkt vorzufinden, wobei die Schnittstellen in der Produktbeschreibung vorhanden sind. Dieser Umstand konnte aber durch Neulöten von Pins des console-vx Boards behoben werden.

- Die gumstix-Boards besitzen keinerlei vorgefertigte Schutzschaltungen gegen erhöhte, beabsichtigt oder unbeabsichtigt auftretende Ströme und Spannungen. Gerade bei einem komplexen Projekt, wie der Staubsaugerroboter es mit der Kombination aus Sensoren, Aktoren und externen Stromquellen ist, führt dies immer wieder zu massiven Problemen und Gefahren für die gumxstix-Boards, im schlimmmsten Fall zur vollständigen Zerstörung der Boards!

2.1.2 Ultraschallsensoren

Um Hindernisse rechtzeitig erkennen zu können, hat sich die Projektgruppe entschieden, sechs Ultraschallsensoren für ihren Staubsaugerroboter zu benutzen. Die Sensoren befinden sich an den folgenden Positionen: vorne, vorne rechts, vorne links, rechts, links und hinten. Eine schematische Darstellung bietet die Abbildung A.3.

Der Schall des Sensors breitet sich im Raum als Welle aus. Als Medium dient die Luft. Die Ausbreitungsgeschwindigkeit hängt von der Dichte der Luft und somit vom Ort ab. Dabei wird die Schallwelle zusätzlich an Stellen mit einer Dichteänderung teilweise reflektiert. Durch jede Dichteänderung wird ein Echo zur Ultraschallquelle reflektiert, welches als Signal wieder auffangbar und auswertbar ist. Die Zeitdifferenz zwischen aussenden und empfangen des Signals gibt bei bekannter Mediumsdichte Aufschluss über die Distanz zwischen Grenzfläche und Sensor. Somit ist eine Längenmessung entlang der Schallausbreitung möglich.

Die Firma Devantech [9] hat eine Serie von kleinen Ultraschallsensoren entwickelt. Ein vergleichbares Modul selbst zu bauen wäre ungenauer und teurer. Die Projektgruppe hat

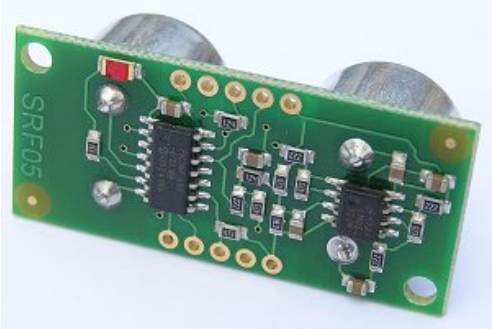


Abbildung 2.4: Der Ultraschallsensor SRF05 [9]

sich für das Modell SRF05 entschieden, siehe Abbildung 2.4. Dieses besitzt eine Reichweite von bis zu 4 Meter. Des Weiteren hat er einen zweiten Modus, in dem sowohl der Start der Messung, als auch das Ergebnis über die gleiche Leitung übertragen werden (siehe Abschnitt über Modus 2). Der Ultraschallsensor kann folglich über einen einzigen Port (Pin) gesteuert werden. Diesen Vorteil verwendet die Projektgruppe für ihre sechs Ultraschallsensoren. Im Folgenden werden die beiden Modi und deren Pinbelegung näher erläutert. Ferner wird auf die Distanzberechnung und die Richtwirkung des Sensors näher eingegangen.

Modus 1

Wie in Abbildung 2.5 zu sehen ist, benutzt der erste Modus seperate Trigger- und Echo-Pins, das heißt, es müssen zwei Ports mit dem Mikrocontroller verbunden werden. Der SRF05 hat einen internen Pull-Up Widerstand, so dass der Modus-Pin unverbunden bleiben kann. Ein kurzer High-Impuls, auf dem Trigger Input, von mindestens 10 Mikrosekunden löst den Messvorgang aus. Anschließend überwacht der Mikrocontroller den Echo-Pin. Die Länge des High-Impulses ist proportional zur Entfernung des Objektes. Die gemessene Zeit, gemessen wird in der Zeiteinheit μS, wird durch die Konstante 58 dividiert, wodurch die Entfernung in Zentimetern erhalten wird.

Modus 2

Der zweite Modus besitzt einen gemeinsamen Pin für das Trigger- und Echo-Signal, siehe Abbildung 2.6. Er hat den Vorteil, dass nur ein Pin des Controllerports belegt ist, denn die Auslösung und das Messen des Ergebnisses erfolgt über den gleichen Port. Um diesen Modus zu aktivieren, muss der Modus-Pin am Ultraschallsensor mit Low (GND) verbunden werden.

Nach Senden eines Auslösesignales, ein mindestens $10\mu S$ langes High-Signal, muss der Port auf Input umgeschaltet werden. Danach wird das High-Signal des Ultraschallsensors abgewartet. Für diese Umschaltung stehen mindestens $700\mu S$ zur Verfügung. Analog zur oberen Beschreibung des ersten Modus, ist die Länge des zurückgelieferten High-Impulses

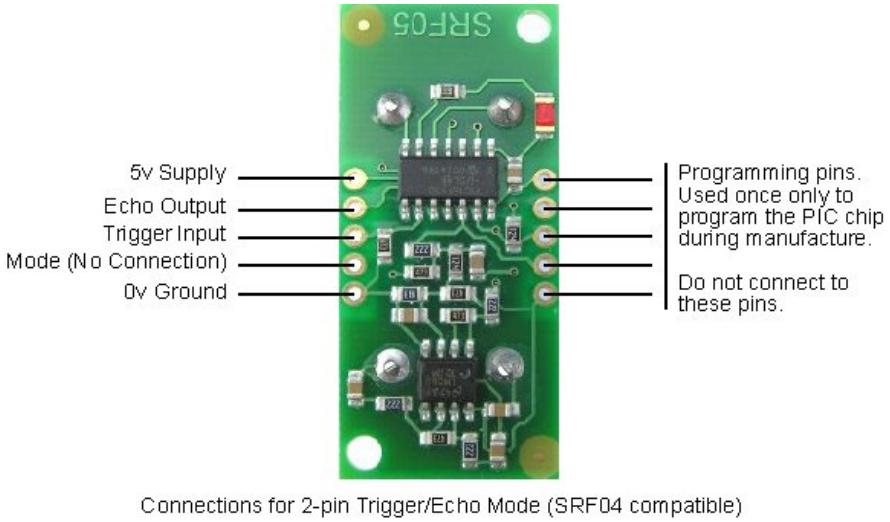


Abbildung 2.5: Anschlüsse für seperaten Trigger- und Echo-Pin [9]

proportional zur Entfernung des Objektes. Indem die gemessene Zeit durch die Konstante 58 dividiert wird, wird die Entfernung in Zentimetern erhalten.

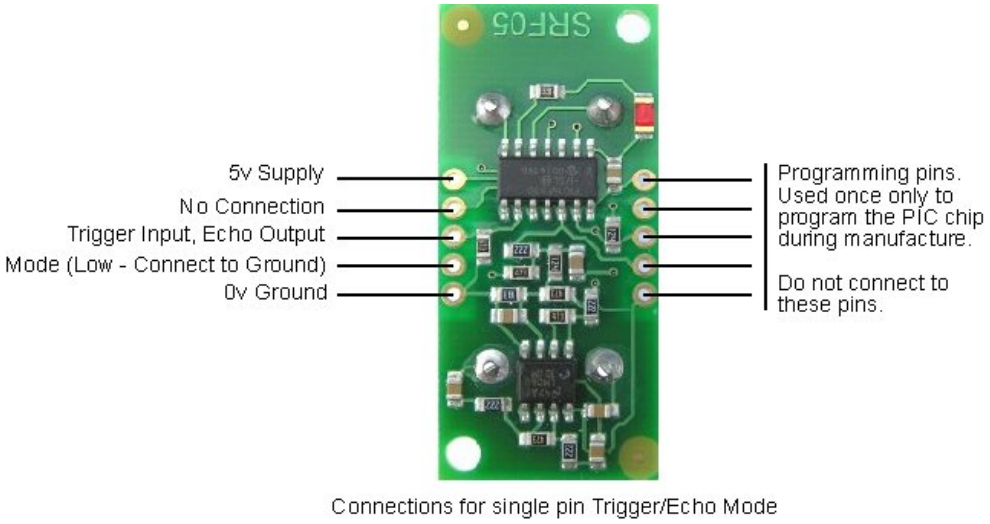


Abbildung 2.6: Anschlüsse für gemeinsamen Trigger- und Echo-Pin [9]

Distanzen berechnen

Die Zeitdiagramme des SRF05 werden für den jeweiligen Modus in den Abbildungen 2.7 und 2.8 gezeigt. Wie zuvor in den Abschnitten über Modus 1 und 2 erwähnt, reicht ein mindestens $10\mu S$ langes High-Signal aus, um den Messvorgang auszulösen. Der SRF05 sendet acht Zyklen hintereinander einen ultrahochfrequenten Ton, bei 40kHz, aus. Während

dieser Zeit hat er einen High-Pegel. Wird ein Echo registriert, wird der Pegel in diesem Fall auf Low gesenkt. Die Länge des High-Pegels ist proportional zur Entfernung des Objekts. Mit einer Zeitmessung lässt sich die Distanz, beispielsweise in Zentimeter, berechnen. Wird kein Echo empfangen, senkt der Ultraschallsensor den Pegel nach 30 Millisekunden auf Low. Bis zu 20 Messungen pro Sekunde sind mit diesem Ultraschallmodul möglich, das heißt alle 50 Millisekunden ist eine Messung möglich. Wichtig ist dabei, dass die Zeit des Echo-Impules möglichst genau ermittelt wird. Je genauer dies geschieht, desto akkurater ist die Entfernungsmessung. Abschließend ist noch zu erwähnen, dass die restlichen fünf Pins, auch Programmier-Pins genannt, ausschliesslich bei der Herstellung dazu dienen, den Flash-Speicher des Chips (PIC16F630) zu programmieren. Das bedeutet, dass keiner dieser Pins verbunden werden darf.

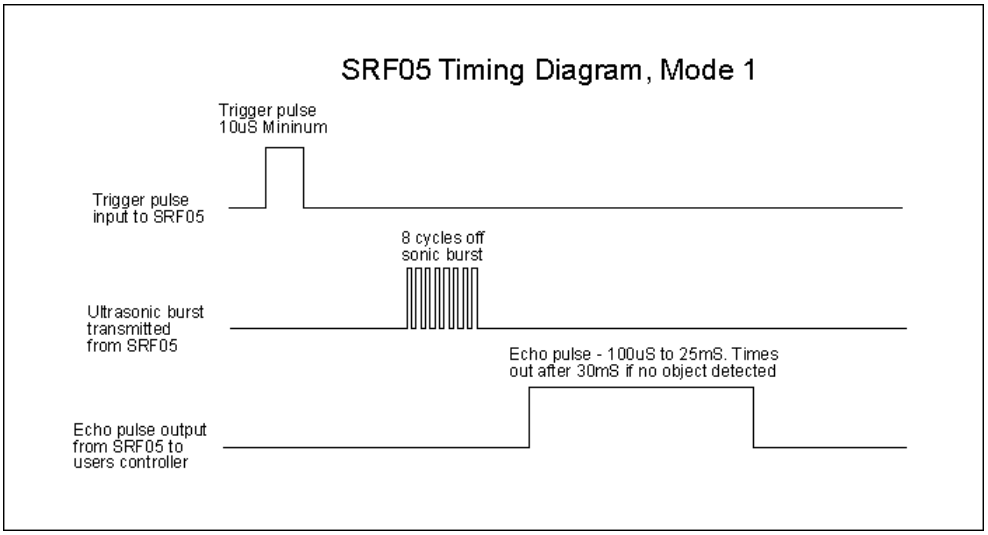


Abbildung 2.7: Zeitdiagramm vom Modus 1 [9]

Richtwirkung des Ultraschallsensors

Die Richtwirkung des SRF05 ist dem Datenblatt des Herstellers entnommen und ist in der Grafik 2.9 zu sehen. Die Ultraschallwellen breiten sich konisch aus. Die Richtung ist nicht änderbar.

2.1.3 Kontaktsensoren

Die Ultraschallsensoren können ein Hindernis erst ab einem Mindestabstand von 10cm messen. Zudem haben sie eine Fehlerrate. In einigen Szenarien kann es dazu kommen, dass die Sensoren ein Hindernis nicht erkennen. Für diesen Fall hat die Projektgruppe sich entschieden den Staubsaugerroboter mit fünf zusätzlichen Mikroschaltern, auch Schnappschalter genannt, zu versehen. Die Entscheidung fiel auf den Schalter MBF5B der Firma

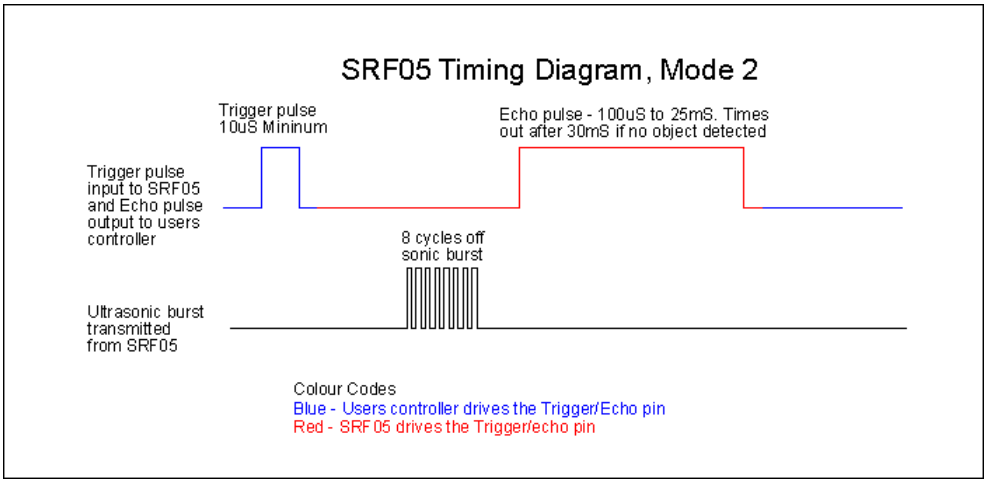


Abbildung 2.8: Zeitdiagramm vom Modus 2 [9]

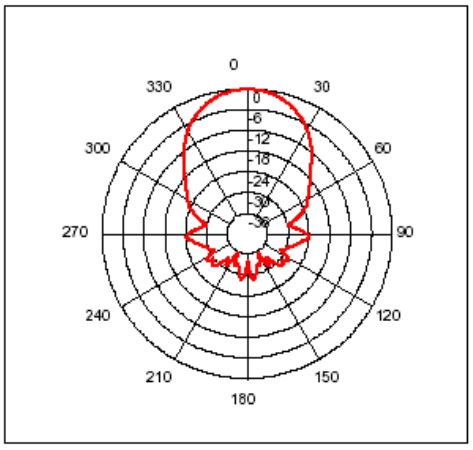


Abbildung 2.9: Richtwirkung des Sensors [9]

Hartmann [13], siehe Abbildung 2.10. Die Abbildung 2.11 zeigt eine Zeichnung des Schalters mit seinen Anschlüssen. Der Kontakt ist in Ruhestellung zwischen COM und NC oder NO geschlossen. Durch Druck auf den Betätiger wird der Schaltvorgang ausgelöst. Dabei wird von Kontakt NC auf NO umgeschaltet. Der Kontaktabstand ist kleiner als 3mm.

Abbildung 2.10: Der Mikroschalter MBF5B [13]

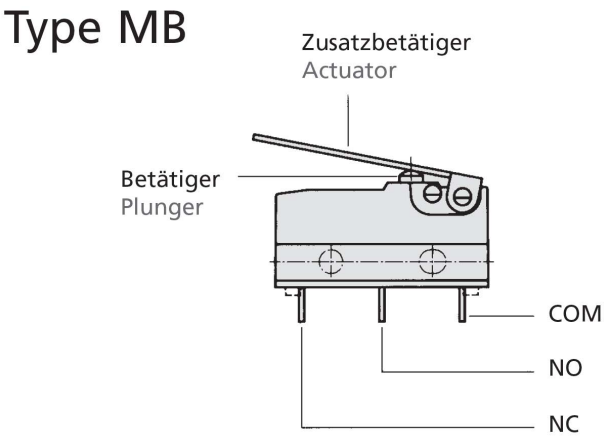


Abbildung 2.11: Anschlüsse des Kontaktsensors [13]

2.1.4 Lichtsensoren

Ein Lichtsensor ändert bei der Einwirkung von Licht seine elektrischen Eigenschaften. Der Reflex Optokoppler CNY70, siehe Abbildung 2.12, der Firma Vishay [31] wird für den Staubsaugerroboter vielfältig eingesetzt. Wie in den folgenden Abschnitten beschrieben, dient er der Projektgruppe als Radencoder und Abgrundsensor. In der Zeichnung 2.13 ist dargestellt, dass der CNY70 in seinem würfelförmigen Gehäuse eine Infrarot-LED als Sender und einen Infrarot-Fototransistor als Empfänger besitzt. Damit kann auf kurze Entfernung das reflektierte Licht der IR-LED durch den IR-Fototransistor gemessen werden. Die Menge des reflektierten Lichts bestimmt, ob der Fototransistor mehr oder weniger leitend wird. Die Reichweite des Sensors beträgt nur wenige Millimeter. Störungen durch Fremdlichteinstrahlung sind eine mögliche Fehlerursache.

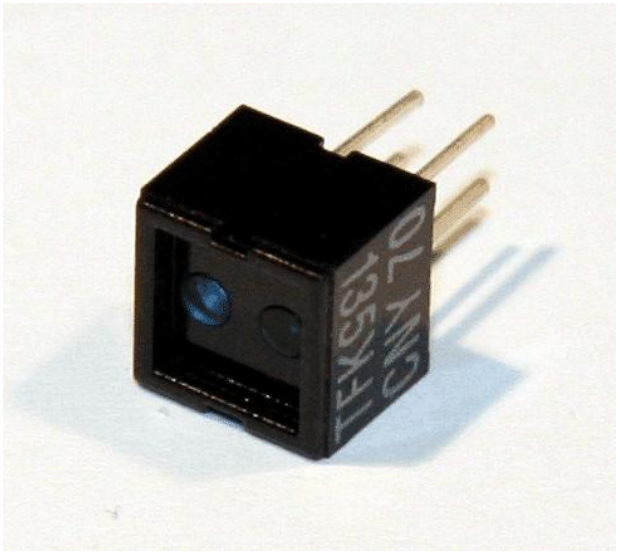


Abbildung 2.12: Der Lichtsensor CNY70 [31]

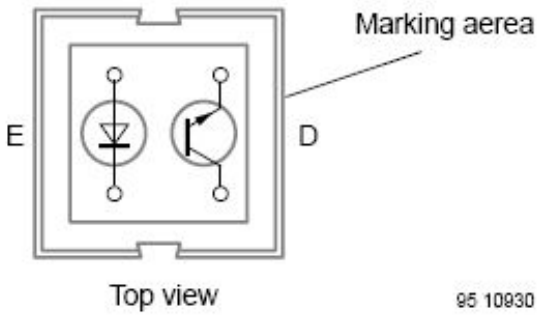


Abbildung 2.13: Steckerbelegung des CNY70 [31]

Radencoder

Mit Hilfe seiner beiden Radencoder kann der RAPTOR die zurückgelegte Wegstrecke bestimmen, eine bestimmte Strecke geradeaus fahren, sowie sich um einen bestimmten Winkel drehen. Auf der Innenseite der Räder sind Encoderscheiben aufgeklebt. Auf ihnen sind insgesamt 30 schwarze und 30 weiße Streifen im Wechsel kreisförmig angeordnet. Ein Radencoder besteht aus einem Lichtsensor, dem CNY70, der so am Roboter angebracht ist, dass er auf die Streifen der Encoderscheiben zeigt. Der Lichtsensor erzeugt analoge Signale. Sie werden für die weitere Verarbeitung durch den Mikrocontroller mit Hilfe eines Schmitt-Triggers in digitale Signale umgesetzt. Ein Schmitt-Trigger erzeugt aus einem analogen Eingangssignal eindeutige digitale Schaltzustände. Aufgrund dessen reicht es, wenn der Mikrocontroller nur noch die Flankenwechsel von High nach Low zählt.

Abgrundsensor

Abgründe, wie beispielsweise Treppen und Tischkanten, erkennt der RAPTOR mit Hilfe von drei CNY70, die als Abgrunddetektoren fungieren. Sie befinden sich an der unteren Front des Staubsaugers. Wie bei dem Maussensor darf der Abstand zwischen Lichtsensor und Boden nicht mehr als 3mm betragen. Problematisch können beispielsweise Linien am Boden sein, die der Abgrundsensor fälschlicherweise als Abgrund erkennt.

2.1.5 Maussensor

Da die Radencoder und die Ansteuerung der Motoren keine hinreichende Sicherheit bei der Erkennung der Länge abgefahrener Strecken gewährleisten können, hat die Projektgruppe sich für einen Maussensor als zusätzliche Navigationshilfe entschieden. Optische Maussensoren sind heutzutage feste Bestandteile moderner Mäuse. Im Gegensatz zu Mäusen mit Rollkugel, in der die Bewegung des Benutzers mechanisch erfasst wird, erfolgt hier eine optische Erfassung der Bewegung. Prinzipiell ist ein optischer Maussensor nichts anderes als eine kleine Kamera, die ständig Bilder der erfassten Fläche aufnimmt und Änderungen mathematisch ermittelt. Somit kann eine Positionsänderung präzise berechnet werden.

Das Project c't-Bot [8] benutzt einen Maussensor, der auch sehr gut die Anforderungen für einen Staubsaugerroboter erfüllt, und wurde somit von der Projektgruppe weitestgehend übernommen. Als Sensorchip wird der ADNS-2610 der Firma Agilent Technologies [3] eingesetzt. Dieser wird gemeinsam mit einer LED zur Beleuchtung des Bodens, einer Linse und einigen weiteren Bauteilen wie Quarz zur Taktgebung auf einer eigenen Platine untergebracht und von unten an den Staubsaugerroboter befestigt. Die Verbindung zu dem Mikrocontroller wird über die beiden Pins SDIO und SCLK realisiert.

Funktionsweise

Der ADNS-2610 besteht aus einem Bilderfassungssystem, einem DSP zur Bildverarbeitung und einem seriellen 2-Pin Bus Interface zur Host-Kommunikation. Der Chip verarbeitet

1512 Bilder pro Sekunde bei einer Sensorgröße von 18x18 Pixeln und einer Auflösung von 64 Graustufen. Die Auflösung beträgt 400 dpi.

Das Funktionsprinzip nennt sich 'Optisches Navigationssystem' und beruht darauf, dass der Bildsensor ständig mikroskopisch kleine Bilder von der Oberfläche aufnimmt, und der DSP diese Bilder ständig miteinander vergleicht. Aus den unterschiedlichen Bildern wird dann die Richtung und die zurückgelegte Strecke errechnet. Als Ausgabe stehen die aktuellen X- und Y-Werte in Registern, die über den seriellen Port abgefragt werden können.

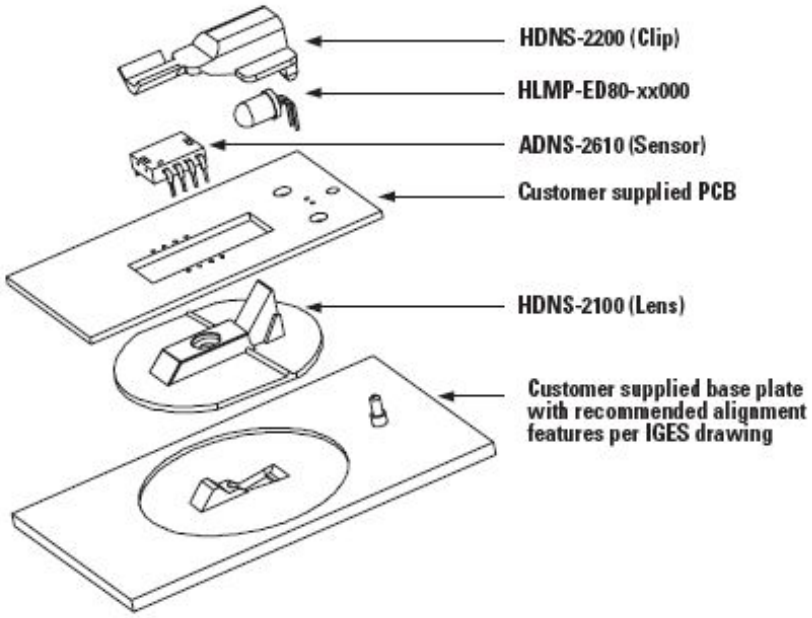


Abbildung 2.14: Bestandteile des optischen Maussensors [3]

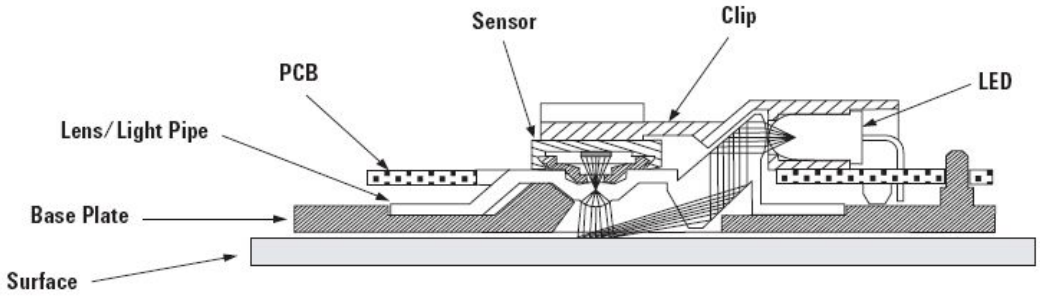


Abbildung 2.15: Aufbau und Funktionsweise der Optik des optischen Maussensors [3]

2.1.6 Kompasssensor

Die Verwendung einer Karte für die Navigation setzt eine zuverlässige Positionsbestimmung des Roboters im Raum voraus. Ein Maussensor liefert zwar die abgefahrene Strecke, kann aber insbesondere auf glatten oder spiegelnden Oberflächen ungenau sein. Aus diesem Grund hat sich die Projektgruppe entschieden, die Fahrtrichtung des Roboters zusätzlich über einen Kompasssensor zu bestimmen.

Ein passendes Modul mit der Bezeichnung CMPS03 wurde bei der Firma Devantech [9] gefunden. Im Vergleich zu Kompasssensoren anderer Hersteller zeichnet sich der CMPS03 durch geringen Preis, fertigem Aufbau des Moduls und Einfachheit des Auslesens der Daten aus. Dieses Modul ist speziell für die Bedürfnisse von Robotern gestaltet worden. Es ermittelt anhand des Erdmagnetfeldes die genaue Himmelsrichtung in 0,1 Grad Schritten. Das Modul arbeitet auch in geschlossenen Räumen korrekt, was insbesondere für den Staubsaugerroboter wichtig ist. Die Ausgabe der Ergebnisse kann entweder als PWM-Signal, oder über einen I^2C-Bus abgefragt werden. Im Folgenden werden die beiden Betriebsarten und die Pinbelegung des Kompasses näher erläutert.

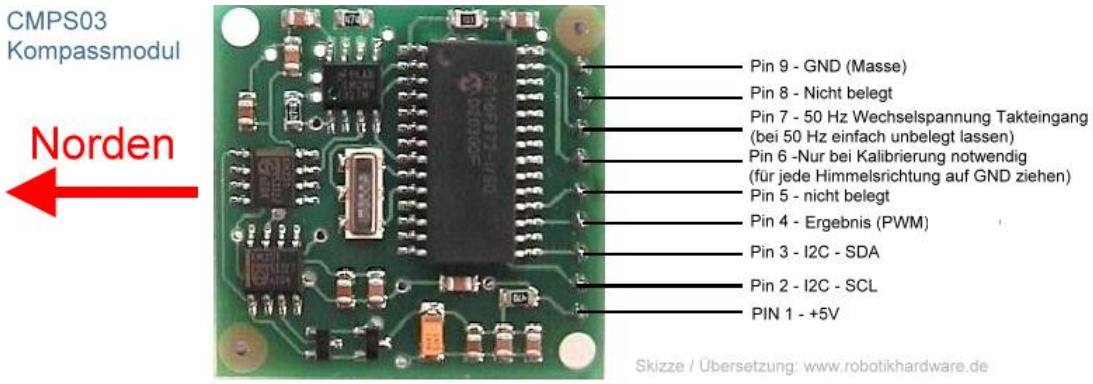


Abbildung 2.16: Pinbelegung des Kompasssensors [9]

Modus 1

Das PWM-Signal an Pin 4 gibt den Kompasswert 0 bis 359,9 Grad in Form eines High Impules aus. Die Länge des High-Impules kann zwischen 1 Millisekunde und 36,99 Millisekunden liegen. Demnach entsprechen 0,1 Millisekunden (100μ s) einem Grad. Es muss somit nur die Signallänge gemessen werden, danach ist eine Umrechnung sehr einfach.

Modus 2

Der I^2C-Bus wird an den Pins 2 und 3 (SDA und SCL) angeschlossen. CMPS03 besitzt keine Pullup-Widerstände wie sie für den I^2C-Bus notwendig sind. Das Masterboard am I^2C-Bus sollte demnach diese I^2C-Leitungen mit ca. 5k bis 47k Widerständen mit +5V

verbinden. Wird der I^2C-Bus bei CMPS03 nicht genutzt, so sollten Pin 2 und 3 über einen Widerstand (ca. 10k bis 47k) mit +5V verbunden sein, damit Störungen vermieden werden.

Die Kommunikation über den I^2C-Bus erfolgt, wie bei fast allen Modulen, über verschiedene Register. Es stehen folgende Register zur Verfügung:

Register	*Funktion*
0	Software Version (Firmware Version)
1	Kompasswert - Ein Byte 0 bis 255 entspricht 0 bis 359,9 Grad
2,3	Kompasswert als Word (also 2 Byte / Low and High) Der Wert von 0 bis 3599 entspricht 0 bis 359,9 Grad.
4,5	Interne Testregister - werden nur vom Hersteller genutzt
6,7	Interne Testregister - werden nur vom Hersteller genutzt
8,9	Interne Testregister - werden nur vom Hersteller genutzt
10,11	Interne Testregister - werden nur vom Hersteller genutzt
12	Unbenutzt, liefert immer 0 zurück
13	Unbenutzt, liefert immer 0 zurück
14	Unbenutzt, liefert undefinierten Wert zurück
15	255 :startet die Kalibrierung (Justierung der Richtungen) 0 : Beendet Kalibierung. Werte werden im EEPROM gespeichert

Tabelle 2.1: Registerbelegung des Kompassmoduls

Die Slave-Adresse ist fest auf Hex C0 (also Dezimal 192) eingestellt. Wie bei I^2C üblich, wird immer erst die SLAVE Adresse und dann das abzurufende Register als Byte versendet. Anschließend erfolgt ein Lesezugriff, in dem die SLAVE Adresse +1 gesendet und danach 1 oder 2 Bytes abgerufen werden. CMPS03 unterstützt die Standard I^2C Taktraten von 100 und 400 KHz. Wird mehr als 100 KHz genutzt, sollte nach Schreiben der Registeradresse eine kurze Pause von ca. 50μ s eingebaut sein. Nachfolgend wird die Pinbelegung des Kompasssensors beschrieben.

Pin 1 und 9 - Stromversorgung

Das Kompassmodul benötigt lediglich 5V am Pin 1 und verbraucht im Durchschnitt ca. 15 mA. Pin 9 wird mit Ground verbunden.

Pin 6 - Kalibrierung

Damit das Modul in jeder Umgebung möglichst genau arbeitet, ist eine einmalige Justierung (Kalibrierung) empfehlenswert. Die Kalibrierung erfolgt indem das Modul exakt waagerecht in alle vier Himmelsrichtungen ausgerichtet wird und jedes Mal dabei Pin 6 kurz mit GND verbundet wird. Das Ergebnis wird intern gespeichert und bleibt auch dann erhalten, wenn keine Spannung anliegt. Alternativ kann auch über den I^2C-Bus die Kalibrierung durchgeführt werden. Dort funktioniert es auf die gleiche Weise, mit dem Unterschied, dass statt des Tasterdrucks eine 255 in das Register 15 geschrieben wird.

Pin 7 - Wechselspannungsfelder

Die Abtastrate der Kompassposition erfolgt bei unbeschaltetem Pin 7 intern gewöhnlich mit 60 Hz. In Umgebungen mit starken Wechselspannungsfeldern kann es wegen des dadurch erzeugten Magnetfeldes zu Ungenauigkeiten kommen. In diesem Fall kann es günstig sein die Netzfrequenz über diesen Pin mit dem Modul zu synchronisieren. Dazu muss das passende 50 Hz Taktsignal mit TTL Pegel angelegt werden. Dies kann das Ergebnis in solch schwierigen Umgebungen verbessern. In der Regel kann der Pin unbeschaltet gelassen werden.

2.1.7 Getriebemotoren

Bei der Entwicklung eines mobilen Roboters ist es erforderlich irgendeine Art von Motor einzusetzen. Bei der Auswahl des richtigen Motors müssen mehrere Kriterien beachtet werden:

- Größe
- Strombedarf
- Steuerbarkeit
- Drehmoment

Die Größe der Motoren müssen, je nach Robotergröße ausgesucht werden, denn die Motoren nehmen auf der einen Seite wichtige Flächenressourcen für andere Bauteile weg. Auf der anderen Seite dürfen die Motoren nicht zu schwach sein, dadurch wäre der Roboter nicht mehr in der Lage sich fortzubewegen.

Klare Schrittweiten und/oder präzise Geschwindigkeiten ermöglichen es den zurückgelegten Weg relativ exakt zu bestimmen, sowie die Fahrt grader Strecken zu realisieren. Der Motor sollte auch präzise halten können, wenn die Abstandsensoren einen Abgrund erfassen, um das Runterfallen des Roboters zu vermeiden.

Das Drehmoment der Motoren bestimmt die Fortbewegung. Ein geringer Drehmoment in Kombination mit hoher Last führen zu einer Überlastung des Motors bzw. Motortreibers. Es existieren Motortreiber (L293D), die bei Überhitzung abschalten. Dadurch besteht aber das Problem, dass der Roboter ständig stehen bleibt.

Für fahrende Roboter kommen nur zwei Motoren in Frage: Getriebemotoren oder Schrittmotoren. Die Projektgruppe hat sich für Getriebemotoren entschieden, da sie zu den einfachsten, kleinsten und universellsten Motoren zählen. Diese Motoren eignen sich besonders gut zur Konstruktion von flinken wie auch langsamen Robotern. Es können aber nur Roboter bis zu 5kg Gewicht angetrieben werden. Die Motorsteuerung für den Getriebemotor ist relativ einfach. Es gibt zwei Anschlüsse, welche mit jeweils unterschiedlicher Polarität belegt werden, um den Motor laufen zu lassen. Je nach Richtung des Stromflusses, dreht sich der Motor entweder vorwärts oder rückwärts. Die Getriebemotoren können abhängig von

ihrer Belastung und dem Untergrund (Fliesen, Teppich, Laminat, etc.), in ihrer tatsächlichen Umlaufgeschwindigkeit voneinander abweichen. Um sicherzustellen, dass zwei verschiedene Getriebemotoren gleich oft drehen, müssen externe Sensoren eingesetzt werden, die die Radumdrehung ständig überwachen und die Information an einen Mikrocontroller senden. Trotz dieser Nachteile werden Getriebemotoren eingesetzt, auch wenn die Fahrt gerader Strecken zu einer Herausforderung wird. Um zu verstehen warum der Schrittmotor nicht in Frage kommt, muss zunächst das Prinzip des Schrittmotors verstanden werden.

Die Schrittmotoren sind grundlegend anders konzipiert als die Getriebemotoren. Der Rotor besteht meist aus einem Permanentmagneten, der durch Anlegen einer Spannung an den Spulen bewegt wird. Jeder Motor enthält zwei Spulen, welche wiederum jeweils zwei Anschlüsse besitzen, und bei angelegter Spannung den Motor um einen Schritt fortbewegen. Die Fortbewegung des Rotors pro Schritt ist relativ präzise, (die Toleranz liegt höchsten bei 5 Prozent). Der Schrittwinkel gibt dabei an, wieviele Schritte notwendig sind, damit das Rad einmal um die Achse dreht, z.B werden bei einem Schrittwinkel von 1.8 Grad 200 Schritte benötigt. Der Schrittwinkel liegt, je nach Bauart des Motors, in der Regel zwischen 1.8 Grad und 18 Grad. Es kann aber über das Getriebe nach belieben verkleinert werden. So ist eine hohe Präzision und Gleichläufigkeit verschiedener Motoren erreichbar. Der Haltemoment eines Motors ist etwa gleich der Größe des Drehmomentes. Der Schrittmotor ermöglicht es mittels der Motordrehschrittanzahl eine präzise Angesteuerung, d.h. der Motor muss z.B. nicht kurzgeschlossen werden, um zu bremsen.

Im Vergleich zu den Getriebemotoren ist die Ansteuerung der Schrittmotoren aufwendiger. Der Schrittmotor benötigt vier Anschlüsse damit der Rotor einen Schritt ausführt. Außerdem müssen spezielle Folgen von Polaritätsmustern abgearbeitet werden. Die Anschaffungskosten sind viel grösser als beim Getriebemotor, denn zwei Getriebemotoren können an einem Motortreiber (z.B. L293D) angeschlossen werden, während jeder Schrittmotor einen eigenen Treiber beansprucht. Im Projekt hat sich herausgestellt, dass die präzise Ansteuerung, die der Schrittmotor gewährleistet, nicht in diesem Umfang benötigt wird, da mittels anderer Sensoren ständig der aktuelle Standort des Roboters neu berechnet und die Fahrt anschließend fortgesetzt wird. Außerdem besteht die Gefahr, dass der Schrittmotor Schritte verlieren kann. Es ergibt sich das selbe Problem wie beim Getriebemotor und es müsste ein zusätzlicher Sensor eingebaut werden, um den Effekt wieder auszugleichen. Der Schrittmotor verliert auch mit steigender Drehzahl an Kraft, also ist der Motortyp nicht für schnelle Roboter geeignet. Ausserdem müsste, wenn ein Schrittmotor eingesetzt wird, ein geeignete Getriebewahl getroffen werden, der die Motorkraft auf die Antriebsräder realisiert. Ohne Getriebe hätten wir folgende Probleme:

- Ein Schrittmotor hätte mit einem Schrittwinkel von 18 Grad eine zu ungenaue Bewegung.

- Die am Rad einwirkende Kraft ist zu groß, der Motor würde zu stark belastet.

Abbildung 2.17: Ein ATMega32 in in PDIP-Bauform

2.1.8 Mikrocontroller

Obwohl der Gumstix selbst auch mit diversen Ein- und Ausgängen für digitale Signale ausgestattet ist, hat sich die Projektgruppe entschieden, die Sensordatenerfassung und Motoransteuerung auf Mikrocontroller auszulagern. Abgesehen davon, dass hierbei erheblich mehr Einfluss bei zeitkritischen Abfragen genommen werden kann, ermöglicht dies eine bessere Trennung zwischen der Hardwareansteuerung und Aufgaben wie Routenplanung und Kartenerfassung. Zudem ist im Fehlerfalle ein Mikrocontroller erheblich günstiger und einfacher auszutauschen als das Linux-Board.

Schon sehr schnell limitierte sich die Auswahl auf die ATMega-Baureihe der Firma Atmel, da die Mikrocontroller dieser Baureihe den Vorteil haben, dass Prozessor und Speicher in einem Chip integriert sind. Zudem werden sie heutzutage in vielen Projekten eingesetzt; von daher existiert für viele Sensoren bereits umfangreiche Dokumentation und gut verständlicher Beispielcode. Nicht zuletzt wird der komplette c't-Bot über einen einzigen ATMega-Mikrocontroller angesteuert, was ein guter Hinweis für die Leistungsfähigkeit dieser Mikrocontroller ist.

Innerhalb dieser Baureihe fiel die Entscheidung auf den ATMega32. Dieser unterstützt eine Taktrate von 16 MHz. Dank seiner guten RISC-Architektur können die meisten Befehle innerhalb eines Taktes abgearbeitet werden und für jeden Einzelbefehl kann eine Ausführungszeit von etwa 62,5 nS angenommen werden - das reicht auch für die schnellsten, (an den Roboter) angeschlossenen Sensoren aus. Weiterhin hat er einen integrierten 32 kB Flash-Speicher. Dieser kann 'In-System' programmiert werden, was beim späteren Debugging ein großer Vorteil sein kann - zusätzliches Aus- und Einbauen des Mikrocontrollers entfällt.

Ein weiteres Argument für den ATMega32 - im Vergleich zu anderen Mikrocontrollern - sind die 32 I/O-Pins (verteilt auf 4 Ports), die auf der einen Seite frei in der Programmierung verwendet werden können, auf der anderen Seite aber auch auf verschiedene, hilfreiche Zusatzfunktionen umgeschaltet werden können. So ist es beispielsweise möglich, an bis zu vier verschiedenen Ausgängen ein PWM-Signal bereit zu stellen, ohne diese Ausgänge ständig softwaregesteuert umschalten zu müssen.

Zunächst wurde im RAPTOR lediglich ein einzelner ATMega32 eingesetzt. Im Laufe der Projektgruppe wuchsen jedoch die Anforderungen an die Hardware, so dass bei einer späteren Version des Controllerboards zwei Mikrocontroller dieses Typs eingesetzt werden. Die Verteilung der Funktionen auf die Mikrocontroller erfolgte nach Anbringung der dazu nötigen Aktoren respektive Sensoren. Beispielsweise werden die Motoren, Radencoder und der Maussensor über den gleichen ATMega32 angesteuert, weil sie alle auf der unteren Platte des RAPTORs liegen. Die Ultraschallsensoren hingegen werden von einem anderen ATMega32 angesteuert, sie sind an der oberen Platine befestigt.
Ursprünglich sollten die beiden Mikrocontroller auf seperaten Boards agieren, welche dann auch direkt an der dazugehörigen RAPTOR-Ebene befestigt werden; dabei wären nur sehr wenige Kabel zur Verbindung der beiden Platten nötig gewesen. Aus Platzgründen wurde diese Idee jedoch verworfen und könnte allenfalls in einer noch späteren Version umgesetzt werden.

Weitere technische Details zum Mikrocontroller selbst befinden sich in der Hardwarebeschreibung und in den Programmierbeispielen.

2.1.9 Kamera

Kameras werden beim Roboter zu zwei Hauptzwecken eingesetzt:

- Durch eine Analyse der Bodenstruktur soll erkannt werden, ob die aktuelle Bodenfläche unter dem Roboter verschmutzt ist und eine Reinung benötigt.
- Eine Analyse von aufgenommenen Bildern aus der unmittelbaren Raumumgebung soll in Verbindung mit anderen Sensordaten dem Roboter in der Objekt- bzw. Hinderniserkennung helfen.
- Ein spezieller Modus des Roboters erlaubt es, Lebewesen (Mensch oder Tier) in einem Raum zu verfolgen. Hierbei wird eine Webcam als Visier benutzt, um das Zielobjekt zu erfassen.

Moderne Webcams, wie sie heutzutage für Videokonferenzen eingesetzt werden, zeichnen sich durch eine kompakte Größe und akzeptabler Bildauflösung aus. Mittlerweile erzielen auch Standardmodelle unter Echtzeitbedingungen ausgezeichnete Ergebnisse (bedingt durch immer leistungsfähigere Hardware von Desktop-PCs heutzutage). Für den Staubsaugerroboter müssen nicht dauerhaft flüssige Videobilder erzeugt werden, stattdessen genügt es, in regelmäßigen Abständen Bilder der Umgebung/Bodenfläche aufzunehmen und diese zu analysieren.

Bei der Benutzung von Webcams für Linuxsysteme besteht das generelle Problem, dass mitgelieferte Treiber bzw. Software für die Kameras grundsätzlich nur für Windows-Systeme konzipiert wurden und der Einsatz der Geräte unter Linux nicht ohne Weiteres möglich ist. Abhilfe schaffen können selbstprogrammierte Treiber von Privatanwendern aus der Linux-Community, allerdings sind diese meist experimenteller Natur und unterstehen keinerlei Qualitätskontrolle durch die Hersteller. Es sei erwähnt, dass ein Treiber unter Linux kompatibel für eine größere Gruppe an Peripherie ist, leider gehöhren die beiden Webcams des Roboters nicht derselben Gruppe an, weshalb zwei verschiedene Treiber ausfindig gemacht und installiert werden mussten.

Aus Sicht der Hardware werden Webcams standardmäßig durch einen USB-Anschluß betrieben, welcher gleichzeitig als Stromquelle und zur Datenübertragung benutzt wird. Unter normalen Desktop-PCs bestehen keinerlei Probleme bei der Stromversorgung, da der Strombedarf durch den Netzbetrieb gedeckt wird. Für den Roboter mit seinen begrenzten Energiekapazitäten sollte die Benutzung der Webcams genau kalkuliert werden, damit es nicht zu Engpässen in der Stromversorgung kommt, zumal andere Sensoren und Aktoren auch Energie benötigen und die Ressourcen dementsprechend verteilt werden müssen. Unter Umständen wäre es sinnvoll, wenn die Stromversorgung der Webcams extern durch seperate Stromleitungen gesichert und der USB-Anschluß lediglich zur Datenübertragung benutzt wird.

Für erste Testbilder in einer Linuxumgebung wurde eine verhältnismäßig billige Webcam der Marke Logitech SweetPea QuickCam Express von der Firma Logitech [18] gekauft. Im Laufe der Projektgruppe stellte sich aber heraus, dass die Auflösung der von der Logitech-Webcam erzeugten Bilder nicht ausreichend hoch ist für eine befriedigende Bodenanaylse, weshalb später zusätzlich eine zweite Kamera (Philips SPC 900 NC) speziell für die Analyse der Bodenstruktur in den Roboter integriert wurde.

Logitech SweetPea Quickcam Express

Abbildung 2.18: Logitech SweetPea QuickCam Express [18]

Laut Herstellerangaben gelten als minimale Systemvoraussetzungen:

- Pentium III 700 MHz oder schneller
- 128 MB Arbeitsspeicher
- 200 MB freier Festplattenspeicher
- 28.8 KBit/s Internetverbindung

Rein faktisch würde das Gumstix-Board des Roboters den Systemvoraussetzungen der Logitech-Webcam nicht genügen, allerdings muss bedacht werden, dass diese Systemvoraussetzungen zumeist für den Privatanwender in Videokonferenzen oder Online-Chats (Echtzeitumgebung!) gelten. Der Roboter hingegen benötigt primitivere Funktionalitäten wie der regelmäßigen Aufnahme von Einzelbildern, somit ist die Rechenleistung des Gumstix-Boards mehr als ausreichend dimensioniert. Die Standardauflösung der Einzelbilder beträgt 320 x 240 Pixel. Theoretisch könnte die Auflösung laut Herstellerangaben durch Grafikinterpolation verdoppelt werden, diese Möglichkeit wurde aus Komplexitätsgründen aber nicht in Anspruch genommen. Die Farbqualität der erzeugten Bilder ist schlechter als wenn sie unter einem Windows-System erzeugt würden. Das ist aber nicht verwunderlich, schließlich wurden sowohl Kamera als auch passende Treiber für den Einsatz unter einem Windows-System optimiert und nicht an die Grafikmodule von Linux angepasst. Interessant ist zu erwähnen, dass unter dem Gumstix Bilder von besserer Qualität erzeugt werden als unter der Linuxumgebung des Hauptrechners. Diese Kamera wird (nach vorne ausgerichtet) oben auf dem Roboter befestigt und visualisiert den vorderen Sichtbereich. In regelmäßigen Abständen wird durch die Software ein neues Bild aufgenommen, welches einerseits extern über das Webinterface gesehen werden kann und andererseits auf Zielobjekte und Hindernisse analysiert werden kann.

Philips SPC 900 NC

Laut Herstellerangaben gelten als minimale Systemvoraussetzungen:

- Microsoft Windows 98SE/2000/Me or XP
- Pentium III 500 MHZ oder schneller
- 128 MB Arbeitspeicher
- 200 MB freier Festplattenspeicher

Diese Kamera wird senkrecht Richtung Boden im unteren Teil des Roboters befestigt. Ihre Aufgabe ist die Erzeugung von Bildern für die Schmutzerkennung. Wichtig in diesem Zusammenhang ist der passende Abstand zwischen Kamera und Boden: Einerseits darf der Abstand nicht zu groß sein, da die erzeugten Bilder unschärfer werden und die Schmutzerkennung beeinträchtigen könnten, andererseits darf der Abstand nicht zu klein sein,

Abbildung 2.19: Philips SPC 900 NC [24]

da bei kleiner erfasster Bodenfläche der Roboter zunehmend mehr fahren und anaylsieren müsste, um den gesamten Boden in einem Raum abzudecken. Zusätzlich muss in der Nähe der Webcam eine passende Beleuchtung angebracht werden, um geeignete Bilder für die Auswertung erhalten zu können.

2.2 Software

In der Robotik gibt es keine universell einsetzbare Software für datenverarbeitende Systeme eines Roboters, wie beispielsweise Sensordatenfusion oder Odometrie. Zwar gibt es Projekte, wie Player-Stage, die eizelne Bereiche, wie allein die Routenplanung oder nur die Bahnplanung der Projektgruppe simulieren, doch wäre ein erfolgreiches und effizientes Zusammenspiel solcher Open-Source-Projekte nicht gewährleistet, da jedes Projekt anders aufgebaut ist und nicht darauf ausgelegt ist mit anderen Projekten verknüpft zu werden. Aus diesen Gründen hat sich die Projektgruppe dazu entschlossen, eigene Software zu entwickeln.

Ein wichtiger Punkt in der Entwicklung war die Festlegung der verwendeten Programmiersprache. Die Entscheidung fiel auf C/C++. Die objekt-orientierte Programmiersprache C++ war eine gute Basis für paralleles Arbeiten an den einzelnen Klassen der Robotersoftware. Mit C ließ sich eine effiziente und hardwarenahe Implementierung für den Mikrocontroller erreichen. Des Weiteren haben sich beide Sprachen in allen Bereichen der Softwareentwicklung etabliert und es existieren umfangreiche Dokumentationen zu dieser Sprache, wie zum Beispiel [16].

Im Folgenden wird beschrieben, in welchen Bereichen vorgefertigte Softwareprojekte, wie Buildroot oder Player-Stage, zum Einsatz kamen und an welchen Stellen eigene Softwarekomponenten, wie die Odometrie oder die Sensordatenfusion erstellt werden mussten. Es wird jedoch nur einen detaillierter Überblick über die verwendete und implementierte

Software vorgestellt. Die Realisierung der Software befindet sich im weiteren Verlauf dieses Dokumentes (siehe Kapitel 4).

2.2.1 Buildroot

Bei Buildroot handelt es sich um eine Skriptsammlung, die bereitgestellte Projekten, wie Kernel, Programme oder Treiber, vereinigt. Sowohl ein Betriebssystem als auch ein Dateisystem lassen sich exakt auf die Anforderungen des eingebetteten Systems mittels Buildroot erstellen. Neben der Bereitstellung des Betriebssystems und des Dateisystems können auch noch Bootloader, und diverse Cross-Compiler von Buildroot bereitgestellt werden.

Buildroot erleichterte es das Potential des Gumstix voll auszuschöpfen. Es konnte auf ein vollwertiges Linux-Betriebssystem zurückgegriffen werden und vorherige Erfahrungen in diesem Bereich waren auf die Arbeit in der Projektgruppe anwendbar. So konnten grundlegende Probleme schnell und effizient gelöst werden.

2.2.2 Player-Stage

Um die ständig weiterentwickelten Programme unter einer standardisierten Umgebung testen zu können, ohne einen einsatzfähigen Roboter zu haben, wurde ein Simulationsprogramm für den Roboter verwendet. Wichtig hierbei war es ein System zu finden, welches alle verwendeten Sensoren unterstützt. Die Wahl fiel auf Player-Stage, da es sich hierbei um ein Open-Source Projekt handelt, welches frei erweiterbar ist und somit zusätzliche Funktionalitäten integriert werden konnten. Des Weiteren wies das Projekt eine solide Benutzergruppe auf, was von großer Bedeutung war, da ihr Wissen genutzt werden konnte, um zum Beispiel spezielle Einstellungen vorzunehmen. Es handelt sich hierbei um eine Kombination aus zwei Systemen. Bei Player handelt es sich um einen Server, der mittels Einbinden von Treibern (beispielsweise Odometrietreiber), die Kommunikation der verschiedenen Komponenten eines Roboters steuert und verwaltet. Um Befehle an den Server schicken zu können oder Sensorwerte empfangen zu können, stellt Player eine TCP Verbindung bereit. Selbst entwickelte Steuerprogramme können so mit Player kommunizieren. Ein spezieller Treiber von Player ist Stage. Dieser dient dazu eine künstliche Umwelt zu erschaffen, in der ein simulierter Roboter bereit gestellt wird, um darauf basierend Sensorwerte zu simulieren und Fahreigenschaften zu testen.

Durch die Kombination beider Komponenten lässt sich Software für den Roboter entwickeln und testen, ohne auf die verwendete Hardware zurückgreifen zu müssen. So können mehrere Teams gleichzeitig Bestandteile der Robotersoftware entwickeln.

Hierzu muss lediglich die Vorbedingung erfüllt werden, eine Schnittstelle zwischen der selbst entwickelten API vom RAPTOR und der von Player-Stage zu implementieren. Dazu wurde eine eigene API definiert, welche die Hardware des Roboters abstrahiert. Durch die erfolgreiche Verwendung von Player-Stage war es möglich, unabhängig vom RAPTOR, Software zu implementieren und testen, zumal der Roboter in der Anfangsphase des Projekts nicht fahrbereit war. Dies ermöglichte es parallel an der Software und an der Hardware zu arbeiten.

2.2.3 Application Programming Interface

Eine API ist eine Programmierschnittstelle zwischen mehreren Softwarebestandteilen. Sie unterstützt viele Aspekte der Softwareentwicklung, so dass zum Beispiel ein Programmierer der höheren Softwareebene keine Kenntnisse zur Ansteuerung von Sensoren und Aktuatoren benötigt. Durch Einbinden der API kann somit der Code intuitiver verwendet und einfacher implementiert werden. So kann die zugrundeliegende Hardware effizient angesprochen werden, was sich als großer Vorteil bei der Fehleranalyse erwiesen hat. Zudem war die API dabei behilflich, die Entwicklung der höheren Funktionen parallel vorzunehmen.

Hauptsächlich dient sie dem Abfragen von Sensorwerten und dem Setzen von Statusvariablen. Daher stellt die Implementierung nur eine Aufbereitung der Daten dar, welche von der Hardware geliefert werden, jedoch keine Interpretation. Beispielsweise stellt die API die Werte des Maussensors zur Verfügung, aber aus den reinen Werten geht nicht die Position des Roboters hervor. Diese und andere Berechnungen, finden sich in der höheren Softwareebene. Somit kann gesagt werden, dass in der Richtung von der Hardware zur Software nur die reinen Rohdaten aufbereitet, aber nicht modifiziert werden. In der anderen Richtung wurde dagegen Wert auf abstraktere Funktionen gelegt. So wird zum Beispiel in den höheren Funktionen kein Wissen darüber nötig sein, wie die genaue Ansteuerung der Motoren realisiert wird. Dies wird von der Implementierung der API umgesetzt.

Diese Kombination bringt zwei große Vorteile mit sich. Es wird die Gefahr verringert, dass auf der Sensorenseite Informationen verloren gehen oder verfälscht werden, da die Auswertung von spezialisierten Algorithmen übernommen wird. Die Aktoren dagegen bekommen für ihre Arbeit alle Informationen, die sie brauchen, in einer möglichst komprimierten Form. Des Weiteren wird die Geschwindigkeit des Gesamtsystems erhöht, da die Sensordaten sehr schnell zur Verfügung stehen und die Aktoren durch die hardwarenahe Implementierung der API effizient angesteuert werden können. Dies alles trug dazu bei, ein Höchstmaß an Effizienz bei der Nutzung der verwendeten Hardware zu erreichen.

2.2.4 Bahnplanung

Der autonome Roboter soll einen kompletten Raum möglichst effizient abzufahren. Basierend auf der von der Kartenerstellung bereitgestellten Karte, berechnet ein Algorithmus eine optimale Route. Der erfasste Raum wird, in der internen Repräsentation des RAPTOR, in mehrere Felder unterteilt, welche vom Bahnplanungsalgorithmus abgearbeitet werden. Hierbei erfasst ein modifizierter Floodfill-Algorithmus alle Felder des Raumes und legt sie auf einem Stack ab, welcher dann abgearbeitet wird. Der Algorithmus versucht möglichst lange, gerade Strecken zu suchen und sie bis zur Wand abzufahren. Anschließend wird ein möglichst langer paralleler Kurs in entgegengesetzter Richtung gesucht. Sobald ein Feld komplett abgearbeitet ist und der RAPTOR in kein benachbartes Feld fahren muss, wird die Routenplanung benutzt, um den kürzesten Weg zu einem weiter entfernten Feld zu berechnen.

2.2.5 Routenplanung

Die Routenplanung berechnet, gegen Angabe eines Start- und Endpunktes, eine effiziente Route. Die Bahn- und Routenplanung unterscheiden sich in der Hinsicht, dass die Bahnplanung eine komplette Route berechnet, in der die ganze Fläche des Raumes abgefahren wird und bei der Routenplanung lediglich der kürzeste Weg zwischen zwei Punkten gesucht wird.

2.2.6 Hauptprogramm

Das Hauptprogramm vereinigt alle Softwarekomponenten des Roboters. Damit alle Komponenten möglichst parallel arbeiten können, wurden die Prozesse in Threads ausgelagert. Somit kann gleichzeitig die Schmutzerkennung arbeiten, während die Robotersteuerung den RAPTOR durch den Raum navigiert. Einzelne Funktionen müssen beim gleichzeitigen Zugriff verschiedener Threads abgesichert werden, damit die Threads sich nicht gegenseitig beeinflussen und es weder zu Inkonsistenzen noch zu Konflikten kommt. Das Hauptprogramm verwaltet dazu Rechenzeit der verschiedenen Threads. Abhängig vom eingestellten Modus saugt der RAPTOR, fährt eine bestimmte Route ab oder folgt einem Objekt.

2.2.7 Odometrie

Mit Hilfe der Odometrie lässt sich die Position des Roboters durch das Beobachten der Räder bestimmen. Auf diese Funktion kann beispielsweise die Robotersteuerung zugreifen, da sie für die Befehlssequenzen der einzelnen Modi notwendig sind.

Die Odometrie ist ein zentraler Bestandteil der Navigation bzw. Orientierung des Roboters, da sie die Schnittstelle zwischen der realen und der internen Repräsentation der Welt bildet.

2.2.8 Kartenerstellung

Für die Bahn- bzw. Routenplanung ist eine vorgefertigte Karte des Raums notwendig. Die Kartenerstellung wird nach dem Einschalten des Roboters aktiviert. Zunächst werden für die Erstellung der Karte alle Wände eines Raumes systematisch abgefahren. Währenddessen sammelt der Roboter mittels geeigneter Sensoren und Fahrmethoden alle Sensordaten, bereitet diese auf und speichert sie in einer Karte, welche den abgefahrenen Raum repräsentiert. Die erstellten Karten können dann gespeichert und wieder geladen werden, sobald der Roboter beim Erfassen des Raumes eine Übereinstimmung mit bereits vorhandenen Karten findet.

2.2.9 Robotersteuerung

Die Robotersteuerung ist die Instanz auf der High-Level-Programmierebene, die alle vorgesehenen Modi und deren Algorithmen für den Roboter mit den echten Fahrbefehlen

realisiert. Sie verwaltet die Modi Bahnplanung, Routenplanung, sowie den Follow-Modus und führt diese mit entsprechenden Sequenzen von Fahrbefehlen aus. Während die Bahn- und Routenplanung eine Route erzeugen, die nacheinander durch Fahrbefehle abgearbeitet wird, wird beim Follow-Modus der Winkel und der Abstand zum zu verfolgenden Zielobjekt gemessen und in Fahrbefehle umgesetzt. Wichtig in diesem Zusammenhang ist auch der Einfluss der Odometrie. Diese ist dafür zuständig, die Robotersteuerung stets über die aktuelle Position des Roboters zu informieren.

2.2.10 Schmutzerkennung

Bei aktuellen Staubsaugerrobotern kommen nur unpraktikable Lösungen im Bereich der Schmutzerkennung zum Einsatz. Diese erkennen den Dreck nur, indem sie ihn aufnehmen. Normalerweise wird hierbei der Schmutz durch geschickte Luftführung gegen eine Metallplatte geschleudert, welche den Aufprall registriert. Anhand der Häufigkeit der Kollisionen wird der Verschmutzungsgrad berechnet.

Die Projektgruppe verfolgt eine andere Strategie. Hierzu wurde eine Kamera installiert, die Aufnahmen des Bodens macht. Die so entstandenen Fotos werden mit Hilfe von Methoden des maschinellen Lernens ausgewertet. Dieser Vorgang der Schmutzerkennung benötigt zwei Phasen. Die erste Phase besteht aus einem Training, in dem der Roboter lernt, wie in einem bestimmten Raum ein sauberer Boden aussieht. Hierfür müssen während der Fahrt mehrere Bilder aufgenommen werden, welche in ein Modell des Bodens einfließen. Dieses Modell stellt die Referenz dar, welche in der zweiten Phase die Bewertung des Bodens ermöglichen soll. Hierbei werden die aktuellen Fotos des Bodens ausgewertet und eine Differenz zu dem Modell des sauberen Bodens ermittelt. Diese Differenz wird Fehler genannt. Anhand des berechneten Fehlers lassen sich Rückschlüsse auf den Verschmutzungsgrad ermitteln.

2.2.11 Sensordatenfusion

Da der RAPTOR verschiedene Sensoren benutzt, ist eine Fusion der Werte notwendig, um einen Wert zu erhalten mit dem gearbeitet werden kann. Des Weiteren liefern die Sensoren aufgrund von Produktionstolleranzen keine genauen Werte. Der Datenfusion geht es darum aus verschiedenen Messungen eine optimale Schätzung zu berechnen, die genauer ist als die einzelnen Messungen der Sensoren. Die Tatsache, dass der RAPTOR mehrere Sensoren verwendet, bietet die Möglichkeit die Sensorwerte mit einander zu kombinieren um eine höhere Genauigkeit zu erzielen. Anhand der Sensorwerte wird ein Schätzwert gebildet, der versucht der realen Situation am nächsten zu kommen. Die Schätzung kann als ein Verfahren definiert werden, das auf Basis von Wissen über die Systemeigenschaften und durch Trennung von Signalen und Störungen einen Schätzwert des Systemzustandes mit einer minimalen Schätzfehlervarianz ermittelt. Als Schätzverfahren wurde der Kalmanfilter angewendet. Dies ist ein rekursives Verfahren, das den Systemzustand mittels einer Gaußverteilung schätzt.

2.2.12 Umgebungsmodell

Da bei der Entwicklung des Roboters versucht wurde besondere Aspekte, wie Schmutzerkennung, zu berücksichtigen, blieb es nicht aus, maßgeschneiderte Software zu entwickeln, die diesen Anforderungen entsprechen. Hierbei ist vor allem die interne Repräsentation der Umgebung zu erwähnen. Dieses Abbild der realen Welt benötigt eine besondere Aufmerksamkeit, es wurden dafür verschiedene Anforderungen formuliert: zum einen muss eine Anpassung an die begrenzte Rechenleistung der verwendeten Hardware möglich sein, zum anderen darf dies nicht die Effizienz des Umgebungsmodells verringern. Hier war es notwendig eine eigene Entwicklung vorzunehmen.

Die Kartenarchitektur basiert auf einer Gitterkarte. Diese bietet durch ihren schnellen Zugriff eine bessere Basis zur Routenberechnung. Sie wird durch verschiedene Techniken der Bildverarbeitung realisiert, die aber rein auf Operationen ganzzahliger Werte aufbaut, welche von der verwendeten Hardware problemlos bewältigt werden kann. Auch handelt es sich um einfacher zu implementierende Algorithmen, welche durch verschiedenste Techniken zusätzlich optimiert werden können. Dadurch sinkt die Fehleranfälligkeit deutlich aufgrund der geringeren Komplexität.

2.2.13 Follow-Modus

Ein zusätzliches Feature, das der RAPTOR beherrscht, ist der Follow-Modus. Die Idee einer solchen Funktion ist unter Anderem die Programmierung von Wegen. Auch könnte der Follow-Modus dazu verwendet werden den Roboter in den zu reinigenden Raum zu führen oder einfach nur den Roboter schwere Lasten fahren zu lassen.

In diesem Modus sind die primären Funktionen, wie Bahn- und Routenplanung, inaktiv. Lediglich die Datenfusion und Odometrie laufen für die Schmutzerkennung im Hintergrund weiter. Mit Hilfe der Frontkamera werden Bilder aufgenommen, die mit verschiedenen Farbfiltern gefiltert werden. Anschließend wird, je nach Einstellung, der größte Farbblock einer bestimmten Farbe in den Bildern gesucht. Dabei wird entweder nach dem größten roten, grünen oder blauen Farbblock gesucht. Mit den gefundenen Blöcken wird anschließend ein Korrekturwinkel berechnet, den der RAPTOR fahren muss. Durch die vertikale Position des Blockes lässt sich der Abstand des Objektes zum Roboter berechnen. Somit ist eine Verfolgung eines Objektes möglich.

2.2.14 Webinterface

Ein Webinterface ist eine Schnittstelle zu einem System, welches über einen Browser angesprochen werden kann. Für den RAPTOR wurde eine solche Webschnittstelle entwickelt, in der dem Benutzer alle Sensordaten angezeigt und bestimmte Funktionalitäten zur Verfügung gestellt werden. Zunächst stellt das Webinterface alle erfassten Sensordaten des Roboters dar, darunter auch die von der Front- und Bodenkamera aufgenommenen Bilder. Des Weiteren ist ein Bild der internen Karte zu sehen, welches durch die Kartenerstellung erzeugt wurde, auf welcher die Position des Roboters abgebildet ist. Darüber befindet sich

ein Menü, mit dem der Modus des Roboters in Routen-, Bahn- und Aufgabenplanung gesetzt und geändert werden kann. Soll der Roboter sich in keinem Modus befinden und somit zum Stillstand gebracht werden, kann “Kein Modus” ausgewählt werden. Sobald die Routenplanung aktiviert ist, ist es möglich dem Roboter per Klick auf die Karte neue Koordinaten zu übergeben, welche er daraufhin anfährt. Außerdem besitzt der Roboter eine terminbasierte Aufgabenverwaltung. Der Benutzer kann über das Webinterface Aufgaben zu bestimmten Zeiten bzw. Perioden konfigurieren und den Roboter automatisch dazu veranlassen, die Bahnplanung zu starten mit der ein Raum komplett abgefahren wird.

3 Hardware

Mit Hilfe der Grundlagen aus dem zweiten Kapitel spezialisiert sich dieses Kapitel auf die Realisierung der Hardware des autonomen Staubsaugerroboters. Im Folgenden werden Einzelheiten zum I²C-Bus erläutert. Ferner erfolgt ein Überblick über die Implementierung des Mikrocontrollers ATMega32. Bevor die Realisierung der Hauptplatine des Roboters beschrieben wird, wird erklärt wie die Hardwaregruppe ihre Platinen hergestellt hat. Des Weiteren wird auf den RS232-I²C-Adapter des Roboters näher eingegangen, den die Projektgruppe bereitstellte. Abschliessend wird die Karosserie beschrieben. Dabei wird auf die Platzierungen der einzelnen Hardwareelemente und die entsprechenden Befestigungsmethoden eingegangen.

3.1 Inter-Integrated Circuit

Der Inter-Integrated Circuit (I^2C, auch IIC oder Inter-IC)-Bus ist ein serieller Zweidraht-Bus, der auf Grund seiner Vorteile heutzutage in vielen Systemen verwendet wird. Als er vor etwa zwanzig Jahren von der Firma Phillips entwickelt wurde, waren die Ziele, mit möglichst wenig Adern auszukommen. Damit konnten Pins an den ICs, Leiterbahnen und somit auch wertvolle Fläche gespart werden. Inzwischen hat sich dieser Bus zu einem Industriestandard in unzähligen Embedded-Lösungen entwickelt.

3.1.1 Verwendung im Roboter

Im RAPTOR wird der I^2C-Bus zur Kommunikation zwischen dem Gumstix, dem Mikrocontroller ATMega32 und dem Kompassmodul CMPS03 verwendet.

- In Bezug auf den Mikrocontroller bietet I^2C den Vorteil, dass auf einen bestehenden Standard zurückgegriffen werden kann und keine neuen Methoden zur Kommunikation überlegt werden müssen. Als Alternative war vorübergehend der RS232-Standard im Blickfeld, dieser hätte aber große Nachteile im Bereich der Geschwindigkeit gehabt. Außerdem besteht so die Möglichkeit, den UART-Port des Mikrocontrollers für Debug-Ausgaben oder andere Anwendungen zu nutzen. Ein zusätzlicher Vorteil ist, dass beide Bausteine Anschlüsse für den Bus anbieten und das Protokoll direkt unterstützen.

- In Bezug auf den Kompass bietet I^2C den Vorteil, dass die Orientierung direkt ausgelesen werden kann und nicht erst aus einem PWM-Signal ermittelt werden muss.

Da der Kompass direkt an den I^2C-Bus angeschlossen wird, müssen dafür keine zusätzlichen Pins am Mikrocontroller oder am Gumstix belegt werden.

3.1.2 Die Technik

Der Bus verwendet zwei Leitungen. Eine Leitung (SCK, Serial Clock) wird für die Taktung verwendet, die andere Leitung (SDA) für die eigentliche Datenübertragung. Beide Leitungen werden über Pull-Up-Widerstände auf einem hohen Spannungspegel gehalten. In vielen Anwendungen sind dies 5V; im RAPTOR erzwingen die Spezifikationen des Gumstix allerdings einen Pegel von 3.3V. Weder der Mikrocontroller, noch das Kompassmodul hatten damit Probleme.

Diese beiden Leitungen sind parallel an die entsprechenden Anschlüsse der Bus-Geräte angeschlossen, somit sind also alle SDA-Pins aller Geräte miteinander verbunden, wie Abbildung 3.1 zeigt.

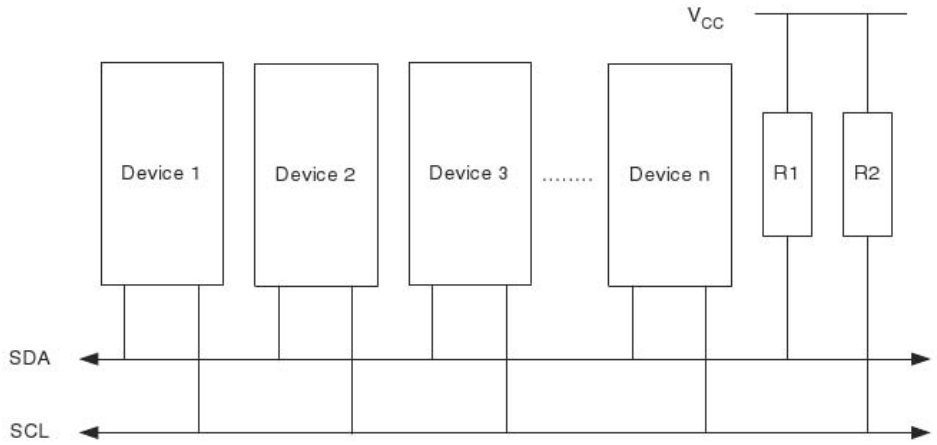


Abbildung 3.1: Anbindung von Geräten am I^2C-Bus [26]

Die Kommunikation zwischen den Geräten geschieht nun dadurch, dass Geräte diese Leitungen auf GND ziehen. Durch die implizite Verundung lesen *alle* Geräte einen LOW-Pegel, wenn auch nur *ein* Gerät die Leitung auf LOW gezogen hat. Umgekehrt lesen *alle* Geräte einen HIGH-Pegel, wenn *kein* Gerät die Leitung auf LOW zieht.

3.1.3 Gültigkeit von Daten

Grundsätzlich gilt, dass ein Bit auf der SDA-Leitung nur dann gültig ist, wenn es während eines gesamten SCK-Impulses konstant bleibt. Wenn der Taktgeber selbst die Daten auf den Bus legt, so wird er sicherlich selber darauf achten. Wenn der Datengeber aber nicht gleich dem Taktgeber ist, so darf er seinen Pegel nicht zwischen der steigenden Flanke und der fallenden Flanke auf der SCK-Ader ändern, siehe Abbildung 3.2.

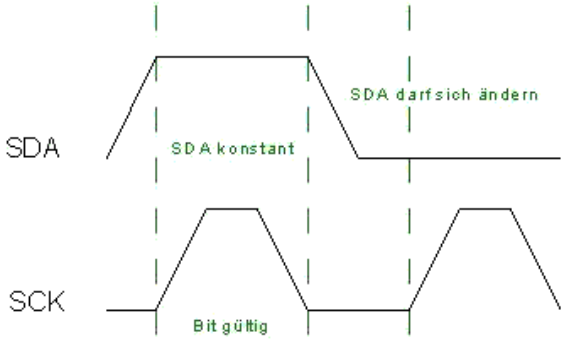


Abbildung 3.2: SDA darf sich während eines SCK-Impulses nicht ändern [26]

Eine Ausnahme von dieser Regelung sind die Start- und die Stopp-Bedingung, die gerade dadurch zu erkennen sind, dass der SDA-Pegel während eines SCK-Impulses geändert wird (siehe Abbildung 3.3):

Startbedingung Zieht ein Gerät die SDA-Leitung auf GND, obwohl SCK auf HI liegt (zum Beispiel zwischen einzelnen Kommunikationspaketen), so signalisiert es damit, dass es zum Master auf dem Bus werden und eine Kommunikation beginnen möchte.

Stoppbedingung Ein ansteigender SDA-Pegel bei SCK-Leitung auf HI-Niveau signalisiert das Ende eines Kommunikationspaketes. Der Bus ist somit wieder frei für die nächste Kommunikation.

“Repeated Start” Wenn ein Master zwar eine Kommunikation beenden, aber sofort eine neue beginnen möchte, ohne den Bus freizugeben, so kann er statt der Stoppbedingung auch eine erneute Startbedingung senden. Dies wird dann als “Repeated Start” bezeichnet. In der Praxis ist solch ein Vorgehen sinnvoll, weil eine Einzelkommunikation nur lesend oder schreibend sein kann, jedoch keine Mischung davon. Wenn also ein Master zunächst ein Datum schreiben und dann vom gleichen Slave ein Datum lesen möchte, so ist es zu empfehlen, nicht erst den Bus freizugeben. Dieser “gemischte” Zugriff mag zunächst unsinnig erscheinen, allerdings möchte der Master dem Slave oftmals mitteilen, *welches* Datum (z.B. Register) er lesen möchte. Hier ist also ein vorheriger Schreibzugriff nötig.

Sollten zeitgleich mehrere Geräte versuchen, eine Kommunikation zu beginnen und Master auf dem Bus zu werden, so werden die beiden (oder mehr) Geräte diese Kollision erkennen und durch eine Arbitrierung einen einzigen Master ermitteln. Im RAPTOR wird ausschließlich der Gumstix die Master-Rolle übernehmen, so dass es hier nicht zu Kollisionen kommen kann.

Somit wäre auch die Idee des “Repeated Starts” zu vernachlässigen.

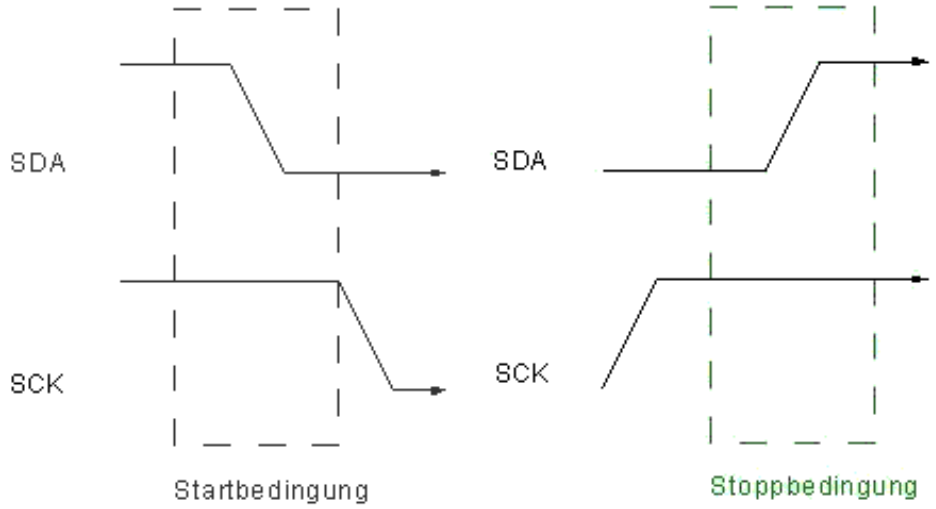


Abbildung 3.3: Start- und Stopp-Bedingung [26]

3.1.4 Bestätigung von Daten

Ein Datenpaket besteht immer aus 8 Bit. Nachdem der Sender diese 8 Bit übermittelt hat, folgt noch ein weiterer SCK-Impuls, in diesem Impuls greift der Sender nicht schreibend auf die SDA-Leitung zu, statt dessen obliegt es der Verantwortung des Empfängers, den SDA-Pegel auf GND zu ziehen - natürlich wieder, bevor auf der SCK-Leitung die ansteigende Flanke erscheint; das ACK-Bit (Acknowledge) ist in diesem Falle ein Bit wie jedes andere auch. Dieses ist dann die Bestätigung, dass der Empfänger die Daten erhalten und gespeichert hat. Abbildung 3.4 stellt solch ein Szenario dar.

Wenn der Empfänger den SDA-Pegel ebenfalls HIGH lässt, so ist das für den Sender ein Zeichen, dass der Empfänger die Daten nicht oder nicht richtig erhalten hat. Wie in diesem Falle zu verfahren ist, ist von der Software abhängig.

Allerdings tritt dieser NACK-Fall (Not ACK/Not Acknowledge) nicht nur bei Fehlern auf, sondern auch, wenn der Empfänger keine weiteren Daten erhalten *will*. Dieser Fall tritt beispielsweise bei dem im RAPTOR verwendeten Kompassmodul auf. Nachdem es adressiert und Register 2 ausgewählt wurde, sendet das Modul nun nacheinander die Register 2,3,4 usw. Da allerdings nur Register 2 und 3 relevant sind, sollte nach dem zweiten ausgelesenen Byte ein NACK gesendet werden.

Natürlich ist auch die danach gesendete Stoppbedingung ein eindeutiges Zeichen für das Ende der Kommunikation, allerdings kann das NACK sinnvoll sein, weil in manchen Anwendungen das Bereitstellen zusätzlicher Bytes für den Sender mit Zusatzaufwand verbunden ist. Durch das Senden des NACK weiß der Sender somit, dass er diesen Zusatzaufwand nicht betreiben muss, da diese zusätzlichen Bytes nicht abgefragt werden.

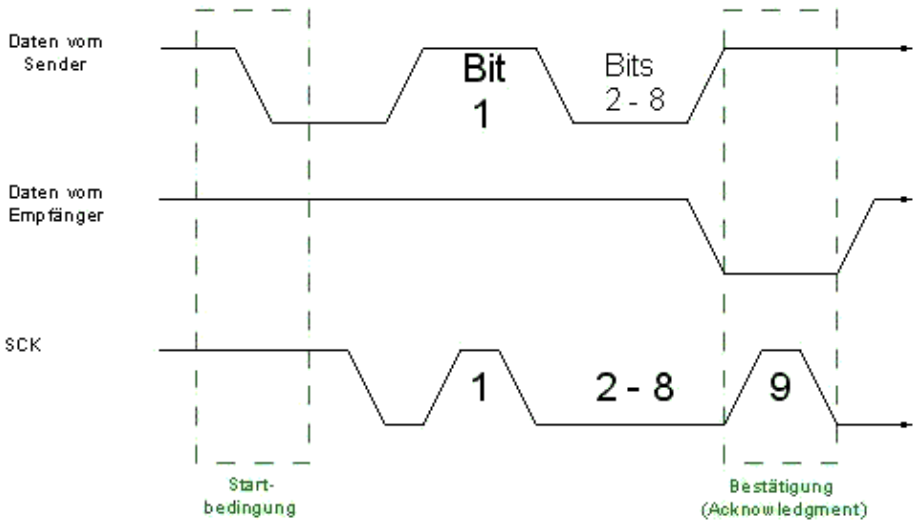


Abbildung 3.4: Der Empfänger quittiert den Empfang eines Paketes, indem er die SDA-Ader beim “9. Bit” auf Null zieht [26]

3.1.5 Adressierung

Das erste Byte nach der Startbedingung ist die Adressierung des Slaves. Der Master wählt aus, mit welchem Slave er kommunizieren möchte, zudem wird an dieser Stelle auch schon der Modus (lesend/schreibend) definiert.

I^2C-Geräte, die als Slaves fungieren können, haben eine Adresse, auf die sie reagieren. Diese lässt sich - je nach Gerät - frei vergeben (wie im Mikrocontroller möglich), teilweise frei vergeben (manche ICs der gleichen Baugruppe sind immer im gleichen “Adressraum”, nur die letzten Bits können manipuliert werden) oder kann auch fest in der Hardware vorgegeben sein (wie im Kompassmodul). Nach der ursprünglichen Definition des Bus hat eine Adresse exakt sieben Bit. Einige Bitkombinationen sind jedoch für andere Zwecke reserviert, so dass nach dieser Reservierung “nur” 114 Geräte angesteuert werden könnten. In manchen Anwendungen reichen diese 114 Adressen nicht aus. Es gibt daher neben der Möglichkeit, mehreren Slaves die gleiche Adresse zu geben, inzwischen auch eine erweiterte 10-Bit-Adressierung. Da im RAPTOR nur vier Teilnehmer und nur drei Slaves am Bus betrieben werden sollten, wurde eine Einarbeitung in diesen Bereich nicht als notwendig angesehen.

Da von einem Datenbyte nur die ersten 7 Bit für die Adresse benötigt werden, wird mit dem achten Bit die Datenrichtung ausgewählt:

- Ist das Bit gesetzt, so wird lesend auf den Slave zugegriffen. Der Master befindet sich im *Master Receiver*-Modus, der Slave wird in den *Slave Transmitter*-Modus geschaltet.

- Ist das Bit nicht gesetzt, so wird schreibend auf den Slave zugegriffen. Der Master befindet sich im *Master Transmitter*-Modus, der Slave wird in den *Slave Receiver*-Modus geschaltet.

Das neunte Bit ist dann - wie schon erwähnt - das ACK-Bit. Hiermit bestätigt der Slave, dass er seine Adresse empfangen hat und "einsatzbereit" ist. I^2C sieht auch eine Möglichkeit vor, allen Slaves am Bus eine Nachricht zukommen lassen - "General Call". Hierbei senden dann alle Slaves ein ACK.

In manchen Beispielquelltexten fällt auf, dass dort als Adresse ein Wert über 127 steht, obwohl nur eine 7-Bit-Adressierung verwendet wird - selbst im Kompass-Datenblatt steht geschrieben, der Kompass hätte die Adresse 192. Da die Datenrichtung mit der Slave-Adresse als ein Byte auf den Bus gelegt wird, wird sie oftmals direkt mit in den Parameter übernommen. In diesem Datenbyte werden die höchstwertigen Bits für die Adresse verwendet und die Datenrichtung als letztes Bit angehängt; von daher ist das Datum doppelt so groß wie die eigentliche I^2C-Adresse - und im Lesemodus nochmals um eines erhöht. Die richtige Adresse des Kompassmoduls ist somit die `96`, das Datenbyte für Schreibmodus lautet 192, das Datenbyte für Lesemodus lautet 193.

3.1.6 Datenübertragung

Nachdem nun der Slave angesprochen und die Datenrichtung festgelegt wurde, findet der eigentliche Datenaustausch statt. Der Master sorgt für die Impulse auf der SCK-Leitung und je nach Modus legen er oder der Slave die Daten auf die SDA-Leitung. Natürlich ist im `Master Receiver`-Modus zu beachten, dass der Master auch das ACK für die erhaltenen Daten zu senden hat.

Hat der Slave verschiedene Möglichkeiten, Daten zu versenden (z.B. durch verschiedene Register), so muss ihm zunächst mitgeteilt werden, welche Daten er senden soll. Wie schon oben erwähnt, bietet es sich dazu an, zunächst im Schreibmodus auf den Slave zuzugreifen und ihm als Datum die Anweisungen für den darauf folgenden Lesezugriff zu übermitteln.

Beispielsweise hat der Kompass ein Register für Firmware, ein Register für die aktuelle Ausrichtung auf Byte-Wert normiert, zwei Register für die aktuelle Ausrichtung in Dezimalgrad (3599 entspricht 359,9 Grad, hierbei werden dann schon zwei Bytes benötigt, aber der zurückgegebene Wert ist deutlich intuitiver) sowie noch Register für Kalibrierung und andere Informationen. Um die aktuelle Ausrichtung als Dezimalgrad zu erhalten, wird zunächt eine `2` (Auswahl des Registers) an das Kompassmodul gesendet und anschließend eine lesende Datenübertragung gestartet.

```
1 SendeStartBedingung
2 SendeAdresseMitSchreibeModus
3 SchreibeDaten(2)
4 SendeRepeatedStart
5 SendeAdresseMitLeseModus
6 LeseDaten(2 Byte)
7 SendeStoppBedingung
```

Listing 3.1: Kommunikation mit dem Kompassmodul

3.1.7 Clock Stretching

An einem I^2C-Bus müssen Komponenten mit unterschiedlicher Geschwindigkeit arbeiten können, daher musste eine Möglichkeit gefunden werden, den Bus für langsamere Teilnehmer anzuhalten. Diese Möglichkeit findet sich beispielsweise im "Clock Stretching":

Wenn der Slave an einem gewissen Punkt in der Übertragung nicht bereit für das nächste Datenbyte ist, drückt er die SCK-Leitung auf GND, bis er wieder bereit ist. Somit wird verhindert, dass der Master den nächsten SCK-Impuls sendet (zumindest, wenn der Master Clock Stretching unterstützt).

Der ATMega32 unterstützt Clock Stretching. Wenn beispielsweise ein Datenpaket eingetroffen ist, wird ein entsprechendes Flag gesetzt. Die SCK-Leitung wird erst dann wieder freigegeben, wenn das Programm es explizit erzwingt. Der Programmierer hat somit ausreichend Zeit, sämtliche Operationen und Vorbereitungen vorzunehmen, die das Datenpaket verlangt.

Da selbst das Auslesen des Datenpaketes durch den Softwarefluss geschehen muss, könnte selbst *die Abfrage des I^2C-Busses an sich* verzögert werden.

In der Praxis sollte die I^2C-Kommunikation zügig abgewickelt werden. Zu diesem Zweck können I^2C-Ereignisse auch Interrupts auslösen.

3.2 ATMega32

Wie weiter oben schon angeführt, wird im Roboter der Mikrocontroller ATMega32 von Atmel (Abbildung 3.5) verwendet. Dieser soll an dieser Stelle etwas genauer betrachtet werden. Da seine technischen Details zu ausführlich zum Auflisten sind, wird eher auf die Bereiche eingegangen, die für den RAPTOR relevant sind. Um sich genauer mit dem Mikrocontroller zu beschäftigen, ist ohnehin eine umfangreiche Lektüre des Datenblattes [5] und der Anleitungen im Netz und aus Büchern unumgänglich.

3.2.1 Programmierhardware

Auch wenn die Programmierung des Mikrocontrollers *in system* erfolgen kann, wird dazu Programmierhardware benötigt. Es gibt auf dem Markt eine Auswahl an Geräten und es

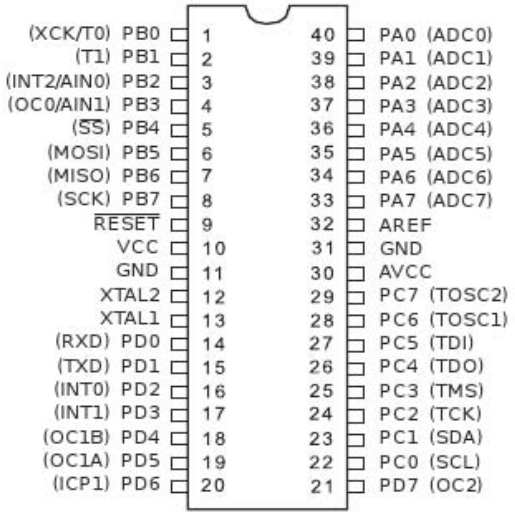


Abbildung 3.5: Pinbelegung des ATMega32

existieren ausführliche Anleitungen zum Selbstbau. Im einfachsten Fall wird der Mikrocontroller quasi direkt an den Parallelport des PCs angeschlossen - was nicht empfohlen wird, da im Fehlerfalle der Port beschädigt werden kann. Sicherer ist die Verwendung von Dioden oder einem kompletten Bustreiber [27]. Zudem gibt es Programmierhardware zum Anschluss an den seriellen Port des PCs ("Serial-ISP"). Aufgrund der großen Verbreitung von USB-Anschlüssen und des Komfort dieses Busses gibt es auch Adapter, um Mikrocontroller über den USB-Port zu programmieren.

AVR ISP mkII

Abbildung 3.6 zeigt ein *AVRISP mkII*, welcher für das Projekt angeschafft wurde. Dabei handelt es sich um einen USB-Programmer direkt von der Firma Atmel. Die Verwendung eines bereits montierten Programmiergeräts bietet den Vorteil, dass dieser auch direkt von der firmeneigenen Software-Suite unterstützt wird und im Gegensatz zu Eigenbau-Lösungen davon ausgegangen werden kann, dass er funktioniert - eine Fehlersuche dort kann entfallen.

Anschluss am Mikrocontroller

Auch wenn es verschiedenste Programmieradapter mit verschiedensten Anschlüssen auf der PC-Seite gibt (klassisch seriell, parallel oder USB), ist der Anschluss auf der anderen Seite standardisiert. Zwar gibt es auch hier noch verschiedene Bauformen:

- Die Firma Atmel definiert eine Belegung an einer sechspoligen Buchsenleiste. Diese findet sich auch am AVRISP mkII.
- In vielen Plänen im Netz wird von einer zehnpoligen Buchsenleiste ausgegangen. Für diese lässt sich einfacher Anschlusshardware finden. Daher hat sich dieser Anschluss als ein "Standard der Hobbybastler" etabliert.

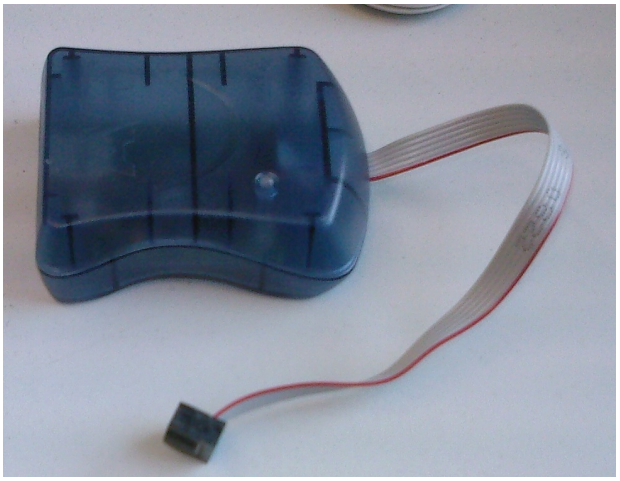


Abbildung 3.6: AVRISP mkII - unser Programmiergerät

- In manchen Anwendungen wird ein neunpoliger D-Sub-Stecker (D shaped subminiature) verwendet - oftmals dient er zur Übergabe auf eine zehnpolige Buchsenleiste.

- Da die für die Programmierung zuständigen Pins an einem ATMega32 sowohl in SMD- als auch in LDIP-Bauform direkt nebeneinander liegen, kann es zur Vereinfachung bei der Platinenerstellung sinnvoll sein, einen einreihig sechspoligen Verbinder zu verwenden.

Im Wesentlichen sind dort aber immer die gleichen Signale enthalten. Tabelle 3.1 listet die verschiedenen Pinbelegungen sowie den Anschluss am ATMega32 auf. Die Details werden später noch erläutert.

Belegung	*ATMega32*	*6 pol Wanne*	*10 pol Wanne*	*9 pol DSub*	*6 pol einreihig*
MISO	7	1	9	5	2
VCC	z.B. 10	2	2	6	5
SCK	8	3	7	4	3
MOSI	6	4	1	1	1
Reset	9	5	5	3	4
GND	z.B. 11	6	4/6/8/10	7/8/9	6
LED	n/a	n/a	3	2	n/a

Tabelle 3.1: Die Pinbelegung verschiedener Stecker

Um die verschiedenen Adern eines Kabels an einer Buchsenleiste zu identifizieren, muss der Stecker so gehalten werden, dass die markierte Ader nach links zeigt. Pin 1 befindet sich dann oben links, die anderen Pins folgen im Zickzackmuster, welches Abbildung 3.7 aufzeigt.

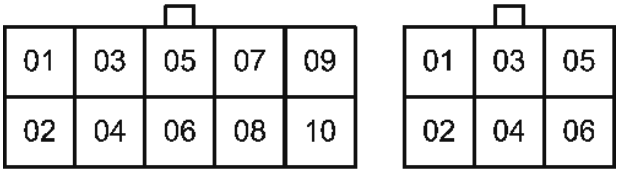


Abbildung 3.7: Pinbelegung von Wannenbuchsen

3.2.2 Inbetriebnahme des ATMega32

Um den Mikrocontroller zum Laufen zu bringen, reicht es aus, die GND-Pins (Pin 11 und Pin 31) mit GND zu verbinden und den Spannungspin (Pin 10) an die Stromversorgung anzuschließen. Dennoch sollten weitere Verbindungen geschaffen werden:

- Zur Spannungsstabilisierung wird ein Kondensator (z.B. 100 nF) zwischen GND und P5V geschaltet. Solche Filter werden sehr häufig an ICs eingesetzt.
- Auch wenn der Mikrocontroller nicht für analoge Messungen verwendet wird, sollte auch AVCC (Pin 30) mit P5V verbunden werden.
- In diesem Aufbau ist der Reset-Pin (Pin 9) unbeschaltet. Generell lässt sich nicht vorhersehen, welcher Pegel an einem unbeschalteten Pin anliegt, eventuell wird ein vorhandenes Rauschen aus der Luft abwechselnd als LOW und als HIGH erkannt. Da der Controller sich bei einem als LOW erkannten Pegel zurücksetzt, kann ein unbeschalteter Reset-Eingang zu ungewünschtem Verhalten führen. Somit sollte dieser Eingang bei fehlender weiterer Beschaltung auf den “sicheren” Pegel P5V gesetzt werden; diese Aufgabe übernimmt ein PullUp-Widerstand. Der Atmega32 verfügt zwar über einen integrierten PullUp-Widerstand am Reset-Pin, bei komplexen Beschaltungen könnte dieser allerdings nicht mehr ausreichen, es empfiehlt sich also, zusätzlich einen externen PullUp-Widerstand (z.B. 10 $k\Omega$) einzusetzen.

Bei dieser Verschaltung hat der Controller noch keinerlei Möglichkeiten, sich mit der Außenwelt zu verständigen (LEDs, Datenverbindungen, etc. . .). Allenfalls ein Ampèremeter könnte über den Stromverbrauch anzeigen, dass der Controller aktiv ist.

Erste Kommunikation mit dem PC

Die zuverlässigste Methode, den Mikrocontroller am PC zu erkennen, ist, das Programmiergerät anzuschließen und die Kennung des Controllers auszulesen. Die dazu nötigen Verbindungen lassen sich aus Tabelle 3.1 entnehmen. Dabei ist jedoch zu beachten:

- Aus Sicherheitsgründen gibt das Programmiergerät am Reset-Pin keinen Pegel aktiv vor. Der PullUp-Widerstand wird weiterhin benötigt.

- Weder kann das Programmiergerät den Mikrocontroller mit Strom versorgen, noch umgekehrt. Beide Bauteile müssen von einer externen Spannungsquelle gespeist werden. Zwar sind mit dem AVRISP mkII einige Basisoperationen schon durch die 5 Volt des USB-Ports möglich, aber für die echte Verwendung ist die Speisung des Controllers eine Notwendigkeit.
- Alle Bauteile in den Aufbauten sollten den gleichen Massepegel haben. Auch das Programmiergerät stellt keine Ausnahme dar.

Programmiersoftware

Zunächst wurde das "AVR Studio" des Herstellers [4] zur Programmierung des Mikrocontrollers verwendet. Diese Software ist kostenlos auf der Homepage des Herstellers erhältlich und unterstützt den von uns verwendeten USB-Programmer nativ. Nach der Einbindung von "WinAVR" [36] kann mit dieser Software auch C-Code kompiliert und übertragen werden.

Später wurde es nötig, auf Linux-Software umzuschwenken; Dies bot zudem den Vorteil einer größeren Flexibilität in den Makefiles. Die bis dahin geschriebenen Programme konnten zum größten Teil ohne Änderungen weiter verwendet werden.

Unter Linux kann die Programmierung des Mikrocontrollers über das Programm "AVRDUDE" [22] vorgenommen werden, ein geeigneter C-Compiler findet sich unter dem Namen "avr-gcc" in der GNU Compiler Collection.

Verbindung zum PC

Während der Entwicklung war es an vielen Stellen nötig, genau zu erfahren, in welchem Programmteil sich der Mikrocontroller gerade befindet - respektive, wieso er nicht so arbeitet, wie es erwartet wird. Hier empfielt sich eine Kommunikationsverbindung zum PC, so dass Debug-Ausgaben erstellt und übermittelt werden können.
Leider ist eine derartige Kommunikation über den Programmieradapter nicht möglich. Zwar verfügt der ATMega32 über eine JTAG-Schnittstelle, die zum Debuggen verwendet werden kann, der von der PG verwendete AVRISP mkII unterstützt diese Schnittstelle jedoch nicht. Die Entwicklung eines zusätzlichen JTAG-Adapters wurde ebenfalls nicht in Betracht gezogen.
Der ATMega32 besitzt jedoch einen integrierten Baustein (USART) für serielle Kommunikation, er kann also einfach mit dem COM-Port eines PCs verbunden werden.

Dennoch ist der direkte Anschluss nicht möglich. Zwar verwendet der ATMega auch das RS232-Protokoll, er nutzt dabei jedoch andere Pegelwerte:

RS232- und TTL-Pegel

Der serielle Port des PCs verwendet das RS232-Protokoll mit RS232-Pegeln:

- Ein logisches HIGH entspricht etwa -10V
- Ein logisches LOW entspricht etwa +10V

Auf der anderen Seite arbeitet der ATMega32 (wie auch die meisten anderen ICs) mit TTL-Pegeln (Transistor-Transistor-Logik):

- Ein logisches HIGH entspricht P5V
- Ein logisches LOW entspricht 0V

Damit ATMega und PC trotzdem miteinander kommunizieren können, muss ein Pegelwandler verwendet werden - Der *MAX232* ist ein Standardbaustein für diese Aufgabe.

Verdrahtung des MAX232

Damit der Pegelwandler ordnungsgemäß funktioniert, sind noch einige Kondensatoren nötig, die genaue Beschaltung lässt sich in Abbildung 3.8 ablesen.

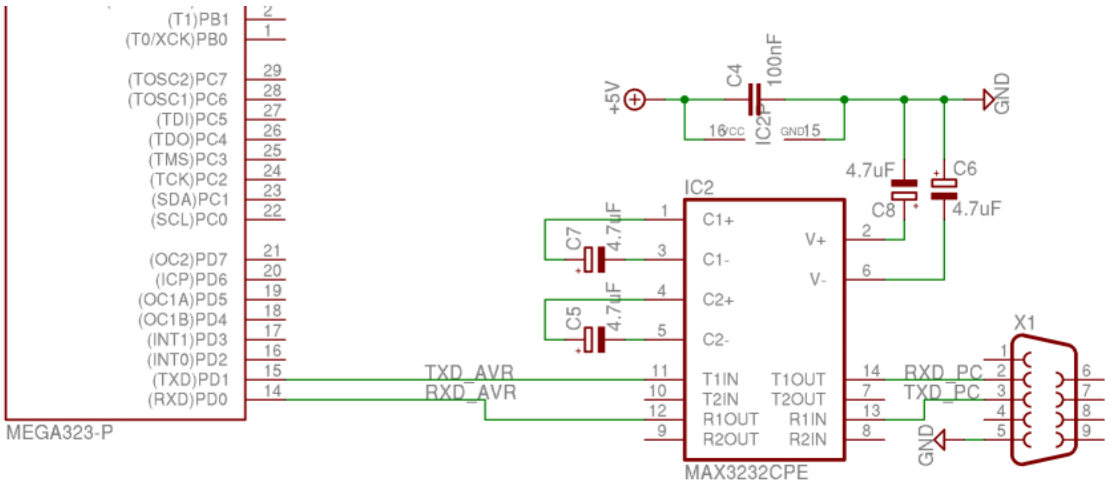


Abbildung 3.8: Verbindung zwischen ATMega und PC

Anschluss am Controllerboard

Da Debug-Ausgaben allenfalls temporär von Nutzen sind, wurden auf den verwendeten Boards die Pegelwandler plus ihre Beschaltung nicht fest eingebaut. Statt dessen wird eine vierpolige Steckleiste (Tabelle 3.2) verwendet, über die ein Pegelwandlerboard kurzfristig angeschlossen werden kann.

Pin	*Belegung*
1	TxD AVR, RxD PC
2	RxD AVR, TxD PC
3	GND
4	P5V zur Speisung des Pegelwandlers

Tabelle 3.2: Die Pinbelegung des RS232-Boardsteckers

Ausgabe im Programm

Um Daten während des Programmablaufes auszugeben, müssen die USART-Register des Mikrocontrollers direkt angesprochen. Vor der ersten Kommunikation ist eine Initialisierung der Baudrate (und anderen Werten wie Stoppbit und Paritätskontrolle) nötig. Zur

Vereinfachung wurden diese Methoden in eine eigene Datei (siehe Listing B.1) ausgegliedert.

Verbindung am PC

Am PC wird nun das bevorzugte Terminalprogramm verwendet (beispielsweise *minicom* unter Linux oder *HyperTerminal* unter Windows). Dabei ist zu beachten, dass bei Flusskontrolle weder "Hardware" noch "Software" eingestellt ist.

3.2.3 Ansteuerung der verwendeten Hardware

Im Folgenden soll dokumentiert werden, wie die im RAPTOR verwendeten Sensoren und Aktoren vom Mikrocontroller angesprochen und ausgelesen werden können.

Ultraschallsensoren

Wie schon in Abschnitt 2.1.2 dokumentiert, muss zunächst ein Trigger-Impuls ausgesendet und anschließend die Länge eines Echo-Impuls gemessen werden. Um Pins zu sparen, werden die Ultraschallsensoren im zweiten Modus betrieben; dies hat zudem den Vorteil, dass ein bestimmter Ultraschallsensor allein durch einen Pin am Controller vollständig identifiziert werden kann.

Die Programmroutinen befinden sich im Anhang unter Listing B.2, zur Messung der Impulsbreite wird ein Timer/Counter des Mikrocontrollers verwendet. Da auch bei maximalen Vorteiler ein Überlauf dieses 8-Bit-Zählers nicht verhindert werden kann, werden hier die Überläufe gezählt.
Um bei nicht angeschlossenen Ultraschallsensoren oder einem Fehlverhalten derselben nicht in einer Endlosschleife zu langen, wird nach einem Timeout automatisch ein Fehlerwert (0) zurückgegeben; diesen geben die Sensoren im Normalbetrieb niemals zurück.

Akku-Messung

Während des Betriebes wird fortwährend die Spannung des internen Akkus über einen Spannungsteiler gemessen; damit kann ein zu niedriger Akkustand frühzeitig erkannt und eine Wechsel bzw eine Aufladung vorgenommen werden, bevor eine extrem niedrige Spannung die Komponenten oder gar den Akku selbst schädigt.
Die Atmega-Baureihe verfügt über einen integrierten Analog-Digital-Wandler; dieser wird bei der Akkumessung verwendet. Der dazugehörige Quelltext befindet sich im Anhang (siehe Listing B.3).

Abgrundsensoren

Dieselbe Routine, die zur Messung der Akkuspannung eingesetzt wird, wird auch zur Abfrage der Abgrundsensoren verwendet; der Pegel dieser Sensoren schwankt je nach Untergrund und Lichtverhältnissen, so dass eine digitale Messung nicht möglich ist.

Akku-LED

Wenn ein gefährlich schwacher Akkustand erkannt wird, leuchtet eine nach außen sichtbare, rote Leuchtdiode auf. Da diese LED an der oberen Platte des Roboters befestigt ist, wird sie von einem anderen ATMega32 angesteuert als von jenem, der die Akkumes-

sung vornimmt. Bewusst wurde darauf verzichtet, eine "Kreuzkommunikation" zwischen den beiden Mikrocontrollern zu konzipieren; die LED wird daher softwareseitig über den Gumstix angesteuert.

Das Controllerboard wurde so angelegt, dass die LED nicht über einen Pin des Mikrocontrollers gespeist wird, sondern direkt von der 5V-Spannungsquelle. Der Atmel-Pin muss lediglich die Masse bereit stellen. Dies bedeutet, dass die LED leuchtet, wenn der Pin auf Ausgang und auf LOW geschaltet wird. Der entsprechende Quelltext befindet sich im Anhang unter Listing B.4.

Beleuchtung für Bodenkamera

Die Bodenkamera zur Stauberkennung befindet sich innerhalb des Gehäuses und nimmt daher Bilder auf, die meist nicht hell genug sind, um Schmutz ausreichend zu erkennen. Um dies auszugleichen, wurden einige ultrahelle LEDs zur Beleuchtung des Bodens genutzt. Auf der anderen Seite würden diese bei durchgängiger Beleuchtung den Stromverbrauch des Aufbaus erhöhen, daher werden diese LEDs nur angeschaltet, wenn ein Bild aufgenommen werden soll. Die Steuerung erfolgt über den Gumstix.

Um den dazugehörigen Controllerpin nicht zu überlasten, wird hier die Masse nicht direkt bereitgestellt, sondern über einen Transistor geschaltet. Daher leuchten diese LEDs, wenn der entsprechende Pin auf Ausgang und auf HIGH geschaltet wird. Die Methoden zur Ansteuerung der LEDs befinden sich ebenfalls im Anhang unter Listing B.4.

Maussensor

Der Maussensor verwendet einen eigenen - internen - Controller. Die Kommunikation mit diesem Controller erfolgt über einen Bus und ein definiertes Protokoll.

Die im RAPTOR verwendete Platine zur Ansteuerung des Maussensors hat bereits die Kontakte SDA und SCL - was sehr analog zum I^2C-Bus aussieht. Tatsächlich ist das Protokoll bis auf einige Kleinigkeiten sehr ähnlich:

- Es ist kein Bus, sondern eine Direktverbindung.
- Es ist keine Hardware-Adressierung nötig.
- Auf die Registeranwahl vom Controller aus folgt sofort die Antwort des Sensors - ohne dass ein Restart nötig wäre.
- Das Bit für die Auswahl der Datenrichtung ist invertiert.
- Die 5V-Default-Spannung auf dem Bus wird vom Controller vorgegeben.

Damit lässt sich die Platine nicht direkt an den I^2C-Bus anschließen. Auf der anderen Seite bedeutet das, dass zwei beliebige Pins am Mikrocontroller verwendet werden können.

Der zur Abfrage verwendete Quelltest (Listing B.5) wurde in groben Zügen der Webseite von Roboternetz.de [26] entnommen, musste jedoch noch ein wenig korrigiert und abgewandelt werden.

Kontaktsensoren

Die Kontaktsensoren haben hier eine Sonderbewandnis, weil sie nicht "ausgelesen" werden, sondern der Controller sofort auf eine Kontaktmeldung reagieren muss. Im Quelltext wird dies dadurch sichergestellt, dass ein Kontakt einen Interrupt auslöst, woraufhin *sofort* die Motoren gestoppt werden. Da dieses Signal sehr wichtig ist und zuverlässig erkannt werden muss, wird hier sicherheitshalber kein interner PullUp-Widerstand des Atmels verwendet, sondern ein externer PullUp-Widerstand eingesetzt.

Der Quelltext zur Konfiguration und Nutzung des Interrupts befindet sich im Anhang unter Listing B.6. Die eigentliche Aktivierung aller konfigurierter Interrupts per `sei()` geschieht jedoch im Hauptprogramm.

Manchmal ist es notwendig, die Motoren auch anzusteuern, wenn ein Kontakt erkannt wurde, dies ist beispielsweise der Fall, wenn bei der Behebung der Kollision durch Rückwärtsfahrt erwartungsgemäß erneut die Kontake auslösen. Dies ist jedoch ein absoluter Sonderfall und wird nur in Ausnahmefällen eingesetzt.

Radencoder

Auch die Radencoder müssen sofort bei Signalankunft behandelt werden, ansonsten könnten Impulse, während das restliche Programm läuft, verloren gehen. Daher sind auch die Radencoder so an den Mikrocontroller angebunden, dass sie bei Pegelwechsel einen Interrupt auslösen, welcher dann sofort bearbeitet wird.

Listing B.7 dokumentiert die Ansteuerung der Radencoder. Hierbei ist zu beachten, dass jeder Radencoder-"Tick" quasi mehrfach gezählt wird.

- Ursprünglich waren die Radencoder als reine Countdown-Zähler konzipiert. Wenn eine vorher eingestellte Anzahl an "Ticks" abgefahren ist, werden die Motoren gestoppt. Somit können Streckenlängen sehr genau abgefahren werden. Für eine nicht durch Abstand bestimmte Fahrt kann der Motorstopp bei Countdown-Ablauf auch abgeschaltet werden.

- Analog zum Maussensor war es jedoch später nötig, auch einen Aufwährtszähler zu implementieren, der wie der Maussensor die abgefahrene Strecke aufsummiert.

Motoren

Für jeden Antriebsmotor werden am Mikrocontroller direkt drei Pins benötigt:

- Über einen Pin wird per PWM-Signal eine pseudo-analoge Spannung zur Regelung der Motorgeschwindigkeit bereit gestellt. Der ATMega32 unterstützt die Ausgabe von PWM-Signalen nativ per Timer/Counter. Timer/Counter1 lässt sich sogar mit zwei Vergleichswerten verwenden, so dass beide Antriebsmotoren über den gleichen Zähler angesteuert, jedoch unterschiedliche Geschwindigkeiten haben können.

- Zwei weitere Pins werden benötigt, um die Drehrichtung des Motors zu übermitteln. Hiermit ist auch ein sofortiger Stopp des Motors möglich. Dieser ist für ein genaues Abfahren von Strecken nötig; Ein Nullsetzen der Geschwindigkeit sorgt noch für ein sanftes Ausrollen.

Die Programmstücke zur Ansteuerung des Motoren befinden sich im Anhang unter Listing B.8. Der größere Fallstrick ist dabei, die Beschaltung zum Motortreiber und zu den Motoren richtig auf die Pins im Quelltext abzubilden.

3.2.4 I2C am Atmel

Die ATMega-Baureihe kann direkt an einen I^2C-Bus angeschlossen werden (I^2C heißt bei Atmel TWI - Two Wire Interface). Das Protokoll muss nicht von Hand implentiert werden, statt dessen werden nur die vorgesehenen Register des Mikrocontrollers beschrieben und ausgelesen. Zudem verhalten sich die beiden ATMega32 auf dem I^2C-Bus wie auch der Kompass als einfache Registermaschinen. Dadurch wird die TWI-Routine zusätzlich entlastet, so dass die eigentliche Abfrage, ob ein Befehl auf dem Bus anliegt (Listing in B.9) sehr kurz und übersichtlich gehalten werden kann.

Diese Routine muss regelmäßig aus dem Hauptprogramm des Mikrocontrollers aufgerufen werden, die Befehlsauswertung wird ebenfalls vom Hauptprogramm vorgenommen.

3.3 Herstellung von Platinen

Einfache Leiterplatten bestehen aus einem elektrisch isolierenden Trägermaterial, dem sogenannten Basismaterial, auf dem eine oder zwei Kupferschichten aufgebracht sind. Die typische Schichtstärke beträgt 35 μm, für Anwendungen mit höheren Strömen werden Stärken zwischen 70 μm und 140 μm verwendet. Als Basismaterial werden in der Projektgruppe RAPTOR mit Epoxidharz getränkte Glasfasermatten verwendet. Auf die Kupferschicht ist Fotopositivlack aufgetragen, der mit einer lichtundurchlässigen Schutzfolie abgeklebt ist. Bei den zweiseitigen Platinen ist die Kupfer-, Fotolack- und Schutzschicht auf beiden Seiten. Die Platinen für den Staubsaugerroboter ist in sieben Schritten manuell von den Mitgliedern der Hardwaregruppe hergestellt worden. Die verwendeten Methoden werden im Folgenden näher beschrieben. Dabei wird auf die gewonnenen Erfahrungen und die eingesetzten Geräte sowie Materialien näher eingegangen.

3.3.1 Layout

Zum Erstellen des Layouts der Platinen wurde das Programm EAGLE (Einfach Anzuwendender Grafischer Layout-Editor) der Firma CADSoft [10] benutzt. Die Software besteht aus mehreren Komponenten: Layout-Editor, Schaltplan-Editor, Autorouter und einer erweiterbaren Bauelementedatenbank. Einige Bibliotheken, wie beispielsweise der Baustein CNY70 (Lichtsensor), mussten hinzugefügt werden. Der Autorouter erzielt nicht immer das gewünschte Ergebnis, so dass manuelle Verbesserungen vorgenommen werden müssen. Zusätzlich ist darauf zu achten, dass die Leiterbahnbreiten nicht zu klein, die Abstände zwischen verschiedenen Leiterbahnbreiten groß genug und eine ausreichende Lochgröße der Pads vorhanden ist.

3.3.2 Belichtung

Das Layout wird zunächst auf eine transparente Folie gedruckt. Danach kann die Platine, ohne Abziehen der Schutzfolie, auf die richtige Größe zugeschnitten werden. Mit der bedruckten Seite nach unten kann das Layout auf die Platine, ohne Schutzfolie, gelegt werden. Bei zweiseitigen Platinen werden die beiden Layoutfolien zuerst übereinander gelegt und anschliessend mit Tesafilm an den Seiten fixiert. Die zurechtgeschnittene Platine kann zwischen die Folien geschoben werden. Das Layout und die Platine wird auf die Glasplatte des Belichtungsgerätes gelegt. Um mögliche Fehlerursachen zu vermeiden, muss darauf geachtet werden, dass das Layout bei diesem Vorgang nicht verrutscht und die Glasplatte sauber ist. Die Vakuumpumpe des Belichtungsgerätes sorgt dafür, dass das Layout nicht verrutschen kann. Der Deckel des Belichtungsgerätes kann anschliessend geschlossen werden, um die Platine eine angemessene Zeit zu belichten. In Abhängigkeit von dem verwendetem Belichtungsgerät, sowie der verwendeten Epoxid-Platine, ergibt sich eine optimale Belichtungsdauer von 2 Minuten und 30 Sekunden.

3.3.3 Entwicklung

Nach der Belichtung kann die Platine entwickelt werden um die belichteten Lackteile abzulösen. Dazu wird eine Natriumhydroxid-Lösung, mit einem Mischungsverhältnis von 1:100, verwendet. Die Platine kann nach etwa einer halben Minute aus der Lösung genommen und anschliessend mit Wasser abgespült werden. Das komplette Layout der Platine sollte sichtbar sein. Es dürfen keine Lackreste mehr vorhanden sein, ansonsten muss der Vorgang wiederholt werden. Ferner sollten die Platinen an den Kupferstellen glänzen an denen der Fotopositivlack abgelöst wurde. Diese Stellen werden beim Eintauchen in das Ätzbad sofort matt, was den beginnenden Ätzprozess zeigt. Ist dies nicht der Fall, muss die Entwicklung wiederholt werden. Die fertig entwickelte Platine sollte dann auf unterbrochene Verbindungen kontrolliert werden, diese können zum Beispiel mit einem wasserfesten Stift repariert werden.

3.3.4 Ätzen

Nach dem Entwickeln der Platine folgt das Wegätzen der freigelegten Kupferschicht. Als Ätzmittel wurde Natriumpersulfat eingesetzt. Das Ätzmittel muss vor Beginn, mit Hilfe eines Heizstabes, auf eine Temperatur von mindestens 40 Grad Celsius und höchstens 60 Grad Celsius gebracht werden. Zusätzlich wird in den Ätzbehälter von unten Luft geblasen um den Ätzvorgang zu beschleunigen. Nach einer ungefähren Dauer von 5-10 Minuten ist die Platine fertig geätzt und kann mit Wasser abgespült werden.

3.3.5 Säubern und Versiegeln

Nach dem Ätzen sind auf der Platine noch Rückstände von Fotopositivlack vorhanden. Dieser kann mit Spiritus entfernt werden. Die gesäuberte Platine wird abschliessend mit

Lötlack besprüht. Die Versiegelung verhindert die Oxidation der Leiterbahnen.

3.3.6 Bohren

Ist die mit Lötlack behandelte Platine trocken, kann gebohrt werden. Es wurde ein Hartmetallbohrer verwendet, da normale High-Speed-Steel-Bohrer (HSS-Bohrer) bei dem Einsatz mit Glasfaserplatinen schnell brechen können.

3.3.7 Löten

Im letzten Vorgang werden die Bauelemente, mit Hilfe eines Lötkolbens, auf die Platine gelötet. Beim Lötvorgang wird Lötzinn erhitzt. Das Flussmittel, das im Lötdraht enthalten ist, reduziert die Oxidation der metallischen Oberflächen und gewährleistet eine zuverlässige mechanische und elektrische Verbindung zwischen den Metallteilen. Fehler beim Löten können mit Hilfe einer Entlötpumpe korrigiert werden. Diese dient zum Absaugen von flüssigem Lötzinn beim Lösen einer Lötverbindung.

3.4 Hauptplatine

Die Hauptplatine, auch Controller-Board genannt, ist die zentrale Platine des Staubsaugerroboters. Auf ihr befinden sich die meisten Bauteile, um diese mittels der beiden Mikrocontroller ATMega32 zu kontrollieren. Ein Mikrocontroller ist für alle Sensoren, die sich auf der unteren Platte des Roboters befinden, zuständig. Der andere ATMega (in SMD-Bauweise) ist dementsprechend für alle Sensoren und Aktoren verantwortlich, die sich auf der oberen Platte befinden. Die Hauptplatine wurde von der Projektgruppe selbst hergestellt. Wie diese hergestellt wurde, ist im vorherigen Abschnitt 3.3 beschrieben. Der Schaltplan A.6 und das dazugehörige Layout A.9 und A.10 der zentralen Platine befindet sich im Anhang. Da die Hauptplatine umfangreich ist, wurden zur Erklärung einige Ausschnitte aus der logischen Ansicht der Schaltung extrahiert. Auf das Kapitel 2 aufbauend, wird hier die Elektronik der Bauelemente behandelt.

3.4.1 Mikrocontroller

Der Mikrocontroller ATMega32 besitzt 32 I/O-Pins verteilt auf vier Ports:

- **Port A:** PA0 bis PA7
- **Port B:** PB0 bis PB7
- **Port C:** PC0 bis PC7
- **Port D:** PD0 bis PD7

In den nachfolgenden Abschnitten wird beschrieben, welcher Pin des Mikrocontrollers wie belegt ist und mit welchem Bauteil er verbunden wird. Zum besseren Verständnis folgt eine Liste mit der Beschreibung der Pinbelegung. Für weitere Details des Bausteins wird hier auf den Abschnitt 3.2 verwiesen. Die 32 I/O-Pins können in acht verschiedene Bereiche eingeteilt werden:

1. Analog-Digital-Wandler
   - `PA0 bis PA7:` ADC0 bis ADC7 sind Eingabepins für analoge Messungen
2. Analog-Komparator
   - `PB2:` AIN0 ist ein externer Eingang des Analog-Komparators, INT2 ist eine externe Interrupt-Leitung
   - `PB3:` AIN1 folgt analog zu AIN0, OC0 kann als Ausgang für Zähler0 verwendet werden (Beispiel: PWM-Modus)
3. SPI-Schnittstelle
   - `PB4:` SS gehört zum SPI-Interface, dieser wird benötigt, um den richtigen Slave am Bus zu wählen
   - `PB5:` MOSI gehört zum SPI-Interface, dieser Datenausgang (als Master) oder Dateneingang (als Slave) wird bei In-System-Programmierung (ISP) verwendet
   - `PB6:` MISO gehört zum SPI-Interface, dieser Dateneingang (als Master) oder Datenausgang (als Slave) wird bei ISP verwendet
   - `PB7:` SCK gehört zum SPI-Interface, dieser Bustakt vom Master wird bei ISP verwendet
4. Timer und PWM
   - `PB0:` T0 kann als Taktgeber für die Zähler 0 und 1 verwendet werden, TCK als externer Taktpin für synchrone USART-Übertragung
   - `PB1:` T1 kann als Taktgeber für die Zähler 0 und 1 verwendet werden
   - `PD4, PD5:` OC1A, OC1B können als Ausgänge für den Zähler1 verwendet werden, beide Pins sind an einem Zähler, es sind aber dennoch verschiedene Signale möglich (Beispiel: PWM)
   - `PD6:` ICP1 kann zusammen mit Zähler1 verwendet werden um den aktuellen Wert einzufangen
   - `PD7:` OC2 kann als Ausgang für Zähler2 verwendet werden
5. I^2C-Schnittstelle
   - `PC0:` SDA ist die Datenleitung der I^2C-Schnittstelle

- `PC1:` SCL ist die Clockleitung der I^2C-Schnittstelle

6. JTAG-Interface
   - `PC2, PC3, PC4, PC5:` TCK, TMS, TDO und TDI gehören zum JTAG-Debug Interface, worüber der Mikrocontroller programmiert und debuggt werden kann
   - `PC6, PC7:` TOSC1, TOSC2 kann als externer Taktgeber für Zähler2 angeschlossen werden
7. Serielle Schnittstelle
   - `PD0:` RX ist Eingang der seriellen Schnittstelle (USART)
   - `PD1:` TXD ist der Ausgang für die Serielle Schnittstelle und externe Takt für den USART
8. Externe Interrupts
   - `PD2, PD3:` INT0, INT1 sind externe Interrupt-Leitungen

Des Weiteren ist der ATMega32 ausgestattet mit:

- `Reset:` Setzt den Controller zurück, wenn ein LOW-Pegel für mindestens zwei Zyklen des Systemtaktes anliegt
- `XTAL1, XTAL2:` Anschluss eines externen Taktgebers (hier: Quarz)
- `AVCC:` Betriebsspannung für den Analog-Digital-Wandler
- `AGND:` Alternative Masse
- `AREF:` Referenzspannung für den Analog-Digital-Wandler
- `VCC:` Versorgungsspannung
- `GND:` Masse

3.4.2 Quarz

Der ATMega32 benötigt ein Quarz-Bauelement mit einer Taktfrequenz von bis zu 16Mhz. Obwohl der ATMega32 einen internen Oszillator besitzt, dient der externe Quarz dazu, dem Mikrocontroller einen genaueren Takt zu geben. Ein Pin vom Quarz ist mit dem XTAL1-Eingang des Mikrocontrollers verbunden und der andere Pin mit dem XTAL2-Eingang. Die Polung des direkt anschließbaren Quarzes ist irrelevant. Beide Pins des Quarzes werden zusätzlich über einen $22pF$-Kondensator mit der Masse verbunden. Die beiden Quarze werden im Schaltplan (siehe Anhang) mit Q3 und Q4 bezeichnet.

3.4.3 Kompasssensor und Gumstix

Abbildung 3.9 zeigt, dass Kompasssensor und Gumstix über eine Steckverbindung an die Hauptplatine anzuschließen sind. Hierfür werden die Pins PC1 (SDA) und PC0 (SCL) des SMD-ATMega32 benutzt. Beide Leitungen sind über zwei PullUp-Widerstände mit jeweils 47k Ω an eine 3,3V Leitung verbunden.

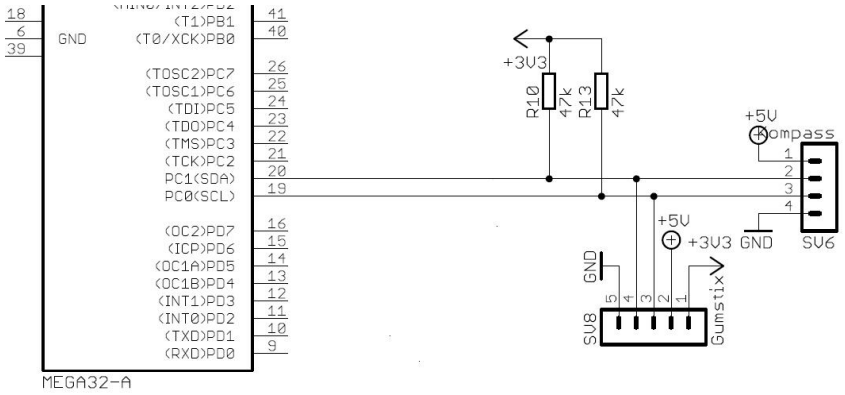


Abbildung 3.9: Realisierung der Hauptplatine - Kompasssensor und Gumstix

3.4.4 Reset

Es ist möglich den Roboter durch einen Taster neu zu starten. Abbildung 3.10 zeigt, wie der Schalter S3 an den invertierenden RESET-Pin des Mikrocontrollers angeschlossen ist. Ist der Taster gedrückt, wird GND durchgeschaltet und der Vorgang ausgelöst. Über einen $10k\Omega$ PullUp-Widerstand bekommt der Taster eine Spannung von 5V. Dies ist nötig, damit er bei unbeschaltetem Zustand einen HIGH-Pegel hat und sich nicht zurücksetzt. Der Kondensator hat eine Kapazität von 100nF. Zusätzlich ist die Leitung an den zweiten Mikrocontroller angeschlossen, so dass bei Betätigung auch beide neu gestartet werden.

3.4.5 Radencoder

Da die Bauelemente CNY70 sich sehr nah an den Drehscheiben der Räder befinden müssen, wurde für jeden dieser Sensoren eine gesonderte Platine zur Stabilisierung bereitgestellt. Auf ihr befindet sich außerdem jeweils ein Vorwiderstand der IR-LED des CNY70 mit 100Ω. Die Signale der Lichtsensoren werden über die Kabel zu einem 40-poligen Stecker auf der Hauptplatine gesendet. Dort sind die Widerstände für den IR-Fototransistor mit jeweils 680Ω montiert, was in der Abbildung 3.11 des Schaltplanauschnittes zu sehen ist. Die beiden Pullup-Widerstände vor dem invertierenden Schmitttriggern, mit IC13A und IC13B bezeichnet, betragen $47k\Omega$. Sie sind an die 5V Spannungsquelle angeschlossen. Der Schmitttrigger gibt das invertierte Ausgangssignal, an den Mikrocontroller weiter: Pin PD2 (INT0) und PD3 (INT1) werden dafür verwendet.

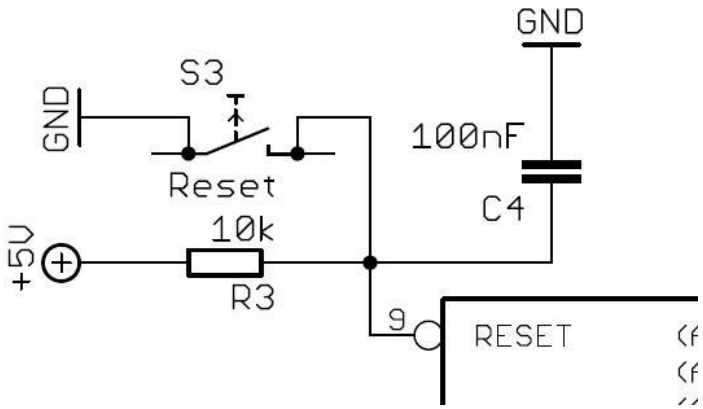


Abbildung 3.10: Realisierung der Hauptplatine - Reset

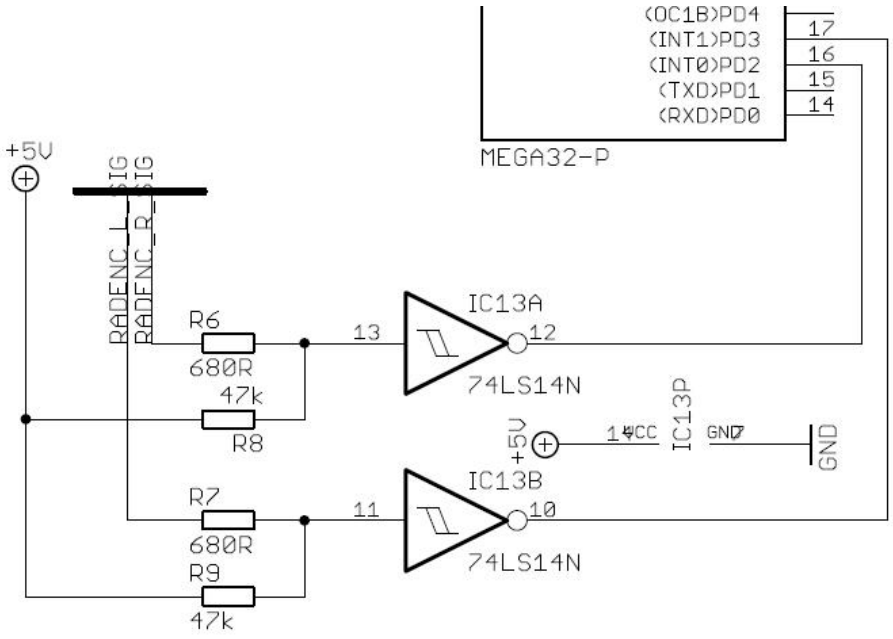


Abbildung 3.11: Realisierung der Hauptplatine - Radencoder

3.4.6 Ultraschallsensoren

Wie in Abschnitt 2.1.2 des zweiten Kapitels bereits erwähnt, werden die Ultraschallsensoren im Modus 2 betrieben. Das bedeutet, dass diese eine gemeinsame Trigger- und Echoleitung benutzen. Hierfür werden die zwei entsprechenden Pins auf GND gesetzt, die die Hauptplatine zur Verfügung stellt. Ebenfalls wird eine 5V Spannungsquelle für einen Pin bereitgestellt. Die sechs Ultraschallsensoren, bzw. dessen Trigger-/Echo-Leitungen sind an die Pins PA0 (ADC0), PA1 (ADC1), PA2 (ADC2), PA3 (ADC3), PA4 (ADC4) und PA5 (ADC5) des SMD-Mikrocontrollers angeschlossen.

3.4.7 Leuchtdioden

RAPTOR besitzt mehrere verschiedene LEDs, die unterschiedlichen Zwecken dienen:

- Betriebszustand
- Akkumulatorzustand
- Bodenbeleuchtung

Für den Betriebszustand und den Akkumulatorzustand existieren am Gehäuse des Roboters zwei LEDs mit einem Durchmesser von 10mm:

Der Nutzer wird über eine rote LED informiert, wenn die Akkuleistung sinkt. Diese Information wird benötigt, da der Prototyp keine Aufladestation hat, die er selbstständig anfahren könnte. Wird der Roboter in Betrieb genommen, signalisiert er dies mit einer gelben LED an dem Gehäuse.

Beide Leuchtdioden wefrden über den Pin PB4 (SS) des Mikrocontrollers in SMD-Bauform geschaltet. Die Vorwiderstände der beiden LEDs betragen 160Ω und befinden sich auf der Hauptplatine.

Damit die Kamera genügend Licht zum Fotografieren hat, besitzt der Staubsaugerroboter auf seiner untersten Ebene sechs Leuchtdioden mit jeweils einem Durchmesser von 5mm. Der Pin PB3 (AIN1/DC0) des ATMega32 benutzt einen Vorwiderstand von 100Ω zum N-MOSFET-Transistor (BD139). Wenn das Licht der Bodenbeleuchtung für die Kamera benötigt wird, wird der Transistor vom Mikrocontroller in Durchlassrichtung gesetzt und die sechs Leuchtdioden die durch Kabel mit der Hauptplatine verbunden ist, erhalten die benötigte Spannung. Die LEDs haben jeweils zusätzlich einen Vorwiderstand von 160Ω.

3.4.8 Motoren

Der RAPTOR besitzt vier Getriebemotoren: zwei für die Räder und zwei für die vorderen Bürsten. Für einen autonomen Staubsaugerroboter muss die Geschwindigkeit und die Fahrtrichtung steuerbar sein. Hierfür muss die Drehrichtung und die Anzahl der Umdrehungen der Motoren variiert werden können. Die Realisierung geschieht über den Baustein L293D. Da die Motoren mehr Strom benötigen als ein Mikrocontrollerausgang liefern kann, muss die Ansteuerung über einen Motortreiber erfolgen.

Die Eingänge sind TTL-Kompatibel (Transistor-Transistor-Logik) und der Baustein hat einen integrierten Überspannungs- und Temperaturschutz. Der Spannungsbereich liegt zwischen 4,5V und 36V und der Ausgangsstrom pro Ausgang beträgt 500mA. Der L293D-Baustein besitzt drei Eingänge und zwei Ausgänge, womit zwei Motoren gesteuert werden können. Für die Hauptplatine reichen somit zwei Motortreiber aus um die Rad- und Bürstenmotoren zu betreiben. Die zwei A-Eingänge pro Kanal bestimmen durch unterschiedliche Polarität die Drehrichtung des Motors, bei zwei gleichen Signalen (LOW oder HIGH) stoppt der Motor. Zudem gibt es pro Kanal noch einen EN-Eingang, wodurch erst bei gesetztem HIGH-Signal ein Stromfluss entsteht, der den Motor vorantreibt. Zur Spannungsversorgung besitzt der Baustein zwei Eingänge: VCC1 und VCC2. VCC1 liefert den Strom für den Baustein selber, während durch den VCC2 -Eingang eine externe Spannungsversorgung der Motoren realisiert werden kann. Diese Funktionalität wird ausgenutzt, um den Motor mit den durch den gewählten Akku maximal möglichen 12V zu betreiben.

Im Folgenden wird näher auf die Realisierung der Motoren auf der Hauptplatine eingegangen.

Bürsten

Die vorderen beiden Bürsten haben eine festgelegte Motordrehrichtung: die rechte Bürste dreht gegen den Uhrzeigersinn, die linke Bürste im Uhrzeigersinn. Damit drehen beide Bürsten nach innen um den Schmutz in die Richtung der Saugöffnung zu befördern. Abbildung 3.12 zeigt, wie zum Festlegen der Drehrichtung die Eingänge 1A und 2A für den linken Motor und 3A und 4A für den rechten Motor geschaltet werden. Somit ist es nicht mehr notwendig, diese per Software zu steuern. Ein weiterer Vorteil ist, dass dafür keine zusätzlichen Pins am ATMega32 belegt werden.

Die Pins 1Y und 2Y werden mit Hilfe einer Steckverbindung über Kabel an den ersten Motor gelötet, analog gilt das für die Pins 3Y und 4Y und den zweiten Motor. Die Anschlüsse GND1, GND2, GND3 und GND4 müssen geerdet werden. Die maximale Fahrtgeschwindigkeit des Roboters wird durch die an VCC2 anliegende Spannung gesteuert, dies entspricht demnach 12V Akkuleistung. Wünschenswert ist hierbei noch eine Regulierung der Fahrtgeschwindigkeit. An VCC1 liegen 5V an, die die erforderliche Spannung für den Motortreiber-IC liefern. Die Pins 1-2EN und 3-4EN (Enable-Eingänge für den rechten und linken Motor) sind ebenfalls beide mit der 5V-Spannungsquelle verbunden. Auch hier ist der Vorteil, dass keine zusätzliche Steuerung per Software erfolgen muss und auch keine zusätzlichen Pins des Atmel-Bausteins belegt sind. Der ATMega32 steuert die Bürsten über seinen Pin PD6 (ICP).

Räder

Für die Motorsteuerung des Rades ist wichtig zu beachten, dass der Roboter sowohl rückwärts, als auch vorwärts mit unterschiedlichen Geschwindigkeiten fahren kann. Anders als bei der Ansteuerung des Motortreiber-ICs für die Bürsten, ist hier die Drehrichtung des Motors nicht festgelegt. Wie Abbildung 3.13 zeigt, sind die Pins GND1, GND2, GND3

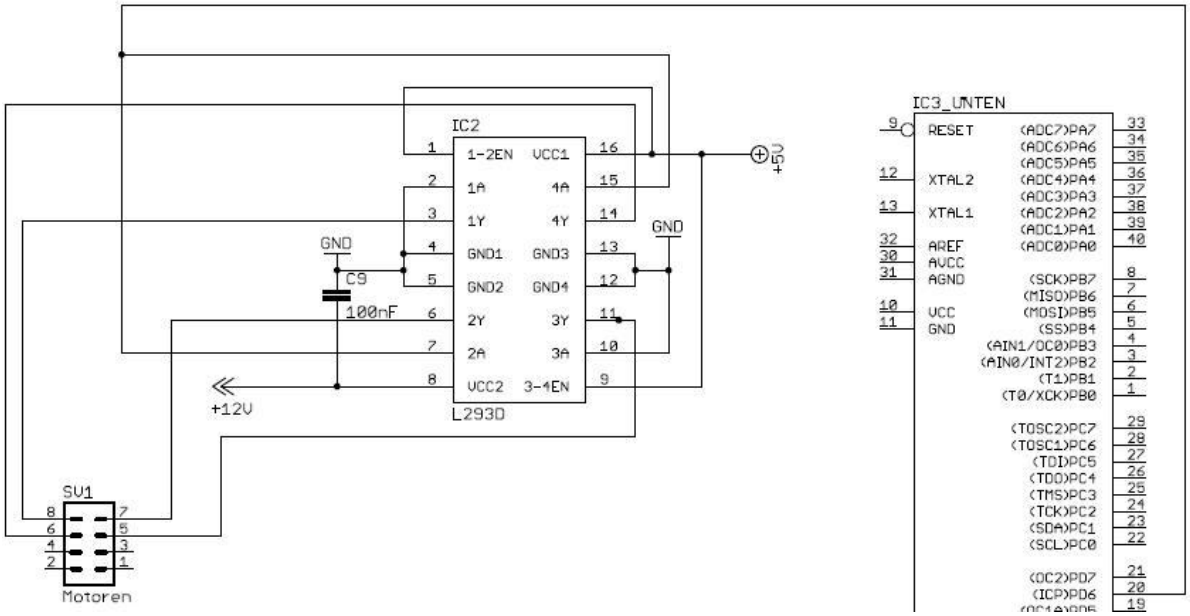


Abbildung 3.12: Realisierung der Hauptplatine - Bürstenmotor

und GND4 auf den logischen Wert Null gesetzt. Die Pins 1Y und 2Y werden über eine Steckverbindung an den ersten Motor der Räder befestigt, die Pins 3Y und 4Y an den zweiten Motor der Räder. Der L293D erhält am Pin VCC1 5V Spannung, für die logische Schaltung, und an VCC2 12V Spannung für die Radmotoren. Um den Motor über die Software ansprechen zu können, besitzt er vier Anschlusspins am ATMega32. Pin 1A kommuniziert mit PC7 des Mikrocontrollers, Pin 2A mit PC6, Pin 4A mit PC5 und Pin 3A mit PC4. Für die Pulsweitenmodulation (PWM) kommuniziert 1-2EN des Motortreiber-IC´s mit PD4 des Mikrocontrollers. Der Pin 3-4EN ist mit PD5 verbunden. Mittels PWM wird die Fahrtgeschwindigkeit gesetzt. Es werden Impulse mit voller Spannung aber variabler Breite an einen Verbraucher gesendet. Dieses modulierte Rechtecksignal wird in einer konstanten Frequenz generiert. Die Breite der Schaltimpulse wird durch das Tastverhältnis (das Verhältnis der Länge eines HIGH- zu einem LOW-Signal innerhalb einer Periode) bestimmt. Dies hat den Vorteil, dass keine Spannung geregelt werden muss. Das PWM-Signal wird den beiden EN-Eingängen des Motortreibers angelegt und durch die Trägheit der Motoren kann, bei entsprechender Pulsbreite, die Geschwindigkeit reguliert werden.

3.4.9 Stromversorgung

Der Akku Kung Long der gleichnamigen Firma [17] ist ein wartungsfreier Blei-Gel-Akku mit einer Spannung von 12V und 4,5Ah. Über eine Steckverbindung ist der Akku mit der Hauptplatine direkt verbunden und liefert die Spannung für die Motoren. Ist die Akkuleistung fast verbraucht, sollte der RAPTOR ausgeschaltet werden um Schäden an den Bauteilen zu vermeiden. Hierfür befindet sich für die 12V ein zusätzlicher Spannungstei-

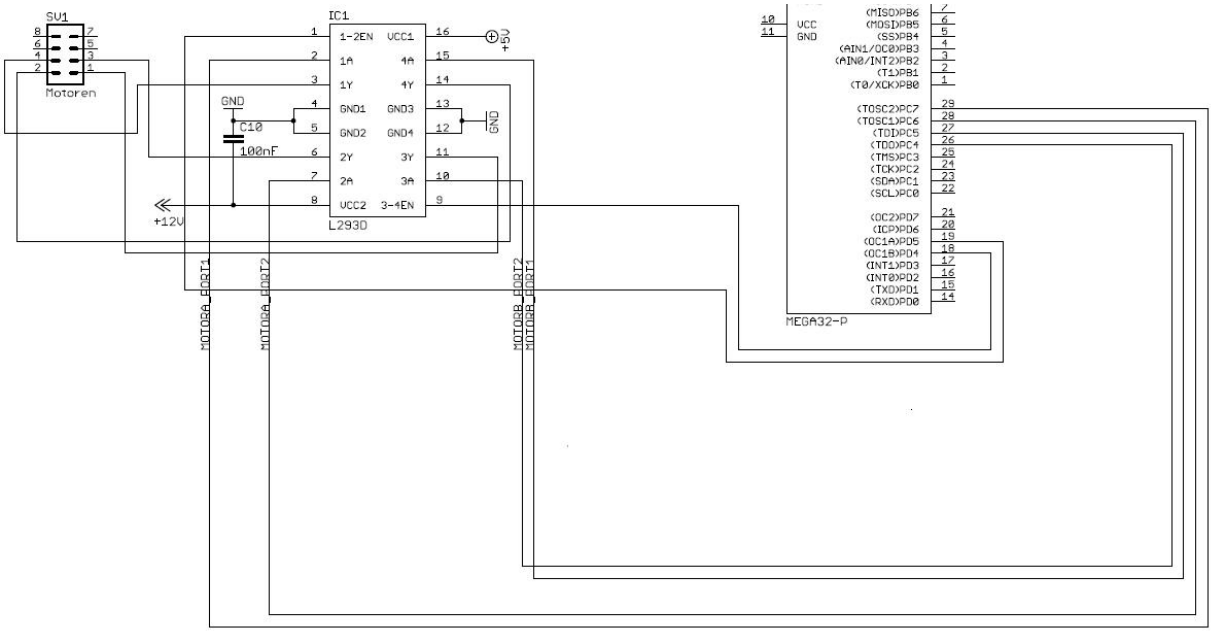


Abbildung 3.13: Realisierung der Hauptplatine - Radmotor

ler auf der Hauptplatine. In Reihe sind die Widerstände $1k\Omega$ und $2k\Omega$ geschaltet. Die Akku-Messung geschieht am Pin PA7 (ADC7) des ATMega32.

Die Sensoren und die Mikrocontroller brauchen 5V Versorgungsspannung. Diese werden über eine zusätzliche Platine gewährleistet. Der Schaltplan hierfür befindet sich in Abbildung 3.14. Hierbei sei zu erwähnen, dass nicht nur die Hauptplatine mit den 5V Spannung der Platine versorgt wird, sondern auch der USB-Hub. Um die Spannung des Akkumulators runter zu regulieren, benutzt die Projektgruppe den Spannungsregler 7805. Der gleiche Schaltungsaufbau mit dem Spannungsregler 7803 anstelle des Bauteils 7805, kann benutzt werden, um die gewünschten 3,3V Spannung für den Gumstix zu erhalten. Da bei der Regelung Hitze erzeugt wird, muss diese durch einen Kühlkörper abgeführt werden. Die Keramikkondensatoren C3 und C4 besitzen 22pF und haben die Funktion eines Tiefpasses. Der Tiefpass soll dafür sorgen, dass die geregelte Spannung am Ausgang der Schaltung sich bei den 3,3V bzw. 5V stabilisiert. Die Elkokondensatoren C1 mit $1000\mu F$ und C2 mit $100nF$ dienen zum Glätten des Signals. Die Schottky-Diode verhindert einen Rückfluß des Stroms.

Leider hat diese Schaltung den Nachteil, dass sie nicht gegen Verpolung geschützt ist. Eine Verpolung kann zur Zerstörung des Spannungsreglers führen. Wenn z.B. der Elkokondensator am Ausgang der Schaltung sich zu schnell entlädt, so entlädt er sich über den Spannungsregler. Die eingesetzte Schottky-Diode übernimmt den so entstandenen inversen Stromfluss. Des Weiteren ist die Schaltung für den Gumstix nicht einsetzbar, da keine Spannungsspitzen abgefangen werden. Treten Spannungsspitzen auf, so kann der Gumstix dadurch zerstört werden.

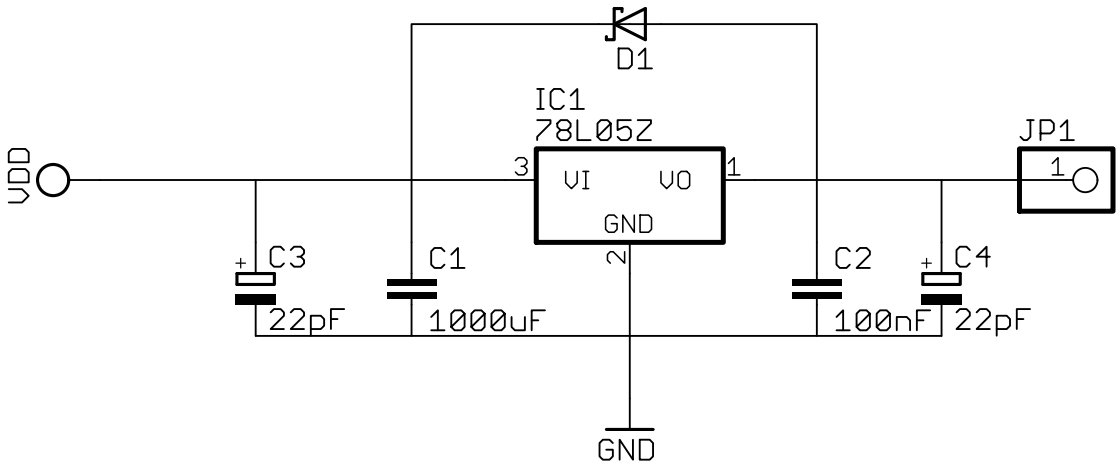


Abbildung 3.14: Realisierung der Stromversorgung

3.4.10 Kontaktsensoren

Wie in Abschnitt 2.1.3 schon erwähnt, ist der vordere Bereich des Staubsaugerroboters mit fünf Kontaktsensoren (sogenannte Bumper) ausgestattet. Falls der autonome Roboter auf ein Hindernis trifft, wird dem Mikrocontroller dies mitgeteilt, so dass dieser entsprechend reagieren kann. Bei dieser Implementierung kann nicht erkannt werden, welcher Bumper ausgelöst wurde. Im idealen Fall sollte der RAPTOR es nicht zu einer Kollision kommen lassen, da die Ultraschallsensoren Hindernisse vorher erkennen sollten. Die Bumper sind demzufolge eine Absicherung gegen eine Fehlfunktion der Ultraschallsensoren. Die fünf Kontaktsensoren sind über einen 40poligen Stecker, der sich auf der Hauptplatine befindet, mit GND verbunden. Als Signalleitung wird der Pin PB2 (AIN0/INT2) des ATMega32 verwendet. Hier befindet sich auch die 5V Versorgungsspannung für die Bumper mit einem Vorwiderstand der $1k\Omega$ beträgt. Sollte ein Kontaktsensor ausgelöst werden, setzt sich GND gegenüber HIGH durch.

3.4.11 Abgrundsensoren

Für die Abgrunddetektion werden ebenfalls Lichtsensoren vom Typ CNY70 verwendet. Wie bei den Radsensoren der Hauptplatine, müssen die Sensoren sehr nah am zu messenden Gegenstand sein. In diesem Fall müssen sie sehr nah am Boden angebracht werden. Die drei Lichtsensoren befinden sich auf Zusatzplatinen. Über Kabel werden diese mit den 40poligen Stecker der Hauptplatine verbunden. Die Vorwiderstände der Lichtsensoren betragen auch hier 100Ω, die PullUp-Widerstände $47k\Omega$. Die drei Abgrundsensoren sind mit den Pins PA0 (ADC0), PA1 (ADC1) und PA2 (ADC2) des ATMega32 verbunden.

3.4.12 Maussensor

Für die korrekte Ermittlung der abgefahrenen Strecke muss der optische Maussensor sehr nah am Boden angebracht werden. Aus diesem Grund wurde der Maussensor auf einer eigenen Platine untergebracht. Als Grundlage für die Schaltung diente der Schaltplan vom c't-Bot [8], welcher entsprechend modifiziert wurde. Im Anhang befindet sich der neue Schaltplan A.5 des Maussensors. Die Verbindung zum Mikrocontroller erfolgt über einen vierpoligen Stecker, welcher Anschlüsse für 5V, Ground, SDIO (Datenleitung) und SCK (Taktleitung) bereitstellt. An die beiden Pins OSC-IN und OSC-OUT des Sensorchips wird der Quarz angeschlossen, welcher den Takt für die Bildaufnahmen vorgibt. Die 5V-Leitung wird an den VDD-Pin des Chips geführt, außerdem speist sie über einen Vorwiderstand von 100Ω die Leuchtdiode. Der zweite Pin der Diode wird an den BC557-Transistor angeschlossen. Die Schaltung des Transistors wird über den LED-CNTL-Pin gesteuert. Schaltet der Transistor, so liegt am zweiten Pin der Diode Ground an. Des Weiteren wird die Ground-Leitung über einen $1\mu F$-Kondensator an den REFA-Pin angeschlossen, und direkt an den GND-Pin des Sensorchips. Ein weiterer $100nF$-Kondensator befindet sich zwischen den Ground und 5V-Leitungen. Die SDIO- und SCK-Leitungen werden direkt vom Stecker an die entsprechenden Pins des Chips geführt.

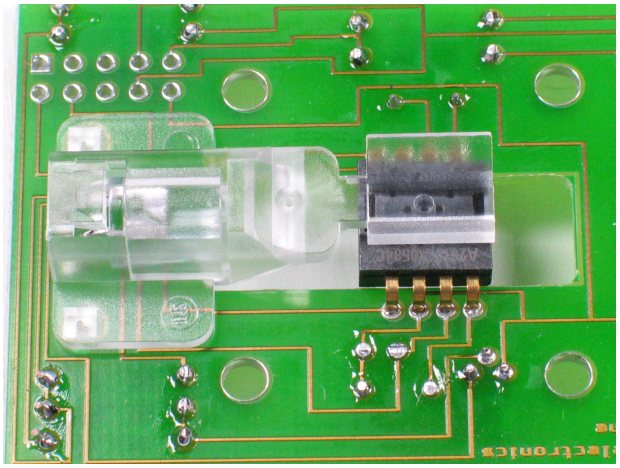


Abbildung 3.15: Der optische Maussensor [8]

3.5 RS232-I2C-Adapter

Der größte Teil der Hardware wird per I^2C angesprochen; da dieser Bus nicht zur Grundausstattung der meisten Rechner gehört, war zu Beginn eine Ansteuerung der Hardware ohne Gumstix nicht möglich. Stellenweise, insbesondere bei Fehlersuche und Tests, war jedoch auch eine Ansteuerung über "normale" PC-Hardware erwünscht und nötig.

Um auch mit regulärer PC-Hardware auf den RAPTOR zuzugreifen, wurde ein RS232-I^2C-Adapter konzipiert und hergestellt. Als gegen Ende der Projektgruppe das Gumstix-

Board nicht mehr zur Verfügung stand und ein PC-Laptop verwendet werden musste, wurde dieser Adapter notwendig, um mit der Hardware kommunizieren zu können.

3.5.1 Beschaltung

Die Schaltung (Abbildung A.7) enthält im Wesentlichen nur die Komponenten, die für den reinen Betrieb des ATMega32, die serielle Kommunikation zum PC sowie zum Ansprechen des I^2C-Bus notwendig sind. Die PullUp-Widerstände können abgeschaltet werden, damit das Board kompatibel zu Hardware ist, die bereits PullUp-Widerstände enthält. Später wurde der Quarz von 16MHz gegen einen Baudratenquarz ausgetauscht, um auch höhere Kommunikationsgeschwindigkeiten fehlerlos zu erreichen.

3.5.2 Protokoll

Während bei einer I^2C-Verbindung die Teilnehmer adressiert werden können, ist RS232 eine direkte Kommunikation zwischen zwei Geräten. Es musste also eine Art Protokoll geschaffen werden, mit welchem die Geräte am Bus adressiert sowie der Lese-/Schreibmodus ausgewählt werden konnte.

Die Kommunikation geht immer vom PC aus, jede Kommunikation über die serielle Schnittstelle beginnt mit einem Steuerbyte:

7 Das höchstwertige Bit wählt den Kommunikationsmodus. Ist dieses Bit gesetzt, soll von einem Busgerät gelesen werden. Ist dieses Bit nicht gesetzt, so soll auf dem Gerät geschrieben werden.

6,5 Adressauswahl, im Raptor werden lediglich drei I^2C-Geräte verwendet, daher reichen zwei Bit zur Adressierung aus. Intern wird die Konstante 96 addiert, somit steht `00` für den Kompass auf Adresse 96.
Adresse `11` (virtuelle Adresse 99) wird für Tests verwendet und kann beispielsweise die reine RS232-Funktionalität auch ohne laufenden Bus verifizieren.

4-0 Die restlichen 5 Bits teilen mit, wie viele Bytes auf das Busgerät geschrieben respektive von dort gelesen werden sollen. Dieses Limit auf 31 Zeichen per Kommunikation stellte im RAPTOR kein Hindernis dar, da niemals mehr als 31 Zeichen pro Einzeldatenaustausch übermittelt werden mussten.

Auf diese Art und Weise war bei jeder Kommunikation nur ein zusätzliches Byte nötig; insbesondere bei häufiger Kommunikation sollte diese Methode den Geschwindigkeitsverlust durch Protokoll-Overhead minimieren.

Nach dem Steuerbyte beginnt die eigentliche Übertragung. Soll von einem Busgerät gelesen werden, so holt der Adapter die entsprechende Anzahl Zeichen und schickt sie anschließend über die serielle Schnittstelle. Wird auf ein Busgerät geschrieben, so werden zunächst alle Daten über die serielle Schnittstelle empfangen und anschließend über den Bus geschickt.

3.5.3 Software

Das Programm des Mikrocontrollers (Listing B.10) implementiert das definierte Protokoll. Um Fehler in der Kommunikation frühzeitig zu erkennen, wird jedes per RS232 vom ATMega32 empfangene Byte zusätzlich zurückgegeben (Echo-Funktionalität).

3.6 Karosserie

Nach einigen Überlegungen fiel die Entscheidung auf einen Roboter, der aus zwei Ebenen besteht. So ist es möglich, alle Komponenten problemlos unterzubringen. Die Ebenen haben eine runde Form. Dadurch wird das Drehen auf der Stelle ohne Anecken ermöglicht. Auf den Ebenen werden die zuvor vorgestellten Bauteile befestigt. Die untere Ebene beherbergt unter anderem die Motoren, den Akku und die Befestigung der Räder. Auf der oberen Ebene ist beispielsweise das Controllerboard, der USB-Hub, die Kameras sowie einige Sensoren wie der Kompasssensor und die Ultraschallsensoren angebracht. Für den Aufbau der zwei Ebenen haben wir uns für zwei runde Aluminiumplatten mit einem Durchmesser von 30cm entschieden und diese vom Institut für Spanende Fertigung an der Technischen Universität Dortmund herstellen lassen. Die beiden Platten werden durch vier Metallstangen mit einem Durchmesser von 0,5cm verbunden. Damit steht das Grundgerüst des Roboters. Die Radkonfiguration sieht zwei Räder und ein Kugelrad vor, die an die untere Ebene montiert werden. Der Durchmesser der Räder beträgt jeweils 10cm und die Breite 2,5cm. Sie werden auf den entgegengesetzten Seiten der Mittelachse der Kreisscheibe angebracht. Dadurch kann sich der Roboter auch auf der Stelle drehen. Das Kugelrad hat nur die Funktion des Stützrades, das im Gegensatz zu den zwei Rädern nicht angetrieben wird. Da die Räder und Motoren sehr viel Platz auf der unteren Platte einnehmen, wurden die meisten Komponenten auf der oberen Platte befestigt. Um die Ecken eines Raumes gut säubern zu können, wurden zwei kleine Bürsten mit einem Durchmesser von 5cm gekauft. Diese befinden sich vorne an der unteren Platte. Für den Fall, dass der Roboter nicht rechtzeitig bremst und doch gegen einen Hinderniss fährt, wurde eine Stoßstange an der untere Platte des Roboters angebracht. Hinter der Stoßstange befinden sich fünf Bumper, die bei Berührung eine Notbremsung auslösen. Die folgenden Abschnitte erläutern die Fertigungsschritte und den Einbau aller angebrachten Teile. Die genauen Konstruktionspläne sind im Anhang A zu finden und enthalten alle eingebauten Komponenten.

3.6.1 Die untere Ebene

In diesem Abschnitt wird beschrieben, welche Komponenten an der unteren Aluminiumplatte befestigt wurden. Ferner wird erklärt, wie die Platte dafür bearbeitet werden musste und was für Befestigungsmethoden ausgewählt wurden.

Bearbeitung der unteren Platte

Bevor mit der maschinellen Bearbeitung der Platte begonnen werden konnte, wurden mit Hilfe eines Zirkels zwei rechtwinklige Diagonalen durch den Mittelpunkt gezeichnet. Anhand dieser Diagonalen wurden dann die genauen Positionen für die folgenden Bearbeitungsschritte festgelegt. Mit Hilfe der Fräsmaschine des Lehrstuhls 12 wurden als erstes

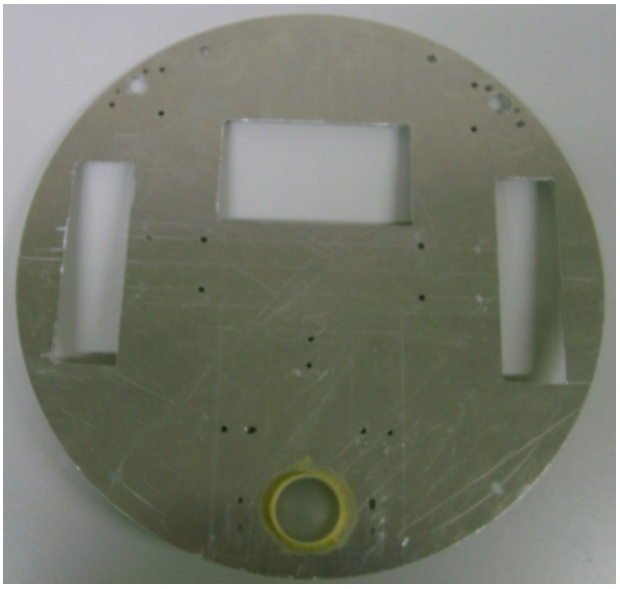


Abbildung 3.16: Bearbeitung der unteren Aluminiumplatte

die Löcher für die Räder ausgefräst. Diese sind jeweils 10,7cm lang und 3,2cm breit. Der Abstand eines Lochs vom Rand der Platte beträgt an der Diagonalen 2,5cm. Danach wurde, weiterhin mit dem Fräser, das Loch für die Bodenkamera ausgeschnitten. Das Loch für die Kamera ist 5,5cm lang und 10cm breit. Um die Bürstenmotoren an die untere Platte zu befestigen, wurden für jeden Motor jeweils vier Löcher gebohrt: ein Loch für die Achse des Motors, der später an den Bürsten befestigt wird und drei Löcher für die Befestigung der Motoren an die Platte. Da die genauen Positionen der drei Löcher zur Motor-Aluplattenbefestigung schwierig auszumessen sind, wurde dafür eine Schablone erstellt. Für das Stützrad wurde im hinteren Teil der Platte, mit einem Spezialwerkzeug für die Fräsmaschine, ein Kreis herausgeschnitten, in das die Halterung des Kugelrades gesteckt wurde. Zur besseren Befestigung wurde diese zusätzlich an der Aluplatte festgeklebt. Das Ergebnis der bisherigen Schritte ist in Abbildung 3.16 zu sehen. Nachdem diese Löcher in die untere Platte gefräst wurden, konnte mit der Befestigung der Räder begonnen werden.

Befestigung der Räder an den Motoren

Um die Räder an den Motoren und diese anschließend an der Platte zu befestigen, wurde ein zylinderförmiger Metallblock mit einer Handsäge, und danach zur Glättung mit der Drehmaschine, auf die Radbreite gebracht. Der Durchmesser hatte schon die richtige Größe.

Die klein geschnittenen Zylinderstücke passten damit genau in die Achse der Räder. Auf einer Seite der Zylinder wurde ein Loch für die Motorachse gebohrt. Senkrecht zu diesem Loch wurde noch ein kleineres Loch von 3mm Durchmesser gebohrt, wodurch der Motor an den Zylinder geschraubt werden konnte (siehe Abbildung 3.17). Auf der anderen Seite der Zylinder wurden zwei kleine Löcher gebohrt. Damit konnte ein kleines Metallstück befestigt werden, dessen Funktion es ist, die Zylinder fest mit den Rädern zu verbinden. Andernfalls hätten sich die Räder beim Fahren lösen können.

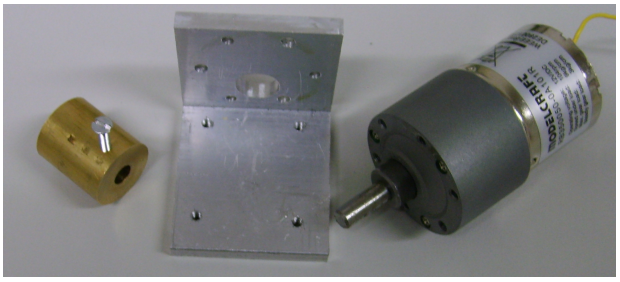


Abbildung 3.17: Bearbeitung der unteren Aluminiumplatte

An den Rädern wurden kleine Stücke mit der Fräsmaschine ausgeschnitten. Die Schraube zur Motor-Zylinderbefestigung dient damit gleichzeitig als Zylinder-Radbefestigung. Damit waren die Räder mit den Motoren verbunden, wie Abbildung 3.18 zeigt. Der nächste Schritt bestand darin die Motoren mit der unteren Ebene zu verbinden.

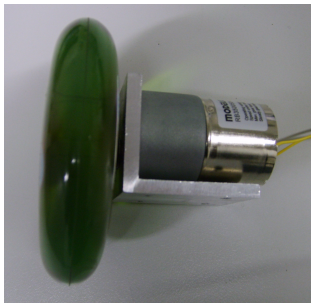


Abbildung 3.18: Bauteile zur Motormontage

Befestigung der Räder an die untere Platte

Am Lehrstuhl 12 der Technischen Universität Dortmund, waren L-förmige Metallstücke vorhanden, die durch Fräsarbeiten auf die im Anhang angegebene Größe gebracht wurden. In eine Seite des rechtwinkligen Metallstücks wurden die Löcher zur Motorbefestigung gebohrt - analog zu den Löchern in der unteren Ebene. Dann wurden sowohl auf der anderen Seite des Winkels als auch auf der Platte selbst vier kleine Löcher gebohrt. Mittels Schrauben konnte dann der Winkel an der Platte befestigt werden.

Als nächstes wurden die Motoren an die L-förmigen Metallstücke geschraubt und dann die Motoren mit den Metallstücken an den vorher bearbeiteten Zylinder geschraubt. Danach wurden die Zylinder in die Löcher der Räder gesteckt. Zuletzt wurden dann die L-förmigen Metallstücke mit den Motoren und Rädern an die Platte geschraubt (Abbildung 3.19).

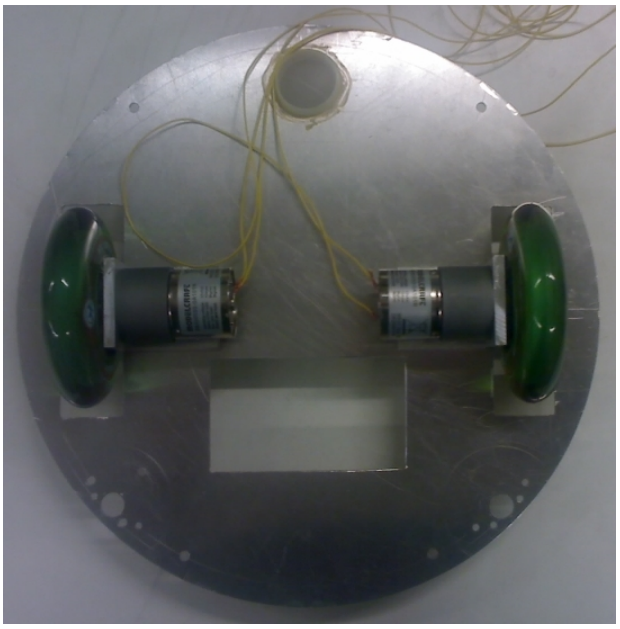


Abbildung 3.19: Befestigung der Räder an der unteren Platte

Befestigung des Kugelrads an die untere Platte

Damit der Roboter später stabil auf dem Boden steht, wurde im hinteren Abschnitt der Platte ein Kugelrad befestigt. Bei der Bearbeitung der unteren Platte wurde schon das Loch ausgeschnitten, in dem dieses Rad angebracht wird. Nach einigen Überlegungen wurde die Entscheidung getroffen, die Halterung des Kugelrades an die Platte zu kleben.

Befestigung des Akkus

Zur besseren Gewichtsverteilung wurde der Akku im hinteren Bereich der unteren Ebene angebracht. Damit die zweite Ebene möglichst tief angebracht werden kann, wurde der Akku auf der unteren Ebene liegend montiert.

Ein am Lehrstuhl vorhandenes rechteckiges Metallstück wurde durch Fräsen in eine L-Form gebracht, ähnlich wie bei der Motorbefestigung der Räder. Danach wurden Stücke mit der Handsäge klein gesägt und Löcher zur Befestigung in die Metallstücke und die Aluplatte gebohrt. Abbildung 3.20 zeigt das Resultat.

Die Metallstücke wurden dann an der Platte befestigt und der Akku eingesetzt. Durch die Platzierung der Metallstücke sitzt der Akku fest auf der unteren Ebene.

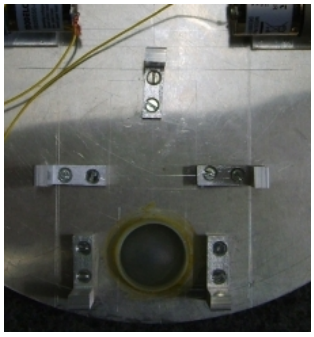


Abbildung 3.20: Befestigung des Akkus an der unteren Platte

Befestigung der Bürsten an der unteren Platte

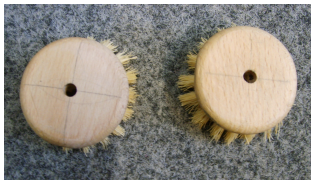


Abbildung 3.21: Bürsten

An den Bürsten, die in Abbildung 3.21 zu sehen sind, wurde jeweils ein Loch für die Befestigung der Motoren gebohrt. Dann wurden die Motoren für die Bürsten an die Platte geschraubt und die Bürsten an den Motoren befestigt (siehe Abbildung 3.22).

Befestigung der Stange an der unteren Platte

Es wurden 5mm dicke Stangen gekauft und diese in 15cm lange Stücke gesägt. Durch zuvor gebohrte Löcher in den Aluminiumplatten wurden die Stangen mit den Platten und damit letztendlich beide Platten miteinander verbunden (Abbildung 3.23).

Befestigung des Maussensors

Der Maussensor dient zur Positionsbestimmung des Roboters. Um die abgefahrene Strecke korrekt zu ermittelt, muss der Maussensor sehr nah am Boden angebracht werden. Er wurde mit 3mm Abstand zum Boden an der unteren Platte des Roboters befestigt. Die Abbildung A.2 zeigt genau an welcher Stelle der Platte der Maussensor positioniert wurde.

Besfestigung der Bumper und Stoßstange

Im idealen Fall sollte es nicht zur Kollision mit Gegenständen kommen. Falls es trotzdem einmal dazu kommt, wurden Bumper und eine Stoßstange an der unteren Platte des Roboters befestigt. Für den Aufbau werden fünf Bumper gebraucht. Diese werden möglichst

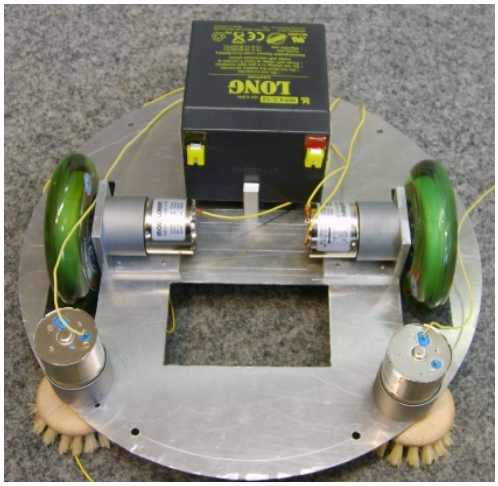


Abbildung 3.22: Befestigung der Bürsten an der unteren Platte

Abbildung 3.23: Befestigung der Stange an der unteren Platte

gleichmäßig an der vorderen Hälfte der unteren Ebene angebracht. Ein Bumper ganz vorne, die restlichen 4 jeweils zu beiden Seiten der Bürstenmotoren. Die Stoßstange wurde vor dem Bumper an der unteren Platte befestigt. Zusätzlich dienen zwei Federn dazu, die Stange nach einem Zusammenstoß mit einem Hinderniss in ihre ursprüngliche Position zu bringen. Die Skizze A.1 zeigt, an welcher Position die Bumper und die Stoßstange angebracht sind.

Befestigung der Abgrundsensoren

Um Abgründe wie beispielweise Treppen oder Tischkanten zu erkennen, wurden drei Lichtsensoren angebracht. Einer wurde vorne an der unteren Platte angebracht, die anderen beiden jeweils vor einem Rad, mit einem Abstand von 3mm zum Boden. Die Abbildung A.2 zeigt an welcher Position die Lichtsensoren angebracht worden sind.

Befestigung der Radencoder

Mit Hilfe der Radencoder kann der Roboter die zurückgelegte Wegstrecke bestimmen, oder eine bestimmte Strecke geradeaus fahren. Für den Aufbau wurde an jedem Rad eine Taktscheibe angebracht. Eine Scheibe besteht aus 24 schwarzen und 24 weißen Streifen, die kreisförmig angeordnet sind. Auf jede Taktscheibe ist ein Lichtsensor gerichtet. Die Lichtsensoren zählen bei einer Raddrehung die Streifen der Taktscheiben. Daraus kann die Anzahl der Radumdrehungen und damit auch die zurückgelegte Distanz berechnet werden.

3.6.2 Die obere Ebene

In diesem Abschnitt wird die Bearbeitung der oberen Ebene beschrieben. Dazu gehört die Befestigung der Ultraschallsensoren, des Kompasssensors, des Controllerboards, der Kameras, des USB-Hubs und der Spannungsversorgungsplatinen. Zusätzlich wurden Komponenten wie das Gitter, der I^2C-Adapter, die Status-LEDs und der An/Aus Schalter angebracht. Abschliessend wird die Montage der oberen Platte an den Stangen und die Besfestigung des Laptops an dem Roboter beschrieben.

Die Ultraschallsensoren

Um Hindernisse rechtzeitig erkennen zu können, wurden sechs Ultraschallsensoren eingebaut. Sie nehmen die Positionen Norden, Süden, Westen, Osten, Nordosten und Nordwesten ein. Diese Betrachtung bezieht sich auf die Front des Roboters. Wiederrum fiel die Entscheidung auf L-förmige Metallstücke. Nach dem Bohren von weiteren Löchern war die Befestigung der Sensoren an den Winkeln und der Winkel an der oberen Ebene möglich (Abbildung 3.24). Die Abbildung A.3 zeigt an welche Positionen die Ultraschallsensoren angebracht wurden.

Abbildung 3.24: Ultraschallsensor an dem Metallstück

Der Kompasssensor

Der Kompasssensor ermittelt anhand des Erdmagnetfeldes die genaue Himmelsrichtung in 0,1 Grad Schritten. Der Kompasssensor wurde auf einer Platine befestigt. Diese Platine, wurde dann an die obere Platte geschraubt. Die Abbildung A.3 zeigt an welcher Position der Kompass angebracht wurde.

Das Controllerboard

Das Controllerboard wird für die hohe Anzahl an Sensoren und Aktuatoren benötigt. Auf ihm befinden sich zwei Motortreiber für die Bürsten- und Radmotoren, der I^2C-Bus, die Spannungsversorgung und alle Anschlüsse für die Sensoren. Das Board wurde auf der inneren Seite der oberen Platte befestigt, damit die Kabel leichter verbunden werden können. Die Skizze der Karosserie (Abbildung A.3) zeigt, an welcher Stelle das Controllerboard angebracht wurde.

Die Kameras

Der RAPTOR benötigt zwei Kameras. Die erste, nach unten gerichtete Kamera, dient der Schmutzerkennung. Die Frontkamera dient dem Follow-Modus. In den folgenden Abschnitten wird erläutert, wo diese beiden Kameras angebracht wurden.

Kamera für die Schmutzerkennung Mit dieser Kamera der Marke Philips werden Bilder des Bodens aufgenommen und analysiert. Zuerst wurde ein Loch in die obere Platte mit der Fräsmaschine gebohrt. Um Bilder des Boden aufzunehmen, wurde die Kamera mit der Linse nach unten gerichtet an der oberen Platte befestigt. Ausserdem wurden LEDs an die Kamera angebracht um den Boden ausreichend zu beleuchten. Die Abbildung A.3 zeigt, an welcher Stelle der Karosserie die Kamera für die Schmutzerkennung vorgesehen hat.

Kamera für den Follow-Modus Die Kamera für den Follow-Modus, der näher in Abschnitt 4.9.1 erläutert wird, dient dazu Menschen zu folgen, um beispielweise Getränke einer Person hinterher zu tragen, oder vom Menschen an eine Stelle des Bodens geführt zu werden, die stark verunreinigt ist. Die Kamera wurde frontal an dem

Roboter angebracht, aber nicht direkt befestigt. Sie wird zwischen Gitter und Laptop festgeklemmt. Die Abbildung A.4 zeigt, an welcher Stelle des Roboters sich die Kamera für den Follow-Modus befindet.

Der I^2C-Adapter

Um einen Datenaustausch zwischen dem ATMega und dem Laptop zu gewährleisten, wurde ein I^2C-Adapter zur seriellen Schnittstelle hergestellt. Der Adapter wurde an ein Gitter befestigt. Zur Befestigung wurde nicht leitendes Material verwendet, um einen Kurzschluss zu vermeiden.

Der USB-Hub

Der USB-Hub stellt vier USB-Anschlüsse zu Verfügung. Er wurde auf der Innenseite der oberen Platte des Roboters angeschraubt. Die Abbildung der Karosserie A.3 zeigt seine Position.

Die Spannungsversorgungsplatinen

Damit die Platinen die gewünschte Spannung erhalten, wurden zwei Versorgungsplatinen hergestellt. Eine versorgt das Controllerboard und die andere den USB-Hub. An welcher Position auf der oberen Platte diese Platinen angebracht worden sind, ist in der Abbildung der Karosserie A.3 zu sehen.

Das Gitter

Damit die Komponenten an der Innenseite des Roboters nicht sehr deutlich zu sehen sind, wurde ein Gitter zwischen den zwei Ebenen des Roboters befestigt. Ausserdem wurden an das Gitter zusätzliche Komponenten, wie beispielsweise der An/Aus Schalter und der I^2C-Adapter, angebracht. Am oberen Rand des Gitters wurde ein PVC-Schlauch mit einem Durchmesser von 9mm befestigt, um das Risiko zu verringern sich mit den scharfen Rändern des Gitters zu verletzen.

Der Laptop

Auf dem Laptop läuft das Linux-Betriebssystem und die gesamte Software, die von der Projektgruppe entwickelt wurde. Um ihn zu befestigen wurden kleine Ecken an der oberen Seite des Gitters ausgeschnitten. Diese Ecken wurden dann mit weichem Schwammstoff gepolstert, damit keine Kratzer an ihn kommen. In der Abbildung A.4 ist zu sehen, an welcher Position sich der Laptop befindet.

Die Status-LEDs und der An/Aus Schalter

Mit den Status-LEDs wird festgestellt, ob der Roboter angeschaltet ist. Leuchtet die gelbe LED, so ist der Roboter aktiv. Leuchtet die rote Lampe, so ist die Akkuleistung knapp.

Die Status-LEDs und der An/Aus-Schalter wurden am Gitter befestigt. So sind sie leicht zu sehen und der Roboter lässt sich einfach anschalten bzw. ausschalten.

Befestigung der oberen Platte an der Stange

Nachdem alle oben genannten Komponenten an der oberen Platte befestigt wurden, konnte die Platte samt Gitter an den Stangen befestigt werden. In Abbildung 3.25 ist der fertig montierte Roboter zu sehen.

Abbildung 3.25: Der fertig montierte RAPTOR

4 Software

Nachdem das zweite Kapitel Einblick in die zu programmierende Software gab, spezialisiert sich dieses Kapitel auf die Realisierung der Software für den RAPTOR. Beginnend mit der Software für den Gumstix, wird danach das Weltmodell des Roboters vorgestellt, das die Programmierung der Gitterkarte beinhaltet. Anschließend wird die Sensordatenfusion vorgestellt, gefolgt von der Odometrie. Des Weiteren wird erklärt, wie die Kartenerstellung im RAPTOR funktioniert. Ferner wird die Simulation mit Player-Stage gezeigt, gefolgt von der Beschreibung des API-Entwurfs. Für eine bessere Interaktion zwischen Benutzer und dem Roboter dient das Webinterface. Die Realisierung wird ebenfalls in diesem Kapitel beschrieben. Die Robotersteuerung umfasst den Follow-Modus, die Bahnplanung und die Routenfahrt. Dabei ist der Follow-Modus eine zusätzliche Besonderheit, die den RAPTOR von gewöhnlichen Staubsaugerrobotern unterscheidet. Der Abschnitt Schmutzerkennung beschreibt, wie schmutzige Stellen auf dem Boden mittels einer Bodenkamera direkt erkannt werden können. Abschließend wird die Funktion des Hauptprogramms beschrieben.

4.1 Gumstix

Der Gumstix besitzt im Auslieferungszustand eine Menge Funktionen, jedoch sind nicht alle für die Nutzung des Roboters wichtig. Zum einen müssen zusätzliche Komponenten, wie der I^2C-Bus eingebunden werden, zum anderen die Funktionen der Erweiterungsboards, wie die Nutzung von LAN, oder des Micro-SD Slots, aktiviert werden. Da entsprechende Sicherheitsmaßnahmen bezüglich der Stromversorgung nicht vorgenommen wurden, ist beim Einsatz der Gumstix kaputt gegangen. Ersatzweise wurde ein Notebook bereitgestellt, welches die Funktionen des Gumstix übernommen hat. Jedoch arbeitete die Projektgruppe die meiste Zeit mit dem Gumstix, die daraus gewonnenen Erkenntnisse werden im Folgenden festgehalten:

4.1.1 Aufbau des Flash-Speichers

Neben dem 128MB großen RAM besitzt der Gumstix einen 32MB umfassenden Flash-Speicher, welcher in mehrere Sektionen aufgeteilt ist. Diese werden an dieser Stelle nur kurz aufgelistet, aber es wird nicht näher auf die genauen Spezifikationen eingegangen.

1:0 In dieser Sektion werden die relevanten Daten und Programme des Bootloaders U-Boot gespeichert. Sollten diese Daten beim Flashen beschädigt werden, wäre es nicht mehr ohne Weiteres möglich den Gumstix zu reparieren. **1:1** Diese Sektion ist reserviert für dynamische Daten, wie zum Beispiel Laufzeitvariablen. U-Boot besitzt einen Softwareschutz

um nicht zufällig die ersten beiden Sektionen zu überschreiben. **1:2** Hier lässt sich das JFFS2 Dateisystem wiederfinden, welches die Root Partition für das Betriebssystem ist. Im Dateisystem ist auch der Linux Kernel enthalten, sowie die Betriebsumgebung.

4.1.2 U-Boot

Bei U-Boot handelt es sich um den Bootloader des Gumstix. Mit diesem gibt es die Möglichkeit ein falsch geflashtes Dateisystem oder einen falsch geflashten Kernel wieder zu reparieren, indem das Dateisystem oder der Kernel noch mal über den Bootloader geflasht wird. Um in die Konsole des Bootloaders zu gelangen, muss sofort nach dem Einschalten des Gumstix eine beliebige Taste gedrückt werden. Sonst wird nach nur 3 Sekunden der Kernel normal gebootet.

4.1.3 Inbetriebnahme

Bevor der Gumstix in Betrieb genommen wird, muss sichergestellt werden, dass eine Stromquelle vorhanden ist, die bei einer Spannung ab 3.3V konstant 600mA Gleichstrom liefert. Ab 4V wird jedoch vermehrt Hitze produziert, was zu Komplikationen führen kann, wenn der Gumstix nicht hinreichend gekühlt wird. Durch die zusätzlichen Geräte, wie SD-Cardreader und Ethernet, hat sich bei geringerer Stromstärke der Gumstix immer wieder nach dem Bootvorgang abgeschaltet.

4.1.4 Konfiguration des Betriebssystems

Als Erstes muss der vom console-vx Zusatzboard bereitgestellte Ethernet-Controller und der micro-SD-Card Slot in Betrieb genommen werden. Diese werden in der Datei /etc/-modules aktiviert. Hierbei kann auch sofort die Host-Funktion der USB Schnittstelle eingebunden werden. Dazu müssen die jeweils vorhandenen Zeilen durch die folgenden Zeilen ersetzt werden:

```
1 # MMC support
2 mmc_block
3 pxamci
4
5 # Ethernet support
6 smc911x
7
8 # USB-Host support
9 ohci-hcd
```

Listing 4.1: /etc/modules

Nach dem nächsten Bootvorgang sollte nun eine micro-SD Karte in das Verzeichnis /mnt/mmc gemountet werden. Für die Ethernetnutzung muss in der Datei /etc/modprobe.conf noch folgende Zeilen

```
1 alias eth0 smc91x
2 alias eth1 smc91x
```

Listing 4.2: /etc/modprobe.conf

durch

```
1 alias eth0 smc911x
2 alias eth1 smc911x
```

Listing 4.3: /etc/modprobe.conf

ersetzt werden. Als Nächstes wird nun der Ethernettreiber eingebunden. Durch den Kompiliervorgang von Buildroot entstehen die in Listing 4.4 aufgeführten Dateien. Diese Dateien müssen nun mit Hilfe einer micro-SD Karte oder einer seriellen Verbindung auf den Gumstix kopiert werden. Die ersten beiden Module müssen in das Verzeichnis /lib/modules/2.6.21gum/kernel/drivers/net/ und die dritte Datei in das Verzeichnis /lib/modules/2.6.21gum/kernel/drivers/usb/host/. Optional kann nun die Bluetooth Funktion, sofern sie nicht genutzt wird, abgeschaltet werden, indem im Verzeichnis /etc/init.d die Datei S30bluetooth gelöscht oder umbenannt wird. Nach einem Neustart des Gumstix kann nun über den Befehl ifconfig festgestellt werden, ob der Ethernet-Controller korrekt eingebunden ist.

```
1  ../gumstix-buildroot/build_arm_nofpu/linux-2.6.21gum/
2    drivers/net/gumstix-smc911x.ko
3  ../gumstix-buildroot/build_arm_nofpu/linux-2.6.21gum/
4    drivers/net/smc911x.ko
5  ../gumstix-buildroot/build_arm_nofpu/linux-2.6.21gum/
6    drivers/usb/host/ohci-hcd.ko
```

Listing 4.4: Ethernet-Treiber-Dateien

4.1.5 Flashen des Kernels/Dateisystems

Um die Software auf dem Gumstix auf dem neuesten Stand zu halten, werden des Öfterne Änderungen am Kernel und auch am Dateisystem vorgenommen. Es wurde daher ein Verfahren entwickelt, welches eine sichere Variante für diesen Vorgang darstellt. Im Folgenden wird deshalb das Flashen des Kernel über das Betriebssystems und den Bootloader beschrieben.

Über das Betriebssystem

Natürlich gibt es auch die Möglichkeit einen kompletten, neuen Kernel oder ein anderes Dateisystem auf den Gumstix zu überspielen. Hierfür werden folgende Dateien benötigt:

- rootfs.arm_nofpu.jffs2 (das Root-Filesystem)

- uImage (der Kernel)

Diese Dateien werden wieder auf den Gumstix kopiert. Die Befehle *df* oder *mount* sollten anzeigen, in welchem Bereich im Speicher das Root-Image liegt. Das Ergebnis sollte dem Beispiel Folgendem ähneln.

```
1 /dev/mtdblock1 on / type jffs2 (rw,noatime)
```

Listing 4.5: in der Konsole

Sollte aber etwas anderes als mtdblock1 ausgegeben werden, muss beim Flashen auch eine andere Zieladresse angegeben werden. Zunächst einmal muss jedoch das Dateisystem neu eingebunden werden - allerdings aus Sicherheitsgründen im Nur-Lese-Modus (ro). Das eigentliche Flashen geschieht dann über folgende Befehls-Reihe:

```
1 # modprobe sa1100\_wdt margin=1
2 # mount -o remount,ro /
3 # echo && flashcp -v /tmp/rootfs.arm_nofpu.jffs2
4   /dev/mtd1 && flashcp /tmp/uImage /dev/mtd2
5    && echo > /dev/watchdog
```

Listing 4.6: Flashen des Dateisystems

Der Watchdog sorgt dafür, dass der Gumstix nach dem Flashen automatisch neu gestartet wird. Am Terminalprogramm sollte der Reboot verfolgt werden können. Sollte alles funktionieren, startet der Gumstix nach der U-Boot-Wartezeit sofort das neue Image.

Über den Bootloader

Der Kernel oder das Dateisystem sollte nur im Bootloader geflasht werden, wenn das Betriebssystem nicht mehr funktioniert, weil das Flashen im Bootloader bis zu einer Stunde dauern kann und durch Tippfehler die ganze Prozedur von neuem beginnt. Als Erstes muss der Gumstix an einen seriellen Port angeschlossen werden. Mit dem Befehl *minicom -s* wird auf dem angeschlossenen Rechner das Konsolenprogramm gestartet, das einen Zugriff auf die Konsole des Gumstix ermöglicht. Doch zuerst müssen im Menü folgende Parameter eingestellt werden:

- Serial Device: /dev/ttyS0
- Bps/Par/Bits: 115200 8N1
- Hardware Flow Control: No
- Software Flow Control: No

Nachdem die Daten gespeichert sind und die Konfiguration verlassen wurde, sollte die Gumstix-Konsole mit *minicom -o* gestartet werden. Um in den Bootloader zu kommen,

muss sofort nach dem Einschalten des Gumstix eine Taste gedrückt werden. Wenn ein "Gum>" erscheint, hat der Vorgang geklappt.

Zuerst sollte das Dateisystem geflasht werden indem das Image des Dateisystems auf den Gumstix kopiert wird. Hierfür wird zunächst das Empfangsprogramm des Bootloaders mit dem Befehl *loady a2000000* gestartet. Dieser Befehl sorgt dafür, dass die über den seriellen Port empfangenen Daten im Speicher an die angegebene Adresse, welche sich im RAM befindet, geschrieben werden. Als nächstes müssen natürlich auch die Daten an den Gumstix gesendet werden. Dies geschieht mittels einer Funktionalität in Minicom. Mit [CRTL]-[A] wird der Menümodus geöffnet. Durch Drücken der Taste [S] wird in das Kommunikationsmenü gewechselt. Hier sollte "yModem" ausgewählt werden. In dem nun auftauchenden Dialog sollte das Dateisystem-Image ausgewählt werden, welches durch Kompilieren von Build-Root entstanden ist. Es trägt normalerweise den Namen "rootfs.arm_nofpu.jffs2". Wenn der Kopiervorgang beendet ist, sollte der Flash-Vorgang mit dem Befehl *pro on 1:0-1 && jera all && cp.b a2000000 40000 ${filesize}* gestartet werden. Um den Kernel zu kopieren, wird wieder das Empfangsprogramm des Bootloaders mit *loady a2000000* gestartet und mittels [CTRL]-[A] in das Menü des Minicom gewechselt, mit [S] in den Kommunikationsmodus, "yModem", und im auftauchenden Dialog das Kernel-Image ausgewählt . Es trägt normalerweise den Namen "uImage". Nach dem Kopiervorgang sollte zum Flashen des Kernel die Befehle aus Listing 4.7 eingetippt werden. Der Kernel und das Dateisystem sollten jetzt geflasht sein. Um U-Boot zu beenden sollte *bootm* eingetippt oder der Gumstix vom Strom getrennt werden.

```
1  # katinstall 100000
2  # katload 100000
```

Listing 4.7: Flashen des Kernel

4.1.6 Eigene Programme

Es gibt zwei Möglichkeiten eigene Programme auf dem Gumstix laufen zu lassen. Die erste Möglichkeit ist im Buildroot den gcc und g++ compiler hinzu zu fügen und die Programme auf dem Gumstix zu kompilieren. Die zweite Lösung wäre ein Cross-Compiling. Dazu wird im gleichen Verzeichnis, in dem sich der Ordner "gumstix-buildroot" befindet, ein neuer Ordner angelegt und die Quelldateien des Programms in diesem Ordner eingefügt. Als Nächstes muss ein Makefile angelegt werden, welcher den Inhalt aus Listing 4.8 enthält. In der ersten Zeile müssen noch die Quelldateien hinzugefügt werden, welche jeweils durch ein Leerzeichen getrennt sind. In die dritte Zeile wird Namen der gewünschten Datei geschrieben. Sollte der eigene Quellcode komplexer sein und müssen Bibliotheken hinzugefügt oder Include-Pfade gesetzt werden, so wird das Makefile um die entsprechenden Parameter erweitert. Wenn alles stimmt, kann der Kompiliervorgang mittels des Befehls *make* gestartet werden. Es kann sein, dass zu Beginn ein paar Fehlermeldungen auftauchen, dass bestimmte .o und/oder .bin-Dateien nicht gefunden werden konnten. Dies könnte darin liegen, dass zur Sicherheit bei jedem Kompiliervorgang alle .o und .bin-Dateien gelöscht werden. Es

kann bei minimalen Änderungen vorkommen, dass der Cross-Compiler keine Änderungen in den Quelltextdateien erkennt und nicht kompiliert. Dies wird durch das Entfernen der .o und .bin-Dateien erzwungen. Die so entstandene Datei muss anschließend nur noch auf den Gumstix kopiert werden.

```
1  SOURCES =
2  OBJECTS = $(SOURCES:.cpp=.o)
3  BINARY =
4
5  all : clean $(BINARY)
6
7  $(BINARY) : $(OBJECTS)
8    ../gumstix-buildroot/build_arm_nofpu/staging_dir/
9    bin/arm-linux-uclibcgnueabi-g++ -o $(BINARY) $(OBJECTS)
10
11 $(OBJECTS) : $(SOURCES)
12   ../gumstix-buildroot/build_arm_nofpu/staging_dir/
13   bin/arm-linux-uclibcgnueabi-g++ -c $(SOURCES)
14
15 clean :
16   rm $(OBJECTS) $(BINARY)
```

Listing 4.8: Makefile

4.1.7 Zusätzliche Hardware

Es werden beim Raptor verschiedene Hardwarekomponenten, wie zum Beispiel Ultraschallsensoren, Kompass und Radencoder verwendet. Diese werden mittels des ATMega Mikrocontrollers (siehe Kapitel 3.2) eingebunden und angesteuert. Doch andere Hardwarekomponenten, wie der WLAN-Stick oder die WebCam ver-fügen über eine USB-Schnittstelle, so dass sie direkt am Gumstix angeschlossen werden können. Anhand des Beispiels einer WebCam wird gezeigt, wie Hardware direkt am Gumstix eingebunden und angesprochen werden kann.

Als erstes werden die richtigen Linux-Treiber für die WebCam benötigt. Der Raptor hat eine Logitech Quickcam Express, also wird der Treiber “GSPCA/SPCA5xx” als Quellcode benötigt.

Zuerst muss, wie in Abschnitt 4.1.6 beschrieben, der Treiber cross-kompiliert werden. Hierfür wird im gumstix-buildroot Verzeichnis ein Ordner angelegt und der Treiber in den Ordner eingefügt. Für die Videofunktionen ist wichtig, dass die entsprechenden Bibliotheken auf dem Entwicklungsrechner installiert sind. In diesem Fall muss die Bibliothek V4L (Video for Linux) vorhanden sein.

Um das cross-compiling erfolgreich durchzuführen, muss noch eine Änderung im Makefile des Treibers vorgenommen werden. Die Zeile

```
1 KERNELDIR := /lib/modules/$(KERNEL_VERSION)/build
```

Listing 4.9: Makefile

muss durch folgende Zeilen ersetzt werden:

```
1 KERNELDIR = ../gumstix-buildroot/build_arm_nofpu/
2    linux-2.6.21gum
3 BUILD_ARM = ../gumstix-buildroot/build_arm\_nofpu
4 CROSS_COMPILE = $(BUILD_ARM)/staging_dir/bin/arm-linux-
```

Listing 4.10: Makefile

Jetzt muss die Zeile

```
1 $(MAKE) -C $(KERNELDIR) SUBDIRS=$(PWD) CC=$(CC) modules
```

Listing 4.11: Makefile

durch

```
1 $(MAKE) -C $(KERNELDIR) SUBDIRS=$(PWD) ARCH=arm
2   CROSS_COMPILE=$(CROSS_COMPILE) modules
```

Listing 4.12: Makefile

ersetzt werden. Dadurch bekommt der Compiler die Informationen, wo sich der Kernel befindet und um welche Architektur es sich handelt. Erneut kann durch den *make* Befehl in einer Konsole der Quellcode kompiliert werden. Es entsteht eine .ko Datei, in diesem Fall das Kernel-Modul "gspca.ko". Diese und die drei zusätzlich in Listing 4.13 aufgeführten Dateien müssen nun auf den Gumstix kopiert werden, und zwar ins Verzeichnis /lib/modules/2.6.21gum/kernel/drivers/media/video/. Zum Schluss muss in der Datei /etc/modules der Name des neuen Moduls hinzugefügt werden, erkennbar in Listing 4.14. Nach einem Neustart des Gumstix sollte die WebCam nun einsatzbereit sein.

```
1  gspca.ko
2  ../gumstix-buildroot/build\_arm\_nofpu/root/lib/modules/
3    2.6.21gum/kernel/drivers/media/video/videodev.ko
4  ../gumstix-buildroot/build\_arm\_nofpu/linux-2.6.21gum/
5    drivers/media/video/v4l1-compat.ko
6  ../gumstix-buildroot/build\_arm\_nofpu/linux-2.6.21gum/
7    drivers/media/video/v4l2-common.ko
```

Listing 4.13: Cross-kompilierte Kernel Module

```
1 # WebCam
2 gspca
```

Listing 4.14: /etc/modules

4.1.8 I2C auf dem Gumstix

Der Gumstix verdex XL6P besitzt keinen separaten Anschluss für den I^2C-Bus. Es existieren aber dennoch zwei Möglichkeiten eine Anbindung an diesen zu ermöglichen. Zum einen ist auf dem Motherboard ein Anschluss für ein 24-poliges Folienkabel vorhanden, welches die I^2C-Anschlüsse liefert. Zum anderen kann auf dem Carrier-Board an vorgesehenen Lötstellen ein Anschluss für den I^2C-Bus angebracht werden. Es kam die zweite Variante zum Einsatz, zu sehen in Abbildung 4.1. Die Belegung der I^2C-Anschlüsse stehen in den Board-Layouts des console-vx[32].

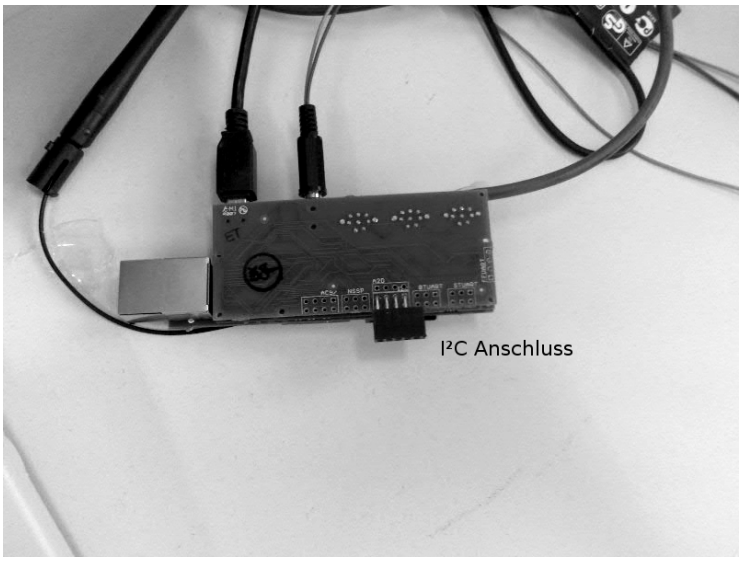


Abbildung 4.1: Anschluss an das Carrier Board

Das Kabel (Abbildung 4.2) musste von der Raptor-Gruppe selbst gebaut werden. Es wurde so entworfen, um eine falsche Benutzung zu vermeiden. Dennoch ist hier große Vorsicht geboten.

Auf dem Kabel sind farbige Markierungen angebracht. Diese kennzeichnen jede Leitung individuell und unverwechselbar.

- blau = X-SCL
- gelb = X-SDA
- schwarz = GND
- weiß = VCC

Um sicher zu stellen, dass auch das I^2C-Modul im Kernel eingebunden ist (was beim Verdex Board der Fall sein sollte), können mit dem Befehl *lsmod* alle eingebundenen Module angezeigt werden. Sollte das Modul “i2c-pxa” oder “i2c-dev” nicht dabei sein, so muss in der Buildroot Konfiguration der I^2C Support aktiviert werden, der Kernel erneut kompiliert

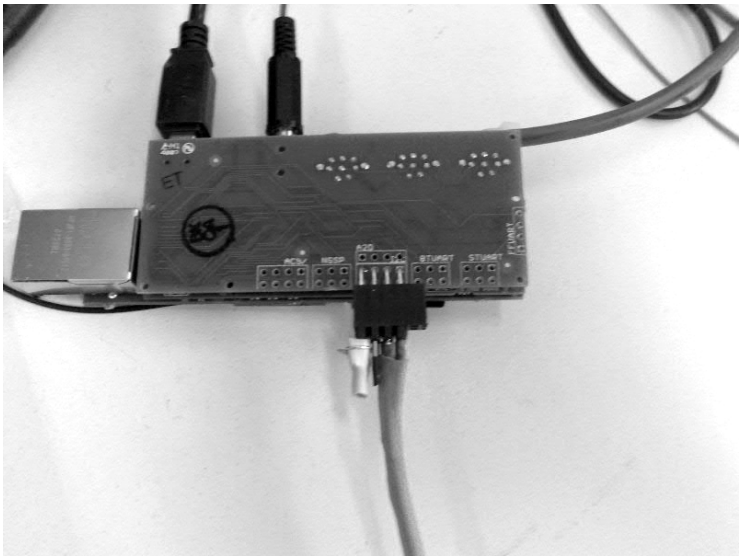


Abbildung 4.2: Carrier Board mit Anschlusskabel

und der neue Kernel geflasht werden. Anschließend müssen in der Datei /etc/modules folgende Zeilen hinzufügt werden:

```
1 # I2C
2 i2c-pxa
3 i2c-dev
```

Nach einem Reboot sollte nun I^2C einsatzbereit sein. In der Ausgabe von *lsmod* ist zu sehen, ob die I^2C Module auch eingebunden sind.

4.1.9 I2C Kommunikation

Zu Testzwecken wird ein simples I^2C Programm[15] benutzt. Am Beispiel des Kompasssensors soll die Kommunikation mittels I^2C verdeutlicht werden.

Der Kompasssensor wird zunächst mit dem I^2C-Kabel verbunden. Die Adresse des Kompasssensors ist "96". Um vom Kompasssensor einen Wert zu erhalten, muss zunächst der Sensor in einen Zustand des Sendens überführt werden, indem ihm ein Integer (in diesem Fall eine "2") als Byte geschickt. Dies erfolgt durch den ersten Befehl aus Listing 4.1.9. An die Adresse 96 wird nun ein Byte mit dem Wert 2 geschickt. Die vom Sensor geschickten Daten können mit dem zweiten Befehl aus Listing 4.1.9 empfangen werden. "–count=2" bedeutet, dass 2 Byte von der Adresse 96 empfangen werden sollen. Es erscheint in der Konsole nun ein Kompasswert in Hexadezimal. Um eine API für die Hardware des Raptors zu erstellen, können die Funktionen im Quellcode vom I^2C genutzt werden.

Für die Kommunikation mit dem Mikrocontroller muss zuerst ein Protokoll erstellt werden, damit sowohl der Gumstix als auch der Mikrocontroller wissen, welche Daten an welche Hardware gesendet werden sollen oder welche Daten von der Hardware zum Gumstix

geschickt werden sollen.

```
1 # i2c 96 SendByte 2
2 # i2c --count=2 96 RcvByte
```

4.1.10 Buildroot

Das Buildroot [33] ist ein System, welches ein standardisiertes Dateisystem Image basierend auf Linux für den Gumstix erstellt. Es werden alle Schritte automatisiert, die zur Erstellung eines brauchbaren Komplettsystems benötigt werden, im Einzelnen wird folgendes durchgeführt :

- Herunterladen aller nötigen Quellcodes und Daten,
- Bauen eines Cross Compilers mit der nötigen Toolchain,
- Kompilieren der gesamten Software, inklusive Kernel und Anwendungen,
- Zusammenführung aller Ergebnisse in ein fertiges Dateisystem, das auf den Gumstix Flash Speicher geladen werden kann.

Im Folgendem wird genauer darauf eingegangen, wie ein Dateisystem Image erstellt wird und wie der Cross Compiler zu benutzen ist.

Kompilieren des Buildroots

Um ein Buildroot zu erstellen, müssen folgende Programme vorhanden sein:

- Subversion,
- GNU Compiler gcc,
- autoconf.

Zunächst wird der aktuelle Quellcode mit Hilfe von Subversion vom Gumstix-Server heruntergeladen. Dabei ist zu beachten, dass, falls ein Proxy benutzt wird, dieser in die Umgebungsvariablen http_proxy und ftp_proxy des Systems eingetragen werden. Jetzt kann mit folgendem Befehl ein check out ausgeführt werden :

$$svn\ \ co\ \ http://svn.gumstix.com/gumstix-buildroot/trunk\ \ gumstix-buildroot.$$

Dabei wird der Quellcode in das Verzeichnis gumstix-buildroot kopiert. Jetzt wird in das gumstix-buildroot Verzeichnis gewechselt und das Buildroot kann konfiguriert werden. Dafür gibt es zwei Möglichkeiten.

Die erste Möglichkeit ist eine schnelle Lösung. Dazu wird der Befehlt

$$make\ \ defconfig$$

eingegeben. In unserem Fall wird bei der “CPU Speed“ die ”600 Mhz“ Option gewählt und die restlichen Fragen werden mit der vorgegebenen Option beantwortet.

Die zweite Möglichkeit ist sehr viel detaillierter. Dazu wird der Befehl

$$make \;\; menuconfig$$

benötigt. Dabei wird ein Menü geöffnet, wo das zu erstellende Image sehr genau konfiguriert werden kann. Es kann zum Beispiel aus einer großen Anzahl von Programmen ausgewählt werden, welche für den Gumstix bereitgestellt werden sollen und welche nicht.

Nachdem die Konfigurierung abgeschlossen ist, kann die Erstellung des Images durch den Befehl

$$make$$

gestartet werden. Nach ungefähr einer Stunde ist das Image im Verzeichnis zu finden. Jetzt kann dieses auf den Gumstix übertragen werden.

Cross Compiler

Ein Cross Compiler ist ein Compiler, der auf einem Hostsystem (in unserem Fall Linux) installiert ist und Quellcode für eine andere Zielplattform (in unserem Fall ARM) übersetzt, so dass das die entstanden Programmdateien auf der Zielplattform ausgeführt werden können.

Der vom Buildroot erstellte Cross Compiler ist für die meiste Software zu gebrauchen, außer natürlich für Software die speziell für eine Architektur geschrieben wurde, die nicht ARM entspricht.

Die meisten Programme lassen sich mit Hilfe von “autoconf” automatisch konfigurieren. Um einen Quellcode zu übersetzten, wird in das Hauptverzeichnis des Quellcodes gewechselt und durch den Befehl

$$./configure \;\; --host = arm - linux$$

autoconf gestartet. Dabei ist zu beachten, dass sich ein Verweis auf den Cross Compiler in der PATH Variable befinden muss. Ist das nicht der Fall, muss diese gesetzt werden. Jetzt kann der Übersetzungsvorgang durch den Befehl

$$make$$

gestartet werden. Wenn die Übersetzung ohne Fehler abgelaufen ist, kann das Programm auf den Gumstix übertragen und ausgeführt werden.

4.2 Weltmodell

Ein Roboter erfasst die Welt mit Hilfe seiner Sensoren. Er errechnet aus diesen Wahrnehmungen ein Modell der Umwelt, welches dazu dient seine Position im Raum zu lokalisieren

und zukünftige Aktionen zu planen. Dieses Modell wird das Weltmodell genannt. Der RAPTOR nutzt hierfür eine Gitterkarte. Diese stellt den bisher erfassten Raum als Koordinatensystem dar. Jeder einzelne Punkt in diesem Koordinatensystem repräsentiert einen Abschnitt im Raum für welchen mehrere Informationen hinterlegt sind. Diese sind zum Beispiel die Belegungswahrscheinlichkeit, welche angibt, mit welcher Wahrscheinlichkeit diese Stelle für den RAPTOR befahrbar ist.

Intern wird für die Gitterkarte die Datenstruktur "Vector" verwendet, die ein Teil der Standard Template Library [28] ist. Sie ist bestens für diesen Anwendungszweck geeignet, denn sie ist dynamisch während der Laufzeit, in ihrer Größe veränderbar und bietet zudem einfachen und sicheren Zugriff auf die in ihr gespeicherten Informationen. Da sich in der Datenstruktur "Vector" wiederum Datenstrukturen einbetten lassen, wird auch von einer Containerstruktur gesprochen. Für den RAPTOR wurde hier eine eigene Klasse entwickelt, welche alle relevanten Informationen, wie zum Beispiel den Grad der Verschmutzung, bereithält.

Neben der Datenspeicherung während der Laufzeit ist auch eine Speicherung des Weltmodells von Bedeutung, welche ein Ausschalten des Roboters überlebt. Hierfür wurde eigens ein speziell angepasstes Dateiformat entworfen, welches die zu speichernden Informationen in möglichst kompakter Form enthält. Da die begrenzende Ressource nicht der Speicherplatz, sondern die Rechenleistung ist, wurde auf eine Kompression der Daten verzichtet. Dies beschleunigt das Einlesen der gespeicherten Karten.

Um weitere Performanzvorteile zu erreichen wurde eine der höheren Funktionen direkt in die Gitterkarte integriert. Es handelt sich um die Routenplanung, welche einen optimalen Weg zwischen zwei Punkten berechnet. Hierfür ist es von großem Vorteil direkten Zugriff auf die Daten der Gitterkarte zu haben, weil nicht der Umweg über die für externe Zugriffe zuständigen Funktionen genommen werden muss. Näheres zur genauen Funktionsweise der Routenplanung befindet sich im Abschnitt Routenfahrt 4.9.3.

4.3 Datenfusion

Alle derzeitigen Roboter haben gemeinsam, dass sie sowohl Sensoren als auch Aktuatoren benötigen um ihre Umwelt zu erfassen und darauf zu reagieren. Umso wichtiger ist es, dass die empfangenen Informationen so genau wie möglich sind. Trotz dieser einfachen Erkenntnis lässt sich dieses Problem nicht so leicht lösen, weil nicht ideale Sensoren das genaue Erfassen der Umwelt fast unmöglich machen. Bedingt durch Fertigungstoleranzen liefert jeder Sensor neben dem Nutzsignal auch einen Rauschsignal. Es gilt nun, diesen Rauschsignal zu reduzieren um die Qualität der Information zu erhöhen, was eine genauere Erfassung der Umgebung möglich macht. Damit ist das Ziel, die Genauigkeit der erfassten Umgebung und die Robustheit des Roboters zu verbessern. Eine Möglichkeit die sich anbietet ist es, Sensoren die dieselbe Umgebung erfassen miteinander zu verknüpfen. Dadurch, dass mehrere Sensoren dieselbe Umgebung erfassen, wird zwar Redundanz erzeugt, aber der Roboter wird durch diese Maßnahme robuster (hinsichtlich Ausfallsicherheit) und der Einsatz unterschiedlicher Sensoren erhöht außerdem die Genauigkeit, da die Schwächen

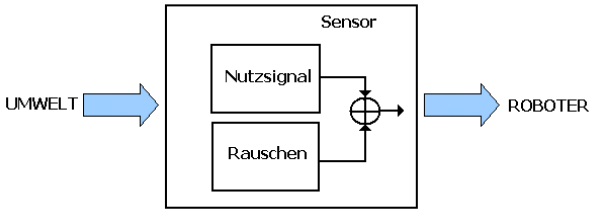


Abbildung 4.3: Nutz- und Störsignal

des einen Sensors durch die Stärken des anderen Sensors kompensiert werden. Durch den Einsatz eines Filters soll auch das Rauschen der einzelnen Sensoren reduziert werden, die bei einer anschließenden Datenfusion die Genauigkeit noch einmal verbessert.

4.3.1 Aufbau der Datenfusion

Bild 4.4 zeigt grob den Aufbau der Datenfusion. Die Sensoren liefern verschiedene Messwerte mit verschiedenen Fehlerwahrscheinlichkeiten. Es erfolgt eine Datenzuordnung, in der Messerwerte geeignet zugeordnet werden. Es können nur gleichartige Daten zugeordnet werden (z.B. mehrere Ultraschallsensoren) oder Daten die den geschätzten Parameter verbessern bzw. die für einen dritten Parameter benötigt werden (z.B. die Lokalisation). Anschließend wird der Schätzwert gebildet, wobei auch vergangene Messwerte als Sensorwert in die Datenzuordnung eingehen. Mit der Sensordatenfusion soll das Fehlerrauschen im Nutzsignal durch kombinieren von Sensordaten reduziert werden. Bei der Kombination zweier Sensoren treten synergetische Effekte auf. Bei der Datenfusion geht es darum, aus verschiedenen Datensätzen oder gleichartigen Datensätzen eine optimale Schätzung zu bestimmen, die besser als die einzelnen Daten sind. Die Datenfusion soll also einen neuen Datensatz produzieren, der inhaltlich mehr Informationen enthält als die einzelnen Daten für sich liefern. Die Tatsache, dass im RAPTOR mehrere Sensoren eingesetzt werden, bietet die Möglichkeit, die Informationen der Sensoren miteinander zu kombinieren und so eine höhere Genauigkeit zu erzielen. Wie Abbildung 4.4 zeigt, fließen die Sensordaten in den Fusionsblock hinein, dort werden sie verarbeitet um so eine optimale Information zu liefern. Durch das Einsetzen redundanter Sensoren wird außerdem die Ausfallsicherheit

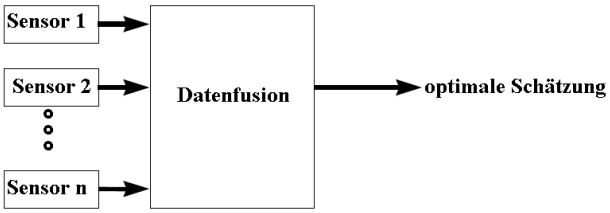


Abbildung 4.4: Fusionsprozess

erhöht. Beispielsweise bleibt im Falle eines Ausfalls eines Ultraschallsensors der RAPTOR

noch funktionsfähig, jedoch vergrößert sich der tote Winkel. Eine gute Fusionsmethode bietet der diskrete Kalman-Filter, der hier zum Einsatz kommt [20] . Dieser Filter kommt häufig bei mobilen Robotern zum Einsatz. Allerdings besteht hier das Problem, dass der Kalman-Filter davon ausgeht, dass alle Daten (Nutzsignale) zur gleichen Zeit erfasst werden, d.h. Daten die zur unterschiedlichen Zeit ankommen erhöhen die Unsicherheit und der geschätzte Parameter wird zu ungenau. Ist aber die zeitliche Verzögerung zwischen den Sensoren sehr klein, der Fehlerterm in der Datenfusion hat in dem Fall einen geringen Einfluss, kann dies vernachlässigt werden. Ansonsten müssen bzw. können die verzögerten Daten extrapoliert werden, was meist zu einer erhöhten Unsicherheit in den Daten führt. Beim Extrapolieren muss nämlich sichergestellt werden, dass nur Daten miteinander fusioniert werden, die zur gleichen Zeit erfasst wurden. Das heisst, für jeden Sensorwert muss die zeitliche Verzögerung erfasst werden, anschließend erfolgt eine zeitliche Zuordnung, sortiert nach der Verzögerungszeit, in der Daten nacheinander extrapoliert werden, damit keine Informationen verloren gehen. Die Daten, die zu einem späteren Zeitpunkt ankommen, werden in einem zweiten Fusionsprozess kombiniert, damit wird eine optimale Schätzung für jede Zeit gewährleistet. Die Messungen, die zum selben Zeitpunkt gemacht

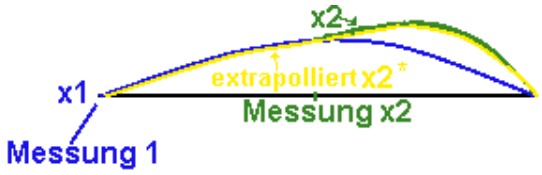


Abbildung 4.5: Extrapolieren der verzögerten Daten

wurden, werden miteinander fusioniert. Anschließend werden die verspäteten Daten mit dem fusionierten Wert weiter fusioniert. Die Genauigkeit verschlechtert sich, wenn z.B. die Radencoderwerte nicht zur gleichen Zeit wie die Ultraschallsensorwerte extrapoliert wurden. Im ersten Schritt gehen in die Schätzung nur die Ultraschallsensorwerte ein. Dieser wird in einem zweiten Schritt mit den Radencoderwerten (die Geschwindigkeit bzw. die zurückgelegte Strecke) weiterverarbeitet. Der geschätzte Parameter verschlechtert sich in dem Fall (siehe Abbildung 4.6). Ohne diesen Ansatz ist der geschätzte Parameter (als grüne Gaußglocke dargestellt) besser, dieser liegt näher am zweiten Parameter z_2.

Die Gaußglocke c (siehe Abbildung 4.6 oben) stellt das endgültige Ergebnis dar, dieser liegt sehr nahe am zweiten Wert. Die Gaußglocke c*(unten), stellt das endgültige Ergebnis dar, dieser Wert liegt sehr weit vom zweiten Wert.

Vorteile der Datenfusion:

- Durch den Einsatz mehrerer Sensoren, ist der Ausfall eines einzelnen Sensors leichter zu verkraften als wenn nur ein einziger Sensor vorhanden ist, der Systembetrieb wird robuster.

- Die Zuverlässigkeit wird verbessert, dadurch dass mehrere Sensoren (redundante Sensoren) das gleiche Objekt erfassen, wird die Glaubwürdigkeit der Beobachtung erhöht.

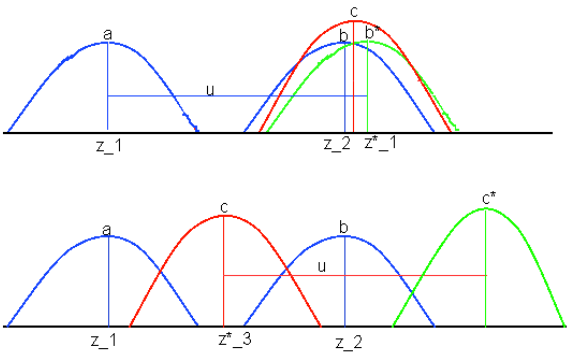


Abbildung 4.6: Oben: Ohne die Werte zu extrapolieren; Unten: Extrapolieren der Werte

- Durch die Verwendung mehrerer Sensoren mit unterschiedlicher Ausrichtung wird die räumliche Erfassung erhöht.

- Unsicherheiten werden durch kombinierte Sensoren reduziert. Also der Einsatz komplementärer Sensoren können die Schwächen des einen Sensors mit den Stärken eines anderen ausgeglichen werden.

Zur Fusionierung der Sensordaten wurde ein Algoritmus auf Basis des in Bild 4.4 dargestellten Modells verwendet.Die Datenfusion arbeitet in zwei Teilen, die von einem Thread gestartet werden. Im ersten Teil wird die zurückgelegte Strecke $\delta x/\delta t$ bestimmt, dieser Wert wird für den Kalmanalgorithmus benötigt. Im zweiten Schritt werden die Update-/ Fusions-Funktionen aufgerufen, die die Sensordaten schätzen und fusionieren. Die zweite, wichtige Hauptfunktion der Datenfusion ist:

```
1   void datenfusion::Sensorvorverarbeitung();
```

Listing 4.15: Funktion zur Schätzung der Sensorwerte und der zurückgelegten Strecke

Die Methode bestimmt zunächst in welchem Zustand der Roboter ist, ausgehend davon wird die Geschwindigkeit bzw. $\delta x/\delta t$ bestimmt, die für den Kalman-Filter benötigt wird.

```
1   void datenfusion::Sensorschaetzung();
```

Listing 4.16: Funktion zum Fusionieren der Sensorwerte

Mit der zweiten Methode wird die geschätzte Geschwindigkeit in die Update-Funktion des Kalman-Filters übergeben, die eine Art Vorausschätzung des Zustandes zum Zeitpunkt $t + 1$ durchführt. Die Methode Sensorschaetzung() fusioniert hier unter anderem auch die vom Maussensor geschätzten Winkel (siehe Abschnitt 4.3.3) mit den geschätzten Kompasswerten. In dieser Methode erfolgt also die Fusionierung der Sensorwerte.

4.3.2 Schätzung

Die Schätzung kann als ein Verfahren definiert werden, dass auf Basis von Wissen über die Systemeigenschaften und durch Trennung von Signalen und Störungen einen Schätzwert des Systemzustandes mit einer minimalen Schätzfehlervarianz ermittelt. Die Abweichung der Schätzung vom wahren Ergebnis kann zwischen 10 und 100 Prozent liegen. Der Grund dafür ist, dass jede Messung mit unvermeidlichen kleinen Fehlern behaftet ist, so dass der wahre Wert nicht geliefert werden kann.

Kalman-Filter

Der in diesem Abschnitt gezeigte Kalman-Filter lehnt sich in weiten Teilen an [6], [7] und [29] an. Der Kalman-Filter wurde 1960 von Emil Kalman entwickelt. Es handelt sich hierbei um eine rekusives Verfahren, das den Systemzustand mittels einer Gaußverteilung schätzt. Dieser wird häufig im Bereich von autonomen Fahrzeugen eingesetzt. Es ist ein stochastischer Zustandsschätzer für dynamische Systeme. Der Kalman-Filter besteht aus mehreren mathematischen Gleichungen. Er ist sehr leistungsfähig, wenn es darum geht vergangene, derzeitige und zukünftige Systemzustände zu schätzen. Die Variablen z_k und z_{k+1} repräsentieren Messwerte, diese können von verschiedenen Sensorquellen stammen oder aber nur aus einer. Die Variablen δ_k und δ_{k+1} sind die dazugehörigen Varianzen. Wie schon vorher erwähnt liefern Sensoren keine Absolutwerte, (die Gründe liegen unter anderem in der Produktion solcher Sensoren, denn die Technologie mit der solche Sensoren entworfen werden unterliegen Grenzen). Der Kalman-Filter berechnet aus den Messwerten und deren zugehörigen Varianzen einen Schätzwert $\hat{x}_{k+1}$, mit einer minimalen Fehlervarianz $\hat{\delta}_{k+1}$. Dieser hat eine kleinere Varianz als die einzelne Varianzen der Messungen bzw. Sensoren.

```
1 void kalmanfilter_init(double pk_wert,double pk_wert_2,
2 double q_wert,double rk_wert,double rk_wert_z, int dimension);
```

Listing 4.17: Kalman-Filter initialisieren

Mit dieser Methode wird der Kalman-Filter initialisiert, d.h. hier werden die Anfangswerte wie Varianzen der Sensorwerte eingetragen. Der Algorithmus arbeitet in einer Schleife. Bevor die Schleife betreten wird, werden die vom Kalman-Filter benötigten Initialisierungswerte gesetzt. Der Kalman-Filter arbeitet in zwei Hauptschritten die in einer Schleife ausgeführt werden. Im ersten Schritt wird abhängig vom letzten Zustand $\hat{x}_k$ und δ_k^2, der neue Zustand $\hat{x}_{k+1}$ und neue Fehlerkovarianz $\hat{\delta}_{k+1}^2$ vorhergesagt. Der zweite Hauptschritt sorgt dafür, dass mittels der vorhergesagten Werte der neue aktuelle Zustand mit der dazugehörigen Fehlerkovarianz geschätzt wird. Dabei geht ein Gewichtsfaktor in die Rechnung ein. Dieser Wert drückt aus, wie stark die Differenz (aktueller Messwert und geschätzter Wert für den Zeitpunkt) in den geschätzten Zustand eingehen. Das System wird durch eine lineare stochastische Differenzgleichung beschrieben [6]:

$$\hat{x}_{k+1} = \hat{x}_k + K_{k+1} * (z_{k+1} - \hat{x}_k) \tag{4.1}$$

```
1  void kalmanfilter_time_update();
```

Listing 4.18: Time-Update-Phase

Diese Methode realisiert die erste Formel und stellt die brechneten Werte bereit, die für die anschließende Fusion benötigt wird.

$$K_{k+1} = \delta^2_{k+1}/(\delta^2_k + \delta^2_{k+1}) \tag{4.2}$$

```
1  void kalmanfilter_fusion();
```

Listing 4.19: Fusions-Phase

Hier werden die berechneten Werten weiterverarbeitet um die Zustände des Roboters, wie beispielsweise Position, Winkelausrichtung, Geschwindigkeit, etc. optimal zu schätzen.

```
1 void messung();
```

Listing 4.20: Aufruf der Sensorwerte

Für den Kalman-Filter ist es wichtig, dass er laufend Sensorinformationen bekommt. Dies wird mit der Methode `void messung()` realisiert. Nach jedem Aufruf übergibt die API der Methode, die aktuellen Sensorwerte. Um den neuen Schätzwert zu erhalten wird der alte Schätzwert (für $k = 0$ ist es der Sensorwert) mit einer Korrektur K_{k+1}, multipliziert und mit der Differenz vom aktuellen Messwert und dem alten Schätzwert addiert. Man nehme an, dass zum Zeitpunkt k ein Sensor eine Messung z_1 liefert. Wie schon erwähnt, kann nun der ausgelesene Wert vom wahren Wert abweichen. Es wird ein Bereich angegeben, in der sich der Wert mit einer gewissen Wahrscheinlichkeit befindet. Dies wird mittels einer Gauß-Glockenkurve dargestellt (siehe Abbildung 4.7). Wie im Bild zu sehen, erreicht die Kurve ihr Maximum an der Stelle des Messergebnisses, die den Wert z_1 darstellt. Der Verlauf der Kurve gibt an, wie gut die Messgenauigkeit ist. Je flacher die Kurve verläuft, desto ungenauer sind die Messungen und desto weniger vertrauenswürdig ist das Ergebnis. Die Varianz $\delta^2_{z_1}$ liefert die Genauigkeit von z_1. Ist die Varianz Null so handelt es sich hierbei um einen exakten Wert. Wird eine zweite Messung z_2 zum Zeitpunkt k_2 mit Varianz $\delta^2_{z_2}$ ausgelesen, so wird sie anschließend mit der ersten Messung kombiniert.

- $K_1 = \delta^2_1$
- $\hat{x}_1 = z1$
- $K_2 = \delta^2_1/(\delta^2_1 + \delta^2_2)$

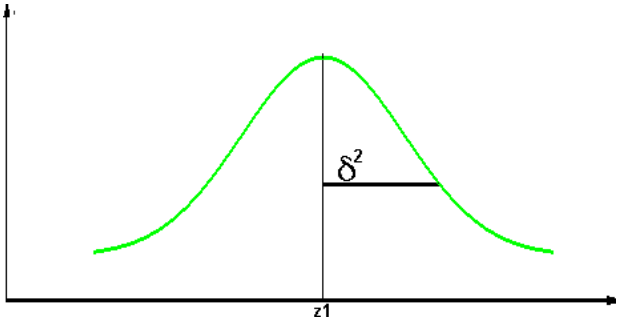


Abbildung 4.7: Qualität einer Messung anhand der Gaußglocke [6]

- $\hat{x}_2 = \hat{x}_1 + K_2 * (z_2 - \hat{x}_1)$

Der Kalman-Gain $K_{k+1} = \delta_k^2/(\delta_k^2 + \delta_{k+1}^2)$ legt fest, wie stark die Differenz zwischen vorherigen Schätzwert und der aktuellen Messung in eine weitere Schätzung eingeht. Die neue Varianz wird durch folgende Formel berechnet:

$$\delta_{k+1}^2 = (1 - \delta_k^2) * K_{k+1}. \tag{4.3}$$

Das Ergebnis in Abbildung 4.8 hat eine geringere Varianz und die Kurve ist steiler. Bei

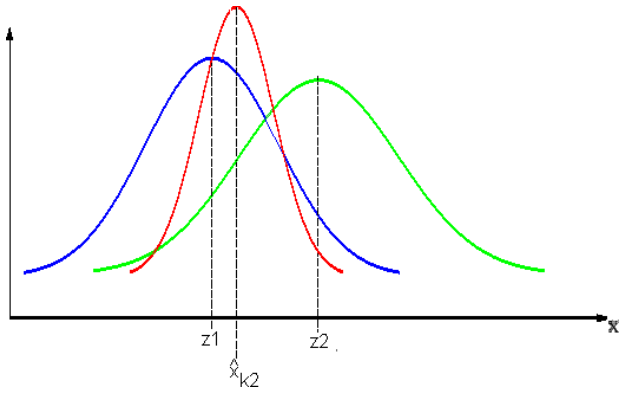


Abbildung 4.8: Varianz zweier Messungen [6]

einer dynamischen Änderung des Signals, wird ein Faktor u in die Gleichung eingerechnet. Dieser Fall tritt ein, wenn z.B. ein Roboter im bewegten Zustand ($\Delta x/\Delta t! = Null$) die Umgebung mit mehreren Ultraschallsensoren abtastet, so ändern sich die Werte der Sensoren mit der zurückgelegten Strecke. Da der Faktor nicht genau angegeben werden kann, enthält auch dieser einen Messfehler der mit w ausgedrückt wird. Um das ganze besser zu verstehen, soll es an einem Beispiel noch einmal verdeutlicht werden. Es soll die Position eines Roboters bestimmt werden. Einfaches Beispiel: der Roboter bewegt sich auf einer Linie und zwar in x-Richtung, die y-Achse soll hier nicht näher betrachtet werden. Da der Roboter im bewegten Zustand Messwerte sammelt, muss die zurückgelegte Strecke mit

einbezogen werden. Es gilt: $(\Delta x/\Delta t) = u + w$. Die Variable w kompensiert die tatsächliche Geschwindigkeit. Kurz vor der zweiten Messung wird ein Schätzwert der Position des Roboters ermittelt, in der er sich zu diesem Zeitpunkt befinden müsste.

$$\tilde{x}_{k+1} = \hat{x}_k + u(t_{k+1} - t_k) + w = \hat{x}_k + u * t_{diff} + w. \tag{4.4}$$

$$\nabla x^2_{k+1} = \nabla x^2_k + \nabla w^2(t_{k+1} - t_k) = \nabla x^2_k + \nabla w^2 * t_{diff}. \tag{4.5}$$

Um die optimale Position zu schätzen, geht die zweite Messung (z_{k+1}) folgendermaßen in die Formel ein:

$$\hat{x}_{k+1} = \hat{x}_k + K_{k+1} * (z_{k+1} - \tilde{x}_{k+1}). \tag{4.6}$$

$$K_{k+1} = \nabla \tilde{x^2}_{k+1}/(\nabla \tilde{x^2}_{k+1} + \nabla x^2_{k+1}). \tag{4.7}$$

Dabei ist

- $\hat{x}_{k+1}$ a-posteriori Schätzung,
- $\hat{x}_k$ a-priori Schätzung,
- K_{k+1} Gewichtung und
- $z_{k+1} - \tilde{x}_{k+1}$ Abweichung.

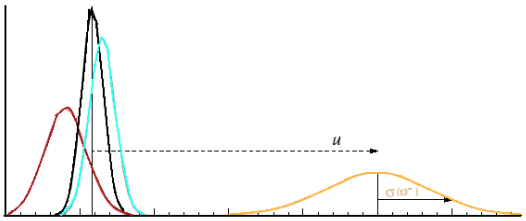


Abbildung 4.9: der Einfluß u [6]

Kalman-Filter für den n-dimensionalen Raum

Die Gleichungen entsprechen dem eindimensionalen Raum. Zur Erfassung des mehrdimensionalen Raums werden die einfachen Koeffizienten durch Matrizen ersetzt und die Messwerte werden in einem n-Dimensionalen Vektor abgebildet.
der geschätzte Systemzustand lautet:

$$\hat{\tilde{x}}_k = A * \hat{\tilde{x}}_{k-1} + B * \hat{\tilde{u}}_k. \tag{4.8}$$

Die Matrix A beschreibt das Systemverhalten. Die Matrix B legt die Gewichtung des Vektors u fest, ohne Gewichtung ist B eine Einheitsmatrix.

$$\hat{\tilde{P}}_k = A * \hat{\tilde{P}}_{k-1} + A^T + Q. \tag{4.9}$$

Q ist die Kovarianzmatrix und erfasst das Rauschen der Sensoren bzw. des Systems.
Der Kalmanzuwachs lautet:

$$\hat{\hat{K}}_k = \hat{\hat{P}}_k * H^T * (H * \hat{\hat{P}}_k * H^T + R)^{-1}. \tag{4.10}$$

Matrix H legt fest, wie stark die aktuelle Messung in den Systemzustand eingeht. Die gelieferten Sensorwerte sind mit einem Rauschen überlagert, diese wird in der Matrix R abgebildet. Die Schätzung muss anschließend aktualisiert werden, und zwar durch folgende Formel:

$$\hat{x}_k = \hat{\hat{x}}_k + K_k * (z_k - H * \hat{\hat{x}}_k). \tag{4.11}$$

Die Fehler- Kovarianzmatrix muss anschließend aktualisiert werden.
In diesem Abschnitt wird an einem Beispiel gezeigt, welchen Einfluss die Einstellungen der Filterparameter auf die geschätzte Größe haben. In diesem Beispiel wird eine simulierte Messung einer skalaren Größe vorgenommen, also eine 5 Volt Spannung. Es wird angenommen dass die gemessene Spannung mit einem Gauß'schen Rauschen überlagert sei. Die Standardabweichung liegt bei 0.1 Volt. Der Startwert $\hat{x}_{k-1}$ bleibt für jede Einstellung unverändert auf Null Volt, damit die Ergebnisse miteinander verglichen werden können. Der Wert von P_{k-1} kann zwischen 0 und 1 liegen. Als erste Einstellung werden folgende Werte festgelegt:

- $P = 1$
- Prozess-Varianz $Q = exp(-5.5)$
- Messkovarianz $R = (0.1)^2 Volt$

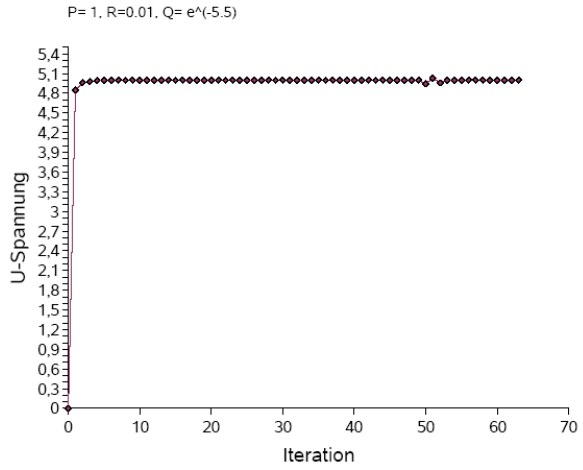


Abbildung 4.10: Verlauf von x_{k1}

Abbildung 4.10 zeigt den Verlauf der geschätzten Werte. Ausreißer (50-60 Iteration) haben bei dieser Einstellung keinen so großen Einfluss auf den geschätzten Zustand. Wird die

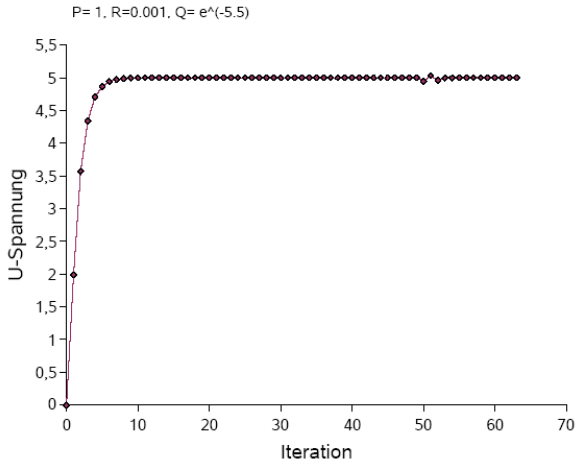


Abbildung 4.11: Verlauf von x_{k2}

Messkovarianz reduziert (siehe Abbildung 4.11), erhöht sich das Vertrauen der gemessenen Werte. Statt ein Bereich nahe bei 5 Volt auszugeben, liefert der Filter bereits Werte, die mehr als die Hälfte darunter liegen ($2Volt- > ca.3.5Volt- > ca.4.3Volt- > ...$). Der Grund dafür ist, dass der Filter die Messungen als sehr zuverlässig einstuft. Die Messungen erhalten eine höhere Gewichtung was zu einer langsamen Anpassung der Spannung an den wahren Wert führt. Wird für Q ein zu hoher Wert gewählt (siehe Abbildung 4.12),

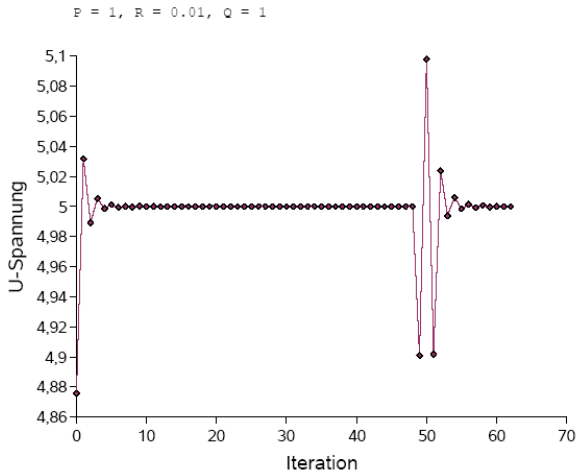


Abbildung 4.12: Verlauf von x_{k4}

verschlechtern sich die Ergebnisse.

4.3.3 Selbstlokalisation (mittels Maus-/ Kompassensor)

Damit der Roboter sich im Raum zu Recht findet, benötigt der Roboter die aktuellen x-/y-Koordinaten. Die Klasse Datenfusion liefert die aktuelle Position des Roboters. Zum Startzeitpunkt liegt der Wert bei $(0, 0)$. Die benötigten Koordinaten werden aus dem Maussensor und Radencoder berechnet, wobei der aktuelle Winkel zur Berechnung der Position aus dem Kompassensor und Maussenor in die Berechnung mit einbezogen werden. Der Kalman-Filter schätzt die aktuellen Werte des Maussensors. Diese werden dann mit den geschätzten Radencoder- und Kompassensorwerten verrechnet. Der Kompassensor ist so platziert, dass die x-Koordinaten des Maussensor nur dann Werte liefern, wenn der Roboter sich dreht. Der Zusammenhang zwischen Drehung und Winkel ist in der Abbildung 4.13 gezeigt.

```
1   void datenfusion::maus_winkel(double x_Mauswert);
```

Listing 4.21: Funktion zur Winkelschätzung mittels Maussensor

Mit dieser Methode wird aus den Maussensorwerten der Drehwinkel berechnet. Dazu wird zunächst der momentane Winkel (Startwinkel) des Roboters übergeben. Um die Methode einzusetzen, wird die Abmessung vom Maussensor zum Mittelpunkt des Roboters benötigt. Aus der Kreisgleichung (siehe Abbildung 4.13) lassen sich Rückschlüsse ziehen, um wieviel Grad der Roboter sich gedreht hat. Die Kreisgleichung ist in vier Bereiche aufgeteilt, die

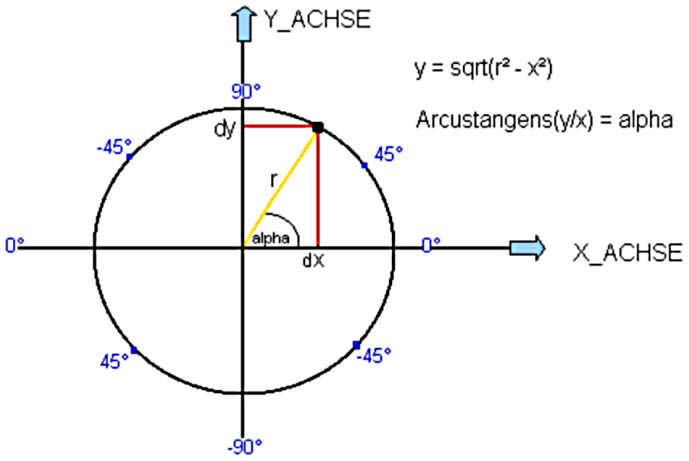


Abbildung 4.13: Zusammenhang Winkel und Mausänderung

eine Änderung der x- und y-Koordinate kennzeichnen.

- Bereich I : x- und y-Koordinate positiv
- Bereich II : x-Koordinate positiv und y-Koordinate negativ

- Bereich III: x- und y-Koordinate negativ
- Bereich IV : x-Koordinate negativ und y-Koordinate positiv

Abhängig von dieser Änderung wechseln sich die Winkelvorzeichen und müssen umgerechnet werden, um den richtigen Winkel zu berechnen. Bei dem Kompasswert und der aus dem

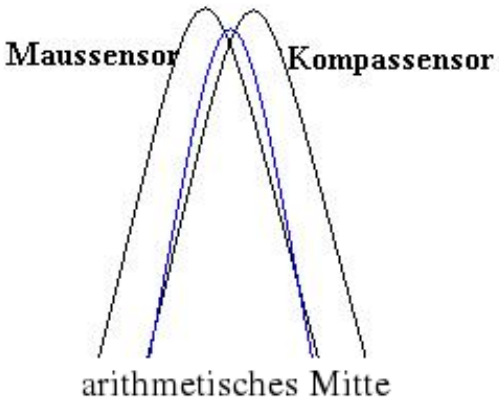


Abbildung 4.14: Maussensor und Kompassensor

Maussensor berechneten Winkel, handelt es sich nicht um absolute Werte. Die Werte enthalten noch Abweichungen (vom wahren Wert). Die Abweichung können reduziert werden, indem das arithmetische Mittel der beiden Werte gebildet wird. Die Roboterposition wird in definierten Zeitabständen aktualisiert. Dazu wird der geschätzte, zurückgelegte Weg δx und die geänderte Orientierung α, ebenfalls geschätzt, mit der zu letztgespeicherten Position des Roboters verrechnet. Hier wird angenommen, dass sich bis zur nächsten Aktualisierung, der Roboter geradlinig bewegt. Da die Zeitspanne sehr klein ist, ist der Fehler der in die Rechnung eingeht auch nur sehr klein. Die Position des Roboters $\wp = (x, y, \alpha)$ wird folgendermaßen berechnet:

- Eingabe $a = (\delta x, \delta\wp)$
- $l \leftarrow f(l, \alpha) = (x + \delta x * sin(\alpha + \delta\wp), y + \delta x * cos(\alpha + \delta\wp), \alpha + \delta\wp)$

Diese Formel berechnet die neue Roboterposition und übergibt die berechneten Werte der Odometrie, die die Werte zur Erstellung der Karte eingesetzt wird.

```
1   void datenfusion::reset(double x, double y);
```

Listing 4.22: Rücksetzfunktion

Die Methode in Listing 4.23 setzt die Koordinaten des Roboters auf einen gewünschten Punkt. Sie wird in der Odometrie als Zusatzfunktion bereitgestellt.

4.3.4 Integration der Boost-Library

Für eine Implementierung der Matrizen inkl. deren Operationen wurden Datenstrukturen der Boost-Library [30] verwendet. Diese stellt eine Vielzahl an vorgefertigten Matrizentypen und -operationen wie z.B. Einheitsmatrizen, inverse Matrizen usw. bereit, welche laufzeiteffiziente Berechnungen ermöglichen.

4.4 Odometrie

Der reale Raum wird über eine implizit darübergelegte Gitterkarte in Gitterzellen eingeteilt, von denen jede entweder von einem Hindernis belegt oder frei sein kann.

Selbstverständlich soll der RAPTOR auch mehrfach im gleichen Raum eingesetzt werden. Eine seiner Stärken besteht gerade darin, einen bereits erkannten und eingespeicherten Raum wiederzuerkennen, anstatt ihn erneut komplett abzufahren und eine neue Karte für ihn zu erstellen.

Dabei ist es sehr unwahrscheinlich, dass er sich beim neuen Start exakt am gleichen Punkt - respektive in der exakt gleichen Gitterzelle - befindet, auf dem er beim letzten Durchlauf durch diesen Raum begonnen hat. Es muss also eine Umrechnung zwischen den Punkten der bereits bekannten Karte und den Punkten der aktuell erkannten Karte vorgenommen werden.

Diese Umrechnung der von der Datenfusion als real gelieferten Umgebungskoordinaten und der im Speicher befindlichen Karten wird von der Odometrie vorgenommen. In der Regel führt das auf eine einfache Transposition hinaus (Abbildung 4.15); bei komplexeren Karten kann jedoch auch eine Skalierung nötig sein.

4.5 Kartenerstellung

Um die von der Bahn- und Routenplanung benötigten Karten auch nutzen können, müssen diese zunächst einmal erstellt werden. Hierzu wird die in Abschnitt 4.2 beschriebene Gridmap genutzt. Das Erstellen der Karte lässt sich zunächst in zwei grobe Berreiche unterteilen: das Interpretieren der Sensorwerte zur Erstellung der Karte und die Raumerkundung, um eine vollständige Karte zu erstellen.

4.5.1 Sensorwerte interpretieren

Der Messbereich des Ultraschallsensors ist ein Kreissektor. Beim Auslesen des Sensors wird der gemessene Abstand zum Hinderniss genutzt, um zunächst 3 Punkte in diesem Sektor zu bestimmen: der erste Punkt ist die Position des Ultraschallsensors in der Karte und zugleich der Mittelpunkt im Kreissektor, der zweite und dritte Punkt sind die Verbindungspunkte des Kreisbogens mit den Kreisradien. Aus Gründen effizienterer Berechnungen wird der Kreissektor in ein gleichschenkliges Dreieck abstrahiert. Da der Ultraschallsensor einen

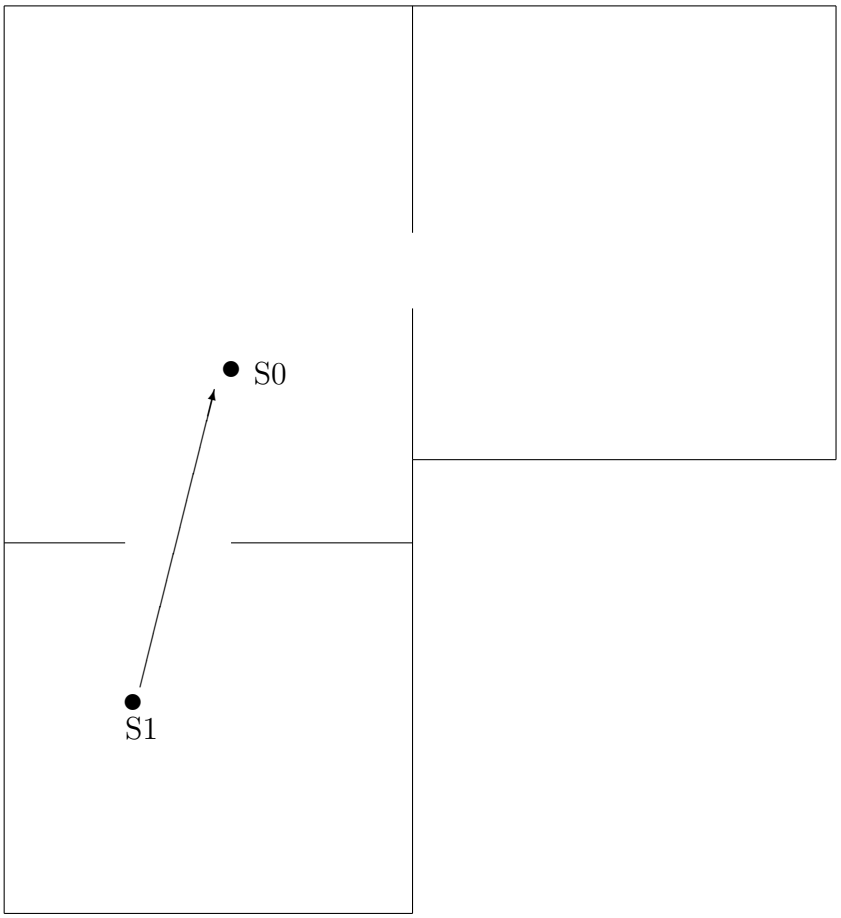


Abbildung 4.15: Transposition zwischen Startpunkten im gleichen Raumkomplex

Messwinkel von 50° hat, lässt sich nicht exakt bestimmen, wo genau auf diesem Kreisbogen sich das Hinderniss befindet. Deshalb wird in diesem Verfahren das Ausschlussprinzip genutzt.

Zunächst werden die Kreisradien gerastert (Mittelpunktschema-Algorithus) [11], so dass sie dem Raster der Gridmap entsprechen. Im Weiteren Verlauf werden alle Grids, die sich im Dreiecks befinden, auf Null gesetzt, was bedeutet, dass es sich bei diesen Grids um keine Hindernisse handelt. Dies geschieht mittels eines Algorithmus, der Polygone füllt [11]. Der Algorithmus fängt beim Mittelpunkt des Kreissektors an. Es werden, parallel zur vom Mittelpunkt gegenüberliegenden Kante, Kanten gebildet und danach überprüft, welche Grids sich auf der Kante befinden. Wenn ein Grid auf der Kante erkannt wird, wird dieser auf Null gesetzt. Nachdem alle Grids auf der Kante neu gesetzt sind, wird eine neue Kante gebildet, die sich immer weiter vom Mittelpunkt entfernt, bis sie an die Kante des Kreisbogens ankommt. Die Grids, die sich an der Kante des Kreisbogens befinden, werden auf 1, somit als Hinderniss, gesetzt. Beim einmaligen Messen des Ultraschallsensors, kann somit ein kleiner Ausschnitt der Karte geschrieben werden. Um eine Karte des kompletten Raumes zu erstellen, muss der RAPTOR nun durch den ganzen Raum fahren, damit eine möglichst exakte Karte des Raumes erstellt werden kann.

4.5.2 Raum erkunden

Die Raumerkundung folgt einem simplen Muster. Zuerst fährt der RAPTOR solange gerade aus, bis er zu einer Wand gelangt. Die Implementierung sieht vor, dass der Roboter sich immer links von einer Wand bei der Erkundung halten soll, also der rechte Ultraschallsensor den direkten Abstand zur Wand misst. Der Roboter dreht sich hierbei solange nach links, bis er sich parallel zur Wand ausgerichtet hat. Eine annähernd optimale Ausrichtung nimmt der RAPTOR ein, sobald der nördliche Ultraschallseonsor einen leicht größeren Wert als der nordöstliche Ultraschallsensor hat. Es wird die Differenz der beiden Werte gebildet. Sobald die Differenz ca. 4cm beträgt, hat sich der Roboter optimal ausrichtet und bleibt stehen.

Nur wenn der RAPTOR parallel zur Wand ausgerichtet ist, werden der linke und der rechte Ultraschallsensor genutzt, um die Karte zu schreiben. Solange nun kein direktes Hinderniss im Weg ist, fährt der Roboter die Wand entlang und zeichnet die Karte auf. Eine Art der Hindernisse sind Ecken im Raum. Durch eine aufwendigere Korrektur der Richtung passt sich der Roboter der neuen Wand an und fährt weiter. Eine Sonderstellung nehmen Ecken ein, deren Winkel kleiner als 90° ist. Hierbei bleibt der RAPTOR komplett stehen und orientiert sich komplett neu. Dabei dreht er sich wieder nach links, bis der Messwert des nördlichen Ultraschallsensors größer ist als der des nordöstlichen.

Kommt der Roboter an einer Kante an, also eine Ecke, die in die entgegengesetzte Richtung zum RAPTOR liegt, wie zum Beispiel eine offene Tür, so fährt der Roboter so weit nach vorne, bis er sich so drehen kann, dass er gefahrenlos an der neuen Wand entlangfahren kann, ohne gegen die Wand zu fahren. Es wird stets ein Sicherheitsabstand zur Wand gehalten, der variabel eingestellt werden kann. Hat der Roboter die Wand komplett abgefahren, ist somit in der Karte eine Grenze eingezeichnet worden. Sofern es noch

unerkundete Flächen in der Karte gibt, deren Flächeninhalt größer als $1m^2$ ist, wird die Bahnplanung aktiviert, um den Raum komplett zu erkunden.

4.6 Player-Stage

Player-Stage besteht aus zwei Komponenten, welche Player und Stage heißen. Player ist eine Softwareschicht die den Zugriff auf die Roboterhardware unterstützt und den Zugriff auf Netzwerkschnittstellen ermöglicht. Stage ist ein Simulator für Player, der neben dem Roboter auch diverse Sensoren zur Verfügung stellt. Mit Stage ist es möglich Software zu testen, bevor sie am realen Roboter eingesetzt wird. Neben Stage gibt es noch Gazebo einen 3D Simulator. Abbildung 4.16 gibt die Zusammenhänge wieder. Player fungiert als Server. Selbst geschriebene Programme können sich über eine Socketverbindung mit Player verbinden. Wenn eine Simulationsumgebung gestartet wird, kann der Benutzer über ein selbstgeschriebenes Programm mit einem Roboter in der Simulationsumgebung mit Hilfe von Player kommunizieren. Es ist auch möglich Player mit einem realen Roboter zu verbinden. Zudem ist es möglich Player über eine Konfigurationsdatei zu konfigurieren. Die Simulationsumgebungen lassen sich ebenfalls konfigurieren. Stage wird benutzt um die Steuerungssoftware für den RAPTOR zu testen.

Die Simulationsumgebung Player-Stage war in der Softwareentwicklung eine grosse Hilfe, vor allem in den Zeiten, in denen die Roboterhardware nicht funktionierte und ein fahrender Roboter zum Testen der geschriebenen Software nicht vorhanden war. Zu Beginn wurde Player-Stage wenig genutzt, da der Roboter bis dato keinen oder nur wenige Ausfälle hatte. Nach dem ersten längeren Ausfall des Roboters stiegen dann alle Softwareteams gezwungenermaßen auf die Simulationsumgebung um und lernten dann erst die Vorteile eines Simulators kennen. Der Umstieg von der Simulation auf den Roboter war mit wenigen Handgriffen im Code durchführbar, das Verhalten war dann aber nicht immer genauso wie es im Simulator war. Das lag unter Anderem daran, dass die Sensorwerte, die aus der Simulationsumgebung gewonnen wurden immer exakt waren, in der realen Welt die Sensormesswerte aber ein Rauschen beinhalten. Dadurch, dass in der Simulation etwas funktionierte was in der realen Welt nicht genauso funktionierte, konnte der Programmierer den eigenen Algorithmus als Fehlerquelle ausschliessen und sich somit die gewonnenen Sensordaten aus dem Roboter einmal genauer ansehen und dort zur Korrektur ansetzen. Neben der problemlosen Umschaltung von der Simulation auf den Roboter und umgekehrt, war das gleichzeitige Arbeiten eine weitere Hilfe. Es konnte parallel am Roboter und am Simulator gearbeitet werden, ohne das Mitglieder der Projektgruppe auf andere warten mussten.

4.6.1 Installation

Im Folgenden wird die Installation von Player-Stage dokumentiert.

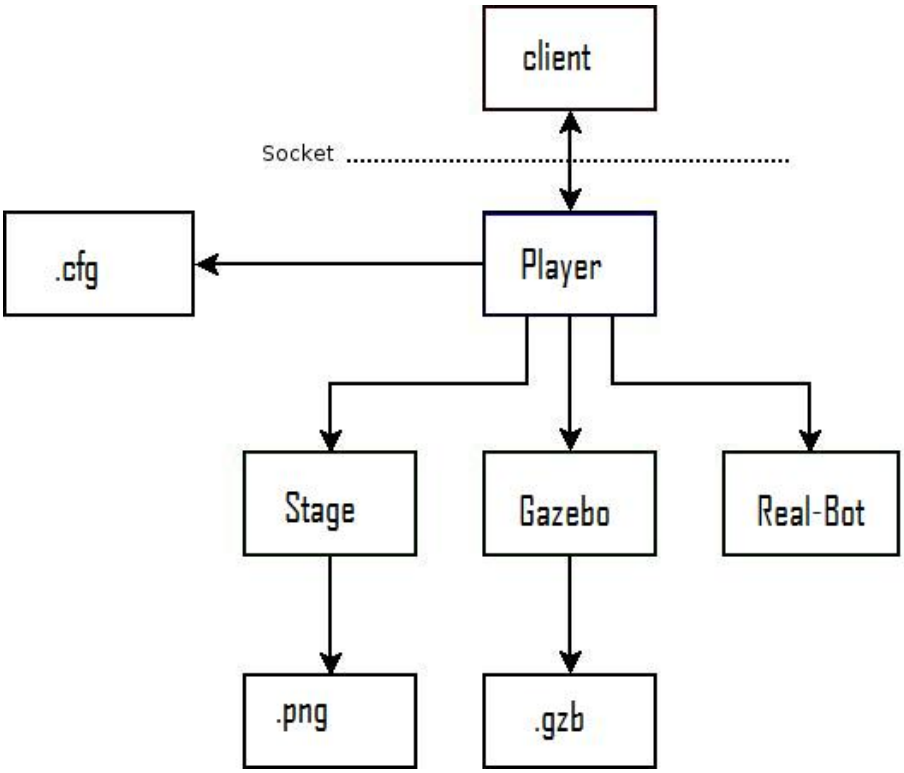


Abbildung 4.16: Komponentenuebersicht Player/Stage/Gazebo [25]

Player-Installation

Diese Installationsanleitung beschreibt die User-spezifische Installation von Player Stage in einem Unterordner "Player" des eigenen Home-Verzeichnisses. Standardmäßig sind die Installationsroutinen von Player-Stage darauf eingerichtet, eine lokale Installation für alle Nutzer vorzunehmen. Daher mussten einige Änderungen an dem empfohlenen Installationsablauf vorgenommen werden.

1. Download der aktuellen Version (player-<version>.tar.bz2) von SourceForge nach /home/<name>/Player.

2. Wechseln nach /home/ <name>/Player.

```
$ cd /home/<name>/Player}
```

3. Entpacken:

```
$ bzip2 -d player-<version>.tar.bz2
$ tar -xvf player-<version>.tar
```

4. Wechseln ins Player-Verzeichnis.

```
$ cd player-<version>
```

5. Setzen von Umgebungsvariablen.

```
$ export PATH=~/Player/bin:$PATH
$ export CPATH=~/Player/include:$CPATH
$ export LIBRARY_PATH=~/Player/lib:$LIBRARY_PATH
$ export PKG_CONFIG_PATH=~/Player/lib/pkgconfig:
        $PKG_CONFIG_PATH
$ export PYTHONPATH=~/Player/lib/python2.2/site-packages:
        $PYTHONPATH
```

6. Überprüfen, dass Flags nicht gesetzt sind:

```
$ env | grep CFLAGS
$ env | grep LDFLAGS
```

7. Falls vorheriger Punkt eine Ausgabe verursacht, müssen die Flags deaktiviert werden:

```
$ unset LDFLAGS
$ unset CFLAGS
```

8. Konfiguration starten.

   ```
   $ ./configure --prefix=/home/<Name>/Player
   ```

9. Kompilieren.

   ```
   $ make
   ```

10. Installieren.

    ```
    $ make install
    ```

Stage-Installation

Bevor Stage installiert werden kann, muss Player installiert und funktionstüchtig sein.

1. Download der neuesten Version von Stage vor dem Entpacken

   ```
   $ tar xzvf stage-<version>.tgz
   ```

2. In das entsprechende Unterverzeichnis wechseln

   ```
   $ cd stage-<version>
   ```

3. Konfiguration starten

   ```
   $ ./configure --prefix=/home/<Name>/Stage
   ```

4. Kompilieren

   ```
   $ make
   ```

5. Installieren

   ```
   $ make install
   ```

4.6.2 Konfiguration des Roboters

Beim Starten der Simulation wird, wenn nicht bewusst geändert, der mitgelieferte Simulationsroboter Pioneer gestartet. Dieser besitzt 16 Sonarsensoren. Da der realer Roboter nur vier besitzt, jeweils einen vorne, links, hinten und rechts, muss der Pioneer an unseren realen Roboter angepasst werden. In dem Ordner "worlds" der Stage-Installation befindet sich die Datei "pioneer.inc". In dieser wird scount 16 auf scount 4 gesetzt. Die vier Sonarsensoren werden durch Anpassen der Koordinaten an die richtigen Stellen gesetzt. Bei dem Pioneer wurden die Koordinaten dazu wie folgt gesetzt:

```
scount 4

   spose[0] [ 0.170 0.0 0 ]
   spose[1] [ -0.050 -0.130 -90 ]
   spose[2] [ -0.250 0.0 -180 ]
   spose[3] [  -0.050 0.130 90 ]
```

Durch diese Einstellungen wird ein Roboter wie aus Abbildung 4.17 generiert. Links im Konsolenfenster der Abbildung 4.17 sind die Sonardaten aus der aktuellen Position des Roboters zu sehen.

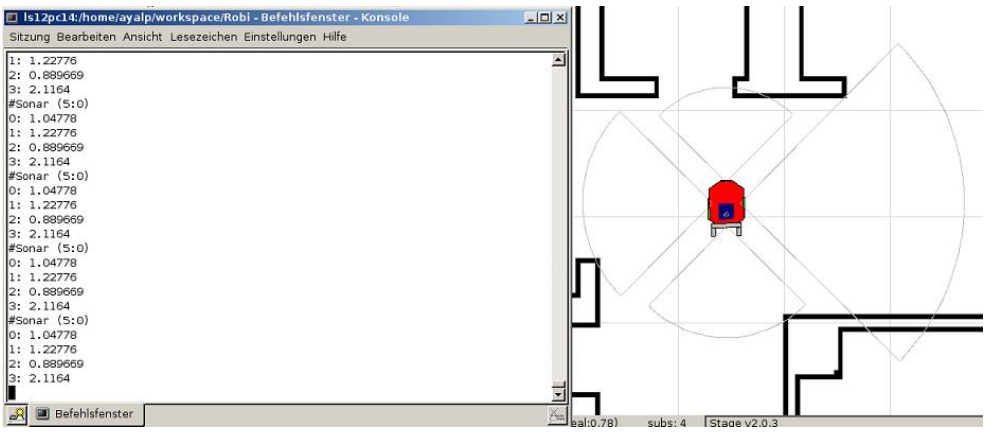


Abbildung 4.17: Simulationsumgebung

4.6.3 Simulation starten

Um Player-Stage letzendlich zu starten, müssen noch einige Umgebungsvariablen gesetzt werden. Dazu muss in der Konsole folgendes eingegeben werden:

```
export LD_LIBRARY_PATH= /home/<Name>/Player/lib
export PKG_CONFIG_PATH= /home/<Name>/Player/lib/pkgconfig/
```

Um eine einfache Welt aus Stage zu laden muss aus dem Verzeichnis

```
/home/<Name>/Player/player-<version>/server
```

die `simple.cfg` Datei wie folgt aufgerufen werden.

```
player /home/<Name>/Stage/stage-<version>/worlds/
                            everything.world.cfg
```

Es stehen auch noch andere Karten zur Verfügung, wie z.B. die `simple.cfg`, die nur aus einigen Räumen und einen Roboter besteht. Diese Karte ist für unsere Zwecke ausreichend. Ist die Simulation erfolgreich gestartet worden, dann müsste eine Karte zu sehen sein wie in Abbildung 4.18. Zu sehen sind dann sechs Roboter in verschiedenen Farben, zwei Geistobjekte und eine Büroumgebung. Wände werden durch schwarze Linien dargestellt.

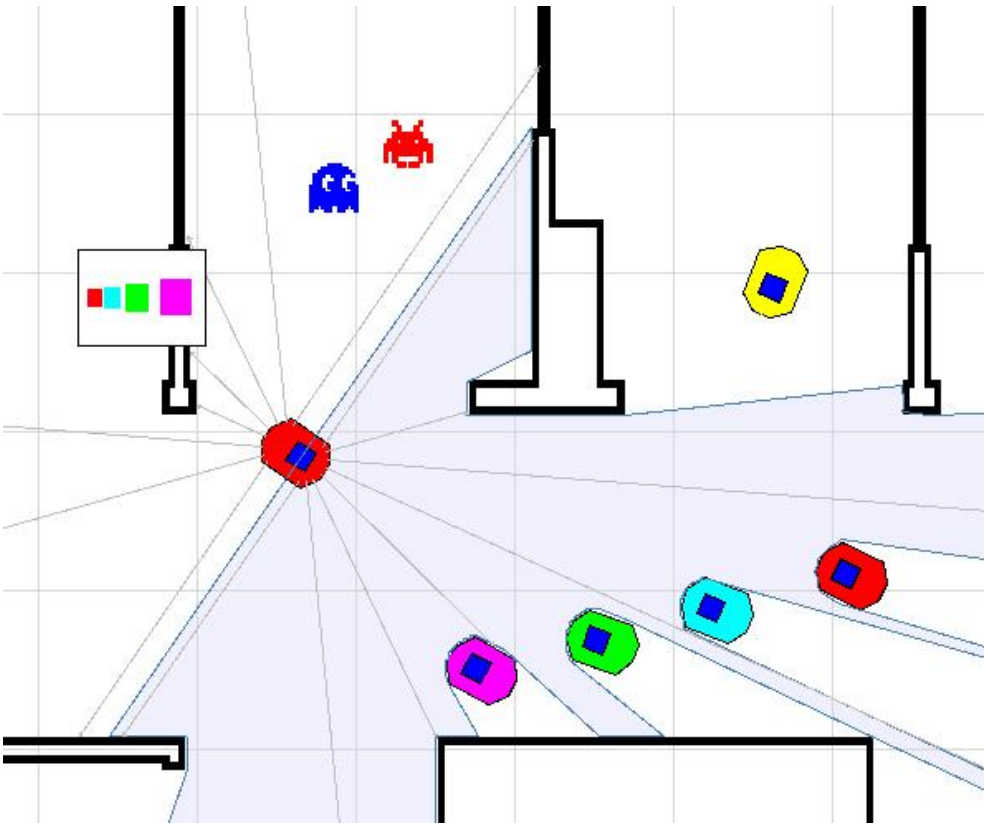


Abbildung 4.18: Simulationsumgebung

4.6.4 Eigene Programme erstellen

Für eigene Programme übernimmt Player die Rolle des Servers, so dass Benutzer ihre eigenen Programme zu Steuerung des Roboters schreiben können. Diese Programme bemerken keinen Unterschied zwischen einen realen Roboter und dem Roboter aus der Simulation.

Die Programme, die in der Simulation den Roboter steuern, funktionieren also auch auf dem realen Roboter ohne oder mit nur geringfügigen Modifikationen.

Um die Funktionen des Simulationsroboters benutzen zu können, muss die Bibliothek
<libplayerc++/playerc++.h>
in das C++ Programm inkludiert werden. Dann stehen dem Programmierer diverse Klassen, in Player-Stage auch Proxys genannt, zur Verfügung. Zuerst wird die Verbindung zu dem Roboter mit der Klasse PlayerClient im Konstruktor der C++ Klasse hergestellt. Zudem wird der Roboter mit den Sonar, den Bumpern und den Position2dProxy verbunden. Diese sind Instanzen der Klassen SonarProxy, BumperProxy und Position2dProxy.

```
1 Roboter():
2   m_roboter("localhost"),
3   m_sonarproxy(&m_roboter,0),
4   m_position2dproxy(&m_roboter,0),
5   m_bumperproxy(&m_roboter,0),
6   m_mapproxy(&m_roboter,0){}
```

Listing 4.23: Rücksetzfunktion

Nachdem der Roboter mit den Proxys verbunden worden ist, ist es möglich z.B. mit der Methode

```
1 void Roboter::setzeMotorGeschwindigkeit
2                (double x,double y, double drehrate)
```

Listing 4.24: Rücksetzfunktion

den Roboter zu steuern.

4.6.5 Makefile

Das Programm muss schliesslich kompiliert werden. Dazu wurde das folgende Makefile geschrieben. Aus den Roboter.cpp und Main.cpp wird die Main.bin erstellt.

```
all : Roboter.o Main.o
  g++ Main.o Roboter.o -o Main.bin `pkg-config \
      --cflags playerc++` `pkg-config --libs playerc++`

Roboter.o : Roboter.cpp
  g++ `pkg-config --cflags playerc++`  -c Roboter.cpp

Main.o: Main.cpp
  g++ `pkg-config --cflags playerc++`  -c Main.cpp

clean:
  rm -f *.o Main.bin
```

Mit dem Aufruf Main.bin startet die Simulation des programmierten Roboters, vorrausgesetzt die Simulationsumgebung Stage wurde, wie bereits beschrieben, gestartet.

4.7 Application Programming Interface

Ein Application Programming Interface, kurz API, dient dazu eine standardisierte Schnittstelle zwischen verschiedenen Programmteilen herzustellen. Es werden hierbei Funktionsaufrufe, sowie deren Wirkung im Gesamtkontext beschrieben, so dass Programmierer, welche diese API nutzen, einen intuitiveren Zugang zu den gebotenen Funktionalitäten erhalten. Im Fall des RAPTOR wurde entschieden eine API zu konstruieren, um hardwarenahe und hardwareferne Programmteile voneinander zu trennen. Dies hat mehrere Gründe. Zum einen benötigen die Entwickler der höheren Funktionen keine Kenntniss über die genaue Ansteuerung der Hardwarekomponenten. Zum anderen kann zu Testzwecken der eigentliche Roboter durch einen Simulator ersetzt werden. Dies ist möglich, weil eine API zwar Funktionsaufrufe definiert, aber nicht deren konkrete Implementierung. So ist es ohne weiteres möglich die Befehle entgegenzunehmen und entweder an reale Hardware oder simulierte Hardware weiterzuleiten.

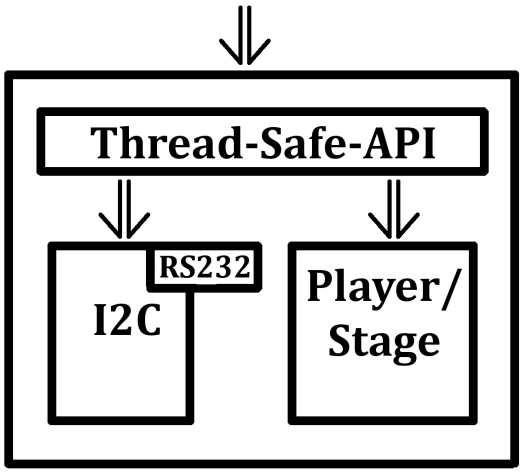


Abbildung 4.19: Ein Darstellung der API

Konkret wurde bei der Entwicklung eine äußerst komplexe Implementierung der definierten API vorgenommen. Wie auf Abbildung 4.19 zu erkennen ist, besteht sie aus mehreren Teilen, welche die Anfragen untereinander weiterreichen. Wird eine Funktion der API aufgerufen, wird diese Anfrage zunächst von dem Modul 'Thread-Safe-API' entgegengenommen. Diese sorgt nun in einem ersten Schritt dafür, dass keine zwei Threads gleichzeitig auf die API zugreifen können. Hierfür werden sogenannte Mutexe gesetzt, welche als Sperre für gleichzeitige Zugriffe dienen. Sollte also gerade ein Thread eine API-Funktion

nutzen und eine zweite Anfrage kommt, bevor die Erste abgearbeitet wurde, wird die Zweite solange aufgehalten, bis die Erste vollständig bearbeitet wurde. Dieses Vorgehen wird immer benutzt und ist somit unabhängig von der weiteren Implementierung der API. Da es sich um eine Kapselung jeder einzelnen Funktion auf die immer gleiche Art und Weise handelt, wurde auf einer sehr abstrakten Ebene programmiert. Der Großteil der 'Thread-Safe-API' wurde in Pre-Compiler-Quelltext geschrieben, was dazu führt, dass der eigentliche Quelltext erst beim Kompiliervorgang erzeugt wird. Dies hat zum Vorteil, dass schnell neue Funktionen in die API aufgenommen werden können, indem das entsprechende Pre-Compiler-Macro benutzt wird. Als Beispiel für dieses Vorgehen dient im Folgenden die Funktion 'readSonar()' , welche als Parameter die Angabe des gewünschten Ultraschallsensors erwartet und den Messwert von diesem zurückgibt. Es wird das Pre-Compiler-Macro 'THREAD_SAFE_FUNC1' verwendet. Dieses wird für Funktionen mit einem Parameter benutzt. Die Angaben in den Klammern geben den Datentyp des Rückgabewertes, den Funktionsnamen und den Datentyp des Parameters an.

```
1 THREAD_SAFE_FUNC1( int, readSonar, int )
```

Listing 4.25: Pre-Compiler-Macro einer Funktion in der Thread-Safe-API

Aus dieser einzelnen Zeile wird folgender Quelltext generiert:

```
1 int thread_safe_api::readSonar( int para_name )
2 {
3   if ( pthread_mutex_lock( m_mutex ) != 0 )
4     cerr<<``FEHLER: API_Mutex-Lock fehlgeschlagen''<<endl;
5   int rueckgabe = real_api.readSonar( para_name );
6   if ( pthread_mutex_unlock( m_mutex ) != 0 )
7     cerr<<``FEHLER: API_Mutex-UnLock fehlgeschlagen''<<endl;
8   return rueckgabe;
9 }
```

Listing 4.26: Pre-Compiler-Macro einer Funktion in der Thread-Safe-API

Wie zu erkennen ist, werden auch Fehlermeldungen generiert, die dazu dienen eventuelle Probleme mit den angesprochenen Mutexen zu lokalisieren. Der eigentliche Funktionsaufruf wird an das Objekt 'real_api' weitergeleitet, welches die angebundene Hardware repräsentiert. Diese kann zum einen reale Hardware sein, oder simulierte.

Im Verlauf der Entwicklung der höheren Funktionen ist es nötig geworden viele Tests mit angebundener Hardware zu machen. Da jedoch nur ein Roboter zur Verfügung steht, musste auf einen Simulator (siehe 4.6.5) zurückgegriffen werden. Um sicherzustellen, dass die Software, welche mit Hilfe des Simulators entwickelt wurde auch mit dem verwendeten RAPTOR funktioniert, mussten alle Funktionsaufrufe, welche in der API definiert sind unverändert bleiben. Es war also nötig alle Befehle der API an den Simulator in angepasster Form weiterzuleiten. Anders als das Ansprechen von realer Hardware, wird der Simulator über Funktionsaufrufe angesprochen. Dies dient einem intuitiveren Programmierstil und ist sehr einfach zu implementieren, da die Funkitonalitäten der API-Funktionen oft

denen der Simulator-Funktionen gleichen. Dennoch birgt die Anbindung eines Simulators ein paar Tücken, wie zum Beispiel die Tatsache, dass der verwendete Simulator über eine Netzwerkverbindung angesprochen wird, wogegen die Roboterhardware über ein Bussystem angebunden ist. Auch die Anbindung mehrerer Bibliotheken, die für den Simulator von essentieller Bedeutung sind, musste geregelt werden. Auch hier wurde wieder auf eine Lösung zurückgegriffen, die automatisiert die entsprechenden Anpassungen während des Kompiliervorgangs vornimmt. Die entsprechenden Pre-Compiler-Befehle abstrahieren die einzelnen Hardwareanbindungen so stark, dass ein Austausch durch die Angabe eines Kompilierparameters möglich wurde. Um den Übersetzungsvorgang zu starten wird zuerst angegeben, ob reale Hardware, oder ein Simulator genutzt wird. Im Falle des Simulators erfordert dies noch die Angabe des Netzwerkports, welcher für die Anbindung benutzt wird. Alle anderen nötigen Einstellungen werden automatisiert vorgenommen und benötigen keinerlei menschliches Zutun mehr.

Da aber auch auf realer Hardware gearbeitet werden soll, muss die API eine Anbindung an die reale Hardware besitzen. Auch diese wird wieder über eine Kompilierparameter ausgewählt. Die Anbindung erfolgt über einen I^2C-Bus, siehe Abschnitt 3.1. Über diesen kommuniziert die API mit dem entsprechenden Mikrocontroller, um diesem den entsprechenden Befehl zu übermitteln. Hierbei werden für die einzelnen Funktionalitäten keine Funktionsaufrufe verwendet, sondern es muss ein Datenstrom erzeugt werden, der über den I^2C-Bus geschickt wird. Es gestaltet sich sehr schwierig eine gute Umsetzung der API-Funktionen auf die Hardware zu erreichen. Zum Beispiel müssen die Messwerte der Radencoder ständig überwacht werden, was aber nicht von den höheren Funktionen erledigt werden kann. Somit muss diese Überprüfung in andere Funktionen eingebettet werden, um die Funktionsfähigkeit sicherzustellen. Dies resultiert aus der Tatsachen, dass die API nur auf Anforderung aktiv wird und nicht im Hintergrund mitläuft. Auch müssen die Funktionen der API so schnell wie möglich beendet werden, um eine Blockade des Programmablaufs zu verhindern. Es steht somit nur sehr begrenzte Rechenzeit zur Verfügung.

Die Kommunikation über I^2C erfordert ein grundlegendes Verständniss der Nutzung von Kernel-Devices. So gibt es nur drei Befehle, die beim I^2C-Bus genutzt werden können. Die Erste ist ‘ioctl()’, was für ‘IO-Controll’ steht. Hierüber werden Grundeinstellungen der Hardware vorgenommen. In unserem Fall ist dies die Bus-Adresse des entsprechenden Mikrocontrollers. Diese Einstellung muss vor jeder Kommunikation vorgenommen werden, weil sich diese seit der letzten Nutzung verändert haben könnte. Der zweite Befehl, welcher zur Verfügung steht ist ‘write()’. Hiermit werden die Daten auf den Bus geschrieben. Der Device-Treiber sorgt automatisch dafür, dass der richtige Empfänger diese Daten erhält. Zu beachten ist, dass nicht die Daten selber als Parameter angegeben werden, sondern die Speicherstelle an der sie zu finden sind. Es bedarf also zusätzlicher Vorbereitungen um einen entsprechenden Datenstrom für den Transfer vorzubereiten. Die dritte und letzte zur Verfügung stehende Funktion ist ‘read()’. Auch hier ist intuitiv klar, dass hiermit Daten vom Bus gelesen werden. Auch hier muss wieder ein Speicherbereich angegeben werden, welche aber dazu dient die vom Bus gelesenen Daten aufzunehmen. Es muss hierbei sichergestellt sein, dass der Speicherbereich entsprechend vorbereitet ist, um diese Daten aufzunehmen. Der Device-Treiber des I^2C-Bus sorgt sowohl bei ‘write()’ als auch bei ‘read()’

dafür, dass die richtigen Anfragen an den Bus gestellt werden und die Kommunikation ungestört ablaufen kann.

Aufgrund von Hardwaredefekten ist es nötig geworden den RAPTOR mit einer anderen als der geplanten Hardware zu betreiben. Diese Tatsache hat zur Folge, dass kein Anschluss für den I^2C-Bus mehr zur Verfügung steht. Es wurde hierfür eine Lösung in Form eines Adapters entwickelt, mit dem die Kommunikation zum I^2C-Bus über eine serielle Schnittstelle realisiert wird (siehe 3.5). Die bereits implementiere API musste daraufhin abgeändert werden, um mit der neuen Situation zurecht zu kommen. Wie bereits weiter oben festgestellt geschieht die Kommunikation mit dem I^2C-Bus über gerade einmal drei Befehle. Das neue Konzept sah nun vor, diese Befehle umzuleiten und statt einen Kernel-Treiber anzusprechen, diese Befehle anderweitig zu verwerten. Die Methode 'ioctl()' zum Beispiel wird anhand seiner Parameter ausgewertet und globale Zustandsvariablen dienen der Speicherung der übergebenen Werte. Auch 'read()' und 'write()' werden auf andere Befehle umgeleitet und der übergebene Datenstrom wird um spezielle Daten erweitert. Es wurde ein sogenanntes Header-Byte entworfen, welches den Beginn einer jeden Kommunikation darstellt. Dieses Header-Byte enthält Informationen über die nächste anstehende Aktion, die Zieladresse der Kommunikation und die Größe des Datenstroms. Diese Werte werden aus dem eigentlichen Datenstrom, der aufgerufenen Funktion und den über 'ioctl()' gesetzten globalen Zustandsvariablen extrahiert und auf entsprechende Weise zusammengestellt. Anschließend wird das Header-Byte mit dem Datenstrom vereint und über die serielle Schnittstelle an der angeschlossenen Adapter versandt.

Eine dritte Anbindung der API wurde bisher noch nicht behandelt und wird auch nur ganz kurz erwähnt. Es kann beim Start des Kompiliervorgangs auch gewählt werden, keinerlei Anbindung an Hardware herzustellen. In diesem Fall kommt eine sogenannten 'leere API' zum Einsatz, welche jeden Funktionsaufruf mit einem Fehler beantwortet. Diese dient dazu die Fehlerbehandlungsroutinen der höheren Funktionen zu testen. Eine Kapselung durch die 'Thread-Safe-API' findet aber aus design-technischen Gründen auch hier statt.

Wie gezeigt wurde, kann auf der Hardwareseite der API eine Vielzahl von Anbindungen realisiert werden. Aufgrund der Modularität des API-Konzeptes kann aber auch eine sehr große Zahl von Programmen auf die zur Verfügung gestellten Funktionalitäten zugreifen. Es wurde daher nicht nur die eigentliche Betriebssoftware des RAPTOR auf die API aufgesetzt, sondern auch einfache Programme. Als Beispiel kann hier die Fernbedienung angeführt werden, die zum Testen der Fahrfunktionen implementier wurde. Sie erlaubt es mittels der Pfeiltasten einer Tastatur den Roboter zu steuern. Auch mehrere Buchstabentasten werden genutzt, um spezielle Kommandos, wie das Schalten von LED oder das Setzen der Geschwindigkeit, zu realisieren. Ein weiteres Beispiel ist das sogenannte Raptordemo. Dieses dient dazu eine effektvolle Vorführung des Raptors zu ermöglichen. Es dient keinem sinnvollen Zweck, außer das Publikum zu begeistern. Es nutzt die Fahrfunktionen und die Ultraschallsensoren. Während des laufenden Programms fährt der RAPTOR solange geradeaus bis ein Hinderniss gefunden wird. In diesem Fall wird eine Drehung gestartet, bis die Geradeausfahrt wieder möglich ist. Es wird also deutlich, dass die API die Entwicklung verschiedenster Software auf vielfälltige Weise unterstützt. Es kann zusammenfassend festgestellt werden, dass die Entscheidung der Definition einer API absolut richtig war und

sich als überaus vorteilhaft erwiesen hat.

4.8 Webinterface

Das Webinterface ist eine HTML-Benutzerschnittstelle des Roboters, mit der der Benutzer alle Sensordaten und Kamerabilder des Roboters einsehen und den Betriebsmodus ändern kann. Die Sensordaten und die Kamerabilder werden vom Hauptprogramm in regelmäßigen Abständen aktualisiert, so dass der aktuelle Zustand der Sensoren des Roboters beobachtet werden kann. Die Bilder der beiden Kameras geben dem Benutzer Aufschluss über die verwendeten Bilder der Berechnungen für den Follow-Modus und die Schmutzerkennung. Zusätzlich kann der Benutzer über das Webinterface Termine für den Aufgabenplaner einstellen. Des Weiteren wird im Webinterface die aktuelle interne Karte des Roboters angezeigt, welche im Laufe des Betriebs erstellt wird. Der durch das Linux-Betriebssystem gestartete Webserver des Roboters ermöglicht es, auf die HTML-Seiten zuzugreifen. Der Zugriff darauf ist über das WLAN-Modul des Roboters gewährleistet. Da das Webinterface als eigenständiges Programm implementiert wurde, ist es losgelöst von der restlichen Softwarestruktur. Dies hat den Vorteil, neben der Portabilität, dass das Webinterface nur Rechenzeit verbraucht, wenn es benötigt wird, was zu Performancevorteilen führt. Es wird lediglich eine Header-Datei integriert, die wichtige Pfade in Form von Präprozessordirektiven bereitstellt. Um das Webinterface zu realisieren, wurden folgende Software-Technologien verwendet:

- HTML zur Darstellung des Webinterfaces in einem Webbrowser
- Cascading Style Sheets (CSS) zur Formatierung und Design der HTML-Seite
- JavaScript um kontextbezogenen Inhalt darzustellen und Bildkoordinaten anzuzeigen
- Common Gateway Interface (CGI) zur client- und serverseitigen Kommunikation
- C++ als Programmiersprache des CGI-Programms

Hypertext Markup Language

Die Benutzerschnittstelle des Roboters sollte möglichst einfach gestaltet werden und jedermann zugänglich sein. Die Entscheidung fiel auf HTML, da Inhalte so strukturiert und mit jedem beliebigen Webbrowser dargestellt werden können. Dazu muss eine WLAN-Verbindung mit dem Roboter hergestellt werden. Das Webinterface ist dann lokal unter `http://raptor/cgi-bin/Webinterface.html` zu erreichen.

Common Gateway Interface

Die Benutzerschnittstelle sollte nicht nur fähig sein, Daten anzuzeigen, sondern auch dem RAPTOR bestimmte Eingaben zu liefern, wie zum Beispiel das Ändern des Modus oder die

Eingabe von Zielkoordinaten für die Routenplanung. Die client- und serverseitige Kommunikation wurde hierbei mittels CGI umgesetzt. Mithilfe von CGI ist es möglich Programme bereit zu stellen, die von HTML-Dateien aus aufgerufen werden und selber dynamisch HTML-Code erzeugen. Um das Programm auszuführen, muss es sich im Verzeichnis „cgi-bin“ des Webservers befinden. Das Programm erzeugt zunächst die Ausgabe „Content-type: text/html“ gefolgt von einer Leerzeile. Danach wird der eigentliche HTML-Code ausgegeben, welcher über CGI an den Webbrowser übertragen und am Bildschirm angezeigt wird. CGI legt nur die Übergabe von Parametern fest und ist deshalb nicht an eine Programmiersprache gebunden, weshalb prinzipiell jede Programmiersprache, die auf dem System ausführbar ist, genutzt werden kann. Erwartungsgemäß fiel die Entscheidung auf C++ als CGI-Programm, da bereits die gesamte Software des Roboters in dieser Sprache implementiert wurde.

C++

Das datenverarbeitende CGI-Programm ist in C++ implementiert worden. Mittels des Programms „Webinterface.html“ werden zunächst alle Daten ausgelesen, die angezeigt werden sollen. Die Datei „sensordaten.txt“ wird vom Hauptprogramm regelmäßig erzeugt und enthält alle Sensorwerte zum Zeitpunkt der Erzeugung der Datei. In einer weiteren Datei „modus.txt“ wird der aktuelle Betriebsmodus des Roboters abgespeichert. Diese Datei wird vom Hauptprogramm ausgelesen, damit eine Modusänderung seitens des Webinterface im Hauptprogramm respektive in der Robotersteuerung initiiert werden kann. Es stehen sechs verschiedene Modi zur Auswahl: „Follow-Mode“, „Routenplanung“, „Bahnplanung“, „Kartenerstellung“, „Aufgabenplaner“ und „Kein Modus“. Im Follow-Modus kann der Roboter anhand der Frontkamera einem Gegenstand oder einem Menschen folgen. Dieser Modus wird in Kapitel 4.9.1 näher beschrieben. Wird der Modus Routenplanung angestoßen, kann der Benutzer dem Roboter konkrete Zielkoordinaten übergeben, die der Roboter daraufhin ansteuert. Hierfür besteht die Möglichkeit mit dem Mauszeiger die im Webinterface angezeigte Karte des Raumes an zu klicken. Sobald der Button „Ziel setzen“ betätigt wird, werden die Koordinaten, die in das untere Textfeld übernommen wurden, in der Modus-Datei abgespeichert, so dass sie von dem Hauptprogramm ausgelesen werden können. Daraufhin wird der Modus geändert, die Route von der aktuellen Position bis zur Zielposition wird berechnet und anschließend vom Roboter abgefahren. Wird der Modus Bahnplanung eingestellt, fährt der Roboter den kompletten Raum in gleichmäßigen Bahnen ab. Eine nähere Beschreibung hierzu befindet sich in Kapitel 4.9.2. Außerdem kann der Modus Kartenerstellung aktiviert werden. Dies hat zur Folge, dass der Roboter, sich an einer Wand orientierend, den kompletten Raum abfährt und dabei eine interne Karte des Raumes erstellt, welche, sobald verfügbar, im Webinterface angezeigt wird. In Kapitel 4.5 wird dieser Modus beschrieben. Als weitere Option steht der Aufgabenplaner zur Verfügung. Mit dem Aufgabenplaner wurde eine Funktionalität implementiert, mit der der Benutzer einmalige, tägliche oder periodische Termine einstellen kann, an denen der Roboter automatisch eine Bahnplanung durchführen wird, vorausgesetzt er ist eingeschaltet. Wird diese Option ausgewählt, erscheint im unteren Bereich der HTML-Seite eine Tabelle, in der alle ein-

gestellten Termine angezeigt werden. Darüber befinden sich mehrere Auswahlboxen, mit denen die Termine hinzugefügt werden können. Das Löschen eines Termins ist ebenfalls möglich. Durch Drücken des Buttons „Aufträge abschicken“ werden die Aufgaben in der Datei „aufgaben.txt“ abgespeichert. Diese Datei kann dann vom Hauptprogramm eingelesen werden, damit die Aufgaben zu den angegebenen Zeiten ausgeführt werden. Der letzte Modus lautet „Kein Modus“ und bringt den Roboter zum Stillstand.

Die Übergabe der vom Benutzer eingegebenen Daten der Web-Schnittstelle geschieht über ein HTML-Formular, welches die Daten per HTTP-Übertragungsmethode „GET“ an den Webserver sendet. Die Daten werden in der Umgebungsvariable „QUERY_STRING“ gespeichert, auf die das CGI-Programm „Webinterface.html“ zugreifen kann. In der Methode „checkQuery()“ wird der String, je nachdem was für eine Eingabe im Webinterface stattgefunden hat, ausgewertet und verarbeitet: Bei einer Modusänderung wird die Modus-Datei entsprechend gesetzt. Außerdem wird sichergestellt, dass auch genau dieser Modus im Webinterface angezeigt wird. Je nach Modus ändert sich auch die HTML-Seite dynamisch. So wird ausschließlich beim Modus „Aufgabenplaner“ die Tabelle zum Erzeugen der Aufgaben angezeigt und im Modus „Routenplanung“ ist der sonst ausgegraute Button zum setzen der Zielkoordinaten verfügbar. Des Weiteren wird geprüft, ob Zielkoordinaten für die Routenplanung übergeben wurden, welche gegebenenfalls in die Modus-Datei hinzugefügt werden müssen. Schließlich werden beim Aufgabenplaner die übergebenen Termine in einem bestimmten Format in die Aufgaben-Datei geschrieben. Nachdem die verschiedenen Konfigurationsdateien ausgelesen und die Benutzereingaben geprüft wurden, wird der HTML-Code erzeugt. Die HTML-Seite besteht aus dem festen Code des Grunddesigns der Seite, den dynamischen Daten, die vom Roboter erzeugt werden und den Benutzereingaben. Nach dem Einlesen und Verarbeiten der Daten, können sie an den richtigen Positionen innerhalb des HTML-Codes platziert werden.

Cascading Style Sheets

Das äußere Erscheinungsbild der Seite wurde mit Cascading Style Sheets realisiert. CSS ist eine ergänzende deklarative Sprache zu HTML, mit der eine HTML-Seite gesondert formatiert werden kann. Darüber hinaus können mittels bestimmter CSS-Eigenschaften auch die Positionierung von HTML-Elementen manipuliert werden. Formatierungen und Positionierungen innerhalb des HTML-Codes fallen damit weg. Der HTML-Code wird an bestimmten Stellen ausgezeichnet, damit das Design mit einer externen CSS-Datei geändert werden kann. Dadurch wird der Inhalt der HTML-Seite von dem Design getrennt.

JavaScript

Es wurde ebenfalls noch die clientseitige Skriptsprache JavaScript verwendet. Der JavaScript-Code wird mit der Datei „raptor.js“ eingebettet und beinhaltet spezielle Zusatzfunktionen. Zum einen sind Methoden implementiert worden, die zum Auslesen der Koordinaten auf der als Bitmap angezeigten Karte dienen. Diese Koordinaten dienen der Routenplanung als Zielposition für den Roboter. Zum Anderen beinhaltet das Skript Methoden, die dafür sor-

gen, dass die passenden Auswahlboxen für den Aufgabenplaner kontextbezogen erscheinen, um eine gewisse Datenkonsistenz beizubehalten.

4.9 Robotersteuerung

Die Robotersteuerung ist die Instanz in der Software, welche die verschiedenen Modi des Roboters in reale Fahrbefehlsequenzen umsetzt. Hierbei wird für jeden Modus eine individuelle Abfolge von Fahrbefehlen aufgestellt, welche dann abgefahren werden. Da sie jeden verfügbaren Modus des Roboters realisiert, werden Instanzen aller Klassen, welche die verschiedenen Modi implementieren, erzeugt und weiterverwendet. Weiterhin werden für eine korrekte Ausführung genaue Sensorwerte und aktuelle Positionskoordinaten benötigt, welche von einem Objekt aus der Klasse Odometrie bereitgestellt werden. Falls eine Odometrie nicht verfügbar ist und keinerlei Informationen über aktuelle Sensorwerte und Positionskoordinaten vorliegen, kann auch eine Blindfahrt des Roboters aktiviert werden, allerdings erreicht sie aufgrund fehlender Umgebungsinformationen keine hohe Zuverlässigkeit und ist stark abhängig von vorgegebenen Parameterwerten. Ihre Ausführung hat daher einen rein testorientierten Hintergrund.

Eine Hauptfunktion der Robotersteuerung, welche auch vom Hauptprogramm aufgerufen wird, ist:

```
1   void robosteuerung::initialize();
```

Listing 4.27: Initialisierung

Diese Funktion enthält alle Hauptfunktionalitäten für die verschiedenen Funktionsmodi des Roboters, wobei die Modi sich durch eine einfache if-else-Fallunterscheidung voneinander ausschließen. Sie läuft in einer unendlichen while-Schleife und wird somit fortlaufend aufgerufen, solange der Roboter eingeschaltet ist. Falls im Moment keine Aktion ausgeführt werden soll, führt sie einen seperat definierten Modus namens 'kein Modus' aus, welche den Roboter zum Stoppen zwingt und ihn in Wartestellung für die nächsten Befehle bringt. Falls der Roboter unerwartet einen ungültigen Modus erhält, wird das gleiche Verhalten erzwungen.

Die Komplexität der Fahrbefehlsequenzen fällt je nach Modus unterschiedlich aus. Während die Bahn- und Routenplanung zur Berechnung der optimalen Fahrstrecke Stacks von Felder benötigen und eine ständige Aktualisierung der Positionskoordinaten erfordern, reicht für den Follow-Modus der Ausrichtungswinkel des Roboters zum zu verfolgenden Zielobjekt. Die Erstellung einer aktuellen Karte des Raumes wiederum benötigt eine Aktivierung der Odometrie.

Die zweite, wichtige Hauptfunktion der Robotersteuerung ist:

```
1   void robosteuerung::fahreStackAb();
2   (optional: void robosteuerung::fahreStackAbBlind();)
```

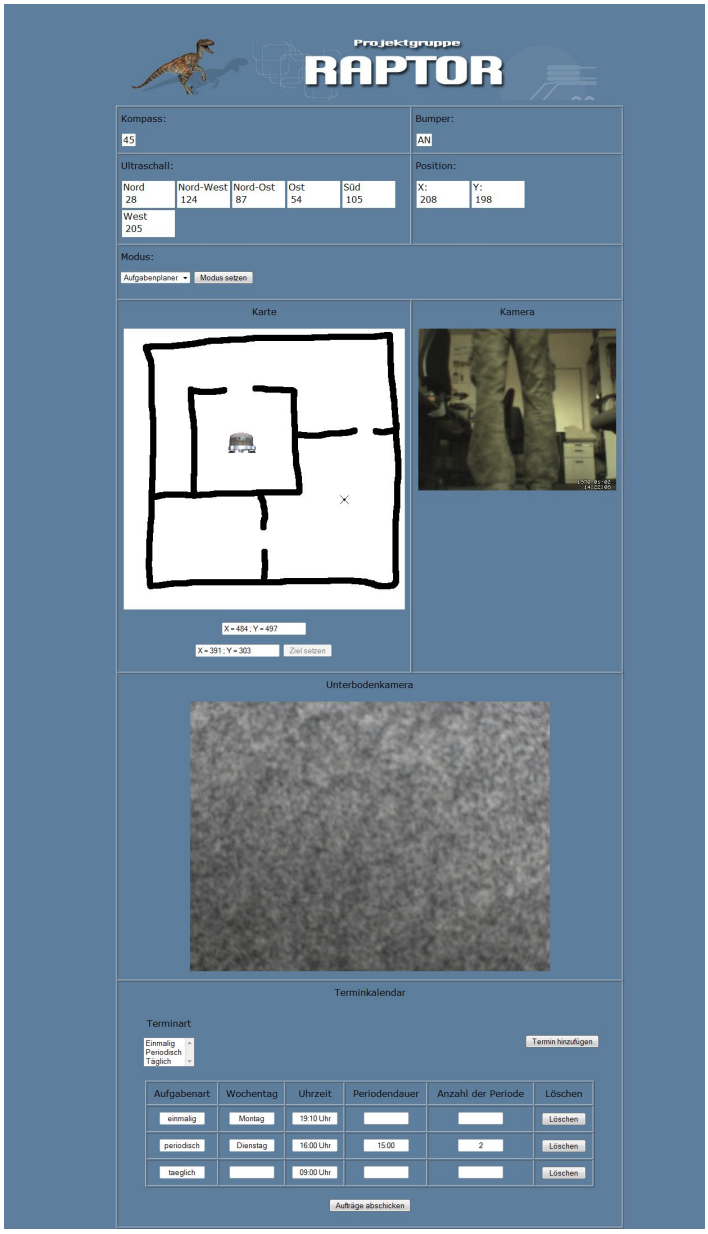


Abbildung 4.20: Webinterface

Listing 4.28: Funktion zur Abarbeitung von Stacks

Diese Funktion wird für die Bahnplanung und die Routenplanung benötigt, da diese ihre Funktionalität dadurch realisieren, dass Stacks von Feldern in einer effizienten Reihenfolge aufgebaut und abgefahren werden. Zusätzlich enthält sie Funktionen für eine primitive Fehlererkennung und Korrektur. Fehlerszenarien kommen zum Vorschein, wenn der Roboter sich an einer falschen Position befindet oder ein vorher nicht einkalkuliertes Hindernis zum Vorschein kommt. Der erste Fall zeigt auf, dass der Roboter physikalisch nicht genau genug fährt und minimal von der Strecke abweicht, somit korrespondieren die reale Position und die ideale Position des Roboters nicht miteinander. Ein solches Verhalten kann nicht vollständig vermieden werden, da ein exaktes Fahren unter realen Bedingungen sehr schwierig zu bewerkstelligen ist. Eine Ausweichlösung wurde realisiert, indem die fehlerhafte Abweichung in der zurücklegten Strecke durch eine zusätzliche Fahrt zur korrekten Position kompensiert wird. Der zweite Fehlerfall erscheint bedingt durch die Tatsache, dass bewegliche Hindernisse erst im Laufe einer Fahrt des Roboters erscheinen. Damit diese Hindernisse rechtzeitg erkannt werden, finden in regelmäßigen Abständen Anfragen an die vorderen Ultraschallsensoren statt. Sollte trotzdem eine Kollision mit einem Hindernis stattfinden, wird der Roboter beauftragt, zurückzuweichen, die Bumper zurückzusetzen und die vorhandene Karte zu aktualisieren. Abschließend wird eine neue Fahrt berechnet.

Zwei weitere, wichtige Funktionen sind:

```
1  int robosteuerung::getWegpunktpositionX(stack<Feld>
2                                          *route);
3  int robosteuerung::getWegpunktpositionY(stack<Feld>
4                                          *route);
```

Listing 4.29: Rückgabe der aktuellen Koordinaten

Sie liefern die aktuelle X- und Y-Koordinate des Roboters, wobei die Funktionen selber wiederum auf die Werte aus der Odometrie zugreifen und sie mit einer umgebungsabhängigen Abweichung versehen. Aus diesen Informationen kann die aktuelle Zielrichtung zur nächsten Position bestimmt werden, realisiert durch die folgende Funktion:

```
1  int robosteuerung::berechneWegpunktrichtung();
```

Listing 4.30: Rückgabe der Zielrichtung zur nächsten Position

Im Gegensatz zu den anderen Modi setzt die Realisierung des Follow-Modus keinerlei Berechnungen mit Datenstrukturen wie Stacks voraus, stattdessen richtet die Robotersteuerung die Fahrbefehle des Follow-Modus danach, welcher aktuelle Winkel und Abstand zum Zielobjekt gegeben ist. Zusammengefasst wird dies in der Funktion

```
1   void robosteuerung::fahrefollow();
```

Listing 4.31: Funktion für den Follow-Modus

für den Follow-Modus, welche auch als einzige Funktion in der Robotersteuerung (neben dem Konstruktor) die Existenz eines Kameraobjekts voraussetzt.

Abschließend sei erwähnt, dass die Robotersteuerung neben seiner Vielzahl an Funktionen auch eine große Menge an Hilfsvariablen zur Abfrage des Roboterzustandes besitzt. Die beiden wichtigsten Variablen sind *modus_changed* und *active*. Die Variable *modus_changed* überprüft, ob der aktuelle Modus des Roboters verändert wurde. Falls dieses stattfand, müssen neue Fahrbefehle zusammengestellt werden, ansonsten wird wie vor der Überprüfung weiter verfahren. Mit Hilfe von *active* wird hingegen überprüft, ob der Roboter sich in einem aktiven Zustand befindet oder im Stillstand verweilt. Diese Variable ist nützlich, falls vom Benutzer keine expliziten Befehle eingegeben werden und der Roboter in Leerlaufzeiten seine täglichen, automatischen Aufgaben abarbeiten kann.

Letztlich symbiotisiert die Robotersteuerung die komplexen Algorithmen der Software mit dem primitiven Fahrbefehlen der Hardware und ist deshalb ein unverzichtbarer Bestandteil des Roboters.

Im Folgenden werden die einzelnen, verfügbaren Modi des Roboters näher erläutert.

4.9.1 Follow-Modus

Der RAPTOR soll im Follow-Modus einem Menschen mit Hilfe einer Kamera, unter Wahrung eines Sicherheitsabstandes, folgen. Dies kann zum Beispiel dazu dienen, den Roboter von einem Ort zu einem anderen zu bekommen, ohne ihn selbst tragen zu müssen. Der Roboter könnte auch, während er einem Menschen folgt, schwere Lasten tragen. Mit der Kamera werden zwei aufeinanderfolgende Bilder aufgenommen und analysiert. Die Differenz, die sich aus den beiden zeitlich unterschiedlichen Bildern ergibt, gibt Aufschluss darüber, in welchen Bildbereichen Bewegung stattgefunden hat und in welchen nicht. Es wurden zwei Verfahren entwickelt um dieser Aufgabe gerecht zu werden. Das erste Verfahren betrachtet ausschließlich die Farben in den beiden Bildern. Somit ist es dem Roboter möglich einem Menschen mit einem bestimmten Merkmal, z.B. einer roten Hose, zu folgen. Das zweite Verfahren bestimmt für einen Block der Größe n x n, mit $n \geq 4$, die korrespondierende Position im darauflegenden Bild. Es ist auch denkbar dies für jedes einzelne Pixel zu machen, im Allgemeinen aber ist der Luminanzwert eines Bildpunktes nicht charakteristisch genug, um den Bildpunkt in einem anderen Bild zu identifizieren. Das verwendete Verfahren ist auch als Blockvergleichs-Verfahren (Block matching) bekannt.

Für das Farberkennungsverfahren werden die Bilder zunächst aus dem RGB-Farbraum in den HSV-Farbraum konvertiert. Dadurch fällt es leichter die jeweils gewünschte Farbe in einem Bild, relativ unabhängig von der Beleuchtung zu finden.

Nachdem beide Bilder konvertiert wurden, wird in dem ersten Bild ein Bereich gesucht, in dem die Farbe am stärksten vertreten ist, dabei ist es auch möglich nach bestimmten

Formen zu suchen, wie z.B. einem hohen roten Rechteck, als Abstraktion für ein rotes Hosenbein. Im nächsten Bild wird in der unmittelbaren Umgebung im Bild wieder nach diesem Muster gesucht. Als Ergebnis liefert die Suche eine korrespondierende Bildspaltennummer, für die Muster nach denen gesucht wurde. Aus diesem Ergebnis wird anschließend der Winkel des gefundenen Musters relativ zur Kamera zugeordnet. Die Berechnung der Winkel zu jeder Spalte eines Bildes wird zu Beginn des Follow-Modus einmal berechnet und für die nachfolgenden Zuordnungen gespeichert. Zur Berechnung werden die Breite und Höhe der gemachten Bilder, Kamerahöhe und der Horizontale sowie der vertikale Öffnungswinkel der Kamera herangezogen. Dabei dient die vertikale Winkelzuordnung zu einer Zeile eines Bildes, der Entfernungsschätzung bei der Verfolgung eines Menschen.

Für das Blockvergleichs-Verfahren werden die RGB-Bilder in Graustufenbilder konvertiert. Dieses Verfahren liefert als Ergebnis Verschiebungsvektorfelder für zwei aufeinanderfolgende Bilder. Die Technik dieses Verfahrens besteht darin, für jeden Block des ersten Bildes, diejenige Position im zweiten Bild zu bestimmen, für die die Ähnlichkeit am größten ist. Die Suche nach der Position größter Ähnlichkeit erfolgt innerhalb eines vorab definierten Suchbereiches, der um die unverschobene Position des Blockes definiert ist. Denkbar wäre, den Suchbereich so groß, wie das Bild zu wählen, dies geschieht aber auf Kosten der Rechenzeit. Der Suchbereich sollte daher so klein wie möglich, aber auch so groß wie nötig sein, um alle Bewegungen im Bild detektieren zu können. Auch die Wahl der Blockgröße hängt davon ab, wie genau die Ergebnisse sein sollen. Hier gilt, umso kleiner die Blöcke umso rechenintensiver und genauer sind die Ergebnisse. Analog gilt, umso grösser die Blöcke umso rechenzeitsparender und ungenauer die Ergebnisse. Für die Bewertung der Ähnlichkeit zwischen Blöcken wird die normierte Kreuzkorrelationsfunktion (normalized cross correlation function, NCCF) verwendet. Möglich wäre es auch andere Funktionen zu verwenden, wie z.B. den quadratischen Fehler (squared error, SE) oder den absoluten Fehler (auch 'displaced frame difference' (DFD) genannt). Ein Vorteil der NCCF ist die Robustheit gegenüber Kameraschwenks [1]: Dadurch werden die gefundenen Verschiebungsvektoren, durch die Bewegung des Roboters während der Verfolgung eines Menschen, nicht verfälscht.

Möglich wäre eine Kombination aus dem ersten und den zweiten Verfahren, um robustere Ergebnisse zu erzielen. Durch die beschränkten Rechenressourcen ist dies aber nicht möglich, auch das NCCF-Verfahren alleine ist mit den bisherigen Ressourcen kaum zu bewältigen.

4.9.2 Bahnplanung

Die Bahnplanung ist einer der internen Modi in der Robotersteuerung, und hat die Aufgabe, in einem bereits erkundeten Raum eine möglichst effiziente Route für die Komplettabfahrt des Raumes zu erstellen und diese dann entsprechend abzufahren. Diese Route soll alle für den Roboter erreichbaren Stellen erfassen, damit die gründliche Reinigung des Raumes gewährleistet ist. Somit stellt die Bahnplanung den „Saugmodus“ dar, der nach dem Einschalten des Roboters und Erkundung des Raumes im Normalbetrieb aufgerufen wird.

Die Bahnplanung arbeitet auf der internen Karte des Raumes, die in GridMap-Format

zuvor von der Odometrie erstellt und abgespeichert wurde. Das schränkt das Einsatzgebiet vom Saugmodus auf bereits erfasste Räume ein, für einen neuen Raum muss somit die Kartenerstellung zuerst ausgeführt werden, bevor die Bahnplanung starten kann. Die abzufahrende Route wird als eine Liste von benachbarten Feldern zurückgegeben, die auf einem Stack gespeichert sind. Somit hat die Route für die Komplettabfahrt nach der Erstellung das gleiche Format wie alle anderen Routen in der Robotersteuerung, und kann mit der „fahreStackAb“-Methode abgefahren werden. In Folgenden wird also nur die Berechnung der Route beschrieben, da diese sich von allen anderen Modi unterscheidet.

Für die Berechnung der Route für die Bahnplanung wird eine eigene Klasse „Bahnplanung“ benutzt, welche eine einzige public-Methode namens „start“ hat. Diese nimmt den Zeiger auf die Karte und die Startkoordinaten (die Stelle, wo der Roboter sich gerade befindet) entgegen, und erzeugt als Ausgabe einen Stack von Feldern, der dann als Route an die Robotersteuerung weitergegeben wird. Die Hauptaufgabe der „start“-Methode besteht in der korrekten Neuinitialisierung von internen Variablen und temporären Werten, und dem Aufruf von Unterfunktionen, welche die eigentliche Route berechnen.

Als Grundlage für die Berechnung der Route wird ein modifizierter Floodfill-Algorithmus [34] verwendet. Dieser Algorithmus kommt ursprünglich aus der Grafikverarbeitung und dient dazu, ausgehend von einem Pixel, zusammenhängende Areale gleicher Farbe zu bestimmen. Es wird unter anderem für das „Eimer“-Tool in Paintprogrammen benutzt, um die Bereiche in einem Bild zu bestimmen, die mit gleicher Farbe gefüllt werden sollen. Ein anderes Verwendungsbeispiel sind Spiele wie „Minesweeper“, wo mit diesem Algorithmus die Felder berechnet werden, die der Spieler geöffnet hat. Der Algorithmus nimmt als Eingabe den Startknoten, die Zielfarbe und die Füllfarbe. Zuerst wird der Startknoten mit der Füllfarbe gefärbt, danach für alle Nachbarn des Startknotens, die mit der Zielfarbe übereinstimmen, wird der Algorithmus rekursiv neu aufgerufen, mit dem Nachbarknoten als neuem Startknoten.

```
1 Flood-fill (node, target-color, replacement-color):
2 if node-color != target-color then return;
3 node-color = replacement-color;
4 Flood-fill (west of node, target-color, replacement-color);
5 Flood-fill (east of node, target-color, replacement-color);
6 Flood-fill (north of node, target-color, replacement-color);
7 Flood-fill (south of node, target-color, replacement-color);
8 return;
```

Listing 4.32: Floodfill-Algorithmus in Pseudocode

In einem zweidimensionalen Array hat jeder Knoten acht Nachbarn, meistens reicht aber die Betrachtung von nur vier (oben, unten, links und rechts von dem Startknoten), damit der Zielbereich komplett gefüllt werden kann. Dieses Vorgehen hilft dabei, Speicher- und Rechenzeitanforderungen an den Algorithmus zu verringern, und wird 4-Way Floodfill genannt. Die Betrachtung aller acht Nachbarn (8-Way Floodfill) dagegen verbraucht mehr Systemressourcen, kann aber dabei helfen, den Zielbereich besser abzudecken.

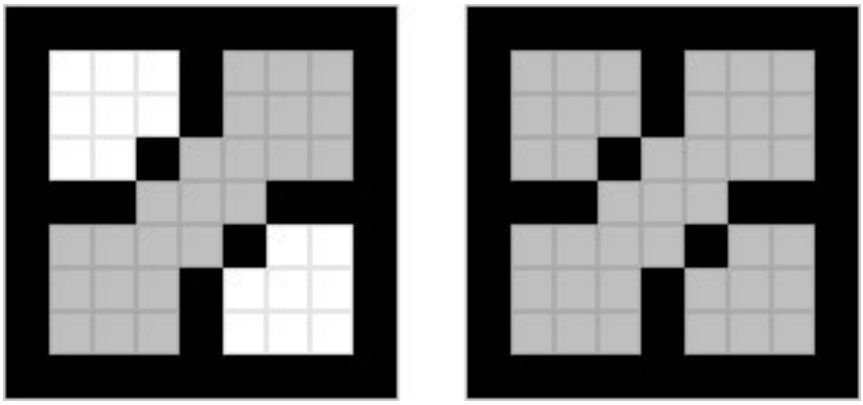


Abbildung 4.21: Unterschiedliche Färbung mit dem 4-Way und 8-Way Floodfill [34]

Durch die Eigenschaft, einen zusammenhängenden Bereich unabhängig von der Form komplett zu füllen, ist der Floodfill-Algorithmus auch sehr gut dazu geeignet eine Route zu erstellen, die den gesamten Raum erfasst, der abgefahren werden soll. Die interne Gitterkarte des Roboters besteht aus einzelnen Feldern, welche die Rolle der Knoten in dem Algorithmus übernehmen. Eine Route entsteht dadurch, dass die Felder in der Reihenfolge ausgegeben werden, in der sie gefärbt wurden.

Der rekursive Ansatz ist sehr einfach zu implementieren und auf Fehler zu überprüfen, allerdings kann er nur bei relativ kleinen Räumen angewandt werden, da sonst der für die Rekursion notwendige Speicherplatz auf dem Programmstack schnell erschöpft wird. Deswegen wurde der Algorithmus so modifiziert, dass anstelle von rekursiven Aufrufen die Knoten in einer Liste (hier: Stack) gespeichert werden, die dann sukzessiv abgearbeitet wird. Zuerst wird ein Knoten vom dem Stack genommen, dann alle seine noch nicht gefärbten Nachbarn auf den Stack gelegt. Anschließend wird der Vorgang solange wiederholt, bis der Stack komplett leer ist. Dieser Ansatz erlaubt es, eine korrekte Route selbst für sehr große Räume zu berechnen (in den Tests konnte sogar für einen 200*200 Meter großen Raum eine Route problemlos erstellt werden).

Die Effizienz der berechneten Route ist dadurch gewährleistet, dass der Algorithmus nur solche Felder betrachtet, die bisher noch nicht gefärbt (also noch nicht besucht) wurden, jedes Feld wird also nur einmal angefahren. Da die Nachbarn eines Feldes immer in einer bestimmten Reihenfolge betrachtet werden, bilden sich lange, gerade Strecken, bis der Algorithmus auf eine Wand trifft. Dann wird eine parallele Strecke in umgekehrter Richtung abgefahren. Somit wird eine mäanderformige Bahn gebildet, die den Raum möglichst komplett abdeckt. Da der Raum aber häufig aus mehreren zusammenhängenden Teilbereichen von Feldern besteht, kann es vorkommen, dass nach Abfahren eines bestimmten Teilbereichs keine benachbarten Felder mehr gefunden werden. Ist der Stack hier noch nicht leer, so wird zu dem letzten noch nicht besuchten Nachbarfeld gesprungen, und ab dieser Stelle die Route weiter berechnet. Durch diesen Sprung können also „Löcher“ in der Bahn entstehen, d.h. zwei Felder, die nicht direkt benachbart sind. Da die Robotersteuerung nur eine Route aus jeweils zueinander benachbarten Feldern annimmt, und die Rückfahrt auf

dem gleichen Weg sehr ineffizient wäre, wird an dieser Stelle mit Hilfe der Routenplanung eine Route zwischen den zwei voneinander entfernten Feldern berechnet. Diese Route wird dann in die eigentliche Bahn an die Stelle zwischen den beiden Feldern eingefügt.

Da die sehr hohe Präzision der Routenplanung bei der Wegberechnung hier nicht notwendig ist, benutzt die Bahnplanung eine modifizierte Version der Routenplanung, die in wenigen Fällen etwas längere Pfade erzeugt, dafür aber 10 bis 20 Mal so schnell arbeitet. Dies ist insbesondere deswegen von Vorteil, da die Sprünge und somit die neuen Routenberechnungen mehrmals vorkommen können.

Abbildung 4.22 zeigt ein Beispiel der Bahnplanung im Simulator.

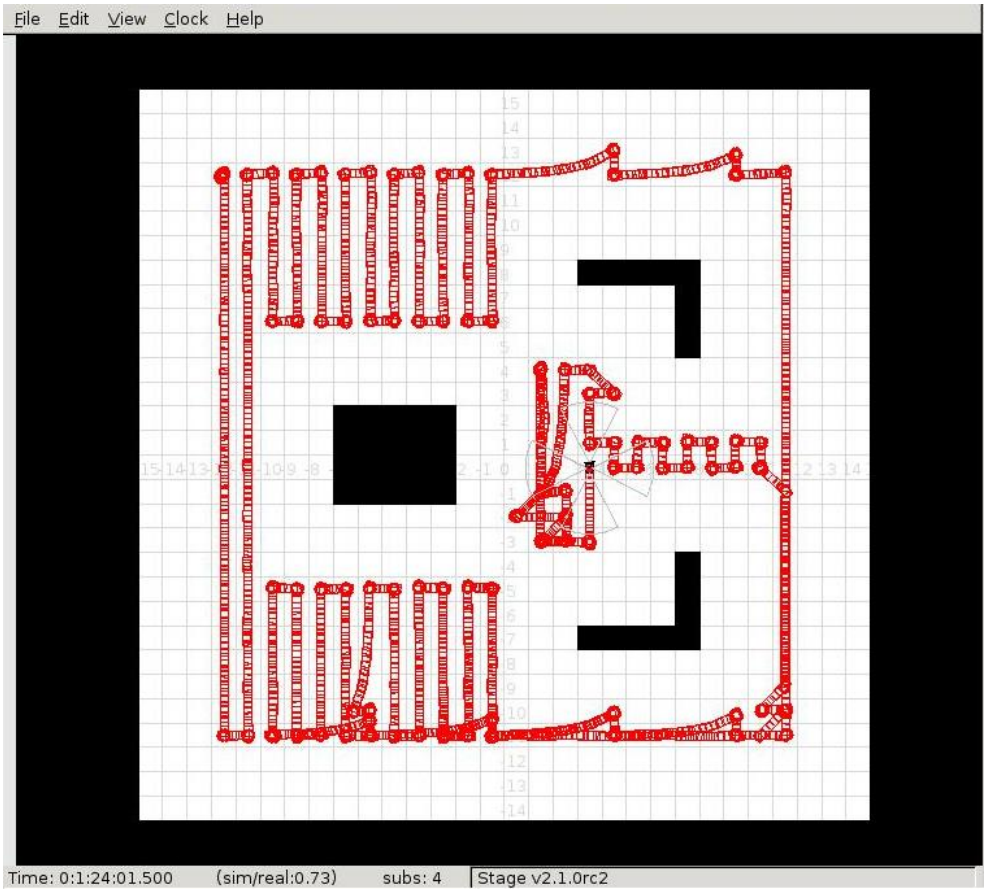


Abbildung 4.22: Bahnplanung im Simulator

4.9.3 Routenfahrt

Es folgt nun die Routenplanung. Diese dient dazu einen möglichst optimalen Weg von einem angegebenen Startpunkt zu einem gewünschten Zielpunkt zu berechnen. Diese Funktionalität ist innerhalb der Gitterkarte gekapselt, weil ein Algorithmus verwendet wird, der direkt auf der Gitterkarte arbeitet und eine speziell für solche Zwecke implementierte

Sondereigenschaft der einzelnen Gitternetzknoten, welche im folgenden als Feld bezeichnet werden, nutzt. Es handelt sich hierbei um eine einfache Variable in den Datenstrukturen der Gitterkarten, welche höhere Funktionen bei Berechnungen unterstützen und entlasten soll. In dem Fall der Routenplanung werden hier Schätzwerte der Entfernungen vom Startpunkt gespeichert und anschließend ausgewertet. Bevor mit der detaillierten Beschreibung dieser Funktion begonnen wird, werden noch ein paar Vorüberlegungen zur Routenberechnung angeführt. Die Gitterkarte ist intern unabhängig von der Größe der einzelnen Felder. Dennoch müssen ihr diese Angaben mitgeteilt werden. Auch die Größe des Roboters ist hier von Bedeutung. Es muss ein gewisser Abstand zu den Objekten im Raum eingehalten werden. Um diesen zu berechnen, werden die gerade erwähnten Werte in Form von Konstanten gebraucht.

Konstanten tragen der Tatsache Rechnung, dass es Werte gibt, die sich nicht ändern. Der Roboter wird nicht plötzlich schrumpfen und die Räume sich auch nicht ausdehnen. Daher können mit Hilfe von Konstanten die Berechnungen zur Routenplanung durchgeführt werden. Hierfür muss aus den beiden Konstanten die Anzahl der freizuhaltenden Felder berechnet werden, die den Abstand vom Zentrum des Roboters zu den Objekten im Raum darstellen. Ist dies geschehen, kann dies bei der Schätzung der Entfernungen berücksichtigt werden. Für diesen Vorgang wird folgender Algorithmus umgesetzt, der sich an dem Bereichswachstumsverfahren [19] orientiert.

```
1   weise Startfeld die Entfernung 1 zu
2   lege Startfeld auf Stack
3   for (alle Felder auf dem Stack) do
4     hole TOS -> aktuelles Feld
5     for (alle Nachbarn des aktuellen Feldes) do
6       wenn (Nachbarfeld nicht belegt)
7       and (Nachbarfeld nicht zu nah an Objekt) do
8         weise Nachbarfeld sein Entfernung zu
9         lege Nachbarfeld auf Stack
10      end
11    end
12  end
```

Listing 4.33: Entfernungsschätzung

Die Entfernungsschätzung kann hier auf vielfältige Art und Weise erledigt werden. Es wurde dabei Wert auf die Verwendung von simplen und schnellen Berechnungen gelegt. Als Nachbarn werden die acht Nachbarfelder und nicht wie beim Bereichswachstumsverfahren nur die vier Nachbarn in den Himmelsrichtungen betrachtet. Die Nachbarn in den vier Himmelsrichtungen bekommen hierbei die Entfernung des aktuellen Feldes + 1. Die anderen vier Nachbarn würden die Entfernung des aktuellen Feldes + $\sqrt{2}$ erhalten. Da diese Präzision aber nicht nötig ist wird hier auf die Wurzelberechnung verzichtet und die Entfernung des aktuellen Feldes + 1.414 angenommen. In Abbildung 4.23 kann ein Beispiel für diese Schätzung betrachtet werden. Die Entfernungswerte wurden auf der Darstellung

gerundet und modulo 10 gerechnet, um auf der Konsole eine verwertbare Darstellung zu erreichen. Mit etwas Überlegung kann schnell der Startpunkt der Routenplanung erkannt werden. Er befindet sich direkt unter dem linken unteren Hindernis auf dem Feld, welches mit einer 1 geschätzt wurde. Der zusätzlich eingehaltene Abstand zu den Objekten und Kartenrändern kann ebenfalls wahrgenommen werden. Letzteres resultiert aus der Tatsache, dass unbekannte Regionen potentiell Hindernisse enthalten können und daher ein Sicherheitsabstand eingeplant werden muss. Nun muss jedoch aus den berechneten Entfernungen die eigentliche Route extrahiert werden. Hierfür wird am angegebenen Zielpunkt gestartet und der folgende Algorithmus umgesetzt:

```
1   lege Zielpunkt auf Stack
2   setze Zielpunkt als aktuelles Feld
3   while (aktuelles Feld != Startpunkt) do
4     for (alle Nachbarn vom aktuellen Feld) do
5       suche kürzeste Entfernung
6     end
7     lege Nachbarn mit kürzester Entfernung auf Stack
8     Nachbar mit kürzester Entfernung wird aktuelles Feld
9   end
10  return stack
```

Listing 4.34: Routenextraktion

Hierbei werden als Entfernungen die zuvor berechneten Entfernungen vom Startpunkt verwendet. Die Interpretation könnte sein, dass Entfernungen als Höhenwerte eines Gebirges angenommen werden. Am Zielpunkt entspringt ein Fluss und der Weg den das Wasser zum Startpunkt wählt ist die kürzeste Route. Hierbei kann es auch mehrere Routen geben, was dazu führt, dass bei der Suche nach dem Nachbarn mit der kürzesten Entfernung mehrere mögliche Kandidaten gefunden werden. In diesem Fall wird beliebig ausgewählt, denn die Länge der Route wird auf jeden Fall minimal sein. Wie die kürzeste Route in dem aufgeführten Beispiel aussieht ist in Abbildung 4.24 zu sehen. Die Zahlen in dieser Abbildung stellen hier nicht die zuerst berechneten Entfernungen dar, sondern die Reihenfolge der Felder modulo 10. Durch den verwendeten Stack werden also nicht nur die Felder auf der Route ermittelt, sondern auch die Reihenfolge in der diese angefahren werden müssen.

Dieses ganze Verfahren hat sich als schnell herausgestellt. Tests auf der von uns verwendeten Hardware zeigten auch bei größeren Karten gute Leistungsresultate. Trotzdem können noch mehrere Optimierungen eingebaut werden. Zum Beispiel kann die resultierende Route auf wenige markante Punkte reduziert und mit verschiedenen Verfahren aus der Bildverarbeitung eine glattere Kurve berechnet werden, die nicht dazu führt, dass sich der RAPTOR auf der Stelle drehen muss, wie es bei einer eckigen Route der Fall ist. Hier muss sich aber noch zeigen, dass der Rechenaufwand an dieser Stelle gerechtfertigt ist. Eine weitere Optimierung kann während der Schätzung der Entfernungen erreicht werden, indem nur vielversprechende Felder ausgewertet werden, die mit einer hohen Wahrscheinlichkeit die optimale Route beschreiben. Dies ist aber wieder eine Optimierung, die gerechtfertigt

sein muss. Es kann durchaus sein, dass die Performanz der durchgeführten Berechnungen nicht weiter ins Gewicht fällt und die Arbeit an anderer Stelle sinnvoller ist.

4.10 Schmutzerkennung

Die Erkennung von Verschmutzungen im Haushalt ist ein Problem, welches mit der heutigen Technik als nicht lösbar gilt. Dies basiert zum einen auf der Anzahl unterschiedlicher Böden und der Menge der verschiedenen Verschmutzungsarten. Die Vielzahl der Kombinationen, welche hier betrachtet werden müssen ist so enorm, dass es nicht möglich ist, diesem Problem direkt zu begegnen. Es werden daher in heutigen Robotern Systeme verbaut, die das aufgenommene Material detektieren [2]. Dieses hat zum Nachteil, dass Verschmutzungen, die nicht von dem Roboter beseitigt werden können, nicht wahrgenommen werden. Dies wäre aber eine wichtige Information für den Menschen, da eine gesonderte Reinigung nötig wird. Bei der Entwicklung des RAPTOR wurde von Anfang an die Vorraussetzung für eine optische Schmutzerkennung mittels einer Webcam geschaffen, um diesem Prolem zu begegnen. Im Folgenden wird beschrieben, wie die genaue Umsetzung dieses Ansatzes aussieht und welche Probleme dabei gelöst werden mussten. Ebenso wird aufgezeigt, welche Beschränkungen die Projektgruppe nicht überwinden konnte und welche Fähigkeiten angestrebt wurden.

Als Hardware kommt eine eigens für diesen Zweck installierte Webcam (siehe Abschnitt 3.6.2) zum Einsatz, welche eine Auflösung von 640x480 Pixeln besitzt. Diese wurde an der oberen Ebene des RAPTOR angebracht, um von dort aus Bilder des Bodens anzufertigen. In der unteren Ebene wurde zu diesem Zweck eine Öffnung gebohrt durch die dies möglich wird. Diese interne Lösung hat den Vorteil, dass das Umgebungslicht weniger Einfluss auf die Kamerabilder hat. Die zusätzlich an der Kamera angebrachte LED-Beleuchtung führt dazu immer eine einheitliche Belichtung des Bodens zu erhalten. Somit wird die Störungsquelle des wechselnden Lichtes größtenteils eliminiert. Dies ist von großer Wichtigkeit, denn die Schmutzerkennung basiert auf Methoden des maschinellen Lernens und diese arbeiten umso besser, je geringer die zu kompensierenden Störungen sind. Wie genau diese Algorithmen arbeiten, wird nun näher erläutert.

Das Vorgehen beim maschinellen Lernen gliedert sich in zwei Phasen. Als erstes eine Trainingsphase in der das Wissen anhand von Testdaten gelernt wird. Zum zweiten die Klassifikationsphase in der das erworbene Wissen auf reale Daten angewendet wird.

4.10.1 Training

Das Training eines Klassifikationssystems benötigt bei komplexeren Aufgaben meist eine sehr hohe Rechenleistung. Es werden eine Unmenge an Trainingsdaten verwendet, welche die Realität in möglichst guter Art und Weise abbilden. Im Falle einer Schmutzerkennung ist dies, wie bereits weiter vorne erwähnt, eine fast unlösbare Aufgabe. Um diesem Problem zu begegnen, wurde bei der Entwicklung des RAPTOR ein Trick angewendet. Da eine Abbildung aller Böden und Verschmutzungsarten in Form von Trainingsdaten nicht

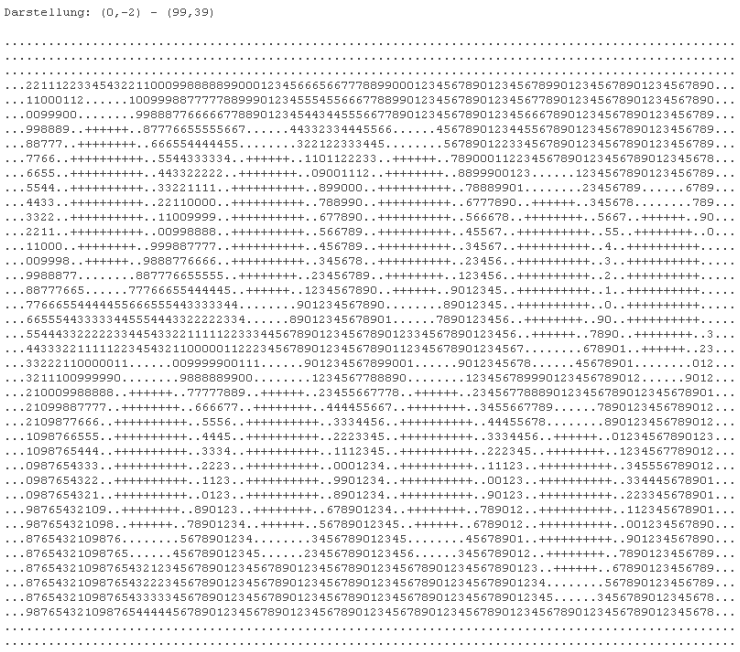


Abbildung 4.23: Entfernungsschätzung für die Routenplanung

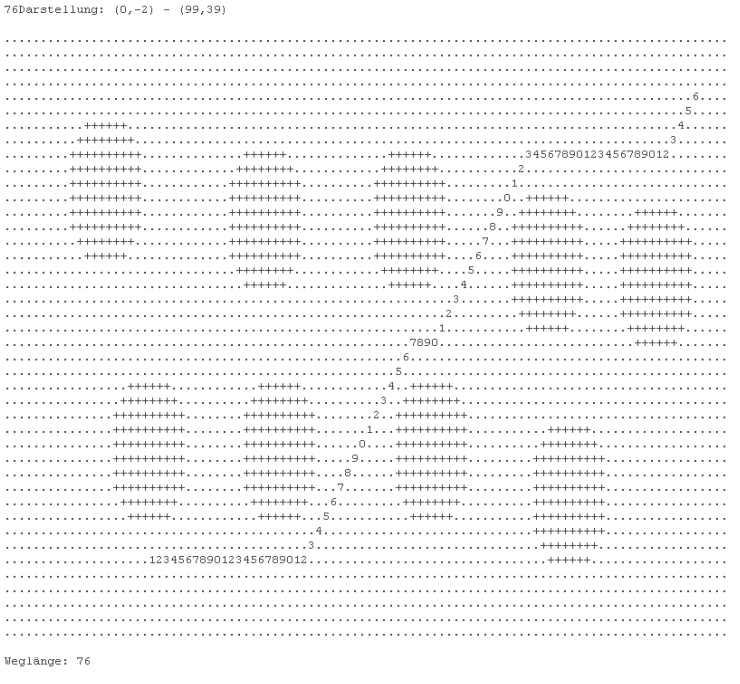


Abbildung 4.24: Die berechnete Route

als Möglichkeit zur Verfügung stand, wird bei der Schmutzerkennung diese Trainingsphase in den laufenden Betrieb des RAPTOR integriert. Dies ist möglich, weil nur der aktuell befahrene Boden analysiert wird und somit alle anderen Bodenbeläge von vornherein ausgeschlossen werden können. Dies stellt aber auch eine Anforderung an den Benutzer. Das Training muss nämlich mit einem sauberen Boden absolviert werden, denn es wird ein internes Modell dieses Bodens erstellt, welches später mit dem dann aktuellen Zustand verglichen wird. Zu Beginn des Trainings wird ein Bild des Bodens erzeugt. Auf Abbildung 4.25 ist zu erkennen, wie ein solches aussieht. Im nächsten Schritt wird von jedem Farbkanal ein Histogramm erzeugt [14]. Dieses Histogramm wird nun in einer Datei gespeichert, bis das Training beendet werden soll. Bis dies jedoch geschieht werden ständig weitere Bilder erzeugt und in ihre Histogramme überführt. Dies dient der Merkmalsreduktion, denn die Betrachtung von 640x480 Bildpunkten mit jeweils drei Farbwerten würde die Fähigkeiten der verwendeten Hardware bei weitem übersteigen. In Form eines Histogramms besteht ein Bild nur noch aus 3x256 Merkmalen.

Wenn das Ende der Trainingsphase erreicht ist, was überlicherweise nach der Aufnahme von 100 Bildern der Fall ist, wird ein abschließender Lernprozess gestartet. Dieser überführt die gesammelten Histogramme (siehe Abbildung 4.26) in eine Normalverteilungsfunktion [21]. Diese besteht zum einen aus dem Mittelwert aller Histogramme und der Standardabweichung. Letztere ist ein Maß dafür, wie chaotisch der Boden ist. Je höher die Standardabweichung ausfällt, desto ungleichmäßiger ist die Farbgebung des Bodens. Diese Umrechnung hat das Bodenmodell (siehe Abbildung 4.27) zum Ergebnis. Dieses besteht aus 3x256x2 Daten, womit klar sein dürfte, dass das Bodenmodell klein genug ist, um eine performante Klassifikation zu erreichen. Dennoch ist der Informationsgehalt so hoch, dass Verschmutzungen auf dem Boden zu einem gewissen Grad erkannt werden können. Dieser Grad ist das Qualitätskriterium, an dem sich die Schmutzerkennung des RAPTOR messen lassen muss.

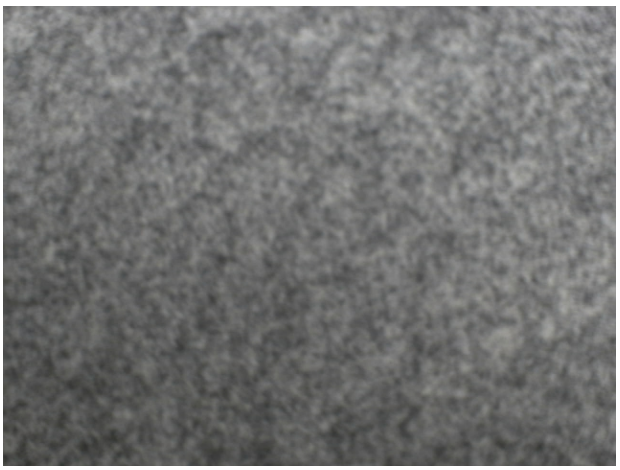


Abbildung 4.25: Bild der Bodenkamera

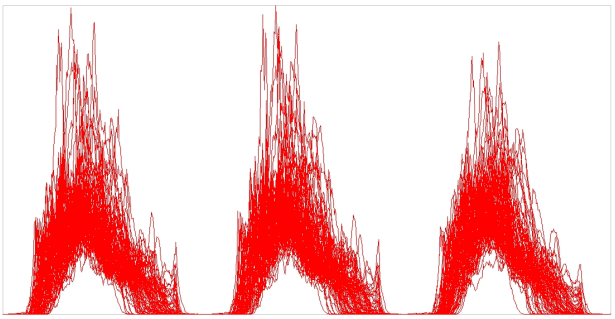


Abbildung 4.26: Histogramme von 100 Bildern der Bodenkamera

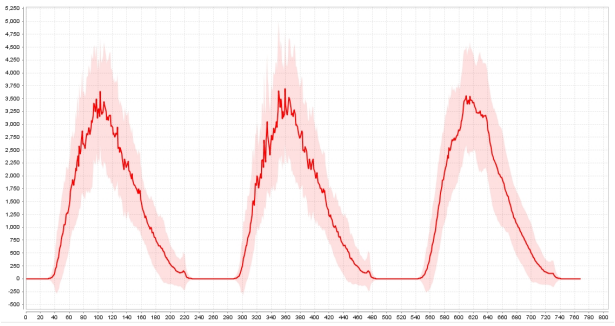


Abbildung 4.27: Gelernte Normalverteilung der Bodenbild-Histogramme

4.10.2 Klassifikation

Die Klassifikationsphase beginnt zunächst wie die Trainingsphase. Es wird ein Bild des Bodens erzeugt und anschließend in ein Histogramm überführt. Wie bereits erwähnt dient dies der Merkmalsreduktion und somit der Performanzoptimierung. Anschließend wird die eigentliche Klassifikation des Bodenzustandes vorgenommen. Hierzu wird das Histogramm des Bodens mit dem gelernten Bodenmodell verglichen. Zu diesem Zweck wird das Histogramm in die Normalverteilungsfunktion des Bodenmodells eingesetzt und wir erhalten einen Wert der Abweichung. Dies wird für jeden einzelnen Histogrammwert vorgenommen und die Ergebnisse werden akkumuliert. Die Betrachtung der Ungleichmäßigkeit des Bodens führt hier dazu, dass in Bereichen des Histogramms, wo sich diese Unordnung auswirkt der Fehler bei einer Abweichung des aktuellen Histogramms von dem gelernten Mittelwerthistogramm kleiner ist, als wenn in diesem Bereich kaum Unordnung zu erkennen gewesen ist. Dieses Vorgehen bewirkt, dass natürliche Schwankungen kompensiert werden und weniger stark in den abschließenden Fehlerwert eingehen, als Abweichungen, welche während des Trainings nicht beobachtet wurden.

Als finaler Schritt in der Klassifikation dient die Einordnung des berechneten Fehlers in eine für den Menschen verständliche Skala. Dieses Problem konnte die Projektgruppe leider aus Zeitgründen nicht mit der Sorgfalt angehen, welche wünschenswert gewesen wäre. Es wurde daher darauf verzichtet eine an den jeweiligen Boden angepasste Einordnung der Ergebnisse vorzunehmen. Statt dessen wurde manuell eine Skala erzeugt, die auf Er-

fahrungen während der Funktionstests beruht. Die weitere Entwicklung in diesem Bereich sollte zunächst zum Ziel haben diese Erstellung der Skala zu automatisieren und somit zuverlässiger zu machen. Es könnte damit ein großes Maß an Verbesserung erzielt werden, weil gerade die Einordnung in eine Skala von der Unordnung der Bodenstruktur abhängt. Auch wenn diese Unordnung bereits in der Normalverteilung berücksichtigt wird, führt ein stark chaotischer Boden dennoch dazu, dass natürliche Schwankungen einen Fehlerwert produzieren können, der als Verschmutzung klassifiziert werden kann. Insgesamt läßt sich aber sagen, dass gute Ergebnisse mit dem zur Zeit exisiterenden Ansatz erzielt werden konnten. Die Erkennung beschränkt sich zwar auf die Erkennung von sogenanntem Partydreck, wie beispielsweise Chips, Gummibärchen und Smarties, aber dieses wird ohne die Änderung des Bodenzustandes erreicht, was eine große Verbesserung gegenüber bestehender Systeme darstellt.

4.11 Hauptprogramm

Das Hauptprogramm dient dazu, die restliche Software des Roboters zu starten und zu verwalten. Es wird umgehend nach dem Start von RAPTOR und dem Booten des Betriebssystems ausgeführt, und regelt alle weiteren Aspekte der Verwaltung restlicher Teilsoftware für den Betrieb des Roboters. Die Aufgabe des Hauptprogramms besteht darin, alle für den Betrieb des Roboters notwendige Softwareteile zu initialisieren und auszuführen. Um die Sensordaten von der Umgebung des Roboters aktuell zu halten, muss die Sensordatenfusion immer laufen, damit die Sensoren in genügend kurzen Abständen abgefragt werden können. Zu diesem Zweck wurde die Datenfusion im Hauptprogramm in einen eigenen Thread ausgelagert, welcher die notwendige Parallelisierung der Berechnungen gewährleistet. Ein anderer Thread nimmt die Robotersteuerung mit den Funktionen zum Ausführen der einzelnen Fahrmodi auf. Somit besteht die Funktionalität des Hauptprogramms darin, alle Objekte der gebrauchten Klassen zu erstellen, zu initialisieren und zu starten. Des Weiteren wird im Hauptprogramm die Threadverwaltung durchgeführt.

Im Folgenden werden der Aufbau und die Aufgabe der einzelnen Threads ausführlicher beschrieben:

```
1 void *starteDatenfusion(void *arg)
```

Listing 4.35: Datenfusion-Thread

Dieser Thread beinhaltet eine endlose while-Schleife, in der alle 200 Millisekunden die „startThread“-Methode der Datenfusion ausgeführt wird. Diese Methode liest dann alle Sensordaten über die API ab.

```
1 void *starteRobosteuerung(void *arg)
```

Listing 4.36: Robotersteuerung-Thread

Dieser Thread ruft die „initialize“-Methode in der Robosteuerung auf, welche die Ausführung der einzelnen Fahrmodi übernimmt (siehe Abschnitt 4.9).

Die Reihenfolge, in der die Threads ausgeführt werden, wird durch die Round-Robin Scheduling-Strategie [35] festgelegt. Bei dieser Strategie wird die Zeit in sehr kleine „Scheiben“ aufgeteilt, welche dann den einzelnen Threads nacheinander in bestimmter zuvor festgelegter Reihenfolge zugewiesen werden. Das ermöglicht eine faire Verteilung der Systemressourcen, so dass kein Thread unendlich lang auf die Ausführung warten muss (das sogenannte „Aushungern“ kommt somit nicht vor). Zusammen mit der Robustheit des Verfahrens und der Einfachheit der Implementierung waren das die Hauptgründe für die Entscheidung, diese Strategie zu wählen.
Aus Performancegründen wurden mehrere Methoden, die ursprünglich als Threads für das Hauptprogramm konzipiert wurden, in eine seperate Klasse ausgelagert und in normale Prozesse umgewandelt. Hierzu gehören vor allem Threads, die eine im Verhältnis zur benötigten Rechenleistung untergeordnete Funktionalität erfüllen. Primär sind es die Prozesse zur Erzeugung von Bildern der Front- und Bodenkamera.

```
1  void starteBilderzeugung_boden();
2  void starteBilderzeugung_front();
```

Listing 4.37: Initialisierung

Zusätzlich wurden auch die Prozesse für die Ausgabe der aktuellen Sensordaten als Textdatei und für die Überprüfung des aktuellen Befehlsmodus hierher verlagert.

```
1  void starteSensordatenAusgabe();
2  void starteGetModusName();
```

Listing 4.38: Initialisierung

Somit existieren im Hauptprogramm nur noch zwei Threads (für die Odometrie/Datenfusion und der Robotersteuerung), welche nicht ausgelagert werden können, da diese beiden Threads häufiger und parallelisiert ausgeführt werden müssen.

Im Prozesszustand kann seitens der Robotersteuerung und der Odometrie weitaus besser kontrolliert werden, wann die Methoden der Threadauslagerung aufgerufen werden. Alle Methoden wurden in einer einzigen Funktion zusammengefasst:

```
1  void execute();
```

Listing 4.39: Initialisierung

Die werden nun weitaus seltener ausgeführt als die verbleibenden zwei Threads im Hauptprogramm. Zum Beispiel wird der aktuelle Betriebsmodus des Roboters nur noch alle 5 Sekunden überprüft, während die Bilder beider Kameras sogar nur noch alle 10 Sekunden aktualisiert werden.

5 Ausblick

Das zweite, dritte und vierte Kapitel beschreibt die bisherige Arbeit der Projektgruppe RAPTOR. Nach Abschluss des einjährigen Projektes hält dieser Abschnitt fest, was für Verbesserungs- und Erweiterungsmöglichkeiten sich ergeben.

Eine Erweiterungsmöglichkeit des Prototyps wäre es, eine Saugvorrichtung zu implementieren. Dabei muss widerrum darauf geachtet werden, ob die Akkuleistung ausreicht oder gegebenenfalls eine bessere Alternative gefunden werden muss.

Aufrgund der festen Position des Maussensors und der Abgrundsensoren (3mm über dem Boden), ist die Gefahr sehr groß, dass diese bei der Fahrt durch Gegenstände beschädigt werden können. Ungewollt besteht die Gefahr, das die Sensoren bei zu hohem Teppich über den Boden schleifen. Diese Sensoren waren zwar günstig, jedoch sind sie im praktischen Gebrauch eines Staubsaugerroboters ungeeignet. Beispielsweise könnte der Maussensor durch eine bessere Linse mit mehr als 3 mm Abstand zum Boden laufen.

Des Weiteren könnte darüber nachgedacht werden, ein weiteres Stützrad am vorderen Teil des Roboters zu montieren. Ohne Akku verlagert sich der Schwerpunkt nach vorne. Ein Stützrad könnte hier helfen, dass dabei keine Sensoren beschädigt werden. Zugleich würde es das Problem lösen, dass sich der RAPTOR bei einem starken Stopp nach vorne "verneigt".

Die Praxis lehrte die Projektgruppe, das die verwendeten Reifen für viele Untergründe zu dünn sind. Auch hier ist der RAPTOR zu sehr vom Boden abhängig.

Wie der Schaltplan der Hauptplatine A.6 zeigt, ist eine Schnittstelle für ein LCD (Liquid Crystal Display) freigehalten worden. Ein LCD am RAPTOR zu montieren, würde den Staubsaugerroboter qualitativ aufwerten. Der Benutzer könnte direkt am Roboter Stati ablesen und müsste nicht das Webinterface dazu benutzen. Zusätzliche Knöpfe als Bedienelemente würden es dem Benutzer weiter vereinfachen, mit dem Roboter zu interagieren.

Die Mikrocontrollerprogramme könnten an einigen Stellen verbessert werden. Beispielsweise sind einige Warteschleifen eher unelegant und einige Funktionen ließen sich durch bessere Ausnutzung von Interrupts optimieren. Auch werden die Stromsparmodi nicht ausgenutzt. Da die Controller nicht die größten Stromverbraucher am RAPTOR sind, stellte das bisher kein Problem dar; dennoch besteht hier Verbesserungspotenzial.

Wie an mehreren Stellen im Endbericht erwähnt wurde, ist der Gumstix irreparabel beschädigt worden. Das Ersetzen durch ein Laptop anstelle eines eingebetteten Systems war nur eine Notfalllösung. Mit Hilfe einer zuverlässigen Schutzschaltung und Stromversorgung sollte der Roboter auch wieder über den Gumstix kontrolliert werden können. Abgesehen von der integrierten I^2C-Funktionalität stellt der Gumstix weitaus mehr Speicher und Rechenleistung zur Verfügung als das verwendete Notebook. Auch unterstützt

das Gumstix-Erweiterungsboard W-LAN-Verbindungen, so dass der zusätzliche USB-Stick entfallen kann.

In der Sensordatenfusion gibt es ebenfalls Verbesserungsmöglichkeiten. Es ist eine Überlegung wert, statt des Kalman-Filters einen Partikel-Filter für die Selbstlokalisation einzusetzen. Der Partikelfilter könnte möglicherweise bessere Werte liefern, jedoch müsste dies in der Praxis mit dem Kalman-Filter verglichen werden, um genauere Aussagen zu treffen. Eine weitere Überlegung ist, dass zur Schätzung der Position die Regressionsanalyse eingesetzt wird. Diese kann so verwendet werden, dass von fünf (oder mehr) Messpunkten, in dem Fall die Koordinaten des Maussensors, eine Funktion bestimmt wird, welche die Werte so genau wie möglich abbildet. Hiermit wäre es möglich eine Prognose für den nächsten Zeitschritt zu stellen. Dieser vorhergesagte Wert könnte anschließend mit dem Kalman-Filter kombiniert werden, um die Positionsschätzung zu verbessern.

Die Gitterkarte benötigt eine effizientere Datenhaltung. Es werden noch Vektoren verwendet; bei einer Neuimplementierung könnten diese komplett weggelassen werden. Mit einer Baumstruktur könnten die Daten komprimiert und komplett im Speicher gehalten werden, so dass der bisherige Overhead der eigens entwickelten Klasse entfallen würde.

Die Kartenerstellung könnte erweitert werden, indem implementiert wird, dass Räume wiedererkannt werden. Das bedeutet, dass der aktuelle Raum während der Kartenerstellung mit gespeicherten Räumen abgeglichen wird, um diese Karten dann zu laden und die Kartenerkennung vorzeitig zu stoppen. Auch kann mit etwas Aufwand die Präzision der Karte erhöht werden. Dies verlangt zugleich eine genauere Sensordatenfusion.

Des Weiteren könnte anhand der internen Karte eine Positionsbestimmung erfolgen. Diese Daten finden zur Zeit keine Beachtung bei der Positionsermittlung. Anhand der erkannten Abstände zu Objekten bzw. Wänden durch Ultraschall und der Blickrichtung des Roboters durch seinen Kompasssensor wäre es möglich, seine Position im Raum zu bestimmen. Dies könnte dann mit der bisherigen Positionsbestimmung abgeglichen werden, um eine genauere Position zu ermitteln.

Die Robotersteuerung könnte eine kontrollierte Geradeausfahrt implementiert haben, die dafür sorgt, dass der RAPTOR eine gerade Linie fährt. Alternativ dazu könnte die zu fahrende Route in eine Bezier-Kurve überführt werden; die daraus resultierende Bewegung wäre "runder" und würde erheblich weicher wirken als der bisherige Stopp des Roboters bei jeder Richtungsänderung. Zusätzlich könnte eine angestrebte Verbesserung sein, dass bei einer Blockade der Route bzw. der Bahn durch ein Hindernis automatisch die Routenplanung zum Umfahren gestartet wird. Dies ist zur Zeit nicht integriert und somit führen unerwartete Hindernisse zu Problemen.

Das Problem der Schmutzerkennung gilt mit der heutigen Technik der Fachwelt als unlösbar. Damit war es der Projektgruppe unmöglich, optimale Ergebnisse zu erzielen. Beispielhaft kann hier aufgeführt werden, dass Verbesserungen vorgenommen werden können, durch eine angemessenere Ausleuchtung des Bodens, durch blockiertes Umgebungslicht, schnellere Aufnahme der Bilder zur Verringerung der Bewegungsunschärfe, schnellerer Prozessor um mehr Merkmale betrachten zu können und eine Kamera mit Autofokus.

Der Betrieb des Hauptprogramms mit Threads war mit der verwendeten Hardware nicht möglich, stattdessen wurden Threadauslagerungen in normale Prozesse implementiert. Da-

durch ging ein beachtlicher Teil an Genauigkeit und Korrektheit verloren.

Das Webinterface könnte deutlich interaktiver werden. Beispielsweise könnte mit Ajax (Asynchronous JavaScript and XML) realisiert werden, dass das Webinterface nicht per Refresh in regelmäßigen Zeitabständen neugeladen werden muss, sondern die Änderungen der Sensorwerte ohne Verzögerung sofort dargestellt werden können. Außerdem wäre es dann möglich, die Fernbedienung in das Webinterface zu integrieren.

Durch einen besseren Algorithmus könnte die Bahnplanung auch die Form des Raumes besser in die Bahnberechnung einbeziehen. Der zur Zeit verwendete Algorithmus deckt zwar den gesamten Raum ab, aber im schlimmsten Fall muss der Roboter zwischen einzelnen Teilbereichen im Raum oft hin und her fahren.

Die Projektgruppe besuchte im September einen Wettbewerb, der in Abschnitt 1.2.2 beschrieben wurde. Eigens hierfür wurde der sogenannte Follow-Modus entwickelt. Diese zusätzliche Funktion dient dazu, dem Menschen zu folgen und dabei beispielsweise ein Tablett mit Getränken zu fahren. Hier wäre eine teurere und bessere Frontkamera sehr von Nutzen. Das würde die Kamerabilder verbessern und dieser positive Effekt würde sich direkt auf den Follow-Modus auswirken. Des Weiteren könnte ein Erwartungsbild berechnet werden, so dass Veränderungen im Bild nicht mehr betrachtet werden, die durch die Eigenbewegung des RAPTOR entstehen. So würden die Bewegungen von anderen Objekten viel besser zu erkennen sein. Bei dem Wettbewerb erhielt dieser Modus ausschließlich positive Resonanz, wie auch seine innovativen Lösungen mit der Kartenerstellung und Schmutzerkennung.

A Hardwarekomponenten

A.1 Skizzen der Karosserie

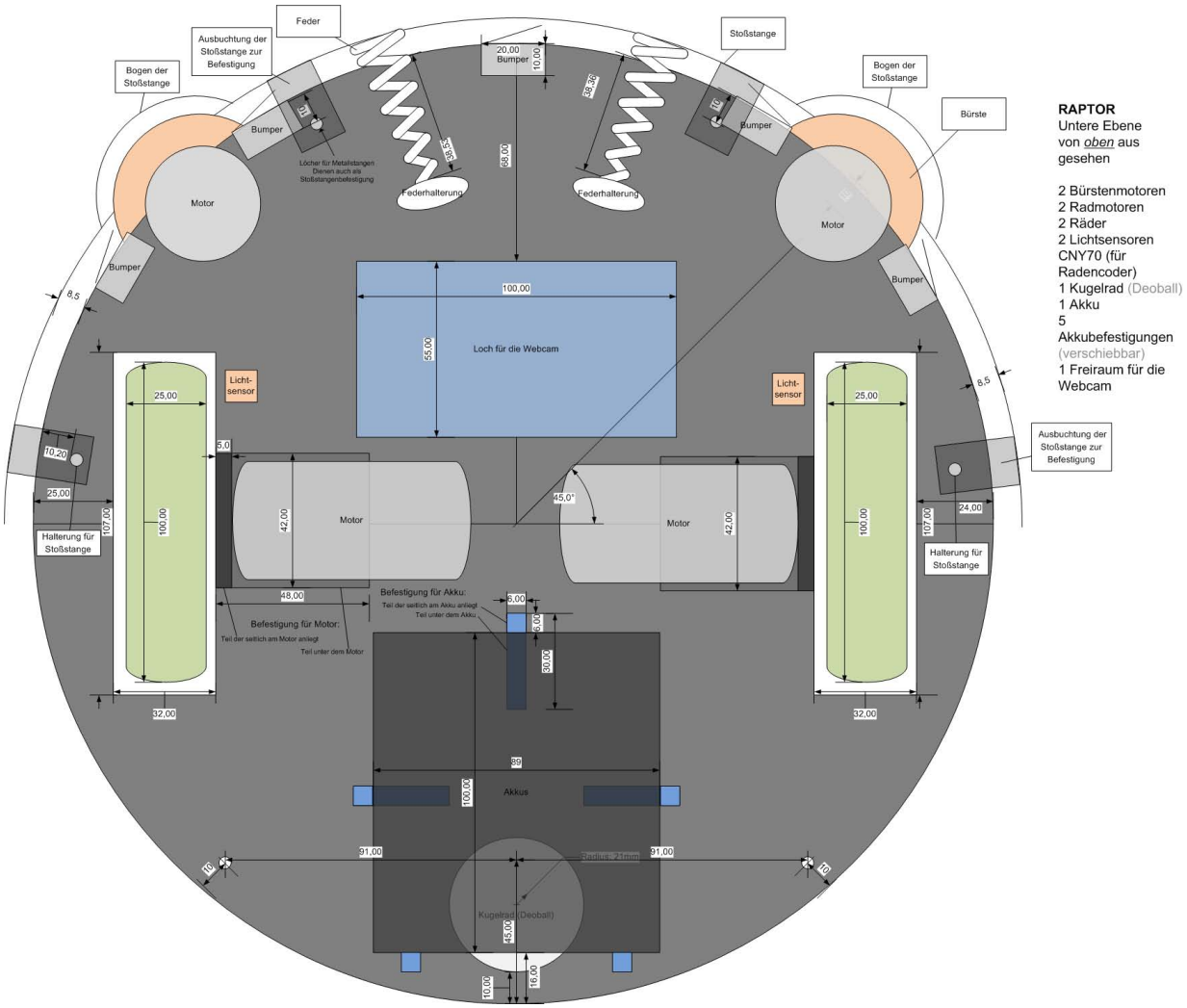


Abbildung A.1: Untere Ebene von oben aus gesehen

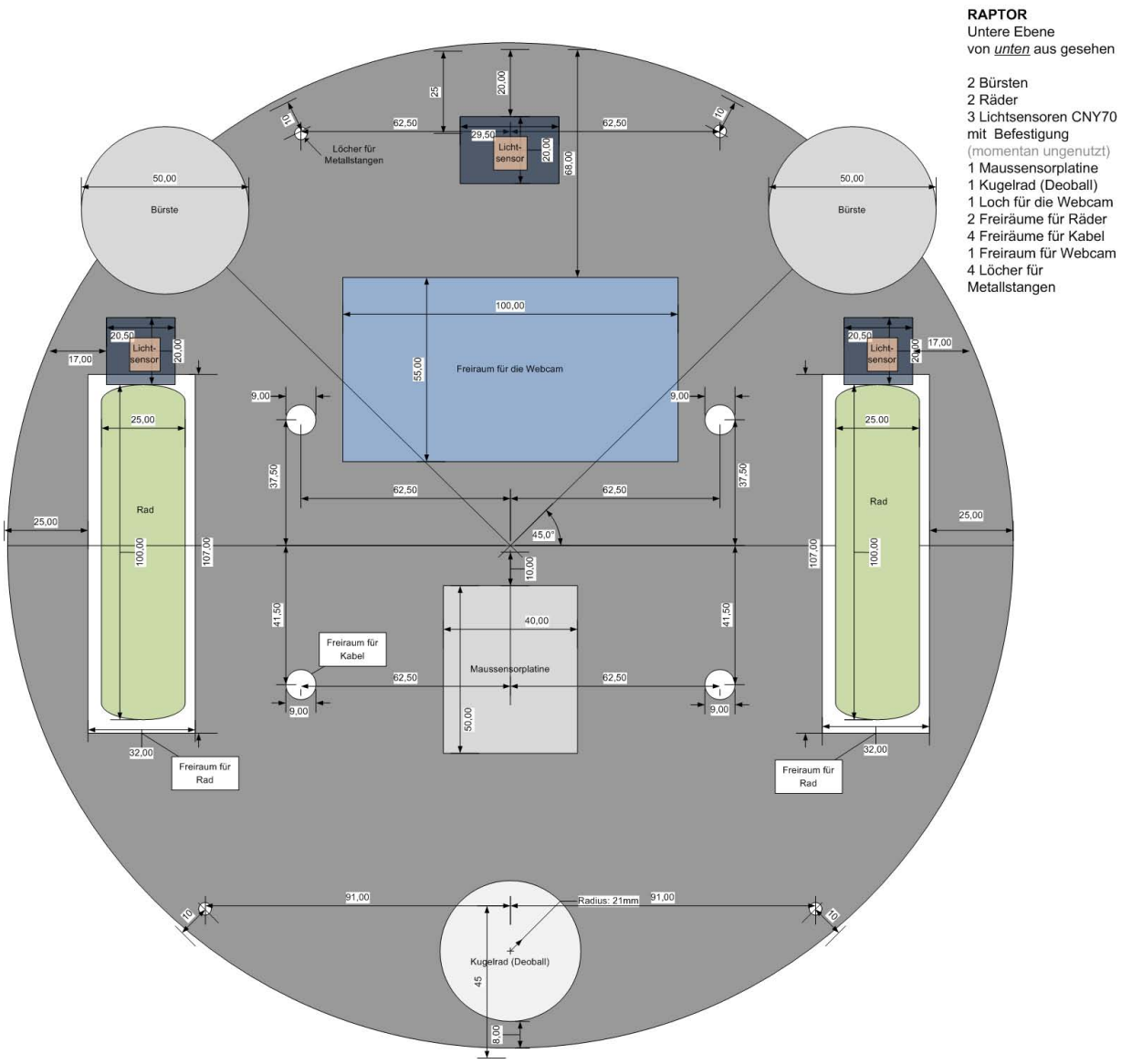


Abbildung A.2: Untere Ebene von unten aus gesehen

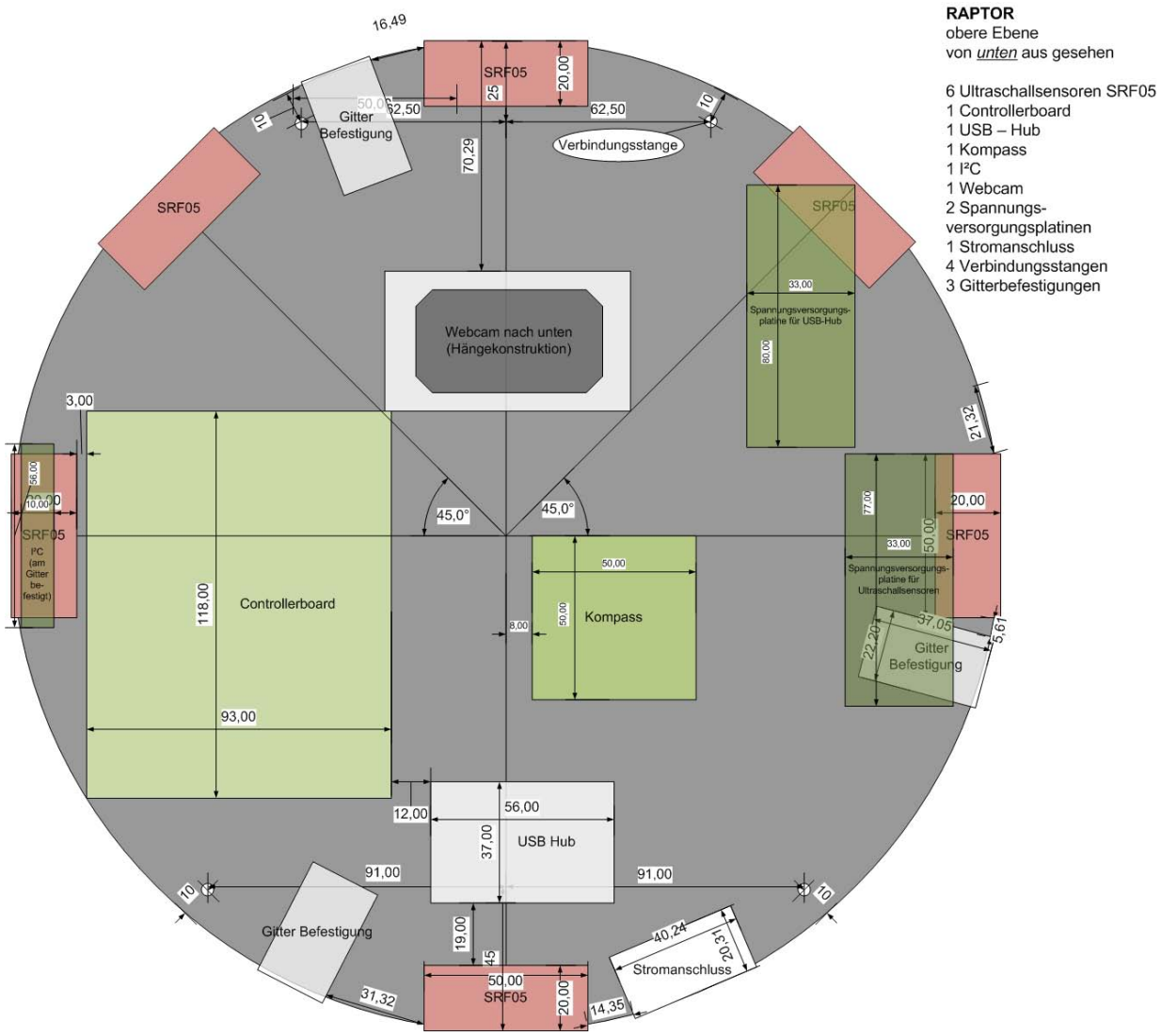


Abbildung A.3: Obere Ebene von unten aus gesehen

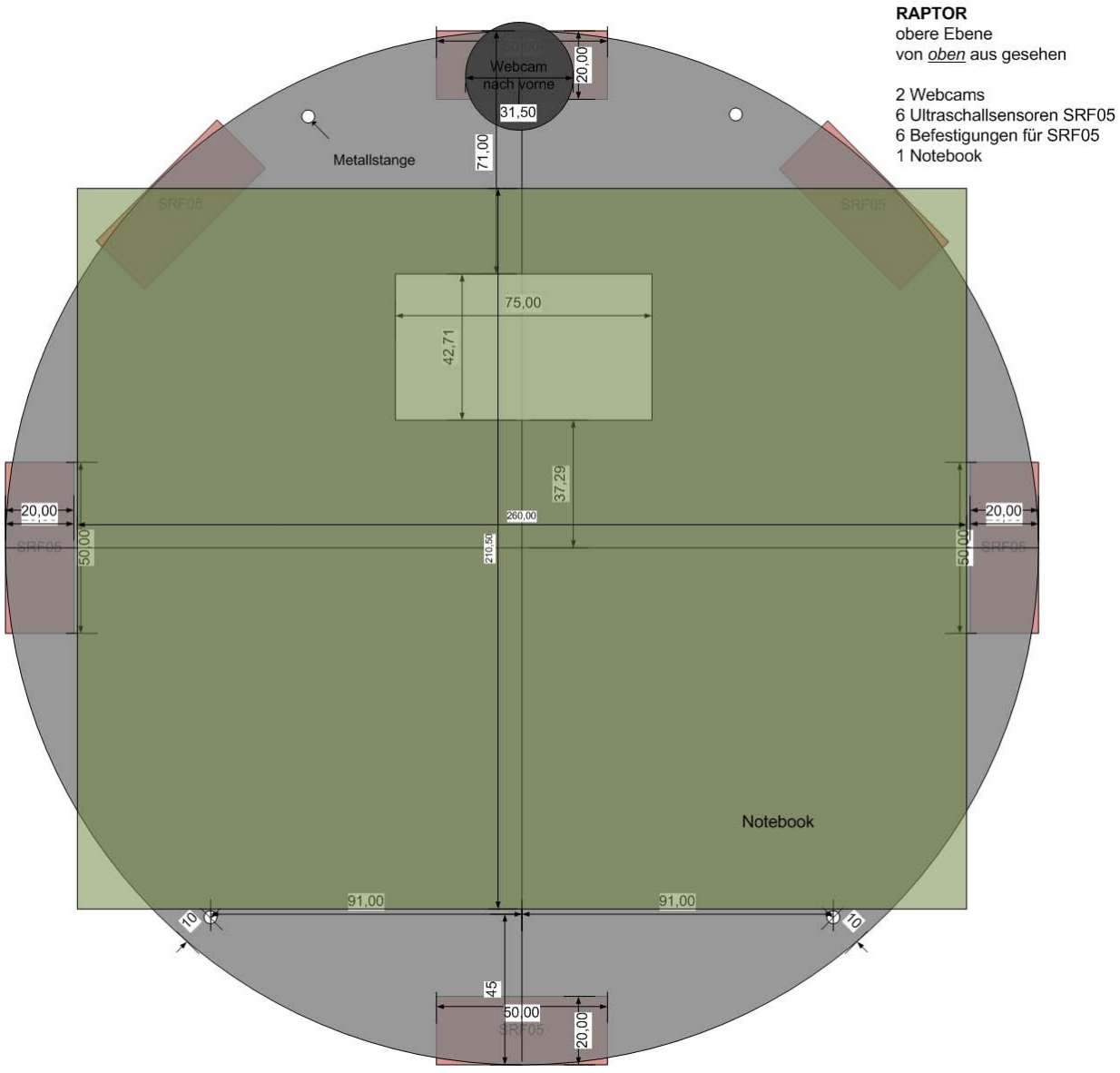


Abbildung A.4: Obere Ebene von oben aus gesehen

A.2 Schaltpläne

A.3 Platinenlayouts

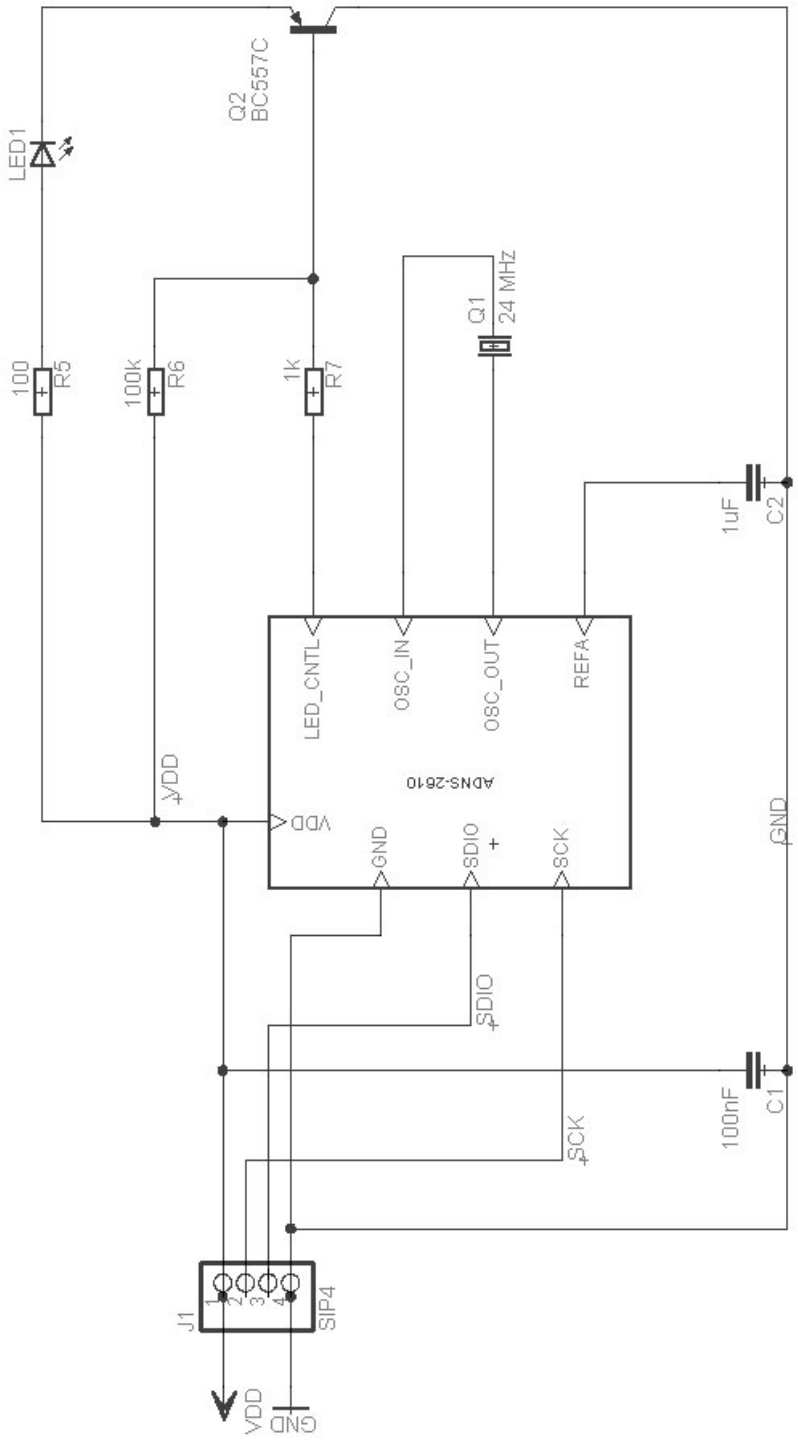


Abbildung A.5: Schaltung der Maussensorplatine

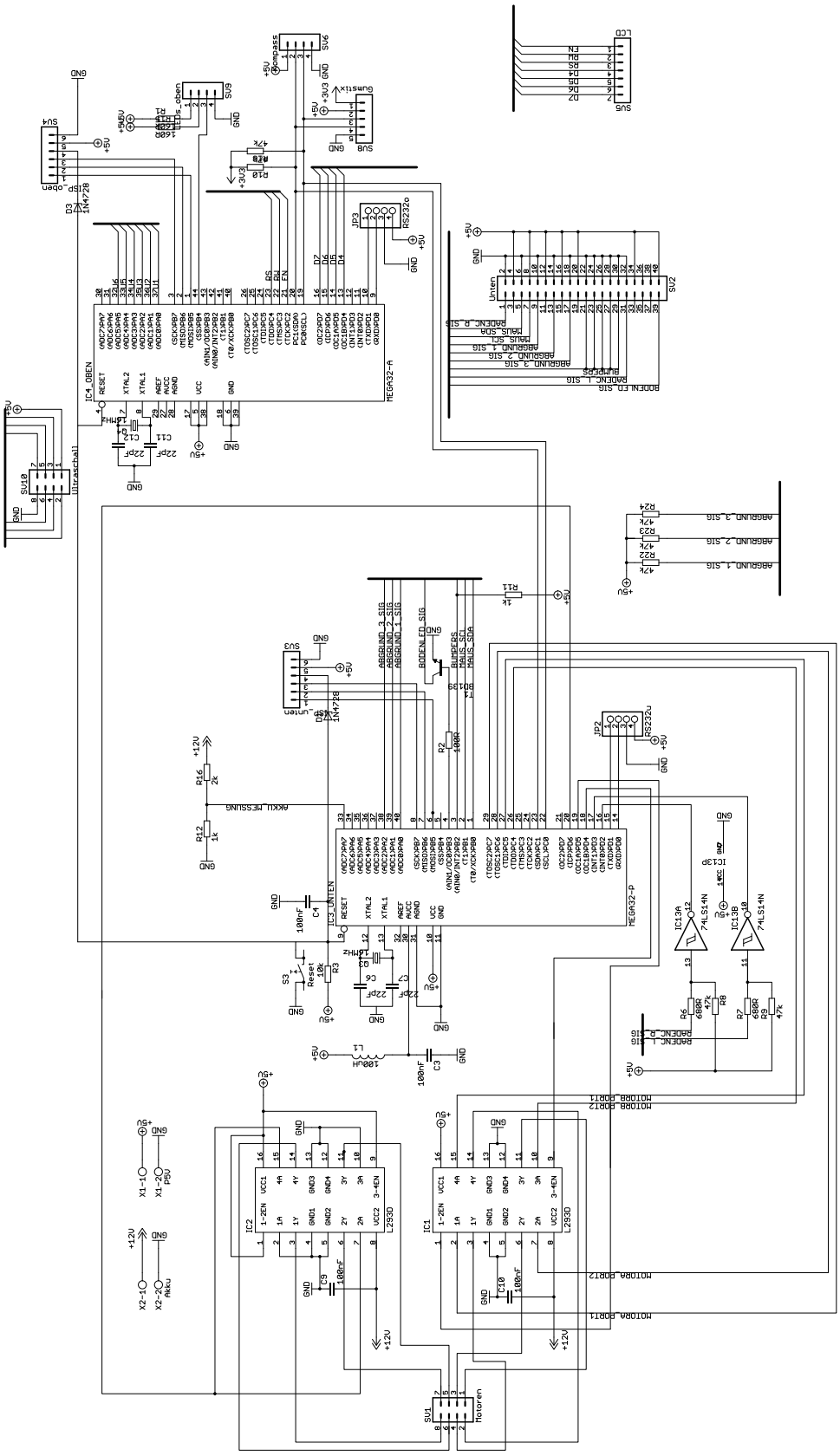


Abbildung A.6: Schaltplan der Hauptplatine

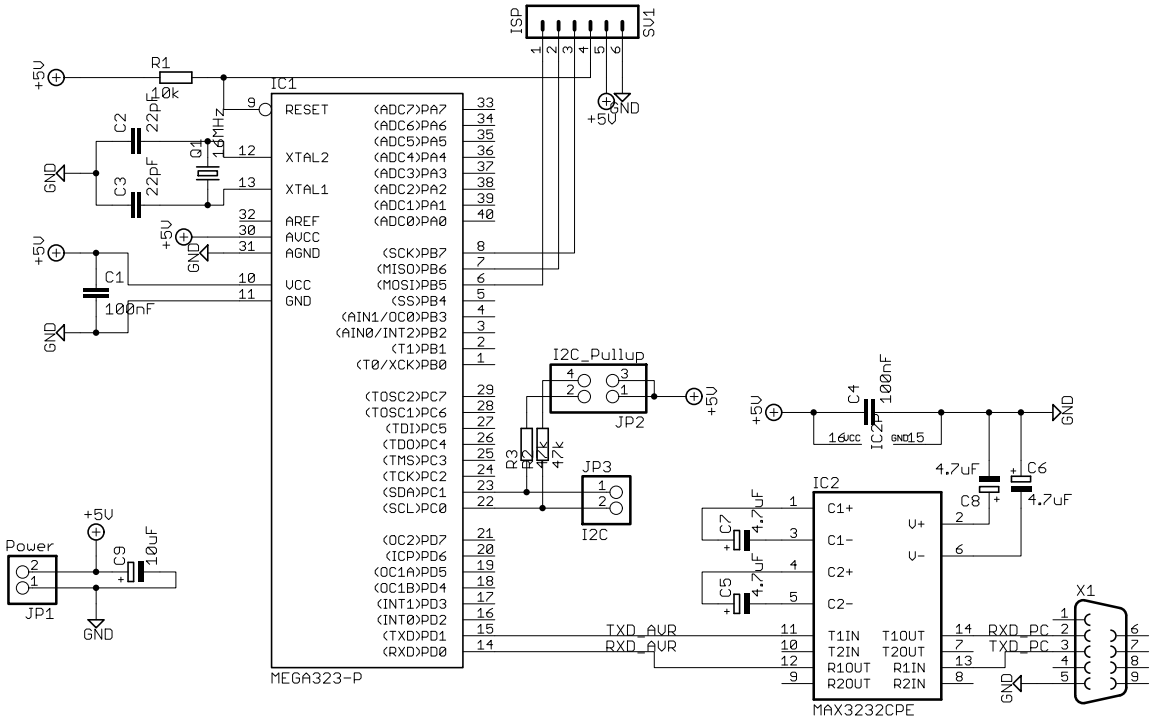


Abbildung A.7: Schaltplan des RS232-I2C-Adapters

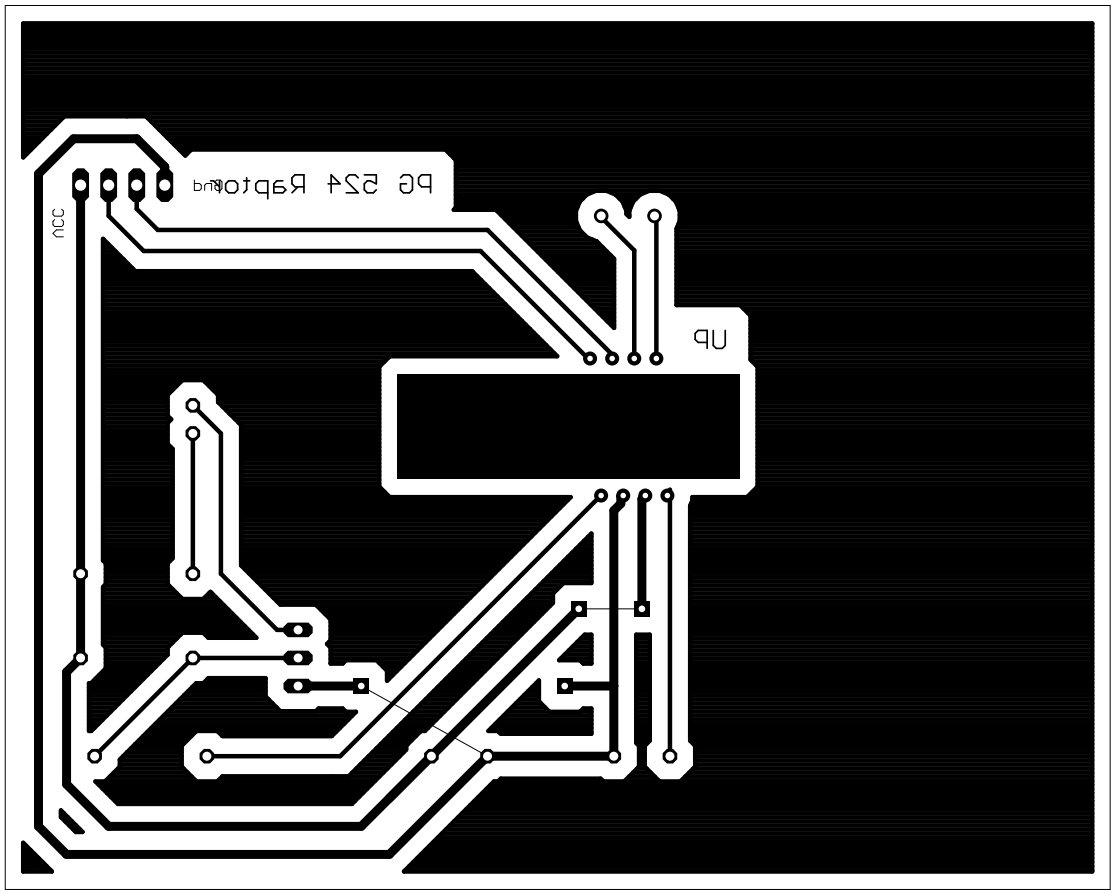


Abbildung A.8: Layout der Maussensorplatine

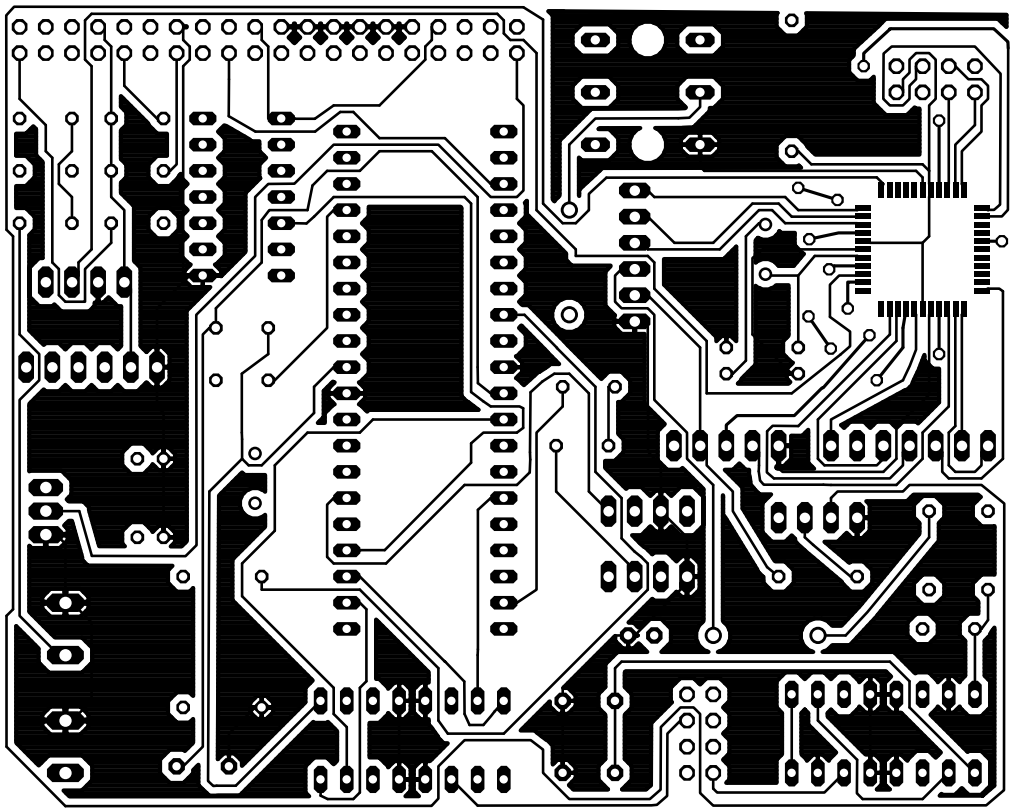


Abbildung A.9: Layout der Oberseite der Hauptplatine

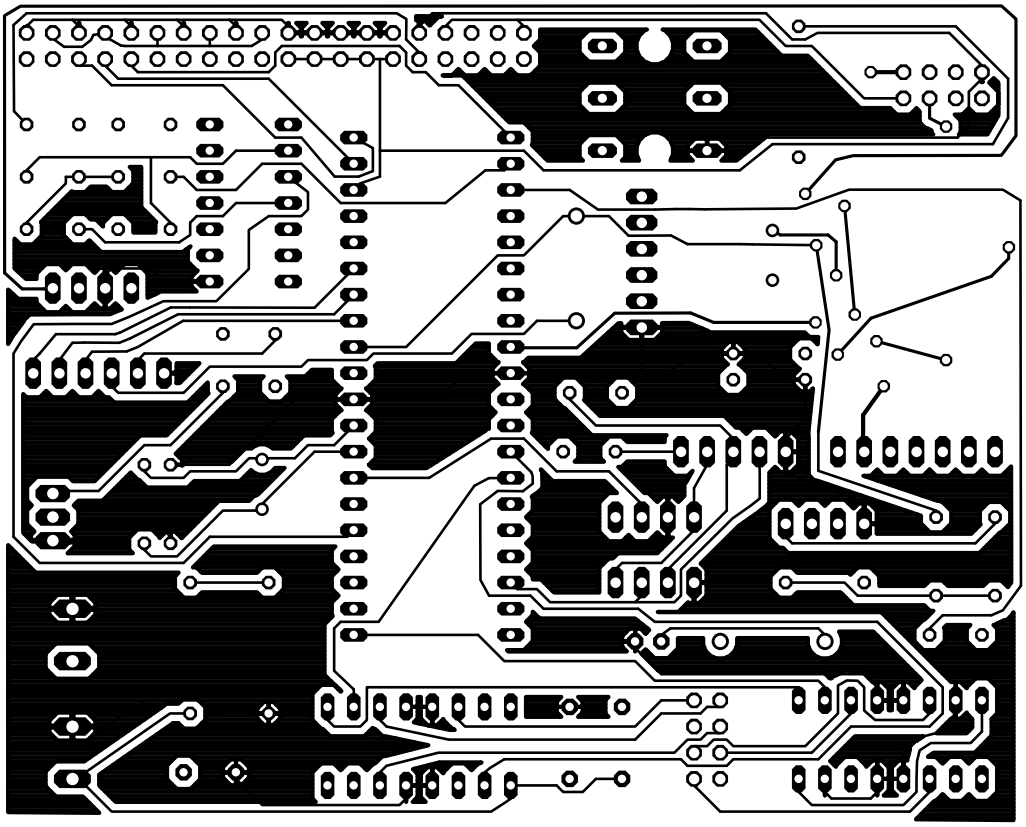


Abbildung A.10: Layout der Unterseite der Hauptplatine

B Mikrocontrollerprogrammierung

```
1  #include <avr/io.h>
2  #include "uart.h"
3
4  #ifndef BAUDEXT
5    #define BAUDEXT 9600L
6  #endif
7  #define BAUDINT ((F_CPU/16)/BAUDEXT - 1)
8
9
10 uint8_t uart_putc(unsigned char c){
11   while (!(UCSRA & (1<<UDRE))){}
12   /* warten bis Senden moeglich */
13   UDR = c; /* sende Zeichen */
14   return 0;
15 }
16
17 uint8_t uart_puts(char *s){
18   while (*s){
19     uart_putc(*s);
20     s++;
21   }
22   return 0;
23 }
24
25 uint8_t uart_init(uint8_t modus){
26   if (modus & (1<<0)){ // Bit 0, LSB: Senden
27     UCSRB |= (1<<TXEN);
28   }
29   if (modus & (1<<1)){ // Bit 1: empfangen
30     UCSRB |= (1<<RXEN);
31   }
32   UBRRH = BAUDINT >> 8;
33   UBRRL = BAUDINT & 0xff;
34   return 0;
35 }
```

Listing B.1: Ausschnitt aus uart.c

```
1  uint16_t uschall_read(uint8_t AuswahlRichtung){
2    uint16_t zeiter;
3
4    uint8_t uschallPin;
5    switch (AuswahlRichtung){
6      case USCHALL_VORNE:
7        uschallPin=USCHALL_PIN_VORNE;
8        break;
9      case USCHALL_RECHTS:
10       uschallPin=USCHALL_PIN_RECHTS;
11       break;
12     case USCHALL_HINTEN:
13       uschallPin=USCHALL_PIN_HINTEN;
14       break;
15     case USCHALL_LINKS:
16       uschallPin=USCHALL_PIN_LINKS;
17       break;
18     #ifdef USE_USCHALL_6DIR
19       case USCHALL_VORNELINKS:
20         uschallPin=USCHALL_PIN_VORNELINKS;
21   break;
22       case USCHALL_VORNERECHTS:
23         uschallPin=USCHALL_PIN_VORNERECHTS;
24   break;
25     #endif
26     default: return -1; break;
27   }
28
29   USCHALL_PORT_DDR |= (1<<uschallPin);
30   USCHALL_PORT_OUT |= (1<<uschallPin);
31   _delay_us(USCHALL_IMPULSBREITE);
32   USCHALL_PORT_OUT &=~ (1<<uschallPin);
33   USCHALL_PORT_DDR &=~ (1<<uschallPin);
34
35   // Timer stoppen und 0 setzen
36   TCNT0=0;
37   zeiter=0;
38   TCCR0 |= (2<<CS00);
39
40   // Warte, bis der Echo-Impuls kommt
```

```
41    while ((USCHALL_PORT_IN & (1<<uschallPin))==0){
42      if(TIFR&(1<<TOV0)){
43        zeiter++;
44        TIFR=(1<<TOV0);
45        if (zeiter>250){ // kommt wohl kein Impuls mehr
46          TCCR0 &=~ (7<<CS00);
47          return 2*58*0x00; // Fehlerwert1
48        }
49      }
50    }
51    // Timer stoppen
52    TCCR0 &=~ (7<<CS00);
53    // Timer neu starten!
54    TCNT0=0;
55    zeiter=0;
56    TCCR0 |= (2<<CS00);
57
58    // Warte, bis Echo-Impuls geht
59    while (USCHALL_PORT_IN & (1<<uschallPin)){
60      if(TIFR&(1<<TOV0)){
61        zeiter++;
62        TIFR=(1<<TOV0);
63        if (zeiter>250){ // kein Ende in Sicht
64          TCCR0 &=~ (7<<CS00);
65          return 2*58*0x00; // Fehlerwert2
66        }
67      }
68    }
69    // Timer Stoppen
70    TCCR0 &=~ (7<<CS00);
71    zeiter=(zeiter<<8)+TCNT0;
72
73    // Halte den Pin zwischen den Messungen auf LO
74    USCHALL_PORT_DDR |= (1<<uschallPin);
75    USCHALL_PORT_OUT &=~(1<<uschallPin);
76
77    return zeiter;
78  }
79  uint16_t zeiter2cm(uint16_t pulsbreite){
80    /*
81     * pulsbreite=1 entspricht 500nS
82     * pulsbreite=2 entspricht 1uS
83     *
```

```
84     * 58uS entsprechen 1cm
85     */
86   return pulsbreite/2/58;
87 }
```

Listing B.2: Ausschnitt aus uschall.c

```
1  uint8_t analog_init(){
2    /*
3     * ADMUX:
4     * REFS: AVCC 01
5     * ADLAR: leftadjust 1
6     * MUX40: PA0 = ADC0:00000
7     */
8    ADMUX |= (1<<ADLAR) | (1<<REFS0);
9    // (0x00); // 0x00=0b00000000;
10   /*
11    * ADCSRA: Bei 16MHZ: Prescaler von 128
12    * ADPS2:0 = 111
13    * ADEN = Enable
14    */
15   ADCSRA|=(ANALOG_PRESCALER<<ADPS0);
16   ADCSRA|=(1<<ADEN);
17 }
18
19 uint8_t analog_abfrage(uint8_t adcAuswahl){
20   /*
21    * loescht die letzten 3 Bits und legt sie
22    * wie in der Auswahl
23    * Auswahl liegt im Bereich zwischen
24    * 0 und 7 (Pinnummer)
25    */
26   ADMUX = (ADMUX & (0xf8))|adcAuswahl;
27     // 0xf8 = 0b1111100
28   ADCSRA|=(1<<ADSC); // Messung Starten
29   while (ADCSRA & (1<<ADSC)); // Warte bis Messung fertig
30   return ADCH; // ADCL muss zuerst ausgelesen werden
31 }
```

Listing B.3: Ausschnitt aus analog.c

```
1  uint8_t leds_init(){ // lediglich auf Ausgang schalten
2    #ifdef USE_LED_BODEN
3      LED_BODEN_PORT_DDR |= (1<<LED_BODEN_PIN);
```

```
4       leds_setzeBoden(0);
5     #endif
6     #ifdef USE_LED_AKKU
7       LED_AKKU_PORT_DDR  |= (1<<LED_AKKU_PIN);
8       leds_setzeAkku(0);
9     #endif
10    return 0;
11  }
12
13  uint8_t leds_setzeBoden(uint8_t neuStatus){
14    // Bei der Akku LED wird die Kathode auf GND gezogen,
15    // Dies geschieht ueber einen Transistor
16    #ifdef USE_LED_BODEN
17      if (neuStatus){
18        LED_BODEN_PORT_OUT |= (1<<LED_BODEN_PIN);
19      }else{
20        LED_BODEN_PORT_OUT &=~(1<<LED_BODEN_PIN);
21      }
22    #endif
23    return 0;
24  }
25
26  uint8_t leds_setzeAkku(uint8_t neuStatus){
27    // Bei der Akku LED wird die Kathode auf GND gezogen,
28    // Dies geschieht direkt
29    #ifdef USE_LED_AKKU
30      if (neuStatus){ // anschalten
31        LED_AKKU_PORT_OUT &=~(1<<LED_AKKU_PIN);
32      }else{ // ausschalten
33        LED_AKKU_PORT_OUT |= (1<<LED_AKKU_PIN);
34      }
35    #endif
36    return 0;
37  }
```

Listing B.4: Ausschnitt aus leds.c

```
1
2   void maus_init(){
3     MAUS_PORT_DDR |= (1<<MAUS_PIN_SDA) | (1<<MAUS_PIN_SCK);
4     // SCK und SDA auf Output
5     MAUS_PORT_OUT |= (1<<MAUS_PIN_SDA) | (1<<MAUS_PIN_SCK);
6     // SCK und SDA auf high
```

```
7    sleepms(100);
8    maus_write(0x00,0x80);
9    // Reset
10   sleepms(10);
11   maus_write(0x00,0x01);
12   // kein Sleep modus
13   sleepms(10);
14 }
15
16
17 void maus_writeByte(uint8_t data){
18   int8_t i;
19   MAUS_PORT_DDR |= (1<<MAUS_PIN_SDA);
20   // SDA auf Output
21   for (i=7; i>=0; i--){
22     MAUS_PORT_OUT &= ~(1<<MAUS_PIN_SCK);
23     //SCK auf Low, Daten vorbereiten
24     if(data&(1<<i)){ // Bit rausschieben
25       MAUS_PORT_OUT |= (1<<MAUS_PIN_SDA);
26     }else{
27       MAUS_PORT_OUT &=~ (1<<MAUS_PIN_SDA);
28     }
29     MAUS_PORT_OUT |= (1<<MAUS_PIN_SCK);
30     // SCK=1 Sensor uebernimmt auf steigender Flanke
31     _delay_us(1);
32     // Sensor Zeit lassen um Bit zu holen
33   }
34   MAUS_PORT_DDR &=~ (1<<MAUS_PIN_SDA);
35   MAUS_PORT_OUT &=~ (1<<MAUS_PIN_SDA);
36   // Hochohmig schalten
37 }
38
39 uint8_t maus_readByte(){
40   int8_t i;
41   uint8_t data=0;
42   _delay_us(3);
43   /*
44    * Sensor Zeit lassen,
45    * um die Daten aus dem Register zu holen
46    */
47   for (i=7; i>-1; i--){
48     MAUS_PORT_OUT &=~ (1<<MAUS_PIN_SCK);
49     /*
```

```
50        * SCK=0
51        * Sensor bereitet Daten auf fallender Flanke vor
52        */
53      _delay_us(1);
54      // Sensor kurz Zeit lassen
55      MAUS_PORT_OUT |= (1<<MAUS_PIN_SCK);
56      // SCK =1 Daten lesen  auf steigender Flanke
57      if(MAUS_PORT_IN & (1<<MAUS_PIN_SDA)){ // Bit einlesen
58        data |= (1<<i);
59      }else{
60        data &=~ (1<<i);
61      }
62    }
63    return data;
64  }
65
66  void maus_write(uint8_t address, uint8_t data){
67    address|=(1<<7); // MSB auf 1 fuer Schreibzugriff
68    maus_writeByte(address);
69    maus_writeByte(data);
70    sleepus(100);
71  }
72
73  uint8_t maus_read(uint8_t address){
74    maus_writeByte(address);
75    sleepus(100);
76    return maus_readByte();
77  }
78
79  int8_t maus_verschiebungx(){
80    return maus_read(0x03);
81  }
82  int8_t maus_verschiebungy(){
83    return maus_read(0x02);
84  }
```

Listing B.5: Ausschnitt aus maussensor.c

```
1  ISR(INT2_vect){
2    datenregister[DATAREG_STATUS]|=
3      (1<<DATAREG_STATUS_BUMPER);
4    if (datenregister[DATAREG_STATUS] &
5      (1<<DATAREG_STATUS_BUMPERIGNORE)){
```

```
6       // Bumper zu ignorieren, nichts machen
7     } else {
8       motor_stopp();
9     }
10  }
11
12  void bumper_init(){
13    BUMPER_PORT_DDR &=~(1<<BUMPER_PIN_BUMPER); // Eingang
14    BUMPER_PORT_OUT &=~(1<<BUMPER_PIN_BUMPER);
15      // kein Pullup
16    MCUCSR &=~ (1<<ISC2); // fallende Flanke triggert
17    GICR |= (1<<INT2); // externer Interrupt2 aktiviert
18  }
```

Listing B.6: Ausschnitt aus bumper.c

```
1   uint8_t radencoder_init(){
2     RADENCODER_PORT_DDR &=~ ((1<<RADENCODER_PIN_LINKS) |
3       (1<<RADENCODER_PIN_RECHTS)); // als Eingang schalten
4     RADENCODER_PORT_OUT &=~ ((1<<RADENCODER_PIN_LINKS) |
5       (1<<RADENCODER_PIN_RECHTS)); // kein PullUp
6
7     datenregister[DATAREG_RADENCODER1_MSB]=0xff;
8     datenregister[DATAREG_RADENCODER1_LSB]=0xff;
9     datenregister[DATAREG_RADENCODER2_MSB]=0xff;
10    datenregister[DATAREG_RADENCODER2_LSB]=0xff;
11    datenregister[DATAREG_RADENCODERSUM_MSB]=0xff;
12    datenregister[DATAREG_RADENCODERSUM_LSB]=0xff;
13
14    datenregister[DATAREG_RADENCODERU1_MSB]=0x00;
15    datenregister[DATAREG_RADENCODERU1_LSB]=0x00;
16    datenregister[DATAREG_RADENCODERU2_MSB]=0x00;
17    datenregister[DATAREG_RADENCODERU2_LSB]=0x00;
18
19    MCUCR |= (1<<ISC01); // fallende Flanke trigger
20    MCUCR |= (1<<ISC11); // fallende Flanke trigger
21    return 0;
22  }
23
24  uint8_t radencoder_ablauf(){
25    datenregister[DATAREG_STATUS]|=
26      (1<<DATAREG_STATUS_RADENC);
27    if (datenregister[DATAREG_STATUS] &
```

```
28        (1<<DATAREG_STATUS_RADENCIGNORE)){
29      // Radencoder zu ignorieren, nichts machen
30    }else{
31      motor_stopp();
32    }
33  }
34
35  uint8_t radencoder_disable(){
36    GICR &=~((1<<INT0) | (1<<INT1));
37    // externer Interrupt0 und Interrupt 1 deaktiviert
38  }
39  uint8_t radencoder_enable(){
40    GICR |= ((1<<INT0) | (1<<INT1));
41    // externer Interrupt0 und Interrupt 1 aktiviert
42  }
43
44  uint8_t radencoder_tickdown(uint8_t registerMSB,
45    uint8_t registerLSB){
46    uint16_t radenc;
47    radenc=datenregister[registerMSB]*256+
48      datenregister[registerLSB];
49    radenc--;
50    if (radenc==0){
51      radencoder_ablauf();
52    }
53    datenregister[registerMSB]=radenc/256;
54    datenregister[registerLSB]=radenc%256;
55  }
56
57  uint8_t radencoder_tickup(uint8_t registerMSB,
58    uint8_t registerLSB){
59    uint16_t radenc;
60    // der aufwaerts zaehlende Radencoder
61    radenc=datenregister[registerMSB]*256+
62      datenregister[registerLSB];
63    radenc++;
64    datenregister[registerMSB]=radenc/256;
65    datenregister[registerLSB]=radenc%256;
66  }
67
68
69  ISR(INT0_vect){
70    // Interrupt-Routine fuer Radencoder; PD2 ist INT0
```

```
71    radencoder_tickdown(DATAREG_RADENCODER1_MSB,
72      DATAREG_RADENCODER1_LSB);
73    radencoder_tickdown(DATAREG_RADENCODERSUM_MSB,
74      DATAREG_RADENCODERSUM_LSB);
75    radencoder_tickup(DATAREG_RADENCODERU1_MSB,
76      DATAREG_RADENCODERU1_LSB);
77  }
78
79  ISR(INT1_vect){
80    // Interrupt-Routine fuer Radencoder; PD3 ist INT1
81    radencoder_tickdown(DATAREG_RADENCODER2_MSB,
82      DATAREG_RADENCODER2_LSB);
83    radencoder_tickdown(DATAREG_RADENCODERSUM_MSB,
84      DATAREG_RADENCODERSUM_LSB);
85    radencoder_tickup(DATAREG_RADENCODERU2_MSB,
86      DATAREG_RADENCODERU2_LSB);
87  }
```

Listing B.7: Ausschnitt aus radencoder.c

```
1   void motor_init(){
2     MOTOR_DIR_PORT_DDR |= 0xf0;
3
4     TCCR1A |=  (1<<WGM10)
5     // Waveform Generation Mode auf "Fast-PWM" setzen (I)
6              | (1<<COM1A1)
7        | (1<<COM1B1);
8     // Output Compare Mode auf "non-inverting Mode setzen"
9
10    TCCR1B |=  (1<<WGM12)
11    // Waveform Generation Mode auf "Fast-PWM" setzen (II)
12             | (1<<CS12)
13             | (1<<CS10);
14    // Frequenzvorteiler auf 1024 setzen
15    TCNT1 = 0x0000; // Vorladen des Timer1
16
17    PWM_DDR |= (1<<PWM_PIN_L) | (1<<PWM_PIN_R);
18    /*
19     * DDR-Pins von OC1A und OC1B,
20     * welche PWM-Signal erzeugen
21     */
22
23    motor_stopp(); // Zu Beginn ohne Fahrt
```

```
24
25    MOTOR_BUERSTEN_PORT_DDR |= (1<<MOTOR_BUERSTEN_PIN);
26    motor_buerste_stopp();
27  }
28
29  void motor_forward(uint8_t speed_left,
30    uint8_t speed_right){
31    motor_l_fw(); motor_r_fw();
32    motor_setzepwm(speed_left,speed_right);
33    radencoder_enable();
34  }
35
36  void motor_backward(uint8_t speed_left,
37    uint8_t speed_right){
38    motor_l_bw(); motor_r_bw();
39    motor_setzepwm(speed_left,speed_right);
40    radencoder_enable();
41  }
42
43  void motor_turn_left(uint8_t speed_left,
44    uint8_t speed_right){
45    motor_l_bw(); motor_r_fw();
46    motor_setzepwm(speed_left,speed_right);
47    radencoder_enable();
48  }
49
50  void motor_turn_right(uint8_t speed_left,
51    uint8_t speed_right){
52    motor_l_fw(); motor_r_bw();
53    motor_setzepwm(speed_left,speed_right);
54    radencoder_enable();
55  }
56
57  void motor_stopp(){
58    radencoder_disable();
59    motor_l_stopp(); motor_r_stopp();
60  }
61
62  void motor_l_fw(){
63    MOTOR_DIR_PORT_OUT |= (1 << MOTOR_L_DIR_PIN_FW);
64    MOTOR_DIR_PORT_OUT &=~(1 << MOTOR_L_DIR_PIN_BW);
65  }
66  void motor_l_bw(){
```

```
67    MOTOR_DIR_PORT_OUT &=~(1 << MOTOR_L_DIR_PIN_FW);
68    MOTOR_DIR_PORT_OUT |= (1 << MOTOR_L_DIR_PIN_BW);
69  }
70  void motor_l_stopp(){
71    MOTOR_DIR_PORT_OUT |= (1 << MOTOR_L_DIR_PIN_FW);
72    MOTOR_DIR_PORT_OUT |= (1 << MOTOR_L_DIR_PIN_BW);
73  }
74
75  void motor_r_fw(){
76    MOTOR_DIR_PORT_OUT |= (1 << MOTOR_R_DIR_PIN_FW);
77    MOTOR_DIR_PORT_OUT &=~(1 << MOTOR_R_DIR_PIN_BW);
78  }
79  void motor_r_bw(){
80    MOTOR_DIR_PORT_OUT &=~(1 << MOTOR_R_DIR_PIN_FW);
81    MOTOR_DIR_PORT_OUT |= (1 << MOTOR_R_DIR_PIN_BW);
82  }
83  void motor_r_stopp(){
84    MOTOR_DIR_PORT_OUT |= (1 << MOTOR_R_DIR_PIN_FW);
85    MOTOR_DIR_PORT_OUT |= (1 << MOTOR_R_DIR_PIN_BW);
86  }
87
88  /*
89   * Setzt die PWM-Werte, wird nur intern aufgerufen
90   */
91  void motor_setzepwm(uint8_t speed_left,
92    uint8_t speed_right){
93    PWM_L = speed_left;
94    PWM_R = speed_right;
95  }
96
97  void motor_buerste_start(){
98    MOTOR_BUERSTEN_PORT_OUT |= (1<<MOTOR_BUERSTEN_PIN);
99  }
100 void motor_buerste_stopp(){
101   MOTOR_BUERSTEN_PORT_OUT &=~(1 << MOTOR_BUERSTEN_PIN);
102 }
```

Listing B.8: Ausschnitt aus motortreiber.c

```
1  uint8_t checkTWI(){
2    if(!( TWCR & (1<<TWINT))){return 0;}
3    // wenn nix los ist, ist nix los.
4    switch (TWSR & 0xf8){
```

```
5       case TW_SR_SLA_ACK: { // Empfang
6         registerauswahl=COMMANDREGS;
7         while ((TWSR & 0xf8) != TW_SR_STOP){
8           TWCR= (1<<TWEN) | (1 << TWEA) | (1 << TWINT);
9     // Schickt ACK
10          while(!(TWCR & (1<<TWINT))){}
11    // warte auf Anforderung
12          switch (TWSR & 0xf8){
13            case TW_SR_DATA_ACK:{ // Raptor sendet noch
14              datenregister[registerauswahl]=TWDR;
15              registerauswahl++;
16              break;
17            } // case
18            case TW_SR_STOP: break;
19            default: TWIerror(TWSR); break;
20          } // switch
21        } // while
22        // Wenn ich hier lande, habe ich ein Stopp bekommen
23        TWCR= (1<<TWEN) | (1 << TWEA) | (1 << TWINT);
24        // ich bin bereit
25        break;
26      } // case Empfang
27      case TW_ST_SLA_ACK: { // senden
28        registerauswahl=datenregister[COMMANDREGS];
29        while ((TWSR & 0xf8) != TW_ST_DATA_NACK){
30          TWDR=datenregister[registerauswahl];
31          registerauswahl++; // irgendwelche Daten
32          TWCR = (1 << TWEN) | (1 << TWEA) | (1 << TWINT);
33    // Daten senden
34          while (!(TWCR & (1<<TWINT))){}
35    // warte auf Anforderung
36        } // while. NACK eingetroffen
37        TWCR= (1<<TWEN) | (1 << TWEA) | (1 << TWINT);
38        // und ich bin wieder bereit
39        break;
40      } // case
41      default: {TWIerror(TWSR);} // default
42    }// switch
43    return 0;
44  }
```

Listing B.9: Ausschnitt aus raptwi.c

```
1  uint8_t twi_senden(uint8_t adresse, uint8_t byteCount){
2    uint8_t zaehler=0;
3    TWCR = (1 << TWEN) | (1 << TWEA) |
4      (1 << TWSTA) |(1 << TWINT);
5    // Startsignal und trigger
6    while (!(TWCR & (1<<TWINT))){} // warte auf Anforderung
7    if ((TWSR & 0xf8)!= TW_START){TWIerror(TWSR);}
8
9    TWDR=(adresse<<1)|0; // Adressauswahl, schreiben
10   TWCR = (1 << TWEN) | (1 << TWEA) | (1 << TWINT);
11   // TWSTA wird geloescht, Daten (Adresse) senden
12
13   while (!(TWCR & (1<<TWINT))){} // warte auf Anforderung
14   if ((TWSR & 0xf8) != TW_MT_SLA_ACK){TWIerror(TWSR);}
15   while (zaehler<byteCount){
16     TWDR=datenregister[zaehler]; // Registerauswahl
17     TWCR = (1 << TWEN) | (1 << TWEA) | (1 << TWINT);
18     // Daten  (Register) senden
19     while (!(TWCR & (1<<TWINT))){}
20     // warte auf Anforderung
21     if ((TWSR & 0xf8) != TW_MT_DATA_ACK){
22       TWIerror(TWSR);
23     }
24     zaehler++;
25   }
26   TWCR = (1 << TWEN) | (1 << TWINT) | (1 << TWSTO);
27 }
28
29 uint8_t twi_lesen(uint8_t adresse, uint8_t byteCount){
30   uint8_t zaehler=0;
31   TWCR = (1 << TWEN) | (1 << TWEA) |
32     (1 << TWSTA) |(1 << TWINT);
33   // Startsignal und trigger
34   while (!(TWCR & (1<<TWINT))){} // warte auf Anforderung
35   if ((TWSR & 0xf8)!= TW_START){TWIerror(TWSR);}
36
37   TWDR=(adresse<<1)|1; // Adressauswahl, lesen
38   TWCR = (1 << TWEN) | (1 << TWEA) | (1 << TWINT);
39   // TWSTA wird geloescht, Daten (Adresse) senden
40   while (!(TWCR & (1<<TWINT))){} // warte auf Anforderung
41   if ((TWSR & 0xf8) != TW_MR_SLA_ACK){TWIerror(TWSR);}
42
43   while(zaehler<byteCount){
```

```
44      if (zaehler==byteCount-1){ // letztes Byte
45        TWCR = (1 << TWEN) | (1 << TWINT);
46        // Auf naechstes Datum warten, dabei NACK senden
47        while (!(TWCR & (1<<TWINT))){}
48        // warte auf Anforderung
49        if ((TWSR & 0xf8) != TW_MR_DATA_NACK){
50          TWIerror(TWSR);
51        }
52      }else{
53        TWCR = (1 << TWEN) | (1 << TWEA) | (1 << TWINT);
54        // Auf naechstes Datum warten, dabei ACK senden
55        while (!(TWCR & (1<<TWINT))){}
56        // warte auf Anforderung
57        if ((TWSR & 0xf8) != TW_MR_DATA_ACK){
58          TWIerror(TWSR);
59        }
60      }
61      datenregister[zaehler]=TWDR;
62      zaehler++;
63    }
64
65    TWCR = (1 << TWEN) | (1 << TWINT) | (1 << TWSTO);
66  }
67
68  void twi_registerLesen(uint8_t adresse,
69    uint8_t registerAuswahl, uint8_t byteCount){
70    datenregister[0]=registerAuswahl;
71    twi_senden(adresse,1);
72    twi_lesen(adresse,byteCount);
73  }
74
75  int main(void){
76    uart_init((1<<0)|(1<<1));
77
78    twi_sender_init();
79
80    uint8_t datenPaket;
81    uint8_t modus;
82    uint8_t adresse;
83    uint8_t byteCount;
84    uint8_t zaehler;
85    for (zaehler=0;zaehler<MAXDATAREGS;zaehler++){
86      datenregister[zaehler]=zaehler;
```

```
87    }
88    READY_LED_PORT_DDR |= (1<<READY_LED_PIN);
89    while(1){
90      zaehler=0;
91      READY_LED_PORT_OUT &=~(1<<READY_LED_PIN);
92      datenPaket=uart_wait_and_getc();
93      READY_LED_PORT_OUT |= (1<<READY_LED_PIN);
94      uart_putc(datenPaket); // Echo
95      modus=(datenPaket & (1<<7))/(1<<7);
96      adresse=(datenPaket & (3<<5))/(1<<5)+96;
97      byteCount=(datenPaket & (31<<0));
98      if (modus==1){ // Lesen vom Slave
99        if (adresse<99){twi_lesen(adresse,byteCount);}
100       for(zaehler=0;zaehler<byteCount;zaehler++){
101         uart_putc(datenregister[zaehler]);
102       }
103     }
104     if (modus==0){ // schreiben auf Slave
105       for(zaehler=0;zaehler<byteCount;zaehler++){
106       datenregister[zaehler]=uart_wait_and_getc();
107       uart_putc(datenregister[zaehler]);
108       if(adresse<99){twi_senden(adresse,byteCount);};
109     }
110   }
111   return 0;
112 }
```

Listing B.10: Das Programm des RS232-I2C-Adapters

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] *Bewegungserkennung in Bildfolgen - ein mehrstufiger Ansatz ; mit einem Geleitwort von H. Niemann & Harald Kirchner.* Wiesbaden :Deutscher Universitaets Verlag,, 1993.

[2] 3000, Kärcher Robocleaner RC. http://www.karcher.de/de/Produkte/Privat/Home_Cleaning/Robocleaner/12691010_Detailinfo.htm.

[3] Agilent-Technologies. http://www.agilent.com.

[4] Atmel. http://www.atmel.com.

[5] Atmel-Corporation. http://www.atmel.com.

[6] brown. http://www.cs.brown.edu/stc/education/course95-96/Kalman-Filters/kalman.html.

[7] carolina university-of-north. http://www.cs.unc.edu//$\sim$welch/media/pdf/kalman_intro.pdf.

[8] C't-Bot. http://www.heise.de/ct/projekte/ct-bot.

[9] Devantech. http://www.robot-electronics.co.uk.

[10] EAGLE. http://www.cadsoft.de.

[11] Foley, James D. & Dam, Andries van: *Grundlagen der Computergrafik.* Addison Wesley GmbH.

[12] Gumstix. http://www.gumstix.com.

[13] Hartmann. http://www.hartmann-codier.de.

[14] Histogramm, Wikipedia. http://de.wikipedia.org/wiki/Histogramm#Histogramm_in_der_Bildverarbeitung.

[15] I2C, Gumstix Wiki. http://www.davehylands.com/gumstix-wiki/i2c/.

[16] Kaiser, Ulrich & Kecher, Christoph: *C/C++ - Von den Grundlagen zur professionellen Programmierung, mit CD.* Galileo Press.

[17] Kung-Long. http://www.klb.com.tw.

[18] Logitech. http://www.logitech.com.

[19] Marshall, David. http://www.cs.cf.ac.uk/Dave/Vision_lecture/node35.html.

[20] Neuburger, Edgar. Einführung in die Theorie des linearen Optimalfilters - Oldenbourg Verlag 1972 - ISBN: 3-486-39361-8.

[21] Normalverteilung, Wikipedia. http://de.wikipedia.org/wiki/Normalverteilung.

[22] OpenSource. http://www.nongnu.org/avrdude.

[23] Patente-Studierende. http://www.patente-studierende.de.

[24] Philips. http://www.Philips.com.

[25] Player-Stage-Gazebo. http://www-home.htwg-konstanz.de/~marcel/Robo/index.html.

[26] Roboternetz. http://www.roboternetz.de/wissen.

[27] Roboternetz. http://www.roboternetz.de/wissen/index.php/AVR-ISP_Programmierkabel.

[28] SGI. http://www.sgi.com/tech/stl/.

[29] stuttgart uni. ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/DIP-2072/HtmlDateien/node49.html.

[30] (uBLAS), Boost Numeric Library. http://www.boost.org/doc/libs/1_36_0/libs/numeric/ublas/doc/index.htm.

[31] Vishay. http://www.vishay.com.

[32] VX, Gumstix Console. http://pubs.gumstix.org/boards/CONSOLE/VX/PCB10003-R1753.

[33] Wiki, Gumstix. http://docwiki.gumstix.org.

[34] Wikipedia. http://en.wikipedia.org/wiki/Floodfill.

[35] Wikipedia. http://en.wikipedia.org/wiki/Round-robin_scheduling.

[36] WinAVR. http://winavr.sourceforge.net/.