Technische Universität Darmstadt
Fachgebiet Simulation und Systemoptimierung
Fachbereich Informatik
Prof. Dr. O. von Stryk

SIMULATION UND SYSTEMOPTIMIERUNG sim

Technische Universität Darmstadt
Institut für Automatisierungstechnik
Fachgebiet Regelungstheorie und Robotik
Prof. Dr.-Ing. J. Adamy
Prof. em. Dr. rer. nat. Dipl.-Ing. H. Tolle

REGELUNGSTHEORIE UND ROBOTIK rtr

Diplomarbeit

Martin Kallnik

Sensordatensynchronisation zum Aufbau eines Navigationskalmanfilters für die mobile Robotik

Eingereicht bei
Prof. Dr. Oskar von Stryk

Betreuung durch
Prof. Dr.-Ing. Jürgen Adamy
Dipl.-Ing. Alexander Rudolph

2003 / 2004

Zusammenfassung

Der mobile Roboter des RTR wurde im Rahmen mehrerer Studien- und Diplomarbeiten um verschiedene Navigationssensoren erweitert. Zur Fusion der Sensordaten wird ein Kalmanfilter verwendet. Bei der Verwendung von Kalmanfiltern tritt das Problem auf, dass die Daten der verschiedenen Sensoren synchron vorliegen müssen, die Sensoren die Daten jedoch in verschiedenen Intervallen liefern. Daher müssen die Daten synchronisiert werden.

In dieser Diplomarbeit soll ein Navigationsprogramm entwickelt werden, das die Daten aller vorhandenen Sensoren ausliest, synchronisiert und an einen Kalmanfilter weiterreicht. Der Kalmanfilter selbst wird in einer weiteren Diplomarbeit von Hendrik Müller entworfen. Besonderes Augenmerk bei der Entwicklung des Navigationsprogramms soll auf dessen Modularität liegen. Diese soll es unter anderem ermöglichen, neue Sensoren problemlos in das Programm einbinden zu können.

Abstract

The mobile robot at RTR has various sensors for navigation. The data of these sensors is combined by a Kalman Filter. The problem by using a Kalman Filter is that it needs the data of all sensors at one time. The sensors deliver their data in individual intervalls. It is therefore necessary to synchronise the data.

In this thesis I will develop a new navigation program. It will deal with all sensors and will synchronise their data. The Kalman Filter itself is not covered in this thesis. It will be developed by Hendrik Müller. Special attention is paid to the fact that the navigation program should be modular in order to plug in new sensors easily.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

API	(Aplication Programming Interface) Schnittstelle der Klassen und Module eines Programms
ARIA	(ActivMedia Robotics Interface for Aplication) Eine C++-Klassenbibliothek zur Ansteuerung eines ActivMedia-Roboters (siehe Anhang C.1)
DNS	(Domain Name Service) Verteilter Verzeichnisdienst im Internet, der einen Rechnernamen (zum Beispiel www.tu-darmstadt.de) in eine IP-Adresse übersetzt
EKF	(Erweiterter Kalmanfilter) Filter zum Schätzen der internen Systemzustände eines Prozesses (siehe Kapitel 4)
GNU	(GNU is not UNIX) Das GNU-Projekt hat es sich zum Ziel gesetzt, ein freies UNIX zu entwickeln, neben einem Betriebssystemkern (GNU/Hurd) entstanden dabei viele Tools und Programme für UNIX/ Linux
GPL	(General Public License) Copyleft-Lizenz des GNU-Projekts. Sie erlaubt es, den Sourcecode zu ändern und die Änderungen weiterzugeben, solange das neue Programm ebenfalls unter der GPL veröffentlicht wird
HTML	(HyperText Markup Language) Textformatierungssprache, die vor allem im World Wide Web Verwendung findet
IP	(Internet Protocol) Standardprotokoll des Internet
LOPS	(Local Positioning System) Stationärer Positionssensor, der die Roboterposition über die Laufzeit von Schallsignalen berechnet (siehe Abschnitt 3.5)
MCU	(Main Controlling Unit) Zentrale Steuereinheit des LOPS-Systems
P2OS	(Pioneer 2 Operation System) Betriebssystem, das auf dem Mikrocontroller des Pioneer 2 Roboters läuft
PDF	(Portable Document Format) Von der Firma Adobe entwickelte Seitenbeschreibungssprache, die ein einheitliches Layout sicherstellt
RPM	(Redhat Packet Manager) Installations-Dateiformat, in dem die meisten Linux-Softwarepakete gespeichert sind

RS232	(Recommended Standard 232) Definition der seriellen Schnittstelle eines PCs. RS232 wird in diesem Text als Synonym für „serielle Schnittstelle“ verwendet
SIP	(Server Information Packet) Datenpaket, das zwischen dem Mikrocontroller und dem Onboardrechner des Roboters ausgetauscht wird
SSH	(Secure Shell) Kryptographisch gesicherte Konsolen-Verbindung zwischen zwei Rechnern. Über diese Verbindung kann auch die graphische Anzeige eines Programms übertragen werden
tar	(tape archiver) Programm zum Erstellen und Verwalten von Dateiarchiven
TCP	(Transmission Control Protocol) Datenprotokoll, das auf einer IP-Verbindung eine bidirektionale Datenverbindung aufbaut, die gegen Datenverlust geschützt ist
UI	(User Interface) Die Benutzerschnittstelle
WLAN	(Wireless Local Area Network) Lokales Netzwerk, das die Daten drahtlos überträgt
X	Die graphische Oberfläche eines UNIX/Linux-Systems

Formelzeichen

α	Winkel zwischen der X-Achse und der Normalen auf eine erkannte Wand (Hessesche Normalform)
ψ	Orientierung des Roboters
ω	Drehung des Roboters in einem Abtastschritt (siehe Abbildung 2.2)
σ	Standardabweichung
A	Systemmatrix des zu schätzenden Systems (Kalmanfilter, siehe Kapitel 4)
B	Eingangsmatrix des zu schätzenden Systems (Kalmanfilter, siehe Kapitel 4)
b	Achsenabschnitt der Ausgleichsgeraden bei der Wanderkennung mit den seitlichen Sonarsensoren (Geradengleichung mit Richtungskoeffizient)
d	Abstand einer Wand vom Ursprung (Hessesche Normalform)
e	Schätzfehler des Kalmanfilters (siehe Kapitel 4)
f	Systemfunktion des zu schätzenden Systems (Kalmanfilter, siehe Kapitel 4)
h	Messfunktion des zu schätzenden Systems (Kalmanfilter, siehe Kapitel 4)
H	Messmatrix des zu schätzenden Systems (Kalmanfilter, siehe Kapitel 4)
K	Kalmanverstärkung (siehe Kapitel 4)
l	Halbe Spurbreite des Roboters (160 mm)
m	Richtungskoeffizient der Ausgleichsgeraden bei der Wanderkennung mit den seitlichen Sonarsensoren (Geradengleichung mit Richtungskoeffizient)
n	Anzahl der zur Wanderkennung verwendeten Hindernispunkte / Hindernispunktepaare
NP	Nullpunkt (der Wert, den der Sensor liefert, wenn die zu messende Größe Null ist)
P	Kovarianzmatrix des Schätzfehlers des Kalmanfilters (siehe Kapitel 4)
Q	Kovarianzmatrix des Prozessrauschens des zu schätzenden Systems (Kalmanfilter, siehe Kapitel 4)

R	Kovarianzmatrix des Messrauschens des zu schätzenden Systems (Kalmanfilter, siehe Kapitel 4)
r	Radius des Kreises den der Roboter fährt (siehe Abbildung 2.2)
RP	Roboterposition im globalen Koordinatensystem
s	Zurückgelegte Strecke des Roboters in einem Abtastschritt (siehe Abbildung 2.2)
s_l	Die Anzahl der Enkoderticks des linken Rades im letzten Abtastschritt
s_r	Die Anzahl der Enkoderticks des rechten Rades im letzten Abtastschritt
SF	Skalenfaktor zur Umrechnung eines Sensorrohwertes in den Messwert
t	Zeitpunkt
T	Dauer eines Abtastschritts
u	Eingangsvektor des zu schätzenden Systems (Kalmanfilter, siehe Kapitel 4)
V	Einflussmatrix des Messrauschens des zu schätzenden Systems (Kalmanfilter, siehe Kapitel 4)
v	Messrauschen des zu schätzenden Systems (Kalmanfilter, siehe Kapitel 4)
v	Translationsgeschwindigkeit des Roboters
v_l	Geschwindigkeit des linken Rades (siehe Abbildung 2.2)
v_r	Geschwindigkeit des rechten Rades (siehe Abbildung 2.2)
W	Einflussmatrix des Prozessrauschens des zu schätzenden Systems (Kalmanfilter, siehe Kapitel 4)
w	Prozessrauschen des zu schätzenden Systems (Kalmanfilter, siehe Kapitel 4)
x	X-Position
x	Zustandsvektor des zu schätzenden Systems (Kalmanfilter, siehe Kapitel 4)
$\hat{x}$	A posteriori Schätzung des Zustandsvektors des zu schätzenden Systems (Kalmanfilter, siehe Kapitel 4)
$\hat{x}^-$	A priori Schätzung des Zustandsvektors des zu schätzenden Systems (Kalmanfilter, siehe Kapitel 4)
y	Y-Position
z	Messvektor des zu schätzenden Systems (Kalmanfilter, siehe Kapitel 4)
$\hat{z}$	Geschätzter Messvektor des zu schätzenden Systems (Kalmanfilter, siehe Kapitel 4)

Kapitel 1

Einleitung

1.1 Ziel der Arbeit

Am Fachgebiet Regelungstheorie und Robotik werden Methoden zur Navigation mobiler Roboter untersucht. Diese Untersuchungen werden an einem *Pioneer 2*-Roboter (Version *PeopleBot*) der Firma *ActivMedia* durchgeführt.

Der Roboter verfügt neben den standardmäßig vorhandenen Sensoren (Odometrie und Sonar) über ein Gyroskop und einen Bodenkorrelator. Außerdem ist im Labor das stationäre LOPS-System aufgebaut. Die Fusion der Sensordaten soll mit einem Erweiterten Kalmanfilter (EKF) erfolgen. Dieser benötigt die Daten der verschiedenen Sensoren zeitgleich in konstanten Intervallen. Die verschiedenen Sensoren arbeiten jedoch alle mit unterschiedlichen Abtastintervallen, auch stehen nicht immer alle Sensordaten zur Verfügung. Die Daten der Wanderkennung stehen zum Beispiel nur dann zur Verfügung, wenn der Roboter vor einer Wand steht, auch können Sensoren ausfallen oder absichtlich abgeschaltet sein.

Im Moment steht noch kein Programm zur Verfügung, das in der Lage ist, die Daten aller Sensoren zu erfassen, zu synchronisieren und an einen online-arbeitenden Erweiterten Kalmanfilter weiterzugeben. Die bisherigen Untersuchungen wurden daher lediglich mit den Daten der Odometrie, des Gyroskops und der Sonarsensoren durchgeführt.

Ziel der Arbeit ist es, ein Synchronisationsprogramm zu entwickeln, das die Daten aller vorhandenen Sensoren ausliest, synchronisiert und an einen Erweiterten Kalmanfilter weiterreicht. Neben dieser Onlinebearbeitung der Sensordaten soll das Programm die Daten in eine Logdatei speichern. Diese Logdatei soll auf einem PC offline bearbeitet werden können. Weiterhin soll sie zur Analyse der Navigation dienen (zum Beispiel zur Auswertung der Positionsfehler der einzelnen Sensoren und des Kalmanfilters) und in Matlab verarbeitet werden können.

Das Programm soll modular aufgebaut sein und es so ermöglichen, leicht weitere Sensoren hinzuzufügen. Die Konfiguration des Programms soll einfach geändert werden können, so dass es möglich ist, einzelne Sensoren ein- und auszuschalten, die Daten bestimmter Sensoren nicht im Kalmanfilter zu verwenden oder die Ausgabe von Fehlermeldungen zu beeinflussen.

1.2 Aufbau der Arbeit

Im ersten Teil der Arbeit werde ich die Grundlagen darlegen. Dieser Teil besteht aus einer Beschreibung des verwendeten Roboters (Kapitel 2), einer Erläuterung der im Moment vorhandenen Sensoren (Kapitel 3) sowie einer Darstellung des Prinzips eines Kalmanfilters (Kapitel 4).

Im zweiten Teil der Arbeit werden die in der Diplomarbeit entwickelten Programme beschrieben. Dazu stelle ich das Programmkonzept (Kapitel 5), die Umsetzung des Programmkonzepts (Kapitel 6) sowie die Möglichkeiten zur Erweiterung des Programms um weitere Sensormodule (Kapitel 7) dar. Anschließend beschreibe ich die Bedienung des Programms (Kapitel 8) sowie die verschiedenen Hilfsprogramme, die die Auswertung der Logdateien erleichtern (Kapitel 9).

Im dritten Teil werde ich abschließend das Synchronisationsprogramm bewerten. Dazu werde ich die Ergebnisse, die mit dem Programm erreicht wurden, darlegen (Kapitel 10) und einen Ausblick auf mögliche Erweiterungen und Verbesserungen des Programms geben (Kapitel 11).

Im Anhang befindet sich das Literaturverzeichnis, eine Beschreibung der beiliegenden Programmdokumentationen (Anhang A) und der verwendeten Dateiformate (Anhang B), eine Erläuterung, welche externen Programme zur Nutzung des Programms nötig sind, (Anhang C) sowie eine Anleitung zur Installation der Software (Anhang D).

Teil I

Einführung

Kapitel 2

Roboter

Abbildung 2.1: *Der verwendete Roboter: Zu sehen sind ein Rad des Roboters, die beiden Sonarsensorarrays und der LOPS-Sender oben auf dem Roboter.*

Bei dem in dieser Arbeit verwendeten Roboter handelt es sich um einen Pioneer 2 DX Roboter in der Version PeopleBot, der von der Firma ActivMedia hergestellt wurde. Der Roboter verfügt über einen Differenzialantrieb, mehrere Sensoren, eine Mikrocontrollerplatine, die die Sensoren und den Antrieb steuert, sowie einen PC als Steuerrechner, auf dem vom Benutzer erstellte Programme ablaufen können. Eine detaillierte Beschreibung des Roboters ist im Handbuch (ActiveMedia Robotics, LLC 2002) nachzulesen.

2.1 Aufbau

Der Pioneer2 DX Roboter ist ein kleiner wendiger Roboter, der in der Version PeopleBot über Aufbau von ungefähr einem Meter Höhe verfügt. Dieser ermöglicht es auch in dieser Höhe Sensoren anzubringen. Die Technik ist im unteren Teil des Roboters untergebracht, hier befinden sich neben dem Antrieb der Mikrocontroller, der Steuerrechner und die Akkumulatoren.

Die beiden angetriebenen Räder sind seitlich angebracht, an der Rückseite verfügt der Roboter über ein frei drehbares nicht angetriebenes Stützrad.

Auf den beiden Plattformen und im Aufbau befindet sich viel Platz für weitere Sensoren oder anderes Zubehör, wie zum Beispiel einen Greifer. Auch die Tragfähigkeit des Roboters bietet noch viel Spielraum, da der Roboter bei einem Eigengewicht von neun Kilogramm bis zu 23 kg Zuladung transportieren kann.

2.2 Antrieb

Der Roboter verfügt über einen Differenzialantrieb. Die beiden Antriebsräder liegen auf derselben Achse und werden getrennt angetrieben, das hintere Stützrad ist frei drehbar und nicht angetrieben. Durch diesen Aufbau ist der Roboter sehr beweglich. Er kann vorwärts und rückwärts fahren, beliebige Kurvenradien einschlagen und sich auf der Stelle drehen. Durch die sehr kompakte (nahezu runde) Bauform kann sich der Roboter selbst in sehr engen Räumen und Gängen bewegen.

Die Bewegung des Roboters wird durch die Geschwindigkeit der beiden angetriebenen Räder bestimmt. Die Bewegung, die der Roboter ausführt, lässt sich nach dem Robotik1-Vorlesungsskriptum (von Stryk 2002) in Abhängigkeit von den beiden Radgeschwindigkeiten als Kreisbahn um ein Drehzentrum beschreiben (siehe Abbildung 2.2). Es gilt, dass beide Räder mit der gleichen Winkelgeschwindigkeit (ω) um das Drehzentrum rotieren. Bei einem Radius (r), mit dem der Roboter um das Drehzentrum rotiert, ergibt sich für die Radgeschwindigkeiten (v_l für das linke und v_r für das rechte Rad):

$$\begin{aligned} v_r &= \omega \cdot (r + l) \\ v_l &= \omega \cdot (r - l) \\ \text{mit:} \quad l &= 0,5 \cdot \mathit{Spurweite} \end{aligned}$$

Für die Vorwärtskinematik ergibt sich:

$$\begin{aligned} \omega &= \frac{v_r - v_l}{2 \cdot l} \cdot T \\ r &= \frac{v_r + v_l}{v_r - v_l} \cdot l \end{aligned}$$

Für $v_l = v_r$ ergibt sich $r = \infty$, der Roboter fährt geradeaus.

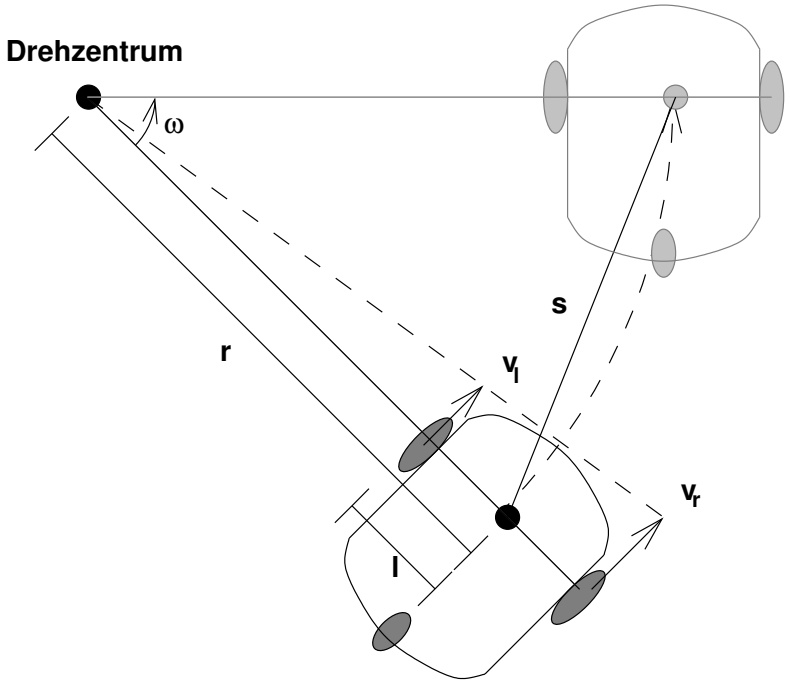


Abbildung 2.2: *Die Kinematik des Roboters: Der Roboter bewegt sich bei konstanten Radgeschwindigkeiten auf einer Kreisbahn mit dem Radius r um ein Drehzentrum.*

Für die während der Zeit T zurückgelegte Distanz s ergibt sich:

$$s = \begin{cases} v_r \cdot T & \text{falls} \quad v_l = v_r \quad (r = \infty) \\ 2 \cdot \sin\left(\frac{\omega}{2}\right) \cdot r & \text{sonst} \end{cases}$$

Dieser Antrieb hat allerdings den Nachteil, dass sich Fehler in den Radgeschwindigkeiten nicht nur auf die Geschwindigkeit des Roboters, sondern auch auf die Bahn auswirken. Deshalb sind gute Geschwindigkeitsregler an beiden Rädern notwendig.

2.3 Mikrocontroller und Onboard-Steuerrechner

Zur Steuerung besitzt der Roboter einen Mikrocontroller und einen Onboard-Steuerrechner. Auf dem Mikrocontroller läuft die Software P2OS, die die Antriebe und die Sensoren, die von ActivMedia auf dem Roboter installiert wurden, steuert. Falls Zubehör (zum Beispiel ein Greifer) vorhanden ist, wird auch dieses an den Mikrocontroller angeschlossen. Der Mikrocontroller berechnet aus den Daten der beiden Radenkoder die Position des Roboters. Diese Position verwendet er, um Fahrbefehle wie zum Beispiel „Fahre 1 m vorwärts“ ausführen zu können.

Der Onboard-Steuerrechner übernimmt die Steuerung auf höherer Ebene. Er kommuniziert mit dem Mikrocontroller über die serielle Schnittstelle. Auf dem Steuerrechner läuft Linux als Betriebssystem, zur Kommunikation mit dem Mikrocon-

troller wird die Softwarebibliothek ARIA (siehe Anhang C.1) verwendet. Der Onboard-Steuerrechner kann auch durch einen Laptop ersetzt werden, der dann mit dem Mikrocontroller kommuniziert.

Über die Serielle Verbindung tauschen die beiden Rechner so genannte Server Information Packets (SIPs) aus. Die SIPs, die der Mikrocontroller an den Steuerrechner sendet, enthalten die Positionsdaten, die Daten der an den Mikrocontroller angeschlossenen Sensoren und allgemeine Zustandsinformationen über den Roboter (zum Beispiel, ob die Motoren angeschaltet sind). Die SIPs vom Onboardrechner an den Mikrocontroller enthalten Steuerbefehle (zum Beispiel Fahrbefehle oder Befehle für Zubehör).

Die Bedienung des Steuerprogramms auf dem Onboard-Steuerrechner des Roboters kann von einem beliebigen Rechner im Internet erfolgen. Dazu loggt sich der Benutzer über SSH auf dem Onboard-Steuerrechner ein und startet das Steuerprogramm, danach kann er das Programm von seinem Arbeitsplatz aus steuern. Zur Vermeidung von Unfällen ist es sinnvoll, dass sich der Benutzer im selben Raum wie der Roboter befindet.

Kapitel 3

Navigationssensoren

Der Roboter des Fachgebiets Regelungstechnik und Robotik ist mit mehreren Sensoren zur Navigation ausgestattet. Neben den Sensoren, über die der Pioneer2 DX Roboter standardmäßig verfügt (Odometrie und Sonarsensoren), wurden weitere Sensoren (Gyroskop, Bodenkorrelator) montiert. Außerdem existiert ein externes Ortungssystem (LOPS), das die absolute Position des Roboters bestimmen kann. In diesem Kapitel möchte ich die einzelnen Sensoren kurz beschreiben, detailliertere Ausführungen sind in älteren Studien- und Diplomarbeiten, auf die bei den einzelnen Sensoren verwiesen wird, zu finden.

Die verschiedenen Sensoren sind an verschiedenen Rechnern angeschlossen. Die mit dem Roboter gelieferten Sensoren sind an den Mikrocontroller des Roboters angeschlossen, die nachträglich installierten Sensoren an die seriellen Schnittstellen des Onboardrechners. Das LOPS-System ist als ortsfestes System an einen externen PC angeschlossen. Abbildung 3.1 zeigt diese Anordnung und alle Verbindungen, über die die Daten der Sensoren zum Onboardrechner gelangen, da auf diesem die Synchronisationssoftware läuft. Die Ausgaben der Software werden mittels einer SSH-Verbindung mit X-Forwarding an einen PC weitergeleitet, von dem aus der Benutzer

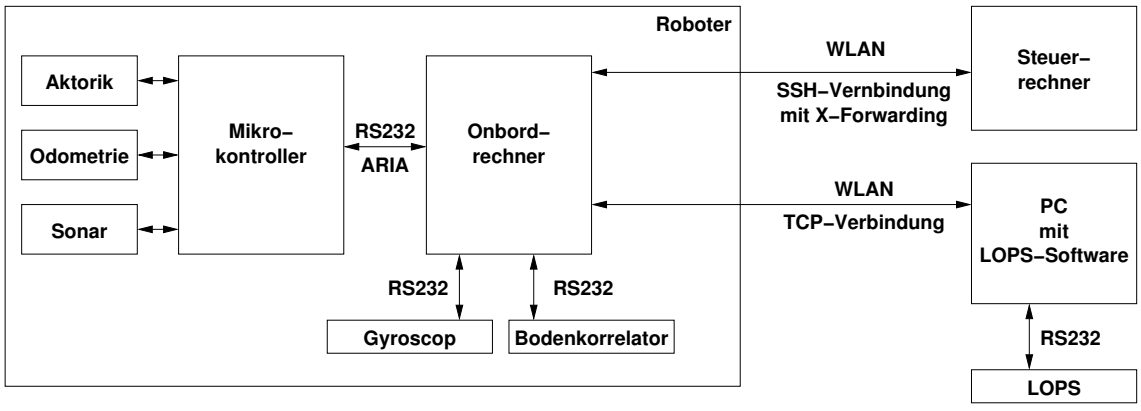


Abbildung 3.1: *Der Anschluss der verschiedenen Sensoren an die beteiligten Computer. Ein Teil der Sensoren ist direkt am Onboardrechner, auf dem das Synchronisationsprogramm läuft, angeschlossen. Zwei Sensoren sind an den Mikrocontroller angeschlossen. Das LOPS-System läuft auf einem stationären PC.*

das Programm und den Roboter steuern und überwachen kann. Steuerrechner und LOPS-Rechner können identisch sein.

Eine Zusammenfassung der vorhandenen Sensoren ist in Tabelle 3.1 am Ende des Kapitels zu sehen. In dieser Tabelle sind die wichtigen Daten aller vorhandenen Sensoren sowie die für den Kalmanfilter relevanten Informationen angegeben.

3.1 Odometrie

Odometriesensoren messen die Bewegung des Roboters. Mit Hilfe der gemessenen Bewegung (Positionsänderung) kann die Position des Roboters fortgeschrieben werden. Bei jedem Abtastschritt des Odometriesensors wird die Bewegung des Roboters auf seine letzte Position addiert.

Der Pioneer 2 DX Roboter besitzt an den beiden Antriebsrädern jeweils einen optischen Enkoder, der die Umdrehung des Rades misst. Aus diesen Werten kann mittels der Kinematik des Roboters der vom Roboter zurückgelegte Weg berechnet werden (siehe Abschnitt 2.2). Die Daten der beiden Enkoder werden vom Roboterbetriebssystem P2OS (auf dem Mikrocontroller) verwendet, um die Position des Roboters zu berechnen. Sie können aber auch vom Onboardrechner angefordert werden, dann werden sie als SIP über die serielle Schnittstelle übertragen. Der Skalenfaktor der Enkoder ($SF_{Odometrie}$) beträgt abweichend von der Dokumentation des Herstellers (ActiveMedia Robotics, LLC 2002) $\frac{1}{152}$ mm pro Tick. Dieser Wert wurde von Jochen Seidenspinner nachgemessen (Seidenspinner 2001/2002).

Die größte Fehlerquelle des Sensors liegt im Schlupf der Räder. Dieser wird beim Pioneer 2 DX dadurch verstärkt, dass sich die Sensoren (mangels Alternative) an den Antriebsrädern befinden und bei diesen der Schlupf beim Beschleunigen und Bremsen besonders stark ist.

3.2 Gyroskop

Ein Gyroskop (Kreisel) misst die Drehung eines Roboters. Auf dem Roboter ist ein Piezokreisel vorhanden, der die Drehung des Roboters mit Hilfe der Corioliskraft bestimmt. Der Kreisel wurde von Andreas Siegel in das Robotersystem integriert (Siegel 2001). Der Sensor ist an die serielle Schnittstelle des Onboardrechners angeschlossen.

Das Gyroskop liefert alle 20,48 ms einen Rohwert. Dieser wird dann in eine Drehrate umgerechnet. Für diese Umrechnung muss der Nullpunkt des Gyroskopsignals, der von der Temperatur des Gyroskops abhängt, bekannt sein. Daher muss das Gyroskop zu Beginn der Messung kalibriert werden. Dazu werden (während der Roboter still steht) mehrere Messwerte aufgenommen und aus diesen der Mittelwert berechnet. Dieser Wert wird bei der Messung dann als Nullpunkt verwendet.

Das prinzipielle Problem des Gyroskops liegt in seiner Drift. Kleine Fehler bei der Kalibrierung des Gyroskops oder eine Temperaturänderung führen dazu, dass das Gyroskop auch im Stand eine Drehrate liefert. Dieser Fehler integriert sich mit der Zeit auf.

3.3 Bodenkorrelator

Der Bodenkorrelator des Roboters bestimmt anhand der optischen Veränderung des Bodens die Translationsgeschwindigkeit des Roboters. Dazu verfügt er über zwei hintereinander angeordnete Sensoren, die die Stärke eines vom Boden reflektierten Lichtstrahls messen. Da sich beide Sensoren auf demselben Pfad bewegen, sind ihre Daten (bis auf Messfehler) identisch. Weil die Sensoren aber hintereinander angeordnet sind, sind die Daten zeitversetzt. Ein Korrelator bestimmt den Zeitversatz der beiden Signale. Aus diesem Zeitversatz und dem bekannten Abstand der beiden Sensoren kann die Translationsgeschwindigkeit des Roboters bestimmt werden. Der verwendete Bodenkorrelator wurde von Hector Bratos in das System integriert und getestet (Bratos 2002).

Bei der Verwendung des Korrelators kann es zu mehreren Fehlern kommen. Da der Korrelator nicht im Drehpunkt des Roboters eingebaut werden konnte, fließt neben der Translation auch die Rotation mit in das Ergebnis ein. Bei einer Drehung des Roboters ist die gemessene Geschwindigkeit zu groß. Da die Rotationsgeschwindigkeit vom Sensor nicht gemessen werden kann, kann der Fehler nicht vom Sensor korrigiert werden. Wenn der Roboter sich auf der Stelle dreht, bewegen sich die Sensoren auf unterschiedlichen Wegen über den Boden, eine Korrelation ist dann überhaupt nicht möglich.

Aber auch beim Geradeausfahren ist es möglich, dass der Bodenkorrelator falsche Werte liefert. Bei den Testfahrten von Hector Bratos ist es besonders bei dynamischen Fahrten des öfteren vorgekommen, dass der Korrelator die Bewegungsrichtung des Roboters falsch bestimmt hat. Dieser Fehler kann über einen anderen Navigationssensor ausgeglichen werden. Da der Bodenkorrelator sowohl eine Vorwärts- als auch eine Rückwärtsgeschwindigkeit liefert, ist es möglich, die Bewegungsrichtung des Roboters mit einem anderen Sensor zu bestimmen und dann die passende Bewegungsgeschwindigkeit des Bodenkorrelators zu wählen.

Da der Bodenkorrelator im Moment defekt und beim Hersteller zur Reparatur ist, steht er für Tests nicht zur Verfügung. Da der Hersteller insolvent ist, ist auch nicht bekannt, ob er jemals wieder verwendet werden kann. Die vorhandenen Korrelatorfunktionen aus der Diplomarbeit von Hector Bratos (Bratos 2002) konnten daher nicht portiert werden.

3.4 Ultraschallsensoren

Der Roboter verfügt über 16 Ultraschallsensoren, mit denen er Abstände zu Hindernissen messen kann. Die Sensoren sind in 2 Arrays mit je 8 Sensoren organisiert. Eines dieser Arrays ist im unteren Bereich des Roboters (20 cm über dem Boden) montiert, das andere im oberen Bereich (102 cm über dem Boden). Die Sensoren beider Arrays decken jeweils den Raum vor dem Roboter ab (-90° bis +90°).

Die Sensoren können Hindernisse in einem Abstand zwischen 10 cm und 5 m erkennen. Die Abfrage der Sensoren erfolgt multiplexed, in jedem Array wird alle

40 ms ein Sensor ausgelesen. Jeder einzelne Sensor wird also alle 320 ms ausgewertet.

Die Werte der Sonarsensoren werden vom Mikrocontroller des Roboters ausgelesen und mit den normalen Server Information Packets (SIPs) alle 100 ms an den Onboardrechner übertragen. Mit der ARIA-Bibliothek (siehe Anhang C.1) können die Messungen der Sensoren abgefragt werden, außerdem lässt sich abfragen, welche der Werte in den letzten 100 ms aktualisiert wurden.

Diese Daten werden dem Kalmanfilter direkt übergeben, außerdem werden nach dem von Richard Woller beschriebenen Verfahren (Woller 2001) mit diesen Daten Wände erkannt. Auch die erkannten Wände stehen dem Kalmanfilter als Eingabewerte zur Verfügung. Außerdem wird mit Hilfe einer geschätzten Position des Roboters versucht, die von den Sonarsensoren erkannten Wände den Laborwänden, die bekannt sind, zuzuordnen. Gelingt dies, kann die geschätzte Position weiter verbessert werden.

Bei dieser Positionsschätzung wird aufgrund des verwendeten Lokalisierungsverfahrens keine eigene Position errechnet. Das Verfahren ist lediglich in der Lage, eine vorhandene Positionsschätzung zu verbessern. Ist die zugrundeliegende Schätzung jedoch zu schlecht, werden die erkannten Wände falsch zugeordnet, was zu keiner Verbesserung der Positionsschätzung führt.

Alternativ könnte versucht werden aus den Sonarwerten, die vom Roboter geliefert werden, direkt die Position zu schätzen. Michel Hunsiker hat diese Möglichkeit in seiner Diplomarbeit untersucht (Hunsiker 2000). Allerdings wurde das neuronale Netz in dieser Arbeit vor allem in einem simulierten Raum getestet. Ein Training im realen Labor fand nur in einigen Teilbereichen statt und der Raum hat sich seit dieser Diplomarbeit auch verändert, daher ist diese Lösung nur mit großem Aufwand verwendbar.

3.5 LOPS — Local Positioning System

Das LOPS-System bestimmt im Gegensatz zu den anderen vorhandenen Navigationssensoren die absolute Position des Roboters. Es wurde von Marcel Blank und Thorsten Kleinschmidt entwickelt und umgesetzt. Das LOPS-System besteht aus einer Sendeeinheit auf dem Roboter und einer Empfangseinheit, die aus einer zentralen Steuereinheit (MCU) und drei Ultraschallempfängern besteht. Die Empfangseinheit ist an einen stationären PC angeschlossen.

Die MCU sendet alle 98 ms ein Funksignal aus, das die Messung initialisiert. Sobald die Sendeeinheit auf dem Roboter das Signal empfängt, sendet sie ein 40 kHz Ultraschallsignal. Dieses wird von den drei Ultraschallempfängern detektiert. Die MCU misst die Zeit zwischen dem Aussenden des Funksignals und dem Detektieren des Ultraschallsignals an den drei Empfängern. Aus diesen Signallaufzeiten lässt sich die Entfernung zwischen dem Roboter und den Ultraschallempfängern bestimmen. Mit Hilfe dieser Entfernungen kann anschließend die Position des Roboters berechnet werden. Die Berechnung der Roboterposition aus den Laufzeiten erfolgt auf dem PC, an den das System angeschlossen ist.

Da das LOPS-System an einen stationären PC angeschlossen ist, muss die ermittelte Position anschließend noch an den Onboardrechner übermittelt werden, dies

geschieht über die WLAN-Verbindung des Roboters.

Das LOPS-System arbeitet sehr genau, daher bietet es sich als Referenz-Messung an, mit der die Ergebnisse der anderen Sensoren und des Kalmanfilters überprüft werden können. Weitere Details zu LOPS sind in (Rudolph et al. 2002*b*), (Blank 2002) und (Kleinschmidt 2002) zu finden.

	Odometrie	**Gyroskop**	**Korrelator**	**Sonar**	**LOPS**
Sensor-rohdaten	linkes und rechtes Enkoderin-krement	Rotations-geschwin-digkeit des Roboters	Transla-tionsinkre-ment	Entfern-ungsmes-sungen der Sonarsen-soren und erkannte Wände	Entfern-ungen der Empfänger vom Robot-er
Sensor-takt	100 ms	20,48 ms	variabel (>40ms)	100 ms	98 ms
Positions-berech-nung	Durch Inte-gration der aus den En-koderwerten berechneten Translatio-nen und Rotationen	Bestimm-ung des Roboter-winkels durch Inte-gration der Rotation	nicht möglich	Korrektur vorhanden-er Positi-onsdaten durch Zu-ordnung erkannter Wände zu bekannten Wänden	Berechnung der Robo-terposition durch Trila-teration
Daten für den Kal-manfilter	linkes und rechtes Enkoderin-krement	Rotations-geschwin-digkeit des Roboters	Transla-tionsinkre-ment	Entfern-ungsmes-sungen der Sonar-sensoren und erkann-te Wände	Entfern-ungen des Roboters von den Empfängern
Art der Synchro-nisation	Summe der jeweiligen Enkoderin-kremente	Durch-schnitt über die Rota-tionsge-schwindig-keiten	Summe der Translati-onsinkre-mente	die neusten Werte der Sensoren bzw. An-sammlung aller er-kannten Wände	die neusten Entfernun-gen und Position des Roboters

Tabelle 3.1: *Die Daten der verschieden Sensoren: Die verschiedenen Sensoren liefern Daten über den Zustand des Roboters, aus diesen Daten können unterschiedliche Informationen über die Position des Roboters gewonnen werden.*

Kapitel 4

Kalmanfilter

Ein Kalmanfilter ist ein Berechnungsalgorithmus zur Schätzung der internen Zustände eines zeitdiskreten Systems mit Hilfe von Sensordaten. Es basiert auf einem zeitdiskreten linearen Systemmodel und einem ebenfalls zeitdiskreten linearen Messmodell. Beide Modelle beinhalten jeweils einen Fehlerterm, der die Störungen des Systems bzw. der Sensoren berücksichtigt. Das Kalmanfilter schätzt die Systemzustände so, dass die Kovarianz des Schätzfehlers minimiert wird.

Die Grundlage eines Kalmanfilters bilden wie bereits erwähnt ein zeitdiskretes lineares Systemmodell und ein zeitdiskretes lineares Messmodell. Diese Modelle beinhalten einen stochastischen Fehlerterm. Diese Terme sind notwendig, da jedes mathematische Modell eines realen Systems Fehler beinhaltet. Diese Fehler haben mehrere Ursachen. So betrachtet jedes Modell nur einen Teil des Systems, unwichtige Teile (zum Beispiel der Schlupf oder das Spiel in den Gelenken) werden ignoriert. Auch können die Konstanten des Systems (zum Beispiel Massen, Widerstände oder Kapazitäten) nicht exakt bestimmt werden und die Sensoren weisen Messfehler auf. Daher ist es besser, mit einem stochastischen Modell zu arbeiten, das diese Fehler berücksichtigt, statt mit einem deterministischen System, das die Fehler ignoriert. Von den Fehlern des System- und des Messmodells wird angenommen, dass diese durch ein weißes gaußverteiltes Rauschen beschrieben werden können.

Die Schätzung der internen Systemzustände basiert auf den Daten der Sensoren. Der Filteralgorithmus ist so gewählt, dass die Kovarianz des Schätzfehlers minimiert wird. Er verwendet alle über das System bekannten Fakten: das Systemmodell, das Messmodell, die Daten aller vorhandenen Sensoren und die Kovarianz der System- und Sensorfehler.

Ein Kalmanfilter arbeitet rekursiv. Das bedeutet, dass die Berechnung in jedem Zeitschritt nur die aktuellen Daten der Sensoren und Systemeingänge sowie die Systemzustände des letzten Zeitschritts benötigt. Dadurch müssen weder sämtliche Daten der vorhergegangenen Zeitschritte gespeichert werden, noch erhöht sich die Rechenzeit in jedem Zeitschritt.

Eine gute Einführung in die Theorie von Kalmanfiltern bieten die Arbeiten von Welch und Bishop (Welch und Bishop 2001) und Maybeck (Maybeck 1979).

4.1 Kalmanfilter

Das Kalmanfilter basiert auf einem zeitdiskreten linearen Modell, das das Verhalten des zu untersuchenden Systems beschreibt:

$$x_{k+1} = A_k x_k + B u_k + w_k$$

Dabei beschreibt der Zustandsvektor $x \in \mathbb{R}^n$ die internen Zustände des Systems, der Eingangsvektor $u \in \mathbb{R}^l$ die Eingänge des Systems und die Zufallsvariable $w \in \mathbb{R}^n$ das Systemrauschen. w ist normalverteilt mit dem Erwartungswert 0 und der Kovarianz Q. Die Systemmatrix $A \in \mathbb{R}^{n \times n}$ beschreibt den Übergang der Systemzustände von einem Zeitschritt in den nächsten, die Eingangsmatrix $B \in \mathbb{R}^{n \times l}$ den Einfluss der Eingänge auf die Systemzustände.

Das Messmodell bestimmt bei bekanntem Systemzustand x mit Hilfe der Messmatrix $H \in \mathbb{R}^{m \times n}$ und dem Messrauschen $v \in \mathbb{R}^m$ die Messwerte der Sensoren ($z \in \mathbb{R}^m$). Das Messrauschen v ist eine normalverteilte Zufallsvariable mit dem Erwartungswert 0 und der Kovarianz R.

$$z_k = H_k x_k + v_k$$

Ziel des Kalmanfilters ist es, eine Schätzung der internen Zustände $\hat{x}$ so zu berechnen, dass die Kovarianz P des Schätzfehlers e minimiert wird.

$$e_k = x_k - \hat{x}_k$$
$$P_k = E[e_k e_k^T]$$

Dazu wird eine erste Schätzung des Systemzustands (a priori Schätzung $\hat{x}^-$) aus der Systemgleichung berechnet.

$$x_k^- = A_{k-1} x_{k-1} + B u_{k-1} + w_{k-1}$$

Anschließend wird die Kalmanverstärkung K bestimmt, mit der eine verbesserte Schätzung des Systemzustands (a posteriori Schätzung $\hat{x}$) bestimmt werden kann. Die Verbesserung der Schätzung wird durch Verwendung der Residuen der Messungen[1] aus $\hat{x}^-$ so bestimmt, dass die Kovarianz des Schätzfehlers minimiert wird:

$$K_K = \frac{P_K^- H^T}{H_k P_k^- H_k^T + R_k}$$
$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H_k \hat{x}_k^-)$$

Bei der Berechnung wird die Kovarianzmatrix P in jeden neuen Zeitschritt übernommen und an den neuen Schritt angepasst:

$$P_k^- = A_{k-1} P_{k-1} A_{k-1}^T + Q_{k-1} \qquad \text{(Überführung der Kovarianzmatrix)}$$
$$P_k = (I - K_k H_k) P_k^- \qquad \text{(Korrektur der Kovarianzmatrix)}$$

[1] Das Residuum der Messung ist die Abweichung zwischen der Messung und der aufgrund der Schätzung des Systemzustands erwarteten Messung (Residuen: $z_k - H_k \hat{x}_k^-$)

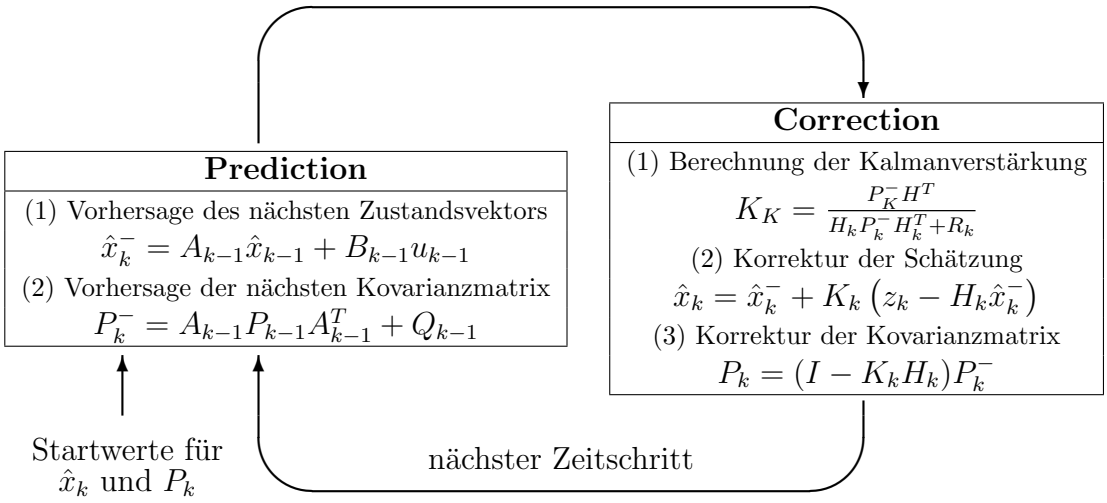


Abbildung 4.1: *Der Algorithmus des Kalmanfilters: Das Kalmanfilter teilt sich in die zwei Phasen Prediction und Correction. Die Prediction-Phase berechnet aus dem aktuellen Systemzustand und den Eingängen die a priori Schätzung $\hat{x}^-$ des Systemzustands. Die Correction-Phase berechnet mit Hilfe der Messungen den a posteriori Schätzwert $\hat{x}$.*

Diese Berechnungen werden zu den zwei Schritten *Prediction* und *Correction* zusammengefasst. Der Schritt *Prediction* schätzt den Zustand des Systems und die Kovarianzmatrix durch Verwendung des Systemmodells, der Schritt *Correction* korrigiert diese erste Schätzung durch Betrachtung der Messungen. Der Algorithmus des Kalmanfilters ist in Abbildung 4.1 dargestellt.

4.2 Erweiterter Kalmanfilter

Um die Einschränkung des Kalmanfilters auf lineare System- und Messmodelle zu umgehen, wurde das Erweiterte Kalmanfilter entworfen. Dieses basiert auf nichtlinearen System- und Messmodellen. Um das Kalmanfilter auch bei nichtlinearen Modellen verwenden zu können, ist es notwendig, die Modelle in jedem Abtastschritt um den Arbeitspunkt zu linearisieren.

Das Systemverhalten und das Verhalten der Sensoren wird bei einem Erweiterten Kalmanfilter jeweils durch ein nichtlineares Gleichungssystem beschrieben:

$$\begin{aligned} x_{k+1} &= f(x_k, u_k, w_k) \\ z_k &= h(x_k, v_k) \end{aligned}$$

Die Schätzung des neuen Systemzustands aus dem alten Systemzustand und der Messwerte $\hat{z}^-$ ist auch mit der nichtlinearen Systembeschreibung direkt möglich:

$$\begin{aligned} \hat{x}_k^- &= f\left(\hat{x}_{k-1}, u_{k-1}, 0\right) \\ \hat{z}^- &= h\left(\hat{x}_k^-, 0\right) \end{aligned}$$

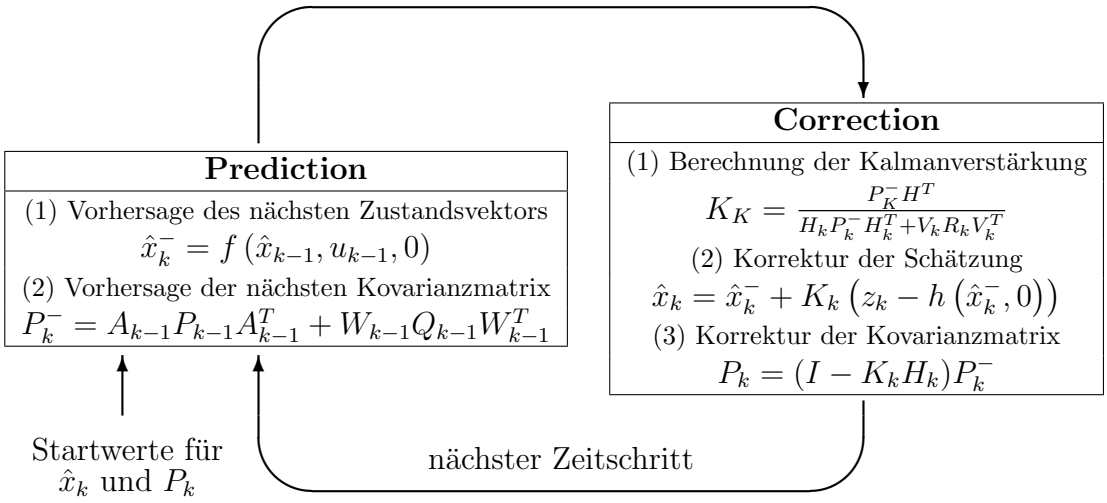


Abbildung 4.2: *Der Algorithmus des Erweiterten Kalmanfilters: Das Systemmodell ist im Gegensatz zum Kalmanfilter nichtlinear. Für die Berechnung der Kalmanverstärkung K und der Kovarianzmatrix P werden daher die Jacobimatrizen der Funktionen f und h verwendet.*

Zur Berechnung der Kalmanverstärkung K und der Kovarianzmatrix P ist es nötig, das System- und das Messmodell um die Arbeitspunkte $\hat{x}^-$ und $\hat{z}^-$ zu linearisieren:

$$x_k \approx \hat{x}_k^- + A_{k-1} \cdot (x_{k-1} - \hat{x}_{k-1}) + W_{k-1}w_{k-1}$$
$$z_k \approx \hat{z}_k^- + H_k \cdot \left(x_k - \hat{x}_k^-\right) + V_k v_k$$

mit:

$$A_k = \frac{\partial f_i}{\partial x_j}(\hat{x}_k, u_k, 0)$$
$$H_k = \frac{\partial h_i}{\partial x_j}(\hat{x}_k, 0)$$
$$V_k = \frac{\partial h_i}{\partial v_j}(\hat{x}_k, 0)$$
$$W_k = \frac{\partial f_i}{\partial w_j}(\hat{x}_k, u_k, 0)$$

Dabei ist $\hat{x}^- \in \mathbb{R}^n$ der geschätzte Systemzustand und $\hat{z}^-$ der geschätzte Messvektor. Die Jabobimatrix A beschreibt die Änderung des Systems bei den Eingängen u_k, die Jacobimatrix H den Zusammenhang zwischen den Systemzuständen und den Messungen. Die Jacobimatrizen V und W stellen den Einfluss der Störungen auf das System und die Messungen dar.

Mit Hilfe dieser Linearisierungen lässt sich sowohl die Kovarianzmatrix P in den neuen Zeitpunkt übertragen und korrigieren als auch die Kalmanverstärkung K

berechnen:

$$
\begin{aligned}
P_k^- &= A_{k-1} P_{k-1} A_{k_1}^T + W_{k-1} Q_{k-1} W_{k-1}^T \\
P_k &= (I - K_k H_k) P_k^- \\
K_K &= \frac{P_K^- H^T}{H_k P_k^- H_k^T + V_k R_k V_k^T}
\end{aligned}
$$

Somit lässt sich das Erweiterte Kalmanfilter ähnlich dem Kalmanfilter berechnen. Da sich die Jacobimatrizen jedoch von Zeitschritt zu Zeitschritt ändern, müssen diese jedes Mal erneut berechnet werden. Dies kann je nach Komplexität der Funktionen f und h sehr aufwendig sein. Der gesamte Algorithmus des Erweiterten Kalmanfilters ist in Abbildung 4.2 dargestellt.

4.3 Bisher verwendete Kalmanfilter

Bisher wurden verschiedene Kalmanfilter entworfen, die die Daten der Odometrie, des Gyroskops und der Sonarsensoren verarbeiten. Diese Kalmanfilter wurden teilweise direkt auf dem Roboter in C und C++ implementiert. Teilweise wurden sie für einen PC in Matlab programmiert. Die Kalmanfilter in C und C++ wurden online auf dem Roboter verwendet, die Matlab-Kalmanfilter wurden offline mit aufgezeichneten Daten oder mit einer Simulation des Roboters verwendet.

Diese Kalmanfilter wurden in den Arbeiten (Rudolph et al. 2002*a*), (Mikhailova 2002), (Rudolph 2003), (Siegel 2001) und (Seidenspinner 2001/2002) beschrieben.

Teil II

Programmbeschreibung

Kapitel 5

Konzept des Programms

In diesem Kapitel wird das Konzept des Programms dargestellt. Zuerst werden die Anforderungen an das Programm erläutert, da diese die Grundlage des Designprozesses darstellen. Anschließend werden die Aufteilung des Programms in die einzelnen Module sowie der Datenfluss durch diese Module beschrieben. Danach werden die wesentlichen Module kurz umrissen.

5.1 Anforderungen

An das Synchronisationsprogramm wurden eine Reihe von Anforderungen gestellt, diese Anforderungen sind im Folgenden beschrieben:

Sensordaten einlesen: Das Programm soll in der Lage sein, die Sensordaten aller vorhandenen Sensoren (Odometrie, Gyroskop, Wanderkennung, Bodenkorrelator, LOPS) einzulesen und zu verarbeiten.

Daten synchronisieren: Die Daten aller angeschlossenen Sensoren sollen synchronisiert werden, damit sie vom Kalmanfilter verarbeitet werden können.

Kalmanfilter aufrufen: Nach der Synchronisation der Daten soll der Kalmanfilter aufgerufen werden, damit dieser aus den Daten die neue Position des Roboters schätzt.

Daten ausgeben: Es soll möglich sein die Daten aller Sensoren sowie die berechneten Positionen auf dem Bildschirm auszugeben.

Daten in Datei loggen: Die Daten aller Sensoren sollen in ihrer Rohform und in der synchronisierten Form gespeichert werden können. Außerdem soll es möglich sein, die berechneten Positionen zu speichern.

Fehlermeldungen loggen: Neben den Sensordaten sollen auch die Fehler in einer Datei gespeichert werden, so dass später festgestellt werden kann, ob während der Messfahrt Fehler aufgetreten sind.

Teilweise Offline-Bearbeitung: Die aufgezeichneten Logdateien sollen nachträglich verarbeitet werden können. Dazu muss das Programm die Dateien einlesen

und die eingelesenen Daten so verarbeiten, als ob sie direkt von den Sensoren eingelesen worden wären. Damit können auf denselben Daten verschiedene Kalmanfilter und / oder Synchronisationsalgorithmen getestet werden.

Leicht konfigurierbar: Das Programm soll leicht konfigurierbar sein. Die Konfiguration soll über eine möglichst selbsterklärende Textdatei erfolgen. In dieser Datei sollen unter anderem folgende Parameter einstellbar sein:

- die verwendeten Sensoren
- das Logging (welche Daten geloggt werden)
- die verwendeten Dateien (Namen der Logdatei)
- wann die Synchronisation startet (welcher Sensor wie oft Daten angeliefert haben muss)
- ob synchronisiert wird
- ob der Kalmanfilter verwendet wird
- welche Daten der Kalmanfilter verwendet
- die Schnittstellen, an denen die Sensoren angeschlossen sind

Modularer Aufbau: Das Programm soll möglichst modular aufgebaut sein, um eine spätere Erweiterung und / oder Verbesserung zu ermöglichen.

Dokumentation des Programms: Alle Funktionen und Datenschnittstellen sollen dokumentiert werden, so dass eine spätere Erweiterung erleichtert wird.

Leichte Erweiterbarkeit um neue Sensoren: Beim Design des Programms soll darauf geachtet werden, dass neue Sensoren leicht integriert werden können.

5.2 Aufbau

Um die Anforderungen an das Programm erfüllen zu können, wurde es in verschiedene Module aufgeteilt. Durch den modularen Aufbau des Programms können einzelne Programmteile ersetzt werden. So ist es möglich, einen anderen Kalmanfilter zu testen, oder auch neue Programmmodule wie etwa einen neuen Sensor zu ergänzen. Aufgaben, die nicht zum eigentlichen Programmablauf gehören, wurden in eigene Hilfsprogramme ausgegliedert, damit das Synchronisationsprogramm möglichst übersichtlich bleibt.

Dabei können als wesentliche Programmmodule die Sensormodule, die Synchronisationsmodule, der Kalmanfilter, das Datenlogmodul, das Fehlermodul, das Fehlerlogmodul, das Robotermodul und die Benutzerschnittstelle genannt werden. Bei der Benutzerschnittstelle und den Sensoren kann ohne Neukompilation des Programms ausgewählt werden, welche Module verwendet werden. Die wesentlichen Module und die wichtigsten Datenströme zwischen diesen Modulen sind in Abbildung 5.1 dargestellt.

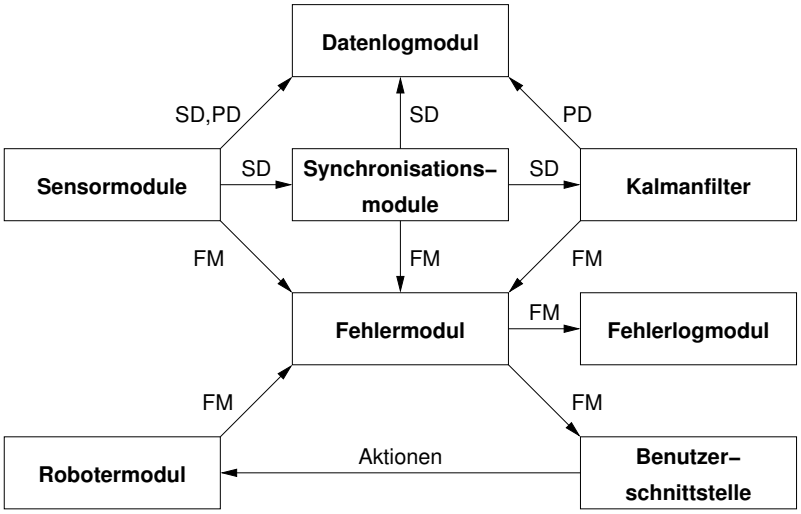


Abbildung 5.1: *Die wesentlichen Module des Synchronisationsprogramms: Das Synchronisationsprogramm besteht aus mehreren Modulen die Daten untereinander Austauschen. Der Datenfluss der Sensordaten ist in der Abbildung 5.2 detailliert dargestellt. (SD = Sensordaten, PD = Positionsdaten, FM = Fehlermeldungen).*

5.3 Datenfluss

Einer der wichtigsten Designpunkte ist die Festlegung des Datenflusses von den Sensoren bis zum Kalmanfilter. Dieser Datenfluss wird von mehreren Datenpuffern unterteilt.

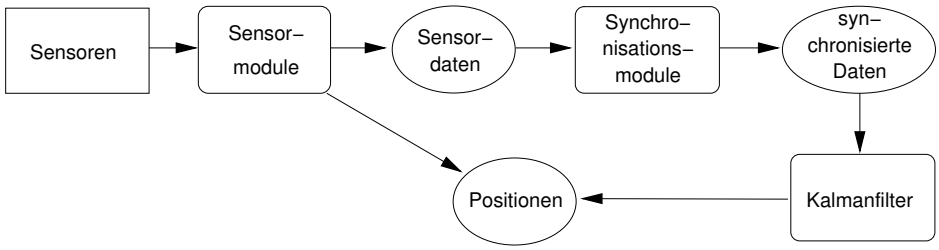
Die Sensormodule lesen die Daten von den Sensoren und verarbeiten sie. Außerdem berechnen sie, wenn möglich, aus den Daten des Sensors die Position des Roboters. Die Sensordaten werden im Sensordatenpuffer gespeichert, die aktuelle Position im Positionsspeicher.

Die Rohdaten der Sensoren werden von den Synchronisationsmodulen weiterverarbeitet. Diese synchronisieren jeweils die Daten eines Sensors und schreiben diese Daten in den Puffer für synchronisierte Daten.

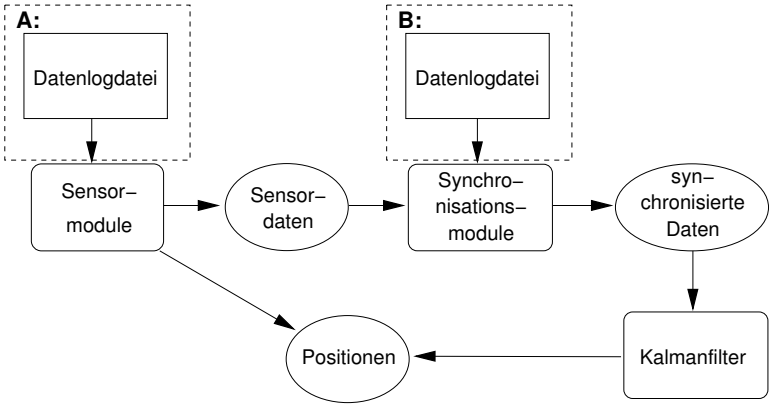
Haben die Synchronisationsmodule ihre Arbeit beendet, wird der Kalmanfilter aufgerufen. Dieser greift auf die synchronisierten Daten zu und berechnet daraus die aktuelle Position des Roboters.

Um den Programmablauf kontrollieren zu können und um die Messfahrt auch später noch bearbeiten zu können, werden alle Daten, die in einen der Datenspeicher geschrieben werden auch in einer Logdatei gespeichert. Diese Daten können später vom Synchronisationsprogramm eingelesen werden, um einen neuen Kalmanfilter zu testen, oder mit Hilfe des Programms Logfileanalyser analysiert werden.

An den Sensordatenpuffer werden besondere Anforderungen gestellt. Die Sensormodule laufen asynchron und in unterschiedlichen Taktraten. Es muss deshalb sichergestellt sein, dass alle Synchronisationsmodule die Daten desselben Zeitabschnitts verarbeiten. Das heißt, es darf nicht vorkommen, dass ein Sensormodul Daten in den in den Sensordatenpuffer schreibt, während die Synchronisationsmo-



(a) *Datenfluss im Online-Modus: Die Daten werden von den Sensoren gelesen.*



(b) *Datenfluss im Offline-Modus: Es werden entweder die Rohdaten (A) oder die synchronisierten Daten (B) aus der Logdatei gelesen*

Abbildung 5.2: *Der Datenfluss im Synchronisationsprogramm: Die Boxen mit abgerundeten Ecken stellen Programmfunktionen dar, die Ellipsen Datenspeicher. Die Daten, die in den Datenspeicher gespeichert werden, können auch in eine Logdatei geschrieben werden.*

dule auf diesen Puffer zugreifen. Gleichzeitig dürfen die Sensormodule jedoch nicht blockiert werden, da ansonsten der Verlust von Sensordaten droht.

Deshalb wurde der Sensordatenpuffer als Shadowbuffer ausgelegt. Das heißt, dass der Sensordatenpuffer doppelt vorhanden ist. Die Sensormodule schreiben in einen der beiden Puffer. Wird die Synchronisation gestartet, wird zwischen den beiden Puffern umgeschaltet, die Sensormodule schreiben in den zweiten Sensordatenpuffer. Dadurch werden weder die Synchronisationsmodule, die auf dem ersten Puffer arbeiten, gestört noch die Sensormodule blockiert. Bei der nächsten Synchronisation werden die Sensormodule auf den ersten Datenpuffer zurückgeschaltet und die Synchronisationsmodule arbeiten auf dem zweiten Puffer. Danach beginnt dieser Ablauf von vorn.

Die anderen Datenspeicher sind als einfache Felder ausgelegt, besondere Vorsichtsmaßnahmen sind hier nicht nötig. Der Puffer mit den synchronisierten Daten wird nur von den Sensormodulen und dem Kalmanfilter verwendet und diese Modu-

le greifen nacheinander auf diesen Puffer zu. Im Positionsspeicher hat jedes Modul ein eigenes Feld, so dass es hier ebenfalls nicht zu Beeinflussungen kommen kann.

5.4 Sensormodule

Die Sensoren unterscheiden sich nicht nur durch die Informationen, die sie liefern, sondern auch durch die Art der Anbindung an das Synchronisationsprogramm. Wie in Abbildung 3.1 (Seite 8) zu sehen ist, sind die Sensoren entweder an den Mikrocontroller, den Onboardrechner oder einen externen Rechner angeschlossen. Dementsprechend unterschiedlich müssen auch die Sensormodule sein.

Das Synchronisationsprogramm legt nur die Schnittstellen zwischen den Sensormodulen und den anderen Programmmodulen fest, wie die einzelnen Sensormodule die Sensoren auslesen, ist nicht vorgegeben. Die Sensormodule laufen entweder als eigener Thread, als Roboter-Task oder als Paketmodul. Threads arbeiten nebenläufig, Roboter-Tasks werden von ARIA in jedem Zyklus (alle 100 ms) einmal aufgerufen, Paketmodule werden immer dann aufgerufen, wenn ein neues SIP vom Roboter eingetroffen ist.

Um das Synchronisationsprogramm flexibel zu halten, werden beim Programmstart nur die Sensormodule eingebunden, die in der Konfigurationsdatei aufgelistet sind. Dadurch kann das Programm an die Messfahrt angepasst werden.

5.5 Synchronisation

Die Sensorsynchronisationsmodule führen die Synchronisation der Daten durch. Für jeden Sensor existiert genau ein solches Modul. Es nimmt die Rohdaten des Sensors entgegen und synchronisiert diese.

Ein eigenes Modul überwacht den Sensordatenpuffer und startet die Synchronisation, wenn die in der Konfigurationsdatei eingestellte Bedingung erfüllt ist. Die Bedingung gibt an, welcher Sensor wie viele Daten geliefert haben muss. So ist es zum Beispiel möglich, die Synchronisation jedes Mal anzustoßen, wenn der Odometriesensor Daten geliefert hat.

Das Synchronisationsmodul schaltet den Sensordatenpuffer um, ruft die Sensorsynchronisationsmodule der einzelnen Sensoren auf und startet anschließend den Kalmanfilter.

5.6 Offline-Modus

Im Offline-Modus werden die Daten nicht von den Sensoren, sondern aus einer Logdatei ausgelesen. In diesem Modus wird die Roboterschnittstelle nicht initialisiert und die Sensormodule werden in einem speziellen Modus gestartet, in dem sie die Sensoren nicht initialisieren. Daher kann das Synchronisationsprogramm im Offline-Modus nicht nur auf dem Onboardrechner des Roboters ausgeführt werden, sondern auf jedem Rechner mit installierter ARIA-Bibliothek.

Nach der Initialisierung werden die gespeicherten Datensätze aus der Logdatei ausgelesen. Je nach Einstellung des Synchronisationsprogramms werden entweder die Datensätze mit den Sensorrohdaten von den Sensormodulen oder die Datensätze mit den synchronisierten Daten von den Synchronisationsmodulen verarbeitet. Die Module bearbeiten die Datensätze so, als kämen sie von den realen Sensoren.

Im Offline-Modus kann dieselbe Messfahrt mit verschiedenen Kalmanfiltern, beziehungsweise mit einem unterschiedlich konfigurierten Kalmanfilter mehrmals nachvollzogen werden. Außerdem ermöglicht dieses Verfahren den Einsatz von Algorithmen, die mehr Rechenleistung benötigen, als der Onboardrechner des Roboters bietet. Die Rechenleistung des Onboardrechners ist relativ beschränkt.

Kapitel 6

Implementierung des Programms

Das Programm setzt auf das Softwarepaket ARIA (siehe Abschnitt C.1) von ActivMedia auf. ARIA ist eine objektorientierte Klassenbibliothek, die es ermöglicht auf die Schnittstelle zum Roboter zuzugreifen.

Die Umsetzung des Programmkonzepts erfolgte in der objektorientierten Programmiersprache C++. Durch den objektorientierten Entwurf ist es sehr leicht möglich, das Programm in verschiedene Module (Klassen) aufzuteilen, die austauschbar sind.

Beim Entwurf wurde darauf geachtet, Aufgaben, die an mehreren Stellen des Programms gelöst werden müssen, in einem eigenen Modul zu bearbeiten. Durch dieses Vorgehen wirkt sich eine Änderung an einer Stelle des Programms auf alle betroffenen Stellen aus. Es entstanden so mehrere Hilfsmodule, die an vielen Stellen des Codes verwendet werden.

In diesem Kapitel werden die Grundzüge der Implementierung dargestellt. Da das Synchronisationsprogramm aus mehr als 50 Klassen besteht, ist es nicht möglich alle Klassen detailliert zu beschreiben. Die Schnittstellen der Klassen sind jedoch ausführlich in den Programmdokumentationen des Synchronisationsprogramms und der Hilfsprogramme (siehe Anhang A) beschrieben.

Bei Fragen zu Details über C++ (zum Beispiel Vererbung) sei auf das Standardwerk „Die C++ Programmiersprache“ (Stroustrup 2000) verwiesen, bei Fragen zur Linuxprogrammierung (zum Beispiel Schnittstellen, Threads und Sockets) auf die Werke „Advanced Linux Programming“ (Mitchel et al. 2001) und „The Linux Programmer's Guide“ (Goldt et al. 1995). Die ARIA-Bibliothek besitzt ebenfalls eine ausführliche Dokumentation (siehe Abschnitt C.1).

6.1 Struktur

Das Programm besteht aus mehreren Modulen. Dabei stellen die Sensormodule, die Synchronisationsmodule sowie der Kalmanfilter die Hauptmodule dar. Diese erledigen die Verarbeitung der Sensordaten.

Neben diesen Programmmodulen gibt es noch mehrere Hilfsmodule. So gibt es zum Beispiel ein Datenlogmodul, ein Fehlermodul, ein Fehlerlogmodul, ein Robotermodul, ein TCP-Socket-Modul, ein Konfigurationsmodul, ein Mathematikmodul

sowie ein Modul für die Benutzerschnittstelle. Das Datenlogmodul speichert die Sensorrohdaten, die synchronisierten Daten und die Positionsdaten. Das Fehlermodul und das Fehlerlogmodul sorgen für eine einheitliche Verarbeitung und Speicherung aller Fehlermeldungen und Mitteilungen. Das Robotermodul steuert den Roboter, es wandelt die Fahrbefehle des Benutzers in Roboterbefehle um. Zur Datenübertragung zwischen zwei Rechnern gibt es das TCP-Socket-Modul, das eine TCP-Verbindung aufbaut. Das Mathematikmodul stellt mehrere häufig verwendete mathematische Funktionen, die nicht in der Standardmathematikbibliothek enthalten sind, bereit.

6.2 Sensormodule

Die Schnittstelle aller Sensormodule ist in der Klasse *SensorModule* festgelegt. Alle Sensormodule besitzen einen Konstruktor und einen Destruktor sowie die Methoden *startSensor()*, *stopSensor()*, *startLogging()* und *readDataFromFile()*.

Der Konstruktor initialisiert den Sensor und das Sensormodul, der Destruktor beendet beide. Die Methode *startSensor()* startet die Sensordatenverarbeitung, die Daten werden jedoch erst nach dem Aufruf von *startLogging()* in den Sensordatenpuffer und die Logdatei geschrieben. Mit der Methode *stopSensor()* wird die Sensordatenverarbeitung angehalten. Zum Verarbeiten der Logdateien im Offlinemodus besitzen die Sensormodule die Methode *readDataFromFile()*, die einen Dateneintrag aus der Datenlogdatei entgegennimmt und ihn verarbeitet.

Jedes Sensormodul, das im Synchronisationsprogramm verwendet wird, muss direkt oder indirekt von der Klasse *SensorModule* erben.

6.2.1 Generische Basisklassen

Von der Klasse *SensorModule* wurden mehrere Klassen abgeleitet, die jeweils eine Betriebsart des Sensors unterstützen und diese Betriebsart so weit wie möglich kapseln. Es gibt Klassen für asynchrone Sensormodule, für synchrone Module sowie für Paketmodule. Asynchrone Sensormodule werden als nebenläufiger Thread ausgeführt, synchrone Module werden in jedem Roboterzyklus von der ARIA-Bibliothek aufgerufen und die Paketmodule werden bei jedem Datenpaket, das für sie bestimmt ist, gestartet.

Die Klasse *ASynchronSensorModule* ist die Basisklasse aller Sensormodule, die einen eigenen Thread verwenden. Sie ist von der Klasse *SensorModule* und der ARIA-Klasse *ArASyncTask* abgeleitet. Die gesamte Thread-Verwaltung wird von der Klasse *ArASyncTask* übernommen, das Sensormodul muss lediglich die Methode *runThread()* implementieren. Diese Methode wird beim Start der Messung als eigener Thread gestartet.

Synchrone Sensormodule, die in jedem Roboterzyklus gestartet werden, werden von der Klasse *SynchronousSensorModule* unterstützt. Diese Basisklasse registriert einen neuen Sensortask (die Methode *sensorTask()*) bei der Roboterklasse der ARIA-Bibliothek. Diese Methode wird nach dem Start der Sensoren in jedem Roboterzyklus aufgerufen.

Die Klasse *PacketHandlerSensorModule* übernimmt die Bearbeitung von Datenpaketen des Mikrocontrollers (SIPs). Diese Basisklasse übernimmt die Zusammenarbeit mit ARIA. Die Methode *packetHandler()* wird von der Klasse jedes Mal aufgerufen, wenn der Roboter ein SIP, das für diese Modul bestimmt ist, gesendet hat. Diese SIPs werden anhand der Paketidentifikation erkannt.

6.2.2 Odometrie

Das Odometriesensormodul arbeitet als Paketmodul. Es verarbeitet die Enkoderdaten, die der Mikrocontroller alle 100 ms schickt. Aus den absoluten Enkoderständen werden für beide Räder die Änderungen seit dem letzten Aufruf des Moduls (s_l für das linke Rad und s_r für das rechte) berechnet. Aus diesen Werten wird die Positionsänderung des Roboters seit dem letzten Abtastschritt bestimmt. Dies geschieht je nach Einstellung in der Konfigurationsdatei auf eine von zwei Weisen.

Beide Berechnungsalgorithmen benutzen das folgende Zustandsraummodell für die Positionsänderung (zu den Bezeichnern siehe Abbildung 2.2):

$$\begin{bmatrix} x \\ y \\ \psi \end{bmatrix}_{t+1} = \begin{bmatrix} x \\ y \\ \psi \end{bmatrix}_{t} + \begin{bmatrix} cos(\psi_t + \frac{1}{2} \cdot \omega_{Odometrie}) \cdot s \\ sin(\psi_t + \frac{1}{2} \cdot \omega_{Odometrie}) \cdot s \\ \omega_{Odometrie} \cdot T_{Odometrie} \end{bmatrix}$$

Die Rotation des Roboters (ω) bestimmen beide Varianten wie in Abschnitt 2.2 beschrieben zu:

$$\omega_{Odometrie} = \frac{v_r - v_l}{2 \cdot l} * T_{Odometrie} = \frac{s_r - s_l}{2 \cdot l} * SF_{Odometrie}$$

Der Unterschied der beiden Varianten liegt in der Bestimmung der zurückgelegten Strecke s. Die erste Variante, die aus den Odometriefunktionen von Jochen Seidenspinner (Seidenspinner 2001/2002) entnommen wurde, nimmt eine gerade Bewegung des Roboters an und berechnet die Strecke daher aus der Zeit und der durchschnittlichen Radgeschwindigkeit:

$$s = \frac{1}{2} \cdot (v_l + v_r) \cdot T_{Odometrie} = \frac{1}{2} \cdot (s_l + s_r) \cdot SF_{Odometrie}$$

Die zweite Variante nimmt eine Bewegung des Roboters auf einer Kreisbahn an und bestimmt die Strecke s wie in Abschnitt 2.2 beschrieben:

$$s = \begin{cases} v_r \cdot T_{Odometrie} = s_r \cdot SF_{Odometrie} & \text{falls} \quad v_l = v_r \\ 2 \cdot \sin\left(\frac{1}{2} \cdot \omega_{Odometrie}\right) \cdot \frac{s_r + s_l}{s_r - s_l} \cdot l & \text{sonst} \end{cases}$$

Die Position speichert das Odometrie-Modul im Positionsspeicher, die Enkoderinkremente im Sensordaten-Puffer. Beide Werte werden auch in die Logdatei geschrieben.

6.2.3 Gyroskop

Das Gyroskopmodul ist ein asynchrones Sensormodul, es läuft in einem eigenen Thread. Im Konstruktor des Gyroskopmoduls wird das Gyroskop initialisiert. Dazu wird aus der Konfigurationsdatei die Schnittstelle, an der das Gyroskop angeschlossen ist, ausgelesen und geöffnet. Anschließend muss das Gyroskop kalibriert werden, da sich der Nullpunkt des Gyroskops in Abhängigkeit von der Temperatur verschiebt. Dazu werden mehrere Werte vom Gyroskop ausgelesen und aus diesen der Mittelwert gebildet. Dieser wird als Nullpunkt ($NP_{Gyroskop}$) für alle weiteren Messungen verwendet. Der Skalenfaktor des Gyroskops ($SF_{Gyroskop}$) ist konstant.

Wenn die Messung gestartet wird, erzeugt das Gyroskop-Modul einen neuen Thread. Dieser fragt die Schnittstelle mit dem Gyroskop regelmäßig ab und berechnet aus den Rohwerten das Winkelinkrement (ω):

$$\omega_{Gyroskop} = (Rohdaten - NP_{Gyroskop}) * SF_{Gyroskop} * T_{Gyroskop}$$

Das Winkelinkrement wird im Sensordaten-Puffer gespeichert. Die Orientierung des Roboters, die durch Aufsummieren der Winkelinkremente bestimmt wird, wird im Positionsspeicher gespeichert. Beide Werte werden auch in die Datenlogdatei geschrieben. Die ebenfalls vom Gyroskop gemessene Temperatur wird nicht verwendet.

Die Funktionen, die das Gyroskop abfragen und die Daten verarbeiten, sind der Diplomarbeit von Andreas Siegel entnommen und wurden an die Programmarchitektur angepasst. Details über das verwendete Gyroskop und die Berechnung des Winkelinkrements sind in dieser Diplomarbeit nachzulesen (Siegel 2001).

6.2.4 Wanderkennung

Die Wanderkennung verwendet die Sonarsensoren. Das Ergebnis der Sonarsensoren liefert der Mikrocontroller alle 100 ms in den normalen SIPs. Daher arbeitet dieses Modul als synchroner ARIA-Task. Die Wanderkennung und Zuordnung läuft in fünf Schritten ab:

1. Auslesen der Sonarsensoren
2. Transformation der Hindernispunkte ins Weltkoordinatensystem
3. Erkennen von Wänden mit den Sensoren eines Arrays (oben und unten)
   (a) Erkennung mit den seitlichen Sensoren (Sensoren 0,7,8 und 15)
   (b) Erkennung mit den vorderen Sensoren
4. Vergleichen der erkannten Wände des oberen und des unteren Arrays
5. Zuordnen der Wände zu den bekannten Laborwänden und Positionsschätzung

Die Schritte zwei bis fünf werden nur durchgeführt, wenn das Sonarsensormodul eine Wanderkennung durchführen soll. Schritt fünf wird nur dann durchgeführt, wenn die Position berechnet werden soll. Bei der Wanderkennung werden mehrere

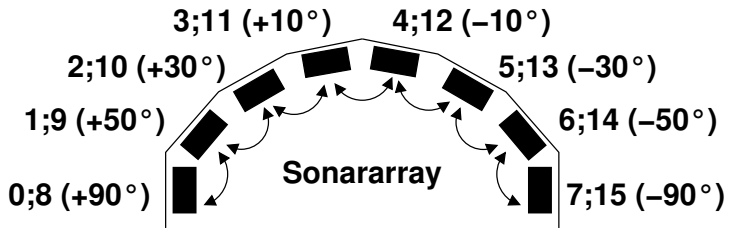


Abbildung 6.1: *Die Sonarsensoren in einem Array: Jedes der Sonararrays besitzt 8 Sonarsensoren. Die Sonarsensoren sind durch eine Nummer gekennzeichnet. Die Nummern zwischen 0 und 7 kennzeichnen die Sensoren im unteren Array, die Nummern zwischen 8 und 15 die Sensoren im oberen Array. Die bei der Wanderkennung verwendeten Sensorpaare sind mit Pfeilen dargestellt.*

Parameter (Anzahl der verwendeten Sonarwerte, Gütekriterien etc.) verwendet, diese Parameter werden vom Sonarsensormodul aus der Konfigurationsdatei ausgelesen, sie können daher leicht angepasst werden. Die Parameter in der Konfigurationsdatei sind in Anhang B beschrieben.

Das zur Wanderkennung verwendete Verfahren orientiert sich an dem von Richard Woller beschriebenen Verfahren (Woller 2001).

Auslesen der Sonarsensoren

Die Sensortask-Funktion überprüft, welche Sonarwerte neu sind (jeder Sonarsensor wird nur alle 320 ms abgefragt, siehe Abschnitt 3.4), und fragt die gemessene Entfernung zu Hindernis sowie die Koordinaten des Hindernisses im Roboterkoordinatensystem ($Hindernis_{lokal}$) ab. Diese Informationen werden in den Sensordatenpuffer und die Datenlogdatei geschrieben.

Transformation der Hindernispunkte ins Weltkoordinatensystem

Soll das Sensormodul auch Wände erkennen, was in der Konfigurationsdatei eingestellt werden kann, werden die Positionen der neuen Hindernispunkte aus dem Roboterkoordinatensystem mit Hilfe der vom Kalmanfilter berechneten Roboterposition ins Weltkoordinatensystem transformiert:

$$Hindernis_{global} = RP + \begin{pmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & \cos(\psi) \end{pmatrix} \cdot Hindernis_{lokal}$$

Die letzten n Hinderniskoordinaten werden nach Sensoren getrennt in Listen gespeichert. Dabei werden Messwerte, deren Entfernung zu groß ist, nicht beachtet, da diese Werte meist auf Messfehler zurückzuführen sind. Die gespeicherten Daten werden zur Wanderkennung genutzt.

Wanderkennung mit den Sensoren eines Arrays

Bei der Wanderkennung werden die Sensoren in zwei Gruppen unterteilt: die Sensoren die zur Seite gerichtet sind (Sensoren 0,7,8 und 15 in Abbildung 6.1) und die

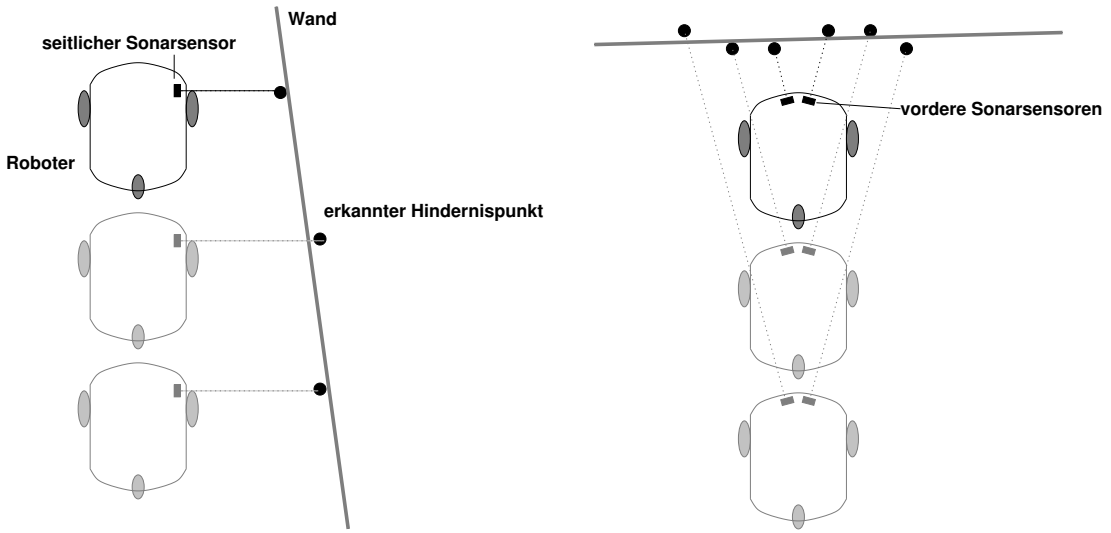


Abbildung 6.2: *Wanderkennung mit den Sonarsensoren: Bei der Wanderkennung mit den seitlichen Sensoren (links) reichen die Daten eines Sensors, da die erkannten Hindernispunkte durch die Bewegung des Roboters weit genug versetzt sind. Bei der Erkennung mit den vorderen Sensoren (rechts) reichen die Daten eines Sensors nicht aus, da die erkannten Punkte zu dicht beisammen liegen. Daher werden die Daten von zwei Sensoren gemeinsam betrachtet.*

Sensoren, die nach vorne gerichtet sind.

Erkennen die *seitlichen Sonarsensoren* ein Hindernis, so fährt der Roboter an dem Hindernis vorbei. Die nacheinander erkannten Hindernispunkte eines seitlichen Sonarsensors liegen daher in einigem Abstand (siehe Abbildung 6.2, linke Seite). So kann mit den Hindernispunkten eines einzigen Sensors bereits eine Wand erkannt werden. Dazu wird durch die letzten n Punkte eine Ausgleichsgerade in der Form $y = m \cdot x + b$ im Weltkoordinatensystem gelegt. Die beiden Parameter m und b lassen sich mit Hilfe der linearen Regression berechnen (Bronstein et al. 2001):

$$m = \frac{n \cdot \sum_{i=1}^{n} x_i \cdot y_i - \sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} y_i}{n \cdot \sum_{i=1}^{n} {x_i}^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

$$b = \frac{\sum_{i=1}^{n} {x_i}^2 \cdot \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} x_i \cdot y_i}{n \cdot \sum_{i=1}^{n} {x_i}^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

Anschließend muss getestet werden, ob die Ausgleichsgerade tatsächlich eine Wand beschreibt. Dazu wird die Standardabweichung der Abstände der Hindernispunkte von der Ausgleichsgerade bestimmt. Ist die Standardabweichung kleiner als der Grenzwert und ist der Abstand der Hindernispunkte zur Wand kleiner als die Standardabweichung, so wird davon ausgegangen, dass die Ausgleichsgerade eine Wand beschreibt.

Danach wird versucht, die Wanderkennung mit mehr als n Punkten durchzuführen. Dazu wird die Berechnung solange mit einem Punkt mehr versucht, bis die oben genannten Kriterien nicht mehr erfüllt werden oder alle gespeicherten Punkte

des betrachteten Sensors verwendet wurden. Die letzte Ausgleichsgerade, bei der alle Kriterien erfüllt wurden, wird als erkannte Wand verwendet. Sie wird in die Hessesche Normalform transformiert und gespeichert.

Bei den *vorderen Sonarsensoren* reichen die Daten eines Sensors nicht aus. Da die Sensoren in Fahrtrichtung des Roboters messen, liegen alle erkannten Hindernispunkte dicht beisammen (siehe Abbildung 6.2, rechte Seite). Daher reichen schon kleinste Messfehler aus, um die Ausrichtung der Wand falsch zu bestimmen. Deshalb werden in diesem Fall die Daten von zwei Sensoren verwendet. Dabei werden auch die seitlichen Sonarsensoren verwendet. Die Sensorpaare sind in Abbildung 6.1 abgebildet.

Durch die n aktuellsten Punktepaare $(x_{i,1}, y_{i,1})$ und $(x_{i,2}, y_{i,2})$ wird jeweils eine Gerade in der Hesseschen Normalform $(x \cdot \cos\alpha_i + y \cdot \sin\alpha_i - d_i = 0)$ im Weltkoordinatensystem gelegt:

$$\alpha_i = \begin{cases} \text{atan2}(-B, -A) & \text{falls} \quad C < 0 \\ \text{atan2}(B, A) & \text{sonst} \end{cases}$$
$$d_i = \mu \cdot C$$

mit:

$$\mu = \begin{cases} +\frac{1}{\sqrt{A^2+B^2}} & \text{falls} \quad C < 0 \\ -\frac{1}{\sqrt{A^2+B^2}} & \text{sonst} \end{cases}$$
$$A = y_{i,1} - y_{i,2}$$
$$B = x_{i,2} - x_{i,1}$$
$$C = y_{i,1} \cdot B + x_{i,1} \cdot A$$

Die Ausgleichsgerade wird durch Bildung der arithmetischen Mittel bestimmt:

$$\alpha = \frac{1}{n} \sum_i \alpha_i$$
$$d = \frac{1}{n} \sum_i d_i$$

Um zu entscheiden, ob die Ausgleichsgerade eine Wand beschreibt, wird ein Test durchgeführt. Dazu werden die Standardabweichungen der Parameter d_i (σ_d)und α_i (σ_α) bestimmt. Sie müssen unter einer Schranke liegen. Ist diese Bedingung erfüllt, so wird analog zu den Seitensensoren sukzessive versucht, ältere Punktepaare hinzuzunehmen, um die Wanderkennung zu verbessern. Es wird abgebrochen, sobald eine der Bedingungen nicht mehr erfüllt ist oder alle gespeicherten Punktepaare benutzt wurden.

Vergleich der erkannten Wände des oberen und des unteren Arrays

Der Roboter verfügt über zwei Sonararrays, eines im unteren Teil und eines im oberen Teil. Dabei befinden sich immer zwei Sonarsensoren übereinander. Gehören die von einem Sonarsensor bzw. einem Sonarsensorpaar erkannten Hindernispunkte zu einer Wand, so muss auch der zugehörige Sensor bzw. die zugehörigen Sensoren im anderen Array eine Wand erkennen. Um die Verwechslung kleinerer Hindernisse mit Wänden zu vermeiden, wird daher überprüft, ob die Wände in beiden Arrays erkannt wurden.

So wird bei jeder erkannten Wand kontrolliert, ob der zugehörige Sensor des anderen Arrays, bzw. die zugehörigen Sensoren, ebenfalls eine Wand erkannt haben. Ist dies der Fall und die beiden Wände stimmen überein (d und α sind nahezu identisch), werden die erkannten Wände zusammengefasst (Mittelwertbildung der beiden Parameter). Stimmen die Wände nicht überein oder wurde nur in einem Array eine Wand erkannt, so werden die Daten der Wanderkennung verworfen.

Zuordnung der erkannten Wände zu den bekannten Laborwänden

Nachdem mit den Sensoren Wände erkannt wurden, muss versucht werden diese Wände einer bekannten Wand des Labors zuzuordnen. Dazu sind die Positionen aller Laborwände in einer Landkarte vermerkt. Diese Landkarte wird aus der Wanddatei (siehe B.1.2) ausgelesen. In der Landkarte sind die Positionen der Wände in der Hesseschen Normalform angegeben. Weiterhin ist für jede Wand das Gebiet vermerkt, in dem sich der Roboter befinden muss, um die Wand erkennen zu können.

Hat der Roboter eine Wand erkannt, werden die Richtung der Wand α und ihr Abstand zum Nullpunkt d mit den Daten der bekannten Wände, die der Roboter von seiner Position aus erkennen kann, verglichen. Bei der Feststellung der Übereinstimmung werden relativ große Fehler zugelassen, da die auftretenden Abweichungen auf Navigationsfehler zurückzuführen sind. Denn wenn die Grenzwerte zu klein gewählt werden, kann die Roboterposition nur dann korrigiert werden, wenn der Roboter seine Position schon sehr gut bestimmt hat. Ziel ist es jedoch, ungenaue Positionsschätzungen zu korrigieren. Wird eine Übereinstimmung festgestellt, wird diese in der Datenstruktur der erkannten Wand vermerkt, so dass diese Informationen auch dem Kalmanfilter zur Verfügung stehen.

Da alle Wände im Labor parallel zu einer der beiden Achsen des Weltkoordinatensystems sind, kann mit Hilfe einer erkannten und zugeordneten Wand eine Koordinate der Roboterposition und die Orientierung des Roboters bestimmt werden.

Der Orientierungsfehler entspricht der Differenz zwischen dem Winkel der erkannten Wand α_{erk} und dem der bekannten Wand α_{bek}. Bei einer zur X-Achse parallelen Wand entspricht der Fehler in der Y-Koordinate der Roboterposition der Differenz zwischen dem gemessenen Abstand des Roboters von der erkannten Wand ($dist_{erk}$) und dem berechneten Abstand zwischen dem Roboter und der bekannten Wand ($dist_{bek}$). Bei einer zur Y-Achse parallelen Wand kann der Fehler in der X-Koordinate bestimmt werden. Die Korrektur der Roboterposition ist in Abbildung 6.3 zu sehen.

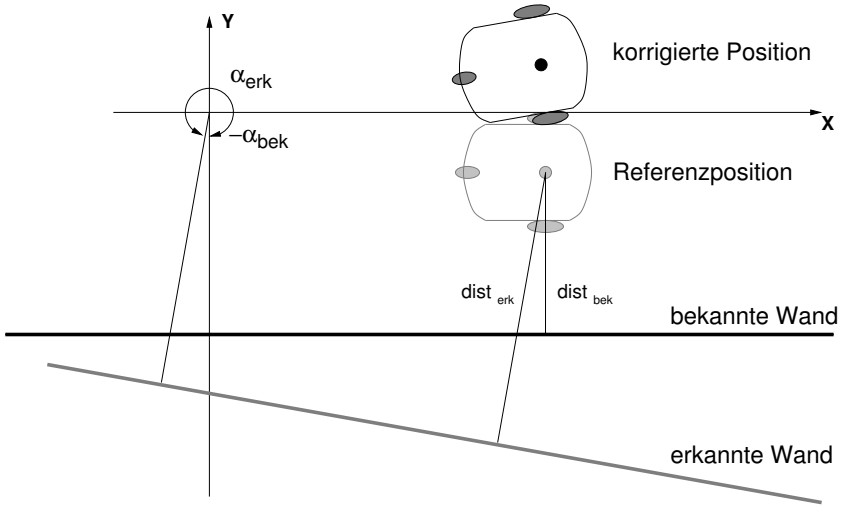


Abbildung 6.3: *Positionskorrektur mit Hilfe des Sonarsensors: Aus dem Unterschied zwischen den Abständen des Roboters zu der erkannten und der bekannten Wand lässt sich der Positionsfehler bestimmen, aus der Differenz der Richtungen der erkannten und der bekannten Wand der Orientierungsfehler.*

Konnten mehrere erkannte Wände zugeordnet werden, so werden die bestimmten Positionsfehler gemittelt. Mit Hilfe dieser Positionsfehler und der Referenzposition kann die korrigierte Position bestimmt werden. In einer Ecke können beide Koordinaten der Roboterposition korrigiert werden. Bleibt eine Koordinate unkorrigiert, so wird ihre Valid-Flag auf falsch gesetzt. Die berechnete Position wird im Positionsspeicher abgelegt.

6.2.5 LOPS

Das LOPS-System ist an einen stationären PC, dem LOPS-PC, angeschlossen. Daher ist das LOPS-Modul zweigeteilt. Es besteht aus einem Programm (lopscpp), das auf dem LOPS-PC läuft, und dem eigentlichen Sensormodul, das auf dem Onboardrechner des Roboters läuft.

Das Programm auf dem LOPS-PC liest die Daten von der LOPS-Hardware über die serielle Schnittstelle und rechnet die gemessenen Signallaufzeiten in Entfernungen um. Dazu wird zwischen den Laufzeiten und der Entfernung der Zusammenhang

$$\mathit{Entfernung} = \mathit{Laufzeit} \cdot \mathit{Signalgeschwindigkeit} + \mathit{Offset}$$

angenommen. Das Offset entsteht durch die Verarbeitungszeiten in den eingesetzten Mikrocontrollern, den Laufzeiten der elektrischen Signale sowie der Laufzeit des Funksignals von der MCU zum Roboter, es ist daher negativ. Die Werte für die Signalgeschwindigkeit und für das Offset werden aus der Konfigurationsdatei ausgelesen.

Aus den Entfernungen des Roboters zu den Empfängern wird die Position des Roboters (RP_{mess}) wie von Torsten Kleinschmidt (Kleinschmidt 2002) beschrieben mittels Trilateration berechnet. Durch Störung der Messung (zum Beispiel durch

Feldname	Datentyp	Größe	Inhalt
Startmarkierung	char	1 Byte	'S'
Typ	char	1 Byte	'D'
Zähler	long	4 Bytes	fortlaufende Nummer des LOPS-Paketes
Distanzen	double[3]	24 Bytes	gemessene Entfernungen
X	double	8 Bytes	X-Position des Roboters
Y	double	8 Bytes	Y-Position des Roboters
Endmarkierung	char	1 Byte	'E'

Tabelle 6.1: *Das Datenformat der Daten-Pakete des LOPS-Programms: Die Daten werden von zwei Bytes zur Synchronisation eingerahmt, um den Verlust von einzelnen Bytes zu Erkennen. Die Daten selbst werden binärkodiert.*

Feldname	Datentyp	Größe	Inhalt
Startmarkierung	char	1 Byte	'S'
Typ	char	1 Byte	'M'
Länge	byte	1 Byte	Länge der Nachricht (unsigned)
Nachricht	char[]	n Bytes	Nachricht
Endmarkierung	char	1 Byte	'E'

Tabelle 6.2: *Das Datenformat der Nachrichten-Pakete des LOPS-Programms: Der Rahmen entspricht dem der Daten-Pakete. Nach der Datenlänge folgt die Nachricht.*

Hindernisse zwischen dem Roboter und einem der Empfänger) kann es vorkommen, dass keine Position berechnet werden kann oder dass eine falsche Position berechnet wird.

Diese falschen Positionen werden ausgefiltert. Dazu werden die letzten berechneten Positionen sowie der Zeitpunkt, zu dem sie gemessen wurden, in einer Liste gespeichert. Aus diesen Positionen wird die aktuelle Roboterposition unter Annahme einer gleichförmigen Roboterbewegung geschätzt. Ist der Abstand zwischen der geschätzten Roboterposition ($\widehat{RP}_{akt}$) und der gemessenen Roboterposition zu groß, wird die Messung verworfen.

Zur Berechnung der geschätzten Roboterposition wird wie folgt vorgegangen: Aus den Positionen der ersten Hälfte der gespeicherten Roboterposition wird die Position RP_{alt} gemittelt, aus der zweiten Hälfte (den neueren Daten) die Position RP_{neu}. Die aktuelle Position RP_{akt} des Roboters wird unter Annahme einer geraden Roboterbewegung geschätzt:

$$RP_{akt} \approx \widehat{RP}_{akt} = RP_{alt} + \frac{t_{akt} - t_{alt}}{t_{neu} - t_{alt}} \cdot (RP_{neu} - RP_{alt})$$

Dabei ist t_x die (gemittelte) Zeit der Positionsmessung RP_x. Nun wird der Abstand der gemessenen von der geschätzten Position berechnet und mit einem Vertrauensabstand ($dist_{ok}$) verglichen. Ist der Abstand größer als der Vertrauensabstand, wird die Messung verworfen. Der Vertrauensabstand ist von dem Alter der

Positionen in der Liste und von dem in der Konfigurationsdatei einstellbaren Grundwert abhängig. Je älter die Messungen sind, desto größer darf der Abstand sein:

$$dist_{ok} = (t_{akt} - t_{alt}) \cdot Grundwert$$

Entfernung, Position und Zähler der Messung werden über einen TCP/IP-Socket an das LOPS-Modul des Synchronisationsprogramms auf dem Roboter übertragen. Dazu wird die Klasse *TcpSocket* verwendet. Das LOPS-Modul arbeitet als asynchrones Sensormodul, das am Socket auf Daten wartet. Es speichert die Position im Positionsspeicher, die Entfernungen und den Index der Messung im Sensordaten-Puffer und schreibt alle Werte in die Datenlogdatei.

Neben den LOPS-Daten werden auch Fehlermeldungen vom LOPS-PC zum Roboter übertragen. Diese werden an den MessageHandler weitergereicht. Das Format der Datenpakete, die zwischen dem LOPS-Programm auf dem LOPS-PC und dem LOPS-Sensormodul ausgetauscht werden, ist in den Tabellen 6.1 und 6.2 beschrieben.

6.2.6 SensorHandler

Um mit mehreren Sensoren umgehen zu können, wurde die Klasse *SensorHandler* programmiert, die die Verwaltung aller Sensormodule übernimmt. Während des Programmstarts initialisiert der *SensorHandler* alle in der Konfigurationsdatei aufgelisteten Sensoren mit den dort genannten Parametern. Anschließend stellt er Methoden zur Verfügung, mit denen alle Sensoren gestartet und gestoppt werden können. Außerdem reicht er im Offlinemodus Logdateieinträge an das zugehörige Sensormodul weiter.

6.3 Wichtige Datenstrukturen

6.3.1 Sensordaten

Der Sensordaten-Puffer speichert alle Sensordaten die zwischen zwei Synchronisationen gesammelt werden. Er ist in der Klasse *SensorDataBuffer* implementiert. Er ist wie in 5.3 beschrieben als Shadowbuffer designt, damit auch während der Synchronisation Daten gespeichert werden können. Er besteht im Wesentlichen aus zwei Arrays, die die Daten der Sensoren aufnehmen. In der Variablen *activeBuffer* ist vermerkt, welcher der beiden Puffer aktiv ist. In den aktiven Puffer legen die Sensormodule die Daten ab.

Die Klasse kapselt alle wichtigen Datenstrukturen und stellt nach außen Methoden zur Verfügung, um Daten einzufügen, den Puffer umzuschalten und die Daten des inaktiven Puffers zu lesen. Durch diese Kapselung der Datenstruktur sowie die Verwendung von Sperrvariablen (Mutexe) ist sichergestellt, dass zu keinem Zeitpunkt zwei Methoden gleichzeitig auf die Verwaltungsinformationen zugreifen.

Die Datenstrukturen für die einzelnen Sensoren sind als Klassen implementiert, die von der generischen Basisklasse *SensorDataEntry* abgeleitet sind. Sie können daher alle in einem Array gespeichert werden.

6.3.2 Synchronisierte Daten

Die synchronisierten Daten werden in der Datenstruktur *SynchonisedData* gespeichert. Diese Datenstruktur enthält einen Array, der die synchronisierten Daten der Sensoren enthält. Die Datenstrukturen, die die synchronisierten Daten eines Sensors aufnehmen, sind alle von der Klasse *SynchronisedSensorData* abgeleitet, können daher gleich behandelt werden.

Die Klasse *SynchronisedSensorData* beinhaltet außer den Datenstrukturen für die synchronisierten Daten auch eine Methode, um synchronisierte Daten aus der Datenlogdatei auszulesen, (*readFromFile()*) und eine Methode, um die Sensordaten dieses Sensors zu synchronisieren (*synchronise()*).

6.3.3 Positionen

Der Positionsspeicher (Klasse *Positions*) ist eine Sammlung aller im Programm berechneten Roboterpositionen. Alle Module, die die Roboterposition berechnen, legen diese hier ab.

Eine Position, die in der Klasse *Position* definiert ist, enthält neben den Datenfeldern für Position und Orientierung des Roboters auch drei Validitäten, die angeben, ob X-Position, Y-Position und Orientierung berechnet werden konnten. Dies ist nötig, da einige Sensormodule die Roboterposition nur teilweise berechnen können. So enthält zum Beispiel die vom LOPS-System berechnete Position des Roboters keine Orientierung und die vom Gyroskopmodul berechnete keine Position.

Die Klasse *Position* enthält auch die Methode *toString()*, die die Positionsdaten in einen String umwandelt und die Methode *toStringValidity()*, die die Validitäten der einzelnem Werte bei der Umwandlung in einen String berücksichtigt.

6.4 IDs

An vielen Stellen des Programms muss aus mehreren gleichartigen Objekten, die in einem Array angeordnet sind, ein bestimmtes ausgewählt werden. So muss zum Beispiel das Odometriesensormodul auf den Odometriedatenspeicher im Sensordatenpuffer zugreifen. Um den Programmcode übersichtlicher und für den Benutzer lesbarer zu gestalten, wurden für diese Zuordnungen IDs programmiert, die sich mit einem symbolischen Namen ansprechen lassen. So ist zum Beispiel *SensorID::odometry* viel leichter zuzuordnen als die Zahl 0, die dasselbe bedeutet.

Eine ID ist als Klasse programmiert (zum Beispiel *SensorID*). Sie besitzt eine *enum*, die alle symbolischen Namen enthält. Da an mehreren Stellen des Programms eine Umwandlung der ID in einen Namen nötig ist, besitzen die IDs auch eine Methode *toString()*, die den Namen des entsprechenden Objektes ausgibt. Außerdem kommt es häufig vor, dass mit IDs gekennzeichnete Objekte in der Konfigurationsdatei angegeben werden, daher haben diese IDs eine Methode *set()*, die die ID aus dem Namen setzt.

Durch die Verwendung von *enum*s ist es beim Compilieren des Programms möglich, eine Typprüfung durchzuführen. Der Compiler kann feststellen, ob bei ei-

nem Funktionsparameter die richtige ID verwendet wurde. Es ist also nicht möglich, statt einer SensorID eine PositionID zu übergeben, was bei einfachen Integerkonstanten oder mittels *#define* definierten Bezeichnern sehr leicht möglich ist.

6.5 Konfiguration

Die Konfigurationen des Synchronisationsprogramms und der beiden Hilfsprogramme lopscpp und logfileanalyser sind in Konfigurationsdateien gespeichert. Das Format dieser Dateien ist in Anhang B beschrieben. Zum Auslesen dieser Datei wurde die Klasse *ConfigurationFile* entworfen. Diese liest die Konfigurationsdatei Wert für Wert aus und gibt die einzelnen Einträge weiter. Dabei ist es möglich, die einzelnen Abschnitte gezielt anzuspringen.

6.6 Benutzerschnittstelle

Die Benutzerschnittstelle wurde in einem eigenen Modul gekapselt. Dieses Vorgehen hat zwei Vorteile: erstens ist es sehr leicht die Benutzerschnittstelle auszutauschen, um zum Beispiel eine aufwendige graphische Oberfläche zu ergänzen, und zweitens müssen die anderen Programmmodule (Sensormodule, Kalmanfilter, etc.) sich nicht mit den Details der Oberfläche auseinandersetzen.

Die Benutzerschnittstelle ist zweigeteilt. Die Ausgabe übernimmt das Modul UserInterface, das in der gleichnamigen Klasse definiert ist. Die Tastatureingaben werden von der ARIA-Bibliothek abgefragt und von der Klasse *KeyHandler* verarbeitet.

Die Klasse *UserInterface* definiert die Schnittstellen des Ausgabemoduls. Sie bietet Schnittstellen, mit denen die einzelnen Module Daten und der MessageHandler die Fehlermeldungen ausgeben können.

Die Klasse *NcursesUserInterface* implementiert ein Ausgabemodul auf Basis der NCURSES-Bibliothek, die einfache graphische Elemente (zum Beispiel Rahmen) in einem Terminal zur Verfügung stellt. Die Klasse *StdoutUserInterface* implementiert ein sehr einfaches Ausgabemodul. Es gibt lediglich die Fehlermeldungen auf der Standardausgabe aus.

6.7 Robotersteuerung

Die Ansteuerung des Roboters wird von der Klasse *ActionHandler* übernommen. Werden dieser Klasse Aktionen (Klasse *Action*) übergeben, wandelt sie diese Aktionen in Roboterbefehle um und reicht sie an den Roboter weiter. Dabei werden Queueing- und Non-queueing-Aktionen unterschieden.

Queueing-Aktionen werden in eine Liste einsortiert und nacheinander ausgeführt, so ist es zum Beispiel möglich eine Trajektorie aus mehreren Fahrbefehlen zusammenzusetzen und diese auf einmal an den *ActionHandler* zu übergeben.

Non-queueing-Aktionen werden sofort ausgeführt. Wird eine Non-queueing-Aktion angefordert, während noch Queueing-Aktionen ausgeführt werden, werden diese

gelöscht und mit der Abarbeitung der Non-queueing-Aktion fortgefahren. Damit ist es immer möglich den Stopp-Befehl (non-queueing) auszuführen, egal welche Befehle gerade ausgeführt werden.

Dem *ActionHandler* können verschieden Befehle übergeben werden:

Motoren an *(non-queueing)* Dieser Befehl schaltet die Motoren der Räder an. Erst nach diesem Befehl reagiert der Roboter auf Fahrbefehle.

Motoren aus *(non-queueing)* Dieser Befehl schaltet die Motoren aus. Danach kann der Roboter erst nach einen „Motor an“-Befehl gefahren werden.

Strecke fahren *(queueing)* Mit diesem Befehl kann dem Roboter eine Strecke übergeben werden, die dieser anschließend geradeaus fährt. Die Streckenmessung erfolgt auf dem Mikrocontroller, sie basiert auf den Odometriewerten.

Drehen *(queueing)* Mit diesem Befehl dreht sich der Roboter auf der Stelle um einen vorgegebenen Winkel. Die Abarbeitung dieses Befehl wird ebenfalls vom Mikrocontroller übernommen.

Kreis fahren *(queueing* oder *non-queueing)* Bei diesem Befehl werden der Kreisradius und die Geschwindigkeit angegeben, die der Roboter fahren soll. Wird auch eine Zeit angegeben, ist die Aktion queueing, der Roboter bearbeitet nach der angegeben Zeit die nächste Aktion. Wird keine Zeit angegeben, ist die Aktion non-queueing.

Radgeschwindigkeiten *(queueing* oder *non-queueing)* Mit diesem Befehl können für beide Antriebsräder Geschwindigkeiten angegeben werden. Wie beim „Kreisfahr“-Befehl kann eine Zeit angegeben werden, dann ist der Befehl queueing, ansonsten ist er non-queueing.

Diese Befehle können auch aus einer Aktionsdatei ausgelesen werden. Das Format dieser Datei ist in Anhang B.1.2 beschrieben. Mit Hilfe dieser Dateien ist es möglich, Trajektorien mehrmals abzufahren, ohne sie jedes Mal erneut eingeben zu müssen. Durch die Verknüpfung von Tasten mit diesen Dateien können komplexe Fahrbefehle im Synchronisationsprogramm verwendet werden. Es ist auch möglich, den ActionHandler so zu konfigurieren, dass er eine Datei beim Programmstart automatisch abfährt.

6.8 Hilfsmodule

In diesem Abschnitt werden einige der zentralen Hilfsmodule näher beschrieben. Diese Module stellen Funktionen zur Verfügung, die an mehreren Stellen des Programms benötigt werden. Die Hilfsmodule zentralisieren diese Aufgaben, so dass eine spätere Änderung dieser Funktionen erleichtert wird und eine mehrfache Programmierung derselben Aufgaben vermieden wird.

6.8.1 Datenlogmodul

Das Datenlogmodul übernimmt das Schreiben und Auslesen von Datenlogdateien. Durch diese Zusammenfassung in einer Klasse müssen sich die Module, die Daten in die Logdateien schreiben, nicht um das Format der Logdateien kümmern. Das Einfügen des Zeitstempels und der Zuordnungszeichen (siehe Anhang B) erfolgt automatisch.

Durch dieses Hilfsmodul ist es auch leicht möglich, das Format der Logdateien zu ändern, ohne dass man alle Sensormodule ändern muss. So würde zum Beispiel die Änderung einer einzigen Zeile Code ausreichen, um die Auflösung der Zeitstempel zu ändern.

6.8.2 Mathematische Funktionen

Bestimmte mathematische Funktionen werden an mehreren Stellen im Programm benötigt. Diese Funktionen sind in der Datei `Math.h` gesammelt. Diese Datei stellt Funktionen für das Quadrieren (*SQR()*), das Bestimmen von Minimum (*MIN()*) und Maximum (*MAX()*) zweier Zahlen sowie das Normalisieren von Winkeln (*normalize()*) zur Verfügung. Neben diesen Funktionen definiert dieses Modul auch noch Konstanten für die Umrechnung zwischen Grad und Radiant (*D2R* und *R2D*).

6.8.3 Fehlermeldungen

Jeder gemeldete Fehler muss im richtigen Format mit dem korrekten Zeitstempel und der richtigen Fehlerklasse sowohl in der Fehlerlogdatei gespeichert als auch im Ausgabemodul angezeigt werden. Diese Aufgaben übernimmt die Klasse *MessageHandler*.

Die statische Methode *message()* nimmt Fehlerklasse und Fehlertext als Parameter entgegen, setzt diese Daten mit dem aktuellen Zeitstempel zu einer Fehlermeldung zusammen und reicht diese Fehlermeldung an die Benutzerschnittstelle und das Fehlerlogmodul weiter. Sollen schon vor der Initialisierung der Benutzerschnittstelle und des Fehlerlogmoduls Meldungen ausgegeben werden, schreibt die Funktion die Meldungen auf die Standardausgabe und speichert sie in einer Liste. Diese Liste wird an die Benutzerschnittstelle und das Fehlerlogmodul weitergegeben, sobald diese initialisiert sind.

Kapitel 7

Hinzufügen weiterer Sensormodule

Das Programm ist modular aufgebaut und ermöglicht es leicht, neue Navigationssensoren hinzuzufügen. Um einen neuen Navigationssensor in die Programmarchitektur einzubinden muss das Programm an mehreren Stellen erweitert werden: Ein Sensormodul, das den Sensor initialisiert und seine Daten ausliest, muss programmiert werden sowie mehrere Datenstrukturen erweitert. Die nötigen Erweiterungen des Programms sind im Folgenden beschrieben.

7.1 Sensormodul

Das Sensormodul muss direkt oder indirekt von der Klasse *SensorModule* abgeleitet sein. Bei der Programmierung eines neuen Moduls ist es sinnvoll, sich je nach Sensor eine der vorhandenen Basisklassen auszusuchen (siehe 6.2.1). Diese bieten für die vorhandenen Anschlussvarianten bereits die nötigen Funktionen. Für neue Sensoren bietet sich meist die Basisklasse *ASynchronSensorModul* an.

Soll die Basisklasse *SynchronousSensorModule* oder *PacketHandlerSensorModule* verwendet werden, ist zu beachten, dass diese im ARIA-Roboterzyklus aufgerufen werden. Ihre Rechenzeit ist aber beschränkt, da der Roboterzyklus alle 100 ms ausgeführt werden muss. Benötigen Sensormodule, die als synchroner Task oder als Paketmodul ausgeführt werden, mehr Rechenzeit, so muss ein Teil des Sensormoduls in einen Thread ausgelagert werden.

Das Sensormodul muss die Initialisierung des Sensors übernehmen und dabei den Sensorstatus setzen. Beim Aufruf der Methode *startSensor()* beginnt die Messung. Das Loggen der Daten in der Logdatei und das Speichern in den Datenstrukturen startet mit dem Aufruf der Methode *startLogging()*. Mit dem Aufruf der Methode *stopSensor()* wird der Sensor (oder mindestens das Logging) gestoppt, im Destruktor wird der Sensor deinitialisiert und die Schnittstelle zum Sensor geschlossen. Das Einlesen der Sensordaten und die Sensordatenverarbeitung werden je nach Basisklasse von einer der Methode *runThread()* (asynchrones Sensormodul), *sensorTask* (synchrones Sensormodul) oder *packetHandler()* (Paketmodul) übernommen.

Für den Offlinemodus enthält jedes Sensormodul die Methode *readFromFile()*, diese liest einen Logeintrag aus der Logdatei ein und schreibt die Daten in den Sensordatenpuffer. Ist das Sensormodul so konfiguriert, dass es die Position des Roboters berechnet, legt die Methode *readFromFile()* diese im Positionsspeicher ab.

7.2 IDs

Damit der Sensor von den anderen Programmmodulen verwendet werden kann, benötigt das Sensormodul eine eigene ID. Dazu wird die Klasse *SensorID* erweitert. In dem enum *SensorIDs* wird eine neue Konstante hinzugefügt. Diese Konstante muss vor *numberOfSensors* eingefügt werden.

Neben der enum müssen auch die Methoden *toString()* und *getLogString()* so erweitert werden, dass sie für die neue Konstante den Namen des Sensormoduls, beziehungsweise den Logmarker des Sensors, ausgeben.

Berechnet das neue Sensormodul auch die Roboterposition, wird auch eine Positions-ID benötigt. Dazu wird die Klasse *PositionID* erweitert. In dieser Klasse muss in der enum *PositionIDs* analog zur Klasse *SensorID* eine neue Konstante eingefügt werden. Anschließend müssen die Methoden *toString()* und *fromLogMarker()* erweitert werden.

Durch diese Änderungen können alle anderen Module die neuen Positionen und das Sensormodul verwenden. Da durch das Einfügen der beiden IDs auch die Werte der Konstanten *numberOfSensors* und *numberOfPositions* erhöht werden, wachsen auch alle Arrays im Programm, so dass keine Probleme durch zu kurze Arrays auftreten.

7.3 SensorHandler

Der SensorHandler bindet alle Sensormodule in das Programm ein. Die Erweiterung des SensorHandler beschränkt sich auf die Einbindung des Sensormoduls während der Initialisierung des SensorHandlers. Dazu wird der Konstruktor der Klasse *SensorHandler* erweitert. Wenn der Sensor verwendet werden soll (Variable *globalConf ->sensor[SensorID::ID des neuen Moduls].active* hat den Wert *true*), wird eine Instanz des neuen Sensormoduls in den Array *sensorModule* eingefügt. Dabei kann man sich an den bereits vorhandenen Sensoren orientieren. Damit die neue Sensor-Klasse dem Sensor-Handler auch bekannt ist, muss die Headerdatei des neuen Sensormoduls eingebunden werden.

7.4 SensorDataBuffer

Die Datenstruktur SensorDataBuffer muss ebenfalls erweitert werden, da die Daten des Sensors auch gespeichert und weiterverarbeitet werden sollen. Dazu wird zuerst eine neue Datenstruktur für einen Datensatz des Sensors angelegt. Diese Datenstruktur muss von der Klasse *SensorDataEntry* abgeleitet sein und die Methode *set()* implementieren. Als Vorlage bieten sich die Datenstrukturen der bereits vorhandenen Sensoren an.

Diese neue Datenstruktur wird in der Klasse *SensorData* hinzugefügt, dazu ist der Konstruktor anzupassen. Die Datenstruktur ist analog zu den vorhandenen Sensordaten-Datenstrukturen in den Array *buffer* einzufügen.

7.5 SynchronisedData

Die Datenstruktur für die synchronisierten Daten muss für den neuen Sensor ebenfalls erweitert werden. Dazu wird eine von der Klasse *SynchronisedSensorData* abgeleitete Datenstruktur angelegt. Diese Klasse muss die Methoden *synchronise()* (Daten synchronisieren), *reset()* (Datenstruktur zurücksetzen) und *readFromFile()* (Daten aus der Logdatei lesen) implementieren.

7.6 Benutzeroberfläche

Durch das Erweitern der IDs wird auch die Benutzerschnittstelle automatisch um Ausgabefelder für die Sensordaten und die neue Position ergänzt. Es muss lediglich kontrolliert werden, ob auf der Oberfläche noch genügend Platz für alle Sensordaten und Positionen ist.

7.7 Kalmanfilter

Neben dem Synchronisationteil des Programms muss natürlich auch der Kalmanfilter erweitert werden, damit die neuen Informationen zu einer Verbesserung der Position des Roboters genutzt werden. Der Erweiterte Kalmanfilter für die hier beschriebene Architektur wird von Hendrik Müller entwickelt. Informationen zur Erweiterung des Erweiterten Kalmanfilters werden daher in seiner Diplomarbeit zu finden sein.

7.8 Hilfsprogramme

Neben dem Synchronisationsprogramm können auch die Hilfsprogramme Logfileanalyser und Logfileconverter (siehe Kapitel 9) erweitert werden.

Zur Auswertung der Positionen, die das neue Sensormodul liefert, und zur Analyse der Taktraten sind keine Erweiterungen notwendig, da die Module PositionAnalyser und TimeAnalyser alle vorhandenen Module bearbeiten. Zur Analyse der Sensorrohdaten wird jedoch ein neues Modul benötigt. Als Vorlage dazu können die vorhandenen Module (zum Beispiel OdometryAnalyser) dienen. Dieses neue Modul muss in der *main*-Routine zum Array *module* hinzugefügt werden. Dazu müssen die Konfigurationsdatei, das Auslesen der Konfigurationsdatei (Klasse *GlobalConfiguration*) sowie die Arraygröße erweitert werden.

Sollen die Daten des neuen Sensors auch in Matlab verwendet werden, muss das Programm logfileconverter erweitert werden. Die synchronisierten Daten des neuen Sensors müssen aus der Logdatei extrahiert und in die Ausgabedateien eingefügt werden.

Kapitel 8

Bedienung des Programms

Das Synchronisationsprogramm wird über die Tastatur gesteuert. Die Konfiguration wird vor dem Programmstart in der Konfigurationsdatei festgelegt. In dieser Datei wird etwa festgelegt, welche Sensoren verwendet werden, welche Daten geloggt werden oder welche Daten der Kalmanfilter verwendet. Sämtliche Einstellmöglichkeiten sind in Anhang B.1 beschrieben.

8.1 Starten

Vor dem Start des Synchronisationsprogramms müssen der Roboter und der Onboardrechner angeschaltet sowie die Ladekabel entfernt werden. Anschließend wird überprüft, ob die Sensoren, die verwendet werden sollen, korrekt angeschlossen sind. Soll das LOPS-System verwendet werden, müssen sowohl die MCU als auch die Sendeeinheit auf dem Roboter mit Strom versorgt werden.

Anschließend loggt man sich von einem der PCs im Labor mittels SSH auf dem Onboardrechner des Roboters (robot-pc) ein. Soll das LOPS-System verwendet werden, muss man sich auch auf dem Rechner gutemiene einloggen (lokal oder über SSH). Danach sollte die Konfigurationsdatei kontrolliert werden. Dabei muss vor allem auf die Einträge Positionsinitialisierung und Actions geachtet werden, da diese zur Folge haben können, dass der Roboter direkt nach dem Programmstart ohne weitere Benutzerbefehle losfährt.

Ist die Konfiguration eingestellt, kann das Programm gestartet werden. Beim Start können dem Programm einige Kommandozeilenparameter übergeben werden.

Parameter	**Bedeutung**
-Monline -Moffline	Programm startet im Onlinemodus (default) Programm startet im Offlinemodus
-CKonfigurationsdatei	Das Programm verwendet die angegebene Konfigurationsdatei (default: default.cfg)

Tabelle 8.1: *Die Kommandozeilenparameter des Programms syncaria: Mit den Kommandozeilenparametern können die Konfigurationsdatei und der Programmmodus gewählt werden.*

Taste	Bedeutung
a	Abfahren der ersten Aktionsdatei
b	Abfahren der zweiten Aktionsdatei
c	Abfahren der dritten Aktionsdatei
d	Motoren ausschalten
e	Motoren anschalten
x / q	Programm beenden
Leertaste	Roboter anhalten
↑	Beschleunigen
↓	Bremsen
←	Links
→	Rechts
k	Roboter fährt Kreis
s	Roboter fährt Quadrat

Tabelle 8.2: *Die Tastaturbelegung des Synchronisationsprogramms. Die Fahrbefehle (Pfeiltasten) entsprechen den Tasten des Saphira-Programms.*

Mit diesen kann die Konfigurationsdatei ausgewählt werden und das Programm in den Offlinemodus versetzt werden. Die Kommandozeilenparameter sind in der Tabelle 8.1 beschrieben.

Wird das LOPS-System verwendet, erscheint nach dem Start des Programms die Mitteilung „LOPS-Modul is waiting for connection on port xxxx“. Daraufhin muss auf dem PC, an den das LOPS-System angeschlossen ist, das Programm lopscpp gestartet werden, das eine Verbindung zum Synchronisationsprogramm auf dem Roboter aufbaut.

Ist die Positionsinitialisierung aktiv, fährt der Roboter nun ein kleines Stück vorwärts, um mit Hilfe des LOPS-Systems seine absolute Position und Orientierung zu bestimmen. Anschließend werden die Fahrbefehle aus der Aktionsdatei abgefahren, falls diese Funktion aktiviert ist. Danach kann der Roboter vom Benutzer gesteuert werden.

8.2 Benutzerschnittstelle

Ist das Programm gestartet kann, es mit der Tastatur gesteuert werden. Um den Roboter zu steuern, müssen zuerst die Motoren eingeschaltet werden (Taste `e`), anschließend kann der Roboter mit den Pfeiltasten gesteuert werden. Es gibt auch einige Tasten, hinter denen sich kompliziertere Fahrbefehle verbergen. Die Taste `k` lässt den Roboter einen Kreis fahren, die Taste `s` ein Quadrat. Die Tasten `a`, `b` und `c` lassen sich mit Aktionsdateien frei belegen (siehe Abschnitt 6.7). Die Tastenbelegung ist in Tabelle 8.2 beschrieben.

Bei Problemen kann der Roboter mit der Leertaste sofort angehalten werden. Mit dieser Taste werden alle noch vorhandenen Fahrbefehle gelöscht und beide Motoren angehalten. Reagiert der Roboter nicht mehr auf Tastendrücke, kann er mit

der Reset-Taste auf der unteren Plattform des Roboters (rote quadratische Taste) gestoppt werden.

Nach der Testfahrt wird das Programm mit der Taste `x` beendet. Anschließend müssen die Logdateien (Daten und Fehlermeldungen) kopiert werden, da das Programm diese beim nächsten Programmstart überschreibt.

8.3 Offline-Modus

Wird das Programm im Offline-Modus gestartet, läuft es ohne Benutzerinteraktion ab. Die Eingabe- und die Ausgabelogdatei sind in der Konfigurationsdatei angegeben, ebenso die zu verwendenden Sensoren. Nach Abarbeitung der Logdateien beendet sich das Programm.

Kapitel 9

Hilfsprogramme

9.1 logfilesorter

Da zwischen dem Lesen eines Datensatzes und dem Schreiben dieses Datensatzes in die Logdatei der Thread unterbrochen werden kann, ist es möglich, dass der Thread, der während dieser Unterbrechung aktiv ist, einen neuen Datensatz erstellt und speichert. Schreibt der alte Thread anschließend seine Daten in die Logdatei, so sind die Einträge in der Logdatei nicht chronologisch geordnet. Da diese Ordnung aber an einigen Stellen vorausgesetzt wird, ist es nötig die Datenlogdateien sortieren zu können.

Das Shell-Skript *logfilesorter* sortiert daher die Datenlogdatei nach dem Zeitstempel und bringt sie so in die richtige Reihenfolge. Das Skript lässt die ursprüngliche Datei unangetastet und schreibt das Ergebnis in eine neue Datei. Die Sortierung erfolgt mit dem GNU-Textutility sort. Der Aufruf erfolgt mit `logfilesorter` *`Quelldatei Zieldatei`*.

9.2 logfileanalyser

Der logfileanalyser dient der Analyse von Logdateien. Er extrahiert aus der Datenlogdatei Informationen und speichert sie in einem Format, das gnuplot (siehe Anhang C.3) verarbeiten kann. Die Ausgabedateien enthalten jeweils nur Informationen eines Moduls. Sie können leicht analysiert werden, zum Beispiel mit Hilfe der GNU-Textutilities. Mit Hilfe von gnuplot lassen sich die Informationen plotten.

Die Ausgabedateien werden im Ausgabeverzeichnis, das in der Konfigurationsdatei festgelegt wird, angelegt. In diesem Verzeichnis legt jedes Modul ein eigenes Unterverzeichnis an, das jeweils die Datendateien, die gnuplot-Ausgabedatei sowie die gnuplot-Steuerdatei (`command`) enthält. Um die Daten plotten zu können, muss gnuplot im jeweiligen Unterverzeichnis mit dem Parameter command aufgerufen werden.

Der logfileanalyser lässt sich mit einer Konfigurationsdatei konfigurieren. Diese Datei bestimmt, welche Module verwendet werden, welche Sensoren diese verwenden sowie das Verhalten von gnuplot (Art der Graphen und Ausgabeformat). Auch die Datenlogdatei, die eingelesen wird, wird in der Konfigurationsdatei festgelegt. Das Format der Konfigurationsdatei sowie alle Einstellungen sind in Anhang B erläutert.

Beim Programmstart kann dem Programm der Name der Konfigurationsdatei übergeben werden, wird kein Dateiname angegeben, wird die Datei `logfile.cfg` verwendet.

9.2.1 Module

Das Programm beinhaltet mehrere Module, die die Logdateien jeweils unter einem eigenen Gesichtspunkt analysieren. Diese Module lassen sich alle in der Konfigurationsdatei konfigurieren. Dort kann bei allen Modulen das gnuplot-Ausgabeformat (Bildschirm, Grafikdatei, etc.) sowie die Art der Graphen (Linie, Impulse, Punkte, etc.) eingestellt werden.

Das *Pfad-Modul* plottet die zurückgelegte Strecke des Roboters. Für jedes Modul (Sensoren und Kalmanfilter), das die Roboterposition berechnet, kann ein Pfad geplottet werden. Dieses Modul ermöglicht es dem Anwender, die Ergebnisse der einzelnen Sensoren untereinander sowie mit denen des Kalmanfilters zu vergleichen.
In der Konfigurationsdatei lassen sich, neben dem gnuplot-Ausgabeformat und der Graphenstile, die Module einschalten, deren Position geplottet werden soll.

Das *Odometrie-Modul* trägt die Enkoderinkremente der beiden Räder über die Zeit ab. Mit diesem Modul kann der Odometriesensor überprüft werden. So lässt sich zum Beispiel leicht erkennen, wie stark die Sensordaten verrauscht sind.

Das *Gyroskop-Modul* trägt die vom Gyroskop gemessenen Drehraten über die Zeit ab. Mit diesem Modul lässt sich die Güte des Gyroskops beurteilen. Neben dem Sensorrauschen lässt sich auch der systematische Fehler bestimmen.

Das *LOPS-Modul* trägt die Entfernungen zwischen dem Roboter und den drei Empfängern über die Zeit ab. Mit diesem Modul lässt sich leicht erkennen, ob das LOPS-System gestört wurde.

Das *Sonar-Modul* trägt die von den Sonarsensoren gemessenen Entfernungen über die Zeit ab. Erkennt der Sensor kein Hindernis, werden die Daten nicht angezeigt.

Mit dem *Zeitmodul* lässt sich das Zeitverhalten des Synchronisationsprogramms überprüfen. Dazu werden die Taktraten der Sensoren und der Synchronisation über die Zeit abgetragen. Mit diesem Modul lässt sich erkennen, ob ein Sensor unregelmäßig abgefragt wird oder ob Sensordaten verloren gegangen sind. In der Konfigurationsdatei lässt sich neben dem gnuplot-Ausgabeformat einstellen, welche Graphen geplottet werden sollen.

Beispiele für die Graphen, die mit diesen Modulen und dem Programm gnuplot erstellt werden können, sind im Kapitel 10 dargestellt.

9.3 logfileconverter

Der Logfileconverter konvertiert die Datenlogdatei in ein Format, das von Matlab leicht eingelesen werden kann und das alle synchronisierten Daten enthält, die für das Testen von Kalmanfiltern in der Simulation benötigt werden.

Der Logfileconverter legt zwei Ausgabedateien an, die Datei `data.log`, die neben den Zeitinformationen die Daten der Odometrie, des Gyroskops und des LOPS-Systems sowie die geschätzte Roboterposition (Kalmanfilter) enthält, sowie die Datei `sonar.dat`, die die Daten der Sonarsensoren enthält. Zu jeder Zeile in der Datei `data.log` gehört genau eine Zeile in der Datei `sonar.dat`. Das genaue Format der erzeugten Dateien ist in Anhang B beschrieben.

Das Programm erhält beim Aufruf zwei Kommandozeilenparameter, der erste enthält den Namen (und eventuell den Pfad) der Datenlogdatei des Synchronisationsprogramms, der zweite den Namen (und eventuell den Pfad) des Ausgabeverzeichnisses. Werden die Parameter nicht angegeben, werden die Werte *data.log* und *output* verwendet.

Teil III

Ergebnisse und Ausblick

Kapitel 10

Ergebnisse

Die in dieser Diplomarbeit entstandenen Programme erfüllen die an sie gestellten Anforderungen. Mit dem Synchronisationsprogramm können Sensordaten aufgezeichnet, synchronisiert und an einen Kalmanfilter weitergereicht werden. Es ist auch in der Lage, den Roboter zu steuern. Dabei können auch Trajektorien in einer Datei festgelegt werden, so dass zu Testzwecken dieselbe Trajektorie mehrfach abgefahren werden kann.

Mit dem Hilfsprogramm logfileanalyser ist es möglich, die Logdateien des Synchronisationsprogramms zu untersuchen: Mit den Graphen, die mit Hilfe dieses Programms und gnuplot erzeugt werden, können Fehler in der Hardware (Sensoren) und der Software (Sensormodule, Synchronisation und Kalmanfilter) leicht erkannt werden.

Mit dem Hilfsprogramm logfileconverter können die Logdateien für die Verarbeitung mit Matlab aufbereitet werden. In Matlab ist es leicht möglich, einen neuen Kalmanfilter zu testen und seine Parameter zu verbessern. Außerdem bietet Matlab die Möglichkeit, beliebige Variablen zu plotten.

Zur Verdeutlichung der Ergebnisse wird im Folgenden die Auswertung einer Messfahrt beschrieben. Als Messfahrt wurde ein Quadrat mit einer Seitenlänge von 3 Metern gewählt. Dieses wurde dreimal gegen den Uhrzeigersinn abgefahren. Bei der Messfahrt wurden die Daten aller Sensoren aufgezeichnet. Der online arbeitenden Kalmanfilter verwendete die Daten der Odometrie und des Gyroskops.

10.1 Ergebnisse des Synchronisationsprogramms

Zur Beurteilung der Daten wurden die Informationen der Logdatei zuerst mit den Programmen logfileanalyser und gnuplot geplottet. Um die Plots nachbearbeiten zu können, wurde als gnuplot-Ausgabeterminal `fig` mit den Parametern `color big metric` gewählt. Die so entstandenen Ausgabedateien wurden mit dem Programm xfig bearbeitet und in das PDF-Format konvertiert. Mit xfig wurden die Achsenbeschriftungen ergänzt und die Farben der Graphen angepasst. Die verwendete Konfigurationsdatei des Programms logfileanalyser ist in der Abbildung 10.1 auf Seite 54 zu sehen.

Im Graphen, der die Daten der Radenkoder darstellt, lassen sich die zwölf Drehungen des Roboters erkennen. Das linke Rad dreht sich rückwärts, während sich das rechte Rad vorwärts dreht. Durch diese Radbewegungen dreht sich der Roboter auf der Stelle. In den Abschnitten, in denen der Roboter geradeaus fährt, lässt sich das Rauschen der Enkoderdaten erkennen. Die Daten des linken Radenkoders schwanken zum Beispiel auf der ersten Gerade nach dem anfänglichen Einschwingen zwischen ca. 4300 und 4700. Der Graph ist in Abbildung 10.3 auf Seite 56 dargestellt.

Auch im Plot der Gyroskopdaten lassen sich die Drehungen des Roboters erkennen. Das Gyroskoprauschen lässt sich am besten während des Stillstands des Roboters beurteilen. Dazu kann direkt auf die Messwerte in der logfileanalyser-Ausgabedatei `Gyroskop/Gyroscop` zugegriffen werden. Bevor der Roboter gestartet wurde, lieferte das Gyroskop meistens die Drehrate $-0.000003835885\frac{1}{s}$. In diesen Messwerten ist der Wert $0.000013599955\frac{1}{s}$ mehrmals eingestreut. Die Differenz dieser beiden Werte lässt sich mit Hilfe des Skalenfakors auf eine Schwankung der Gyroskoprohdaten von einem Bit zurückrechnen. Im Stillstand am Ende der Fahrt schwankt das Gyroskop zwischen den Werten $-0.000003835885\frac{1}{s}$ und $-0.000021271724\frac{1}{s}$. Die Drift des Gyroskops war während dieser Messung also minimal. Der Plot der Gyroskopdaten ist in Abbildung 10.4 auf Seite 56 dargestellt, die Ausschnitte aus der Datei `Gyroskop/Gyroscop` sind in der Abbildung 10.2 auf Seite 10.2 zu sehen.

Beim Graph der Sonarmesswerte wurden durch Änderung an der gnuplot-Kommandodatei (`command`) die Daten der meisten Sensoren entfernt. Nur die Daten der Sensoren 7 und 15 wurden geplottet. Diese Sensoren sind nach rechts gerichtet (siehe Abbildung 6.3 auf Seite 34). An den mit Dreiecken markierten Stellen, die jeweils der ersten Seite des Dreiecks entsprechen, kann man erkennen, dass der Roboter sich der Wand nähert. Nach dem Stillstand des Roboters (letztes Dreieck) unterscheiden sich die Werte der beiden Sensoren um 70 cm. Der untere Sensor (Sensor 7) misst den Abstand zum Pfosten im Labor, während der obere Sensor (Sensor 15), der etwas weiter hinten angebracht ist, den Abstand zur Wand hinter dem Pfosten misst. Der Graph ist in Abbildung 10.5 auf Seite 57 dargestellt.

Die vom LOPS-System gemessenen Entfernungen des Roboters von den drei Empfängern ist in Abbildung 10.6 auf Seite 57 zu sehen. Es fällt auf, dass die Daten des LOPS-Systems nur ein sehr geringes Rauschen aufweisen, es gibt lediglich einige wenige Fehlmessungen. Da diese Fehlmessungen jedoch stark von den zeitlich angrenzenden Messungen abweichen, ist es sehr einfach, diese Fehler auszufiltern. Bei den 2024 Messungen während der Testfahrt war es dem LOPS-System lediglich 404-mal nicht möglich, die Position des Roboters zu bestimmen. Bei diesen Datensätzen weist jedoch meist nur einer der drei Empfänger einen Fehler auf. Werden die Daten von einem Kalmanfilter verarbeitet, kann dieser die geschätzte Roboterposition auch mit nur einer korrekten Entfernung verbessern.

Die Ergebnisse der Positionsberechnungen der Sensormodule und des Kalmanfilters sind in der Abbildung 10.7 auf Seite 58 zu sehen. Der Pfad des LOPS-Moduls kann als Referenzmessung dienen. Seine Daten sind etwas verrauscht, jedoch hat sich gezeigt, dass seine Werte um lediglich 1 cm um die tatsächliche Position schwanken. Im Stand kann man dies am Anfang der logfileanalyser-Ausgabedatei `Position/LOPS` erkennen, die Messwerte schwanken kaum. Auch eine Überprüfung

mit einem Maßband bestätigt dies. Die Ergebnisse der Odometrie ist schlecht, die berechnete Endposition liegt mehr als 150 cm neben der tatsächlichen Endposition des Roboters. Die vom Kalmanfilter berechnete Position des Roboters weicht ca. 30 cm von der tatsächlichen Position ab. Dabei ist zu beachten, dass dieses Ergebnis mit einer frühen Version des Kalmanfilters erzielt wurde. Diese berücksichtigt nur die Ergebnisse der Odometrie und des Gyroskops. Die Daten der Sonarsensoren und des LOPS-Systems wurden nicht verwendet, auch wurden die Parameter noch nicht angepasst.

Im Taktraten-Graphen ist zu erkennen, dass die Synchronisation ca. alle 100 ms erfolgt. Wenn die logfileanalyser-Ausgabedatei `Time/Synchronisation` sortiert wird (z. B. mit dem Befehl `sort -g -b --key=2 Synchronisation > Sync.sort`), kann man in dieser Datei erkennen, dass die Taktzeit zwischen 95 und 105 ms schwankte. Die meisten Werte (über 85 %) lagen jedoch zwischen 98 und 102 ms. Die Synchronisation wurde von der Odometrie getriggert. Daher weist die Odometrie die gleichen Taktraten auf. Auch die Taktrate des Gyroskops schwankt nur unwesentlich. Lediglich die Taktrate des LOPS-Systems weist größere Schwankungen auf. Die minimale Dauer eines Taktes betrug 11 ms, die maximale 270 ms, die meisten Werte (über 85 %) liegen zwischen 90 und 110 ms. Hier zeigen sich die Probleme des LOPS-Systems. Denn die Daten werden auf einem externen Rechner erfasst und anschließend mittels WLAN zum Onboardrechner übertragen.

10.2 Ergebnisse des Offline-Kalmanfilters

Mit Hilfe des Hilfsprogramms logfileconverter kann die Sensordatenlogdatei des Synchronisationsprogramms in ein Format umgewandelt werden, das von Matlab eingelesen werden kann. Dort können die bereits bestehenden Kalmanfilter mit den Daten getestet und um die neuen Sensoren erweitert werden.

Einige Graphen der Matlabauswertung sind in Abbildung 10.9 auf Seite 59 zu sehen.

```
[Main]
data.log
default
yes yes yes yes yes yes

[Position]
"fig color big metric"
kalman lines
lops lines
odometrie lines

[Lops]
"fig color big metric"
dots dots dots

[Odometry]
"fig color big metric"
lines lines

[Gyroscop]
"fig color big metric"
dots

[Sonar]
"fig color big metric"
dots dots dots dots dots dots dots dots
dots dots dots dots dots dots dots dots

[Time]
"fig color big metric"
yes
dots
gyro dots
lops dots
```

Abbildung 10.1: *Die Konfigurationsdatei des Programms logfileanalyser: Als Ausgabeformat wurde das Vektor-Format fig gewählt, da sich dieses leicht mit dem Programm xfig bearbeiten lässt.*

```
   ...
   3597 -0.000003835885
   3618 -0.000003835885
   3638 -0.000003835885
   3659 -0.000003835885
   3679 -0.000003835885
   3700 -0.000003835885
   3720 -0.000003835885
   3744 -0.000003835885
   3761 -0.000003835885
   3781 -0.000003835885
   3802 -0.000003835885
   3822 -0.000003835885
   3844  0.000013599955
   3863 -0.000003835885
   3884 -0.000003835885
   3904 -0.000003835885
   3925 -0.000003835885
   3955 -0.000003835885
   3970 -0.000003835885
   ...
 205341 -0.000003835885
 205362 -0.000003835885
 205382 -0.000003835885
 205403 -0.000003835885
 205423 -0.000003835885
 205444 -0.000003835885
 205464 -0.000003835885
 205485 -0.000021271724
 205505 -0.000003835885
 205526 -0.000003835885
 205546 -0.000003835885
 205567 -0.000003835885
 205587 -0.000003835885
 205607 -0.000003835885
   ...
```

Abbildung 10.2: *Ausschnitte aus der logfileanalyser-Ausgabedatei* `Gyroskop/Gyroscop`*: Während der Roboter am Beginn und Ende der Messung still steht, schwanken die Werte des Gyroskops leicht. Beim Vergleich der beiden Bereiche fällt auf, dass die Drift des Gyroskops minimal war.*

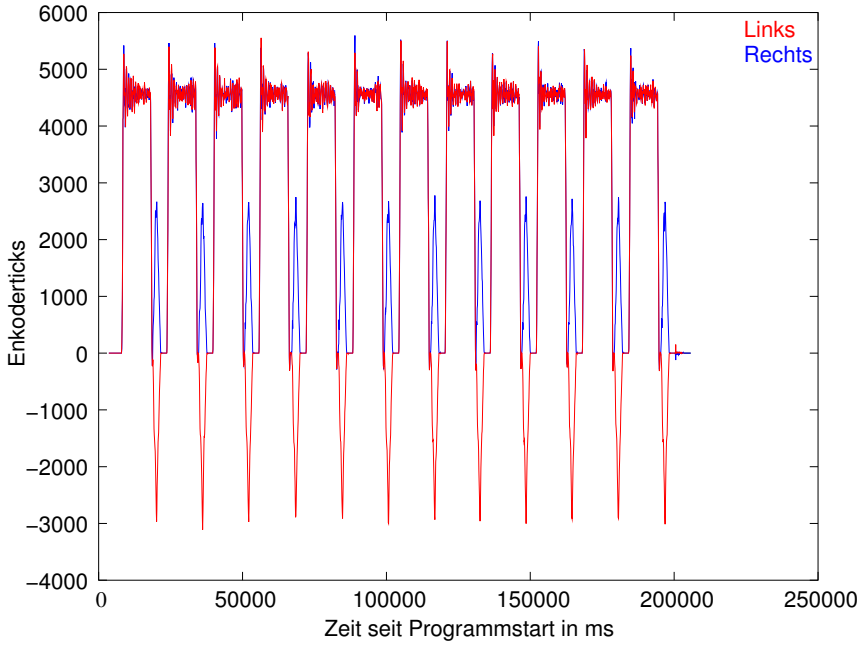


Abbildung 10.3: *Plot der Odometriedaten: Im Graphen lassen sich die zwölf Drehungen des Roboters leicht erkennen. Das linke Rad dreht sich während der Drehung rückwärts, das rechte Rad vorwärts. Auf den Geraden fällt das Rauschen der Radenkoder ins Auge.*

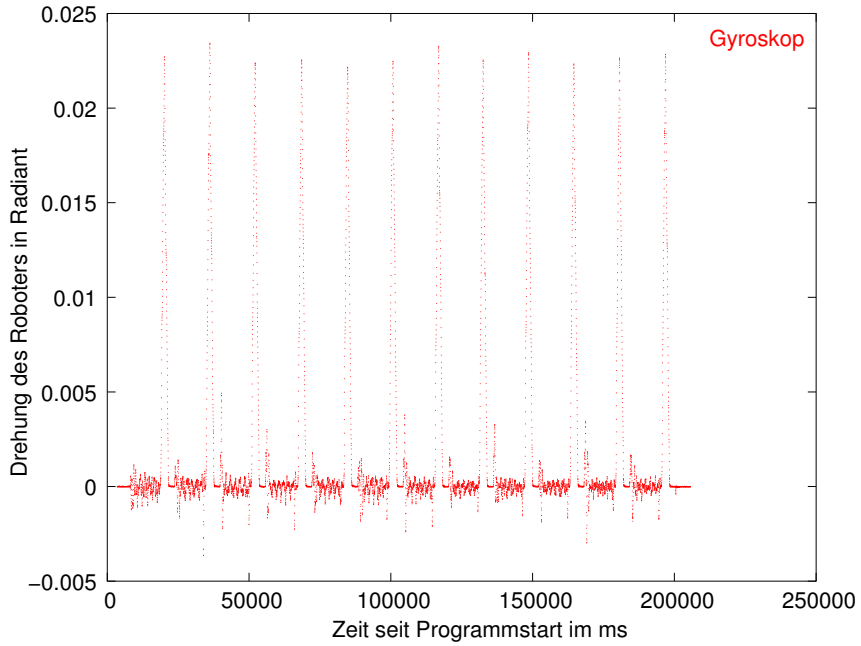


Abbildung 10.4: *Plot der Gyroskopdaten: Auch in diesem Graphen lassen sich die zwölf Drehungen des Roboters erkennen. Am Anfang und am Ende des Graphen kann man erkennen, das das Gyroskop im Stand nur über ein sehr schwaches Rauschen verfügt.*

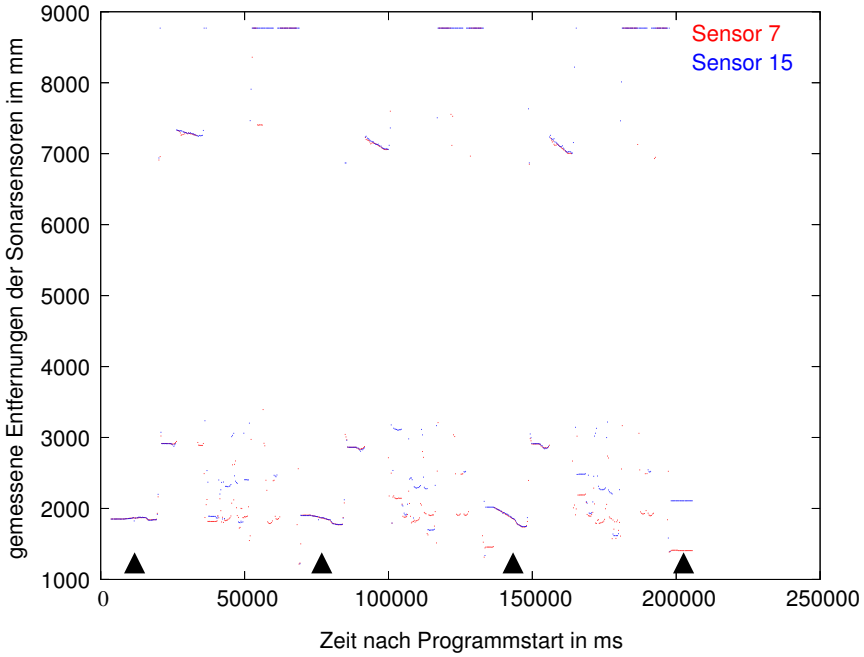


Abbildung 10.5: *Plot der Sonardaten: Der Graph zeigt die gemessenen Entfernungen der nach rechts gerichteten Sonarsensoren 7 und 15. An den mit Dreiecken markierten Stellen sieht man, dass der Roboter im Abstand von ca. 2 m an der Laborwand entlangfährt.*

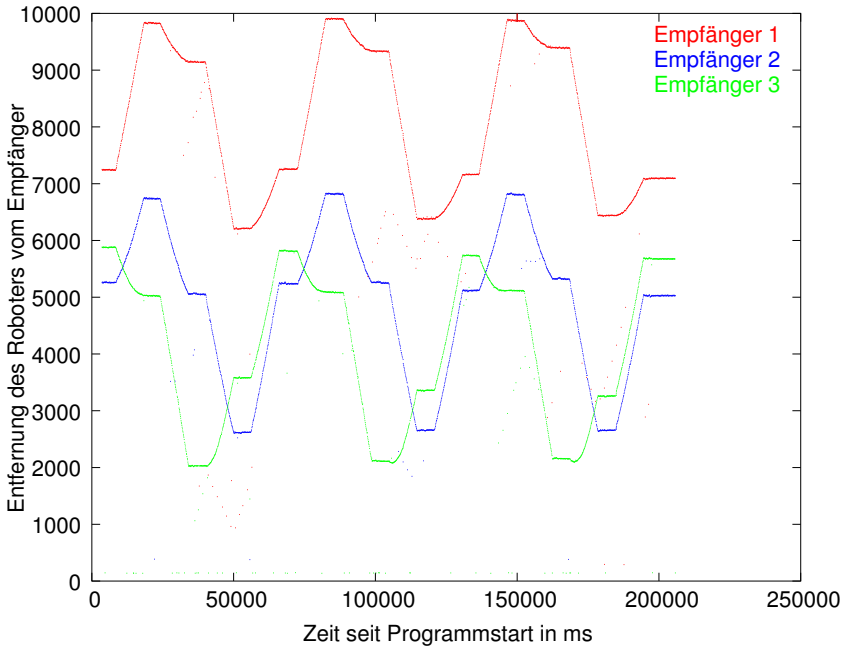


Abbildung 10.6: *Plot der LOPS-Daten: In diesem Graphen kann man erkennen, dass nur wenige der Messungen falsch sind. Die Daten sind kaum verrauscht, es gibt nur einige Ausreißer. Diese können leicht ausgefiltert werden.*

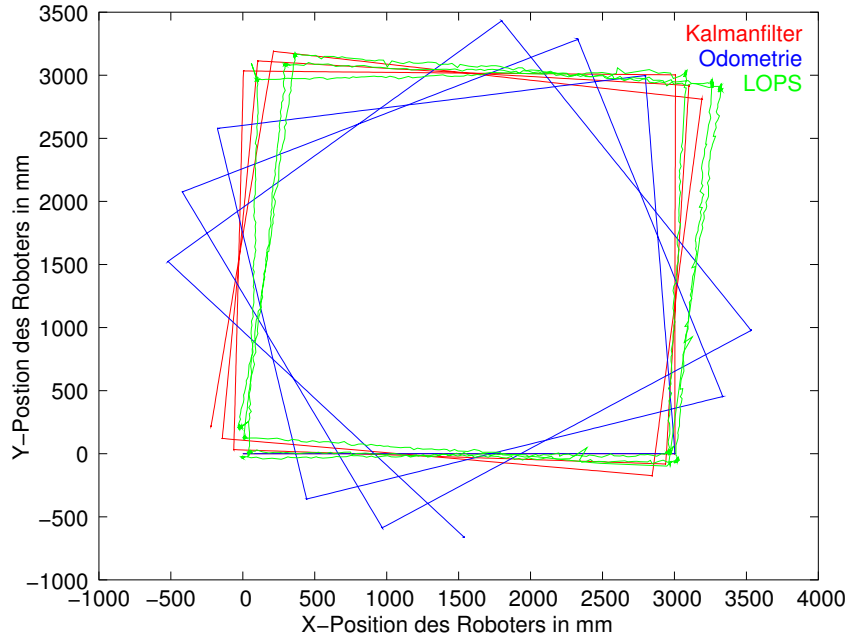


Abbildung 10.7: *Plot der Positionen: Die vom LOPS-System gemessene Position des Roboters ist sehr zuverlässig. Sie schwankt um ca. 10 mm um die tatsächliche Roboterposition. Die Ergebnisse der Odometrie sind sehr schlecht, die Endposition ist ca. 1500 mm falsch. Das Ergebnis des Kalmanfilters ist wesentlich besser.*

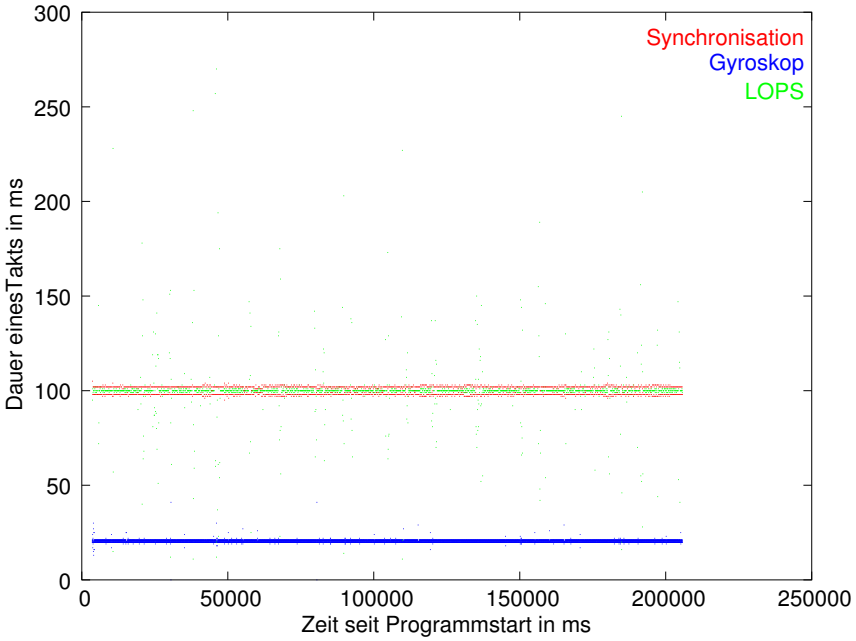
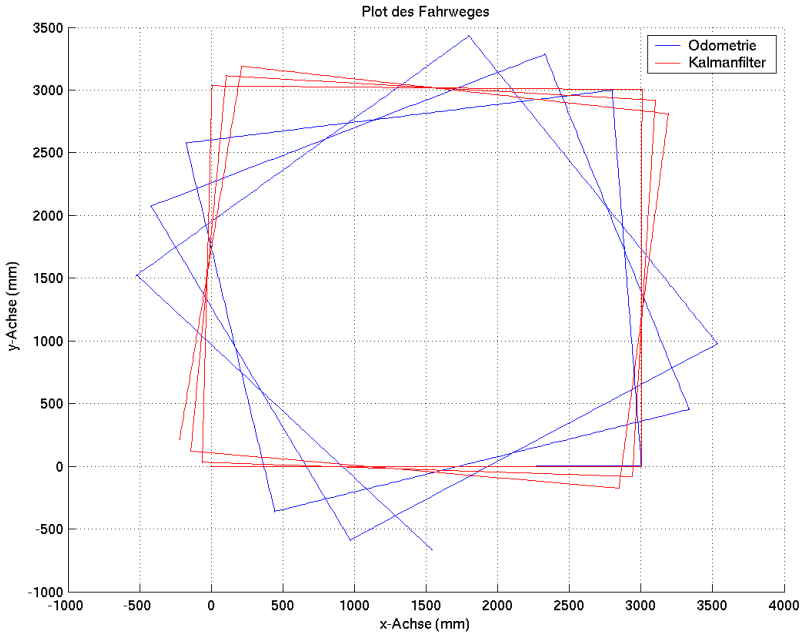
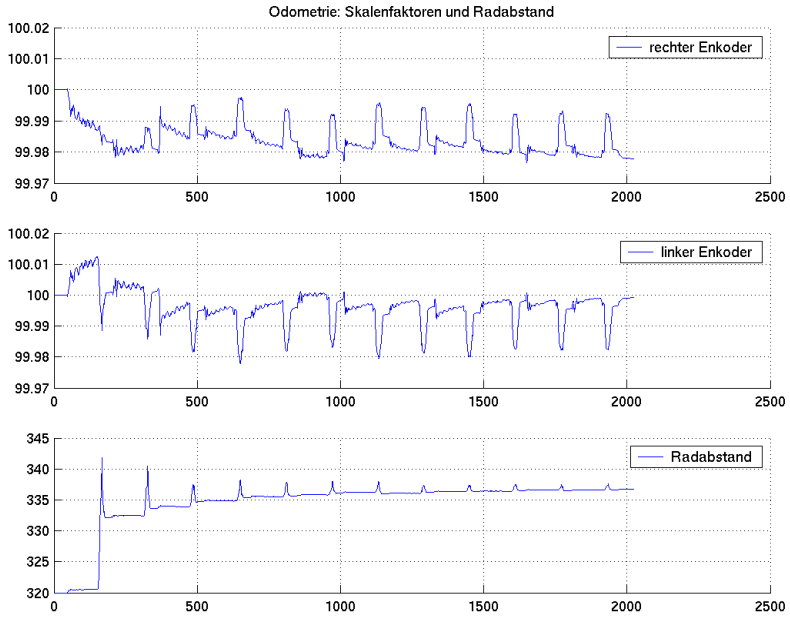


Abbildung 10.8: *Plot der Taktdaten: Im Graphen ist zu erkennen, dass die Taktraten des Gyroskops und der Synchronisation kaum schwanken. Die Taktraten des externen LOPS-Systems schwanken jedoch relativ stark, was an der Anbindung des LOPS-Systems über das WLAN liegt.*



(a) Die berechneten Roboterpositionen



(b) Die Parameter der Odometrie

Abbildung 10.9: *Plots des Offline-Kalmanfilters in Matlab: Der verwendete Kalmanfilter schätzt neben der Roboterposition (a) auch einige Parameter des Robotersystems und der Sensoren. Der Graph (b) zeigt die Schätzungen der Skalenfaktoren der beiden Enkoder sowie des Radabstands. Nähere Informationen zum Kalmanfilter werden in Hendrik Müllers Diplomarbeit zu finden sein.*

Kapitel 11

Ausblick

Das Synchronisationsprogramm funktioniert, es ist möglich mit seiner Hilfe Sensoren und Kalmanfilter zu testen. Während der Diplomarbeit sind jedoch auch einige Einschränkungen aufgefallen:

- Durch die Verwendung mehrerer Rechner ist eine genaue Zeitstempelung der Sensordaten nicht möglich.
- Die Rechenkapazität des Onboardrechners ist eingeschränkt.
- Durch den Ausfall des Korrelators ist nur noch ein Sensor zur Messung der Translationsgeschwindigkeit vorhanden.

Diese Probleme verhindern nicht den Einsatz des Synchronisationsprogramms, jedoch könnte die Lösung dieser Probleme den Einsatzbereich des Synchronisationsprogramms erweitern. Besonders die eingeschränkte Rechenleistung des Onboardrechners behindert das Testen aufwendigerer Navigationsroutinen stark. Denn zum Testen dieser Routinen muss immer erst eine Sensordatendatei aufgezeichnet werden, um diese anschließend offline bearbeiten zu können.

Im Folgenden werde die Einschränkungen näher beschrieben sowie Wege skizziert, wie diese Probleme gelöst werden könnten.

11.1 Zeitstempel

Das Synchronisationsprogramm versieht alle Daten (Sensordaten, synchronisierte Daten und Positionsdaten) mit einem Zeitstempel. Der Zeitstempel der Sensordaten kann jedoch erst in dem Moment, in dem die Daten vom Programm gelesen werden, hinzugefügt werden. Der genaue Zeitpunkt der Datenentstehung (der Zeitpunkt also, an dem der Sensor die Daten gemessen hat), ist dann jedoch nicht mehr bekannt, da die Zeit zwischen dem Aufzeichnen der Daten und dem Vorliegen der Daten im Programm nicht genau messbar ist und von vielen Faktoren abhängt.

Die Zeitstempel der Sensoren, die direkt an den Onboardrechner angeschlossen sind, sind sehr genau. Die Sensorthreads fragen die seriellen Schnittstellen blockierend ab. Das heißt, dass die Threads aufgeweckt werden, wenn neue Daten vorliegen,

so dass sie diese direkt verarbeiten können. Daher ist der Zeitverlust bei diesen Sensoren sehr gering. Bei einer zunehmenden Auslastung des Onboardrechners kann diese Latenz jedoch steigen.

Schwieriger sind die Daten der an den Mikrocontroller angeschlossenen Sensoren. Wie viel Zeit vergeht, bis das SIP mit den Enkoderdaten vom Mikrocontroller durch die ARIA-Bibliothek zum Sensormodul gelangt ist, lässt sich nicht abschätzen. Die Zeiten scheinen jedoch im Verhältnis zur Anbindung mit Hilfe von Saphira (Seidenspinner 2001/2002) besser geworden zu sein. Die Intervalle zwischen den Datenpaketen sind mit ARIA im Gegensatz zu Saphira relativ konstant. Die Zeitpunkte der Sonarmessungen werden nicht vom Mikrocontroller übermittelt, die Sonardaten werden nur alle 100 ms vom Mikrocontroller geliefert, obwohl alle 40 ms zwei Sensoren ausgelesen werden.

Auch die Daten des externen LOPS-Sensors sind nur schwer mit einem Zeitstempel zu versehen. Zwischen dem Auslesen der Daten in der MCU und dem Ankommen der Daten im Synchronisationsprogramm vergeht einige Zeit (Auslesen der Daten über die serielle Schnittstelle, Berechnen der Entfernungen und der Roboterposition auf dem PC, Senden der Daten über WLAN), die nicht genau messbar ist.

Zur Lösung dieses Problems, müssten die Daten direkt im Sensor mit einem Zeitstempel versehen werden (oder zumindest im Mikrocontroller und im LOPS-PC). Durch diese Maßnahme hätte die Übertragungszeit zum Onboardrechner keinen Einfluss mehr auf den Zeitstempel. Allerdings ist dies mit dem verwendeten Betriebssystem des Mikrocontrollers (P2OS) nicht möglich, es müsste durch eine Eigenentwicklung ersetzt werden. Ein derartiges Betriebssystem ist am Fachbereich Informatik an den Fachgebieten Integrierte Schaltungen und Systeme (Krieglstein 2000) sowie Simulation und Systemoptimierung (mehrere laufende Diplomarbeiten) entwickelt worden. Mit diesem Betriebssystem müsste allerdings die gesamte Anwendungssoftware umgestellt werden, da das neue Betriebssystem über andere Befehle mit dem Onboardrechner kommuniziert als P2OS.

Außerdem müsste eine Zeitsynchronisation des Onboardrechner mit dem LOPS-Rechners erfolgen. Das weit verbreitete Network Time Protocol (NTP) kann zur Lösung dieses Problems (Zeitsynchronisation dieser beiden Rechner) allerdings nicht angewendet werden, da dieses Protokoll zur Zeitsynchronisation mehrerer Server entworfen wurde und daher zu ungenau arbeitet. Die hier nötige Zeitsynchronisation im Millisekundenbereich kann folglich nicht erreicht werden. Somit wäre auch hier eine Eigenentwicklung notwendig.

11.2 Rechenkapazität des Onboardrechners

Der Onboardrechner des Roboters verfügt lediglich über eine sehr eingeschränkte Rechenkapazität (233 MHz-CPU). Daher können aufwendige Berechnungen (zum Beispiel eine Wanderkennung mit Hilfe einer Hugh-Transformation) nicht auf dem Roboter durchgeführt werden. Außerdem fällt die Oberfläche des Synchronisationsprogramms sehr schlicht aus, da diese Oberfläche über eine SSH-Verbindung auf einen PC übertragen wird. Diese Übertragung benötigt vor allem durch die Verschlüsselung der Daten auf dem Roboter bei aufwendigen Oberflächen viel Rechenleistung.

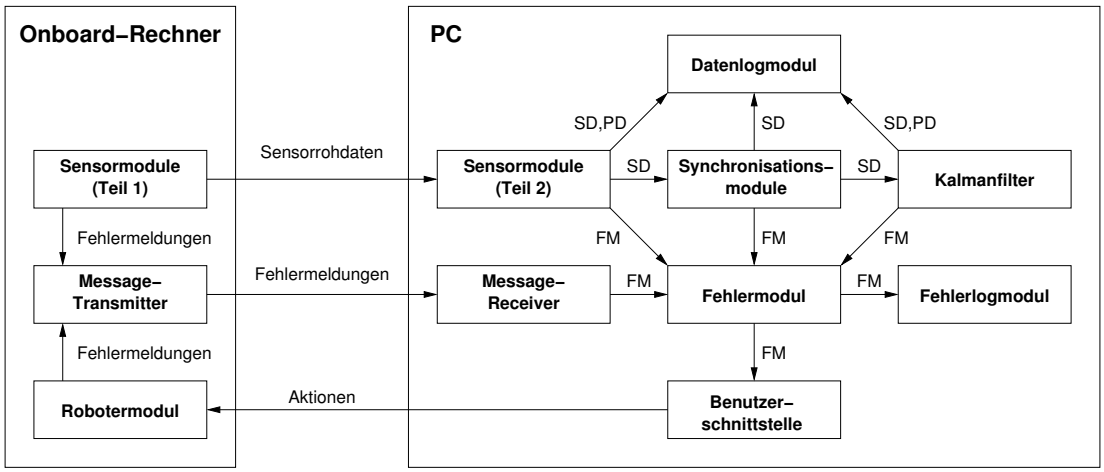


Abbildung 11.1: *Die Verteilung der Module bei der Aufteilung des Programms: Die Datenerfassung und die Steuerung des Roboters laufen auf dem Onboardrechner des Roboters, die anderen Module auf einem PC. (FM = Fehlermeldungen, SD = Sensordaten, PD = Positionsdaten)*

Zur Lösung dieser Probleme wäre es möglich, die Verarbeitung der Sensordaten auf einem PC durchzuführen. Dies ist im Moment nur im Offlinemodus möglich. Es wäre besser, wenn dies auch im Onlinemodus möglich wäre. Dazu müsste das Programm zweigeteilt werden. Auf dem Onboardrechner würden die Daten gesammelt und an den PC übertragen, wenn die Daten eines Abtastschritts vorlägen. Auf dem PC würde dann die Verarbeitung der Sensordaten, die Synchronisation sowie die Filterung durch den Kalmanfilter erfolgen. Im Gegenzug würden die Steuerbefehle vom PC an den Onboardrechner übertragen, wo sie an den Mikrocontroller weitergereicht würden.

Die Sensormodule müssten für diese Architektur zweigeteilt werden, der erste Teil liest die Rohdaten der Sensoren aus und wird auf dem Onboardrechner ausgeführt, der zweite Teil führt die aufwendigen Teile der Sensordatenverarbeitung durch (zum Beispiel die Wanderkennung) und läuft auf dem PC. Außerdem müsste auf dem Onboardrechner ein Modul laufen, das die Fehlermeldungen an den PC überträgt. Dieses Modul existiert bereits (*MessageTransmitter*), es wird vom LOPS-Programm verwendet, um die Fehlermeldungen an den Onboardrechner zu schicken.

Auch die Robotersteuerung müsste geändert werden. Der *ActionHandler* müsste auf dem Onboardrechner ausgeführt werden, die Benutzerkommandos jedoch auf dem PC entgegengenommen werden. Die übrigen Module könnten direkt übernommen werden. Diese Aufteilung der Module ist in Abbildung 11.1 zu sehen.

11.3 Messung der Translationsgeschwindigkeit

Durch den Ausfall des Korrelators kann die Translationsgeschwindigkeit nur noch durch die Odometrie bestimmt werden. Eine Überprüfung des Odometrieergebnisses ist nicht mehr durch einen zweiten Sensor möglich.

Als Ersatz für den relativ fehleranfälligen Bodenkorrelator wäre die Installation einer optischen Maus auf dem Roboter möglich. Eine optische Maus bestimmt wie der Bodenkorrelator die Translation durch eine Beobachtung der optischen Bodenmerkmale. Eine optische Maus bietet neben dem günstigen Preis noch weitere Vorteile gegenüber dem verwendeten Bodenkorrelator. Sie ist wesentlich kleiner und könnte dadurch eventuell direkt im Roboterdrehpunkt befestigt werden. Auch misst sie die Bewegung des Roboters nicht nur in einer Richtung, sondern in der gesamten Ebene.

Die Integration einer optischen Maus in das Robotersystem dürfte keine großen Probleme darstellen, da mehrere Maustreiber für Linux unter der GPL veröffentlicht sind und daher im Quellcode vorliegen.

Teil IV

Anhang

Literaturverzeichnis

ActiveMedia Robotics, LLC (2002). *Pioneer 2 / PeopleBot Operations Manual*.

ActiveMedia Robotics, LLC (2003). *Aria Reference Manual, Version 1.3.0*.

Alexander Rudolph (2003). *Quantification and Estimation of Differential Odometry Errors in Mobile Robotics with Redundant Sensor Information*. The International Journal of Robotics Research 22(2), Seiten 117 – 128.

Alexander Rudolph, Andreas Siegel und Jürgen Adamy (2002*a*). *Ein integriertes Navigationssystem für einen mobilen Roboter*. In: *Eingebettete Systeme – Entwurf und Anwendungen versteckter Computer* (Sorin Alexander Huss, Hrsg.). Seiten 84 – 89. Nummer 1/2002 In: *Thema Forschung*. Technische Universität Darmstadt.

Alexander Rudolph, Marcel Blank, Thorsten Kleinschmidt und Jürgen Adamy (2002*b*). *LOPS – Lokales Positionsbestimmungssystem*. Technischer Bericht. FG Regelungstheorie & Robotik, TU Darmstadt.

Andreas Siegel (2001). *Integration eines Gyroskops in ein mobiles Robotersystem und Design eines Erweiterten Kalman Filters zur Fusion von Kreisel- und Odometriedaten*. Diplomarbeit. FG Regelungstheorie & Robotik, TU Darmstadt.

Bjarne Stroustrup (2000). *Die C++-Programmiersprache*. 4. Auflage. Addison-Wesley Verlag. München.

Dimitri van Heesch (2003). *Doxygen Manual*. http://www.stack.nl/~dimitri/doxygen/download.html#latestman.

Greg Welch und Gary Bishop (2001). *An Introduction to the Kalman Filter*. Technischer Bericht. Department of Computer Science, University of North Carolina at Chapel Hill.

Hector Bratos (2002). *Methoden zur Bestimmung der Translationsgeschwindigkeit eines mobilen Roboters*. Diplomarbeit. FG Regelungstheorie & Robotik, TU Darmstadt.

I.N. Bronstein et al. (2001). *Taschenbuch der Mathematik*. 5. Auflage. Verlag Harri Deutsch. Frankfurt am Main.

Inna Mikhailova (2002). *Implementierung eines indirekten Kalman-Filters zur Fusionierung von Odometrie- und Gyroskopmessdaten eines mobilen Roboters*. Studienarbeit. FG Regelungstheorie & Robotik, TU Darmstadt.

Jochen Seidenspinner (2001/2002). *Zeitsynchronisation von Odometrie- und Gyroskopmessdaten eines mobilen Roboters*. Studienarbeit. FG Regelungstheorie & Robotik, TU Darmstadt.

Marcel Blank (2002). *Konzeptwahl zur Koordinatenbestimmung eines mobilen Roboters mit Hilfe eines Trilaterationsverfahrens*. Studienarbeit. FG Regelungstheorie & Robotik, TU Darmstadt.

Mark Mitchel, Jeffrey Oldham und Alex Samuel (2001). *Advanced Linux Programming*. New Riders. Boston. Online verfügbar unter http://www.advancedlinuxprogramming.com.

Michel Hunsiker (2000). *Situations identification for a mobile robot using sonar data and a Growing Neural Gas Network*. Diplomarbeit. FG Regelungstheorie & Robotik, TU Darmstadt.

Oskar von Stryk (2002). *Robotik 1*. Vorlesungsskriptum zur Vorlesung „Robotik 1“. Sommersemester 2002. FG Simulation und Systemoptimierung, TU Darmstadt.

Peter S. Maybeck (1979). *Stochastic Models, Estimation, and Control*. Band 1. Academic Press, Inc.

Richard Woller (2001). *Methoden zur sonarbasierten Gewinnung von globaler Positionsinformation für einen mobilen Roboter*. Diplomarbeit. FG Regelungstheorie & Robotik, TU Darmstadt.

Sven Goldt, Sven van der Meer, Scott Burkett und Matt Welsh (1995). *The Linux Programmer's Guide*. http://www.iblio.org/pub/linux/docs/linux-doc-project/programmers-guide/INDEX.html.

Thomas Williams et al. (1998). *gnuplot — An Interactive Plotting Program. Version 3.7*.

Thorsten Kleinschmidt (2002). *Möglichkeiten zur Erzeugung von Referenzkoordinaten zur Performance-Analyse von Wegintegrationsverfahren mobiler Roboter*. Studienarbeit. FG Regelungstheorie & Robotik, TU Darmstadt.

Tim Krieglstein (2000). *Implementierung eines Betriebssystems für radgetriebene nicht-holonome Roboter mit Zeitstempeln*. Diplomarbeit. FG Integrierte Schaltungen und Systeme, TU Darmstadt.

Anhang A

Programmdokumentation

Die Programme wurden mit Hilfe des Dokumentationswerkzeugs Doxygen (siehe Anhang C.3) dokumentiert. Doxygen erzeugt aus dem Quelltext, der mit speziellen Kommentaren versehen wurde, eine Dokumentation aller Klassen, Funktionen und Datenstrukturen. Diese Dokumentation beschreibt das gesamte API (Aplication Programming Interface). Sie stellt alle Informationen zur Verfügung, die benötigt werden, um bei einer Erweiterung des Programms die vorhandenen Klassen zu benutzen oder zu ändern.

Die Dokumentation der Programme liegt auf der CD in den Formaten PDF und HTML vor. Sie kann aber auch aus dem Sourcecode erzeugt werden. Dazu ruft man im jeweiligen Programmverzeichnis (das Verzeichnis in dem die kdevelop-Pojektdatei *.kdevprj liegt) das Programm doxygen mit dem Parameter Doxyfile auf. Die HTML-Startseite liegt anschließend unter *Programmname*`-api/html/index.html`. Die PDF-Dokumentation muss noch aus den LaTeX-Dateien erzeugt werden, dazu wechselt man ins Verzeichnis *Programmname*`-api/latex/` und ruft dort den Befehl `pdflatex refman` zweimal hintereinander auf.

Die Dokumentation beschreibt die Methoden aller Klassen, ihre Parameter und Rückgabewerte. Außerdem zeigt sie graphisch die Verbindungen (Vererbung und Besitz) zwischen den einzelnen Klassen an. Soll eine Funktion geändert werden, kann in der Dokumentation nachgeschlagen werden, wo diese Funktion benutzt wird. Mit diesen Informationen können die Folgen der Änderung leicht abgeschätzt werden.

Die Erzeugung der Dokumentationen kann mit Hilfe der Konfigurationsdateien `Doxyfile` konfiguriert werden, diese kann mit dem Tool *doxywizard* bearbeitet werden. Die verschiedenen Einstellmöglichkeiten sind in der doxygen-Dokumentation (van Heesch 2003) beschrieben.

Wird das Programm später erweitert oder geändert, sollte darauf geachtet werden, dass die neuen Klassen mit doxygen-Kommentaren dokumentiert bzw. die Kommentare bei den geänderten Klassen angepasst werden. Das Format der doxygen-Kommentare ist in der doxygen-Dokumentation dargestellt.

Um doxygen verwenden zu können sind einige weitere Programme notwendig, diese sind in Anhang C.3 beschrieben.

Anhang B

Dateiformate

Die im Rahmen dieser Diplomarbeit entstandenen Programme verwenden mehrere Dateien mit unterschiedlichen Dateiformaten. Die Programme lesen ihre Konfiguration aus Konfigurationsdateien und schreiben die Ergebnisse in Logdateien. Die Dateiformate sind so gewählt, dass die Dateien problemlos von einem Menschen gelesen und editiert werden können.

B.1 Konfigurationsdateien

Die Programme lesen ihre Konfiguration beim Start aus einer Konfigurationsdatei. Mit diesen Dateien lässt sich das Verhalten der Programme beeinflussen. Wenn zu jeder Messung die Konfigurationsdatei gespeichert wird, lässt sich später genau nachvollziehen, mit welchen Einstellungen die Daten aufgezeichnet wurden. Da die Programme den Namen der zu verwendenden Konfigurationsdatei als Kommandozeilenparameter akzeptieren, ist es sehr einfach möglich, die Programme mit unterschiedlichen Konfigurationen zu starten.

Fast alle Konfigurationsdateien werden mit Hilfe der Klasse *ConfigurationFile* eingelesen, daher ist ihr Format identisch. Die Konfigurationsdateien bestehen aus einer Aufzählung von einzelnen Werten, die in Abschnitte eingeordnet sind. Durch die einzelnen Abschnitte ist es möglich, dass alle Programmmodule ihre Einstellungen in einer gemeinsamen Datei speichern, ohne dass sich die Module gegenseitig beeinflussen. Neben den Werten kann eine Konfigurationsdatei Kommentare enthalten, die es erlauben, einzelne Werte zu beschreiben, um die Konfiguration leicht anpassen zu können.

Die Werte werden von der Klasse *ConfigurationFile* sequenziell ausgelesen. Die einzelnen Werte sind nur durch die Position in ihrem Abschnitt zugeordnet, nicht durch einen Bezeichner. Daher ist die Reihenfolge der Werte von Bedeutung, es ist nicht möglich, einen Wert auszulassen, da ansonsten alle folgenden Werte in diesem Abschnitt falsch zugewiesen werden. Lediglich am Ende eines Abschnittes ist es denkbar einen Wert als optional zu spezifizieren und ihn dann auszulassen.

Die einzelnen Werte sind durch mindestens einen Zeichenzwischenraum (Whitespace: Leerzeichen, Tabulator, Zeilenumbruch oder Seitenumbruch) getrennt. Soll ein Wert aus mehreren Worten bestehen (Zeichenzwischenräume enthalten), muss er

in doppelte Hochkomma (") eingeschlossen werden. Die Zeichen # und [sind in den Werten nicht erlaubt, da sie einen Kommentar beziehungsweise einen Abschnittsnamen einleiten.

Das Zeichen # leitet einen Kommentar ein. Dieser Kommentar endet am Zeilenende. Kommentare werden von der Klasse *ConfigurationFile* überlesen, sie dienen nur der Erläuterung der einzelnen Werte für den Benutzer. Von der Möglichkeit Kommentare in die Konfigurationsdateien einzufügen, sollte großzügig Gebrauch gemacht werden, da nur sie es einem anderen Anwender ermöglichen, die Konfigurationsdatei zu verstehen.

Um die Konfiguration mehrerer Module in einer Datei festlegen zu können ohne dass sich die Konfigurationen gegenseitig beeinflussen, sind die Konfigurationsdateien in Abschnitte unterteilt. Jeder Abschnitt beginnt mit seinem Abschnittsnamen, der in eckigen Klammern eingeschlossen ist, zum Beispiel „`[Main]`“. Der Abschnitt endet mit dem Beginn des nächsten Abschnitts oder dem Dateiende. Ein Ausschnitt aus einer Konfigurationsdatei ist in Abbildung B.1 auf Seite 72 zu sehen.

B.1.1 Verwendete Konfigurationsdateien

Die Programme syncaria, lopscpp und logfileanalyser verwenden Konfigurationsdateien, um ihre Einstellungen zu laden. Dabei verwenden lopscpp und syncaria dieselbe Datei, da einige Einstellungen bei beiden Programmen identisch sein müssen. Das Format der Konfigurationsdateien ist in den Tabellen B.1 auf Seite 77 (syncaria und lopscpp) und B.2 auf Seite 78 (logfileanalyser) beschrieben. Die Tabellen listen jeweils die einzelnen Werte in ihrer Reihenfolge auf. Jeder Abschnitt wird durch eine Überschrift eingeleitet.

B.1.2 Besondere Konfigurationsdateien

Es gibt zwei weitere Konfigurationsdateien, die nicht dem allgemeinen Konfigurationsdateiformat entsprechen. Die Wanddatei beschreibt die Position der Wände im Labor sowie die Bereiche, von denen aus die Wände vom Roboter erkannt werden können. Ihr Dateiformat wurde von Richard Woller (Woller 2001) übernommen. Jede Wand wird in dieser Datei durch eine Zeile mit 10 Werten beschrieben. Das Format dieser Datei ist in Tabelle B.3 auf Seite 79 beschrieben.

Die Aktionsdateien beschreiben Fahrbefehle. Diese Fahrbefehle können mit Tasten verbunden oder beim Programmstart automatisch ausgeführt werden. In diesen Dateien beschreibt jede Zeile einen Fahrbefehl. In die Dateien können Kommentarzeilen eingefügt werden. Dazu muss das erste Zeichen der Zeile ein „#“ sein. Zwischen dem Fahrbefehl und seinen Parametern beziehungsweise zwischen zwei Parametern muss jeweils mindestens ein Leerzeichen stehen. Das Dateiformat ist in Tabelle B.4 auf Seite 79 beschrieben.

B.2 Logdateien

Die Ausgaben des Programms syncaria werden in Logdateien geschrieben. Dabei werden die Sensordaten und die Fehlermeldungen in unterschiedlichen Dateien gespeichert, um eine einfachere Verarbeitung der Sensordatendateien zu ermöglichen. Das Format beider Logdateien ist zeilenbasiert, jeder Eintrag belegt genau eine Zeile.

Zu jedem Eintrag in einer Logdatei gehört auch ein Zeitstempel, dieser erlaubt es auch die beiden Logdateien zusammenzufügen, um beispielsweise zu sehen, ab wann sich ein Fehler auf die Sensordaten ausgewirkt haben kann. Der Zeitstempel gibt die seit Programmstart vergangenen Millisekunden an. Das Feld ist dabei groß genug, um Langzeitmessungen über einen ganzen Tag durchführen zu können.

Die Fehlermeldungen bestehen aus drei Feldern, der Fehlerklasse, die angibt wie schwer ein Fehler ist, dem Zeitstempel sowie der eigentlichen Fehlermeldung. Ihr Format und die Bedeutung der einzelnen Fehlerklassen sind in Tabelle B.5 auf Seite 80 dargestellt. Ein Ausschnitt aus einer Fehlerlogdatei ist in Abbildung B.2 (Seite 73) dargestellt.

Die Sensordateneinträge bestehen aus den Feldern Quelle, Art, Zeitstempel und Logeintrag. Das Feld Quelle gibt das Modul an, das den Logeintrag geschrieben hat (zum Beispiel „Odometrie“). Das Feld Art beschreibt welche Daten geloggt wurden (zum Beispiel „Rohdaten“ oder „Position“). Der Logeintrag hat je nach Quelle und Art ein eigenes Format.

Das Format der Sensordatendatei ist in der Tabelle B.6 auf Seite 81 (Format des gesamten Eintrags und Bedeutung der Einträge für Quelle und Art) dargestellt. Tabelle B.7 auf Seite 82 (Format der eigentlichen Daten) beschreibt das Format der Logeinträge. Ein Ausschnitt aus einer Sensordatendatei ist in Abbildung B.3 auf Seite 74 zu sehen.

B.3 Ausgabe des Programms logfileconverter

Das Programm logfileconverter wandelt die Datenlogdateien des Programms syncaria in ein Format um, das leicht von Matlab gelesen werden kann. Da mit Matlab nur der Kalmanfilter getestet wird, werden nur die synchronisierten Daten konvertiert. Das Ergebnis der Konvertierung wird in zwei Dateien gespeichert. Die Datei `data.dat` enthält die Zeitdaten, die Daten der Radenkoder, des Gyroskops und des LOPS-Systems sowie die geschätzte Position des Roboters. Die Datei `sonar.dat` enthält die Daten der Sonarsensoren. Das Format dieser Dateien ist in Tabelle B.8 auf Seite 83 zu sehen.

B.4 Ausgabe des Programms logfileanalyser

Die Ausgabedateien des Programms logfileanalyser sind an das gnuplot-Format angepasst. Da sie jedoch jeweils nur die Daten eines Sensors enthalten, eignen sie sich auch um die Ausgabe dieses einen Sensors zu beurteilen. So lassen sich etwa statistische Untersuchungen leicht durchführen.

Das Programm logfileanalyser legt in seinem Ausgabeverzeichnis für jedes Modul ein eigenes Unterverzeichnis an. In diese Unterverzeichnisse legt es die Ausgabedateien der Module ab. Ihr Format ist in Tabelle B.9 auf Seite 84 beschrieben.

```
[Main]

# realer Roboter (robot) oder Simulator (sim)
real

# Benutzerschnittstelle (stdout oder ncurses)
stdout

# Welcher Sensor triggert die Synchronisation (Sensornamen)
odometry

# nach wievielen Datenpaketen
1

# Kalmanfilter benutzen (yes oder no)
yes

...

#################################################################
[Sensors]

# Welche Sensoren werden wie verwendet
# für jeden Sensor eine Zeile
# Format der Zeile:
#   Sensorname
#   Daten synchronisieren (yes oder no)
#   Position berechnen (yes oder no)
#   Rohdaten loggen (yes oder no)
#   synchronisierte Daten loggen (yes oder no)
#   Position loggen (yes oder no)
#   im Kalmanfilter benutzen (yes oder no)


odo   yes yes yes yes yes yes
gyro  yes yes yes yes yes yes
walls yes yes yes yes yes no
lops   yes yes yes yes yes no
```

Abbildung B.1: *Ausschnitte aus der Konfigurationsdatei der Programme syncaria und lopscpp: Die Ausschnitte stellen einen Teil der globalen Konfiguration sowie die Einstellungen der Sensoren dar. Die Kommentare ermöglichen es, die einzelnen Werte zu erläutern, so dass sie verständlich sind.*

```
D        30 connection
D       478 connected
I       479 Configuration read succesfully. Configuration is:
I       483 ProgramMode: online
I       484 Triggering Sensor: Odometry, Trigger count: 1
I       485 LogLevel: 4
I       485 Kalmanfilter: on
I       486 SensorConfiguration (one sensor per line):
I       487 Odometry: synchronise, calculate position, log raw, log s
I       488 Gyroscop: synchronise, calculate position, log raw, log s
I       489 Walls: synchronise, calculate position, log raw, log sync
I       489 Lops: synchronise, calculate position, log raw, log synch
D       490 SensorHandler initialisation
D       491 PM: online
D       492 Odometry konstruktor
D       495 initialising gyroscop
D       500 NoUserInterface Thread gestarted
D      1512 end initialising gyroscop
I      1518 Lopsmodul is waiting for connection on port 4554
I      2793 Alle Sensormodule initialisiert
D      2794 Kalmanfilter using OdoAndGyro
I      2795 initialised
D      2796 Synchronisations Thread gestartet
I      2798 *** initilising current Robotposition using lops, please
I      2799 driving robot...
D      2797 Gyroskop Thread gestartet
D      2800 Lops Thread gestartet
D      2857 Lops-Program gestarted
I      5797 making first meassurement
I      9756 driving robot again...
I     14748 making second meassurement
I     18946 *** initialising complete, robot ready for driving ***
D    317467 Gyroskop Thread beendet
D    317508 Lops Thread beendet
D    317509 vor robot.disconnect()
D    318508 nach robot.disconnect()
D    318509 synchronisation thread beendet
D    319497 NoUserInterface Thread beendet
D    319498 nach Aria shutdown
I    319498 Program ended normal
```

Abbildung B.2: *Die Fehlerlogdatei des Programms syncaria: Die erste Spalte beschreibt die Schwere des Fehlers, die zweite den Zeitpunkt des Fehlers, in der dritten Spalte steht die Fehlermeldung.*

```
S T        0 20. 1.2004 13: 5:35
G R     3536 -0.000003835885
G P     3536        0        0 -0.00000 0 0 1
W R     3537      0      0      0      0      0      0      0      0      0
O C     3542        0        0 1
G C     3542 -0.000003835885 1
W C     3542      0      0      0      0      0      0      0      0      0
L C     3542        0        0        0        0        0 0 0
S S     3542        0        0        0 0 0 0
K P     3542        0        0  0.00000 1 1 1
O R     3542        0        0
O P     3542        0        0  0.00000 1 1 1
G R     3556 -0.000003835885
G P     3556        0        0 -0.00001 0 0 1
L R     3572    28658     7251     5258     5883        0        0 0
L P     3572        0        0        0 0 0 0
G R     3577 -0.000003835885
G P     3577        0        0 -0.00001 0 0 1
G R     3597 -0.000003835885
G P     3597        0        0 -0.00002 0 0 1
G R     3618 -0.000003835885
G P     3618        0        0 -0.00002 0 0 1
G R     3638 -0.000003835885
G P     3638        0        0 -0.00002 0 0 1
W R     3641      0      0      0      0      0      0   8845   1547
O C     3647        0        0 1
G C     3647 -0.000019179423 5
W C     3647      0      0      0      0      0      0   8846   1548
L C     3647     7251     5258     5883        0        0 0 1
S S     3647        0        0  0.00000 1 1 1
K P     3647        0        0  0.00000 1 1 1
O R     3647        0        0
O P     3647        0        0  0.00000 1 1 1
G R     3659 -0.000003835885
G P     3659        0        0 -0.00003 0 0 1
L R     3667    28659     7237     5268     5876        0        0 0
L P     3667        0        0        0 0 0 0
```

Abbildung B.3: *Ausschnitt aus der Datenlogdatei des Programms syncaria: In der ersten Spalte steht das Modul, das den Eintrag geschrieben hat, in der zweiten die Art des Eintrags und in der dritten der Zeitpunkt, an dem die Daten erzeugt wurden. Danach folgen die eigentlichen Daten.*

Das Format der Konfigurationsdatei von syncaria und lopscpp		
Name	**Format**	**Beschreibung**
[Main]		
Anbindung	Anbindungsname	Art der Anbindung (Simulator oder realer Roboter)
Benutzerschnittstelle	UI-Name	Die verwendete Benutzerschnittstelle
Trigger	Sensorname	Sensor, der die Synchronisation triggert
Triggeranzahl	Zahl (>0)	Anzahl der Rohdaten dieses Sensors, nach denen getriggert wird
Kalmanfilter	Yes/No	Soll der Kalmanfilter benutzt werden?
Positionsinitialisierung	Yes/No	Soll die Roboterposition mit Hilfe des LOPS-Systems initialisiert werden?
Datenlogdatei	Dateiname	Name (und Pfad) der Datenlogdatei
Debuglevel	Zahl (0-4)	Wie viele Debugnachrichten sollen angezeigt werden?
Fehlerlog	Yes/No	Sollen die Fehlermeldungen protokolliert werden?
Fehlerlogdatei	Dateiname	Name (und Pfad) der Fehlerlogdatei
[Sensors]		
Sensorname	Sensorname	Name des Sensors
Synchronisieren	Yes/No	Sollen die Daten synchronisiert werden?
Position	Yes/No	Soll die Position aus den Daten berechnet werden?
Rohdatenlogging	Yes/No	Sollen die Rohdaten geloggt werden?
Synchronisation loggen	Yes/No	Sollen die synchronisierten Daten geloggt werden?
Positionslogging	Yes/No	Soll die Position geloggt werden?
Kalmanfilter	Yes/No	Soll der Kalmanfilter die Daten verwenden?
Diese 7 Felder wiederholen sich für jeden benutzten Sensor		
[Odometry]		
Kreisberechnung	Yes/No	Soll die neue Positionsberechnung benutzt werden? (siehe Abschnitt 2.2)
[Gyroscop]		
Schnittstelle	Name	Der Name der Schnittstelle (z.B. `/dev/ttyS1`)
		Fortsetzung folgt

Fortsetzung: Das Format der Konfigurationsdatei von syncaria und lopscpp		
Name	**Format**	**Beschreibung**
[Walls]		
Wanderkennung	Yes/No	Sollen die Wände erkannt werden?
Historienlänge	Zahl	Die Anzahl der gespeicherten Sonardaten
max. Distanz	Zahl	Die maximale Entfernung zum Hindernis
min. Punkte (Seite)	Zahl	Minimale Anzahl von Punkten für eine Wanderkennung mit den seitlichen Sensoren
min. Punkte (Paare)	Zahl	Minimale Anzahl von Punkten für eine Wanderkennung mit Sensorpaaren
max. σ_d (Seite)	Zahl	Maximale Standardabweichung der Entfernung der Punkte zur Wand bei den Seitensensoren
max. σ_d (Paare)	Zahl	Maximale Standardabweichung der Entfernung der Wand vom Nullpunkt bei den Sensorpaaren
max. σ_α (Seite)	Zahl	Maximale Standardabweichung des Winkels der Wand bei den Sensorpaaren
max. Δd (oben / unten)	Zahl	Maximale Differenz der Distanz zwischen dem oberen und dem unteren Sonararray
max. $\Delta\alpha$ (oben / unten)	Zahl	Maximale Differenz des Winkels zwischen dem oberen und dem unteren Sonararray
Wanddatei	Zahl	Der Name der Datei mit den bekannten Laborwänden
max. Δd (Zuordnung)	Zahl	Maximale Abweichung zwischen der erkannten und der bekannten Wand (Distanz in mm)
max. $\Delta\alpha$ (Zuordnung)	Zahl	Maximale Abweichung zwischen der erkannten und der bekannten Wand (Winkel in Grad)
[Lops]		
Robotername	Rechnername	Der DNS-Name des Onboardrechners
Port	Portnummer	Der Port, auf dem der Onboardrechner auf eine Verbindung wartet
Schnittstelle	Name	Schnittstelle, an die das LOPS-System angeschlossen ist
Steigung n	Zahl	Kalibrationsdaten (Geschwindigkeit des Empfängers n)
Offset n	Zahl	Kalibrationsdaten (Offset des Empfängers n)
Die letzten beiden Felder wiederholen sich für alle 3 Empfänger		
X-Position n	Zahl	X-Position des Empfängers n in mm
Y-Position n	Zahl	Y-Position des Empfängers n in mm
Die letzten beiden Felder wiederholen sich für alle 3 Empfänger		
Gewichten	Yes/No	Sollen die einzelnen Daten gewichtet werden?
Positionsfilter	Yes/No	Sollen die Positionen gefiltert werden?
Historienlänge	Zahl	Länge der Positionshistorie
		Fortsetzung folgt

Fortsetzung: Das Format der Konfigurationsdatei von syncaria und lopscpp		
Name	**Format**	**Beschreibung**
[Offline]		
Ergebnisdatei	Datei-name	Name (und Pfad) der Ergebnisdatei
Fehlerdatei	Datei-name	Name (und Pfad) der Fehlerdatei
Synchronisierte Daten	Yes/No	Sollen die synchronisierten Daten verwendet werden? Bei Nein werden die Rohdaten verwendet.
[Action]		
Auto	Yes/No	Soll der Roboter die erste Aktionsdatei beim Programmstart automatisch abfahren?
Datei n	Datei-name	Der Name der Aktionsdatei n
Das letzte Feld wiederholt sich für alle 3 Aktionsdateien		

Tabelle B.1: *Das Format der Konfigurationsdatei von syncaria und lopscpp. Das Programm lopscpp verwendet nur die Daten im Abschnitt [Lops], das Programm syncaria die Daten aller Abschnitte.*

Name	**Format**	**Beschreibung**
		[Main]
Logdatei	Datei-name	Name der zu analysierenden Logdatei
Output	Verzeich-nisname	Name des Ausgabeverzeichnisses
Pfadplot	Yes/No	Sollen die Positionen geplottet werden?
LOPS-Plot	Yes/No	Sollen die LOPS-Entfernungen geplottet werden?
Gyroskopplot	Yes/No	Sollen die Gyroskopdaten geplottet werden?
Odometrieplot	Yes/No	Sollen die Enkoderinkremente geplottet werden?
Sonarplot	Yes/No	Sollen die Sonarsensordaten geplottet werden?
Timeplot	Yes/No	Sollen die Taktraten geplottet werden?
		[Position]
Format	String	Das gnuplot Ausgabeterminal mit Optionen
Position	Positions-name	Name der zu plottenden Position
Stil	String	Der Linienstil, mit dem dieser Pfad geplottet wird
Die letzten beiden Felder wiederholen sich für jede Position		
		[Odometry]
Format	String	Das gnuplot Ausgabeterminal mit Optionen
Stil	String	Der Linienstil, mit dem die Daten geplottet werden
Das letzte Feld wiederholt sich für beide Räder		
		[Gyroscop]
Format	String	Das gnuplot Ausgabeterminal mit Optionen
Stil	String	Der Linienstil, mit dem die Daten geplottet werden
		[Sonar]
Format	String	Das gnuplot Ausgabeterminal mit Optionen
Stil	String	Der Linienstil, mit dem die Daten geplottet werden
Das letzte Feld wiederholt sich für alle 16 Sonarsensoren		
		[Time]
Format	String	Das gnuplot Ausgabeterminal mit Optionen
Synchronisation	Yes/No	Soll die Synchronisation ausgewertet werden?
Stil	String	Der Linienstil, mit dem die Synchronisationsdaten geplottet werden
Sensor	Sensor-name	Name des auszuwertenden Sensors
Stil	String	Der Linienstil, mit dem die Daten geplottet werden
Die letzten beiden Felder wiederholen sich für jeden Sensor		

Tabelle B.2: *Konfigurationsdateiformat des Programms logfileanalyser. Die Möglichkeiten beim Graphenstil und beim Ausgabeterminal (Ausgabeformat und Optionen) sind in der Gnuplot-Dokumentation (Williams et al. 1998) beschrieben.*

Name	Inhalt
X-Position	Die X-Position der Mitte der Wand in mm
Y-Position	Die Y-Position der Mitte der Wand in mm
Richtung	Die Richtung der Wand in Grad
Länge	Die Länge der Wand in mm
Sichtbar X0	die linke Grenze des Sichtbarkeitsbereichs (gobale X-Koordinate in mm)
Sichtbar X1	die rechte Grenze des Sichtbarkeitsbereichs (gobale X-Koordinate in mm)
Sichtbar Y0	die untere Grenze des Sichtbarkeitsbereichs (gobale Y-Koordinate in mm)
Sichtbar Y1	die obere Grenze des Sichtbarkeitsbereichs (gobale Y-Koordinate in mm)
d	der Parameter d (Hessesche Normalform) der Wand in mm
α	der Parameter α (Hessesche Normalform) der Wand in Grad
Die Felder sind jeweils mit einem Leerzeichen getrennt.	

Tabelle B.3: *Das Format der Wand-Datei: Jede Zeile der Wanddatei beschreibt eine Wand. In dieser Zeile befinden sich 10 Felder, die die Position der Wand und den Sichtbarkeitsbereich festlegen.*

Befehl	Parameter	Beschreibung
#	keine	Kommentar
e	keine	Motoren anschalten
s	Fahrstrecke	Fährt die angegebene Strecke (in mm) geradeaus
r	Drehung	Dreht den Roboter um die angegebene Drehung (in Grad)
v	links rechts Zeit	Die beiden Antriebsräder werden mit den angegebenen Geschwindigkeiten rechts und links für die im Parameter Zeit angegebene Zeit (in ms) bewegt.
c	Radius Geschwind. Zeit	Der Roboter fährt die angegebene Zeit auf der durch die Parameter Geschwindigkeit und Radius (in mm) bestimmten Kreisbahn.

Tabelle B.4: *Das Format der Aktionsdateien: Jeder Fahrbefehl wird in einer eigenen Zeile angegeben. Diese Zeile besteht aus dem Befehl und seinen Parametern. Zwischen dem Befehl und den Parametern sowie zwischen den einzelnen Parametern muss jeweils ein Leerzeichen stehen.*

Felder eines Logeintrags		
Name	Grösse	Inhalt
Fehlerklasse	1 Zeichen	Die Fehlerklasse
Zeitstempel	9 Zeichen	Der Zeitpunkt des Eintrags
Fehlermeldung	beliebig	Die Fehlermeldung
Die Felder sind durch Leerzeichen getrennt		

Fehlerklassen		
Name	Zeichen	Bedeutung
Information	I	Information, kein Fehler
Warning	W	Fehler, der zu *keinem* Datenverlust geführt hat
Error	E	Fehler, der zum Verlust von Daten geführt hat
Fatal	F	Schwerwiegender Fehler, Programmabbruch
Debug	D	Debugnachricht, die bei der Fehlersuche helfen soll

Tabelle B.5: *Das Format der Fehlerlogdatei: Jede Fehlermeldung besteht aus einer Zeile, die in drei Felder unterteilt ist. Der untere Teil der Tabelle beschreibt die verschiedenen Fehlerklassen.*

Quellen		
Name	Grösse	Inhalt
Quelle	1 Zeichen	Das Modul, das den Eintrag geschrieben hat
Art	1 Zeichen	Die Art der Daten
Zeitstempel	9 Zeichen	Der Zeitpunkt des Eintrags
Daten	beliebig	Die Daten
Die Felder sind durch Leerzeichen getrennt		

Quellen	
Zeichen	Modul
S	System
C	Korrelator
G	Gyroskop
K	Kalmanfilter
L	LOPS
O	Odometrie
W	Wanderkennung

Art	
Zeichen	Bedeutung
C	Synchronisierte Daten
P	Positionsdaten
R	Rohdaten
S	Synchronisation beendet
T	Startzeit des Programms
W	Wanddaten

Tabelle B.6: *Das Format der Datenlogdatei Einträge: Diese Tabelle beschreibt den Aufbau der Datenlogdatei und die Bedeutung der Kurzzeichen. Das Format der Dateneinträge ist in Tabelle B.7 beschrieben.*

Namen	**Größe**	**Inhalt**
Startzeit des Programms (`S T`)		
Zeit	19 Zeichen	Startzeit im Format TT.MM.JJJJ HH:MM:SS
Positionsdaten (`x P`) / Synchronisation beendet (`S S`)		
X-Position	8 Zeichen	X-Position des Roboters in mm bzw. „0“, falls sie nicht bekannt ist.
Y-Position	8 Zeichen	Y-Position des Roboters in mm bzw. „0“, falls sie nicht bekannt ist.
Drehung	8 Zeichen	Drehung des Roboters in Radiant bzw. „0“, falls sie nicht bekannt ist.
X-Valid	1 Zeichen	„1“ wenn die X-Position bekannt ist, sonst „0“
Y-Valid	1 Zeichen	„1“ wenn die Y-Position bekannt ist, sonst „0“
Drehung-Valid	1 Zeichen	„1“ wenn die Drehung bekannt ist, sonst „0“
Gyroskopdaten (`G R` und `G C`)		
Drehrate	15 Zeichen	Drehrate des Gyroskops in Radiant
LOPS-Daten (`L R` und `L C`)		
Entfernung	8 Zeichen	Entfernung vom Roboter zum Empfänger n in mm
Das Feld wiederholt sich für alle drei Empfänger		
Odometriedaten (`O R` und `O C`)		
Links	8 Zeichen	Zählerstand des linken Radenkoders
Rechts	8 Zeichen	Zählerstand des rechten Radenkoders
Sonardaten (`W R` und `W C`)		
Die beiden letzten Felder wiederholen sich für alle 16 Empfänger		
Neu n	1 Zeichen	Ist der Wert des Sonarsensors n neu?
Das letzte Feld wiederholt sich für alle 16 Empfänger		
X-Position	5 Zeichen	X-Position des Hindernisses in mm (Roboterkoordinaten)
Y-Position	5 Zeichen	X-Position des Hindernisses in mm (Roboterkoordinaten)
Das letzte Feld wiederholt sich für alle 16 Empfänger		
Entfernung n	5 Zeichen	gemessene Entfernung des Sonarsensors n in mm
Das letzte Feld wiederholt sich für alle 16 Empfänger		
Wand erkannt	1 Zeichen	1 wenn eine Wand erkannt wurde, ansonsten 0
α	xx Zeichen	Der Winkel der Wand (Hessesche Normalform)
d	xx Zeichen	Der Abstand der Wand (Hessesche Normalform)
Zuordnung	2 Zeichen	Die Nummer der globalen Wand, die erkannt wurde
Die letzte vier Feld wiederholt sich für alle 9 Wände		
Die Datenfelder sind jeweils durch Leerzeichen getrennt		

Tabelle B.7: *Das Format der Dateneinträge in der Datenlogdatei. Das Format ist abhängig von der Art der gespeicherten Daten. Bei den Dateneinträgen, die synchronisierte Daten enthalten, befindet sich hinter den gennanten Feldern naoch ein 1 Zeichen langes Feld, das angibt, aus wievielen Datensätzen der synchronisierte Daten berechnet wurde.*

Name	Größe	Inhalt
Die Datei data.dat		
Zeitpunkt	8 Zeichen	Zeitpunkt der Synchronisation in ms seit Programmstart
Takt	4 Zeichen	Zeit seit der letzten Synchronisation in ms
Enkoder links	8 Zeichen	Enkoderinkrement des linken Rades in Ticks
Enkoder rechts	8 Zeichen	Enkoderinkrement des rechten Rades in Ticks
Gyroskop	15 Zeichen	Drehrate des Gyroskops in Radiant
Entfernung 1	8 Zeichen	Entfernung des Roboters vom ersten LOPS-Empfänger in mm
Entfernung 2	8 Zeichen	Entfernung des Roboters vom zweiten LOPS-Empfänger in mm
Entfernung 3	8 Zeichen	Entfernung des Roboters vom dritten LOPS-Empfänger in mm
LOPS-X	8 Zeichen	Vom LOPS-System berechnete X-Position des Roboters in mm bzw. „0“, falls diese nicht berechnet werden konnte
LOPS-Y	8 Zeichen	Vom LOPS-System berechnete Y-Position des Roboters in mm bzw. „0“, falls diese nicht berechnet werden konnte
LOPS-Valid	1 Zeichen	„1“, falls die Position vom LOPS-System berechnet werden konnte, ansonsten „0“
X-Position	8 Zeichen	X-Position des Roboters in mm bzw. „0“, falls sie nicht bekannt ist.
Y-Position	8 Zeichen	Y-Position des Roboters in mm bzw. „0“, falls sie nicht bekannt ist.
Drehung	8 Zeichen	Drehung des Roboters in Radiant bzw. „0“, falls sie nicht bekannt ist.
X-Valid	1 Zeichen	„1“ wenn die X-Position bekannt ist, sonst „0“
Y-Valid	1 Zeichen	„1“ wenn die Y-Position bekannt ist, sonst „0“
Drehung-Valid	1 Zeichen	„1“ wenn die Drehung bekannt ist, sonst „0“
Die Datei sonar.dat		
X-Position n	6 Zeichen	X-Position des Hindernisses des n-ten Sonarsensors in mm (Roboterkoordinaten)
Y-Position n	6 Zeichen	Y-Position des Hindernisses des n-ten Sonarsensors in mm (Roboterkoordinaten)
Die beiden letzten Felder wiederholen sich für alle 16 Empfänger		
Die Datenfelder sind jeweils durch Leerzeichen getrennt		

Tabelle B.8: *Das Format der Ausgabedateien des Programms logfileconverter. Die Daten einer Synchronisation stehen in jeweils einer Zeile der beiden Dateien data.dat und sonar.dat*

Name	Größe	Inhalt
Gyroskop/Gyroskop		
Zeit	8 Zeichen	Der Zeitpunkt der Messung in ms seit Programmstart
Drehrate	15 Zeichen	Die Drehrate in Radiant
Lops/Empfaenger n		
Zeit	8 Zeichen	Der Zeitpunkt der Messung in ms seit Programmstart
Entfernung	8 Zeichen	Die Entfernung des Roboters vom Empfänger in mm
Diese Datei ist für jeden der drei Empfänger vorhanden		
Odometrie/Links und Odometrie/Rechts		
Zeit	8 Zeichen	Der Zeitpunkt der Messung in ms seit Programmstart
Entfernung	8 Zeichen	Das Enkoderinkrement des Rades
Diese Datei ist für beide Enkoder vorhanden		
Position/Modul		
X-Position	8 Zeichen	Die X-Position des Roboters in mm
Y-Position	8 Zeichen	Die Y-Position des Roboters in mm
Diese Datei ist für jede untersuchte Position vorhanden		
Sonar/Empfaenger n		
Zeit	8 Zeichen	Der Zeitpunkt der Messung in ms seit Programmstart
Entfernung	8 Zeichen	Die Entfernung bis zum Hindernis in mm
Diese Datei ist für jeden der 16 Empfänger vorhanden		
Time/Modul		
Zeit	8 Zeichen	Der Zeitpunkt der Messung in ms seit Programmstart
Dauer	5 Zeichen	Die Dauer seit dem letzten Ereignis
Diese Datei ist für jedes untersuchte Modul vorhanden		
Die Datenfelder sind jeweils durch Leerzeichen getrennt		

Tabelle B.9: *Das Format der logfileanalyser-Dateien: Die Dateien können mit Hilfe von gnuplot geplottet werde. Außerdem sind sie für statistische Untersuchungen geeignet.*

Anhang C

Benötigte Programme und Bibliotheken

Zur Übersetzung und Benutzung des Programms werden einige externe Programme und Bibliotheken benötigt. Die meisten dieser Softwarepakete sind in einer normalen Linux-Distribution enthalten, die anderen sind auf der CD zur Diplomarbeit zu finden. Das größte benötigte Softwarepaket ist die Bibliothek ARIA, die zur Kommunikation mit dem Mikrocontroller des Roboters benötigt wird. Neben dieser Bibliothek werden noch einige kleinere Programme und Bibliotheken zur Verarbeitung der Logdateien sowie zum Erstellen der Dokumentation benötigt. Diese werden im Folgenden beschrieben.

C.1 ARIA — ActivMedia Robotics Interface for Aplication

Zur Ansteuerung des Roboters wird die Programmbibliothek ARIA (ActiveMedia Robotics, LLC 2003) eingesetzt. Diese bietet ein objektorientiertes Interface zur Ansteuerung der Roboter von ActivMedia. ARIA ist in C++ geschrieben und bietet Schnittstellen für die Sprachen Java und Python. Das Softwarepaket ist unter der Open Source Lizenz GNU GPL (GNU General Public License) veröffentlicht und kann daher vom Anwender an seine eigenen Bedürfnisse angepasst werden.

Zur Verwendung der Programme ist die Installation von ARIA nötig. Die Entwicklung und die Tests basieren auf der Version 1.3.0 vom 8.8.2003. Die Software kann vom ActivMedia-Server[1] bezogen werden und ist auf der CD enthalten.

Bei der Installation des RPM-Paketes traten jedoch einige Probleme auf, die durch Anpassungen im Makefile und in den Sourcecode Dateien behoben werden mussten:

- Im Makefile musste aus den Zeilen, die mit *CXXLINK=* und *CXXSTATICLINK=* beginnen, die Bibliothek *-lstdc++* entfernt werden.

[1] www.activrobots.com

- Die Datei `examples/paramTest.cpp` musste gelöscht werden (bzw. ins Verzeichnis `archieves` verschoben werden). Anscheinend sind die Klassen, die in diesem Test verwendet werden, nicht mehr Teil der aktuellen ARIA Version. Die `cpp`-Dateien der von `examples/paramTest.cpp` benötigten Klassen liegen im Verzeichnis `archieves`.
- In der Datei `src/ArConfig.cpp` musste die Zeile 245 geändert werden: Statt „*ArConfig::parseFile(const char *filename, bool continueOnErrors=false,*“ muss es „*ArConfig::parseFile(const char *filename, bool continueOnErrors,*“ heißen.
- In der Datei `src/ArArg.cpp` musste die Zeile 235 verbessert werden: „*return false;*“ muss in „*return NULL;*“ geändert werden.

Die so geänderte Version konnte anschließend mit `make everything` übersetzt und mit *make install* installiert werden. Nach der Installation, war es erforderlich die Dateirechte neu zu setzen, da nach der Installation sämtliche Dateien für alle Benutzer schreibbar sind und daher (vor allem wenn die Bibliothek vom Benutzer root nach `/usr/local/Aria` installiert wird) Angriffspunkte auf die Systemsicherheit bieten.

Auf der CD zu dieser Diplomarbeit ist auch das Paket Aria.tar.bz2 beigefügt, das die ARIA-Software enthält, und bei dem die nötigen Anpassungen bereits gemacht wurden. Auch sind die Benutzerrechte in diesem Paket restriktiv gesetzt (nur der Benutzer hat Schreibrechte, andere Nutzer haben nur Leserechte). Dieses Paket kann an eine beliebige Stelle im Dateisystem installiert werden, es sind (bei Installation ins eigene Home-Verzeichnis) keine root-Rechte nötig.

Um die installierte Bibliothek nutzen zu können, muss sie der Linker beim Programmstart finden können. Dazu kann der Pfad zu `Aria/lib/` zum Beispiel in der Umgebungsvariable `LD_LIBRARAY_PATH` eingetragen sein. Damit ARIA seine Paramterdateien (in denen Daten zu den Robotern stehen) finden kann, muss der ARIA-Pfad entweder in der Datei `/etc/Aria` oder in der Umgebungsvariablen `ARIA` stehen. Sind beide Einträge vorhanden, wird die Umgebungsvariable verwendet.

Neben einer korrigierten Version des Sourcecodes enthält das erstellte tar-Archiv auch eine erweiterte Dokumentation. Die Dokumentation des ARIA-Softwarepakets wird mit dem Tool doxygen erstellt. Durch Änderung einiger doxygen-Parameter wurde erreicht, dass die Dokumentation einer Klasse auch alle geerbten und private / protected Klassenelemente enthält. Das Aufnehmen der geerbten Klassenelemente erleichtert die Verwendung einer Klasse, da der Entwickler alle ihre Elemente auf einer Seite sieht. Die Aufnahme der protected Elemente erleichtert es dem Entwickler, eine Klasse zu entwerfen, die von einer ARIA-Klasse erbt, denn Klassen haben Zugriff auf protected-Elemente ihrer Eltern.

Außerdem enthält die HTML-Version den Sourcecode der Klassen, so dass man bei Unklarheiten in der Dokumentation direkt im Code kontrollieren kann, was eine Funktion genau bewirkt. Die PDF-Version der Dokumentation wurde um Hyperlinks, mit denen man leicht den im Dokument vorhandenen Verweisen folgen kann, erweitert.

C.2 NCURSES

Für die Benutzerschnittstelle des Programms syncaria wird die NCURSES-Bibliothek benötigt. Diese Bibliothek musste gepatcht werden, um die Zusammenarbeit mit ARIA zu ermöglichen. Mit der normalen Version, die auch auf den Rechnern des RTR installiert ist, ist es nicht möglich den Roboter zu steuern, da mit ihr die Verbindung zum Mikrocontroller (und auch zum Simulator) unterbrochen wird.

Nachdem bei der Version 5.3 der NCURSES-Bibliothek die Zeile 129 in der Datei `ncurses-5.3/ncurses/tinfo/lib_raw.c` (`buf.c_cc[VMIN] = 1;`) auskommentiert wurde, trat das Problem nicht mehr auf. Auswirkungen dieser Änderung auf andere Funktionen sind nicht aufgefallen. Soll die geänderte NCURSES-Bibliothek kompiliert werden, so ist zu beachten, dass die benötigten shared-libraries nur dann erzeugt werden, wenn das `configure`-Skript mit dem Parameter `--with-shared` aufgerufen wird.

Die geänderte Version der NCURSES-Bibliothek ist auf der CD enthalten und auf den Rechnern des RTR sowie dem Onboardrechner des Roboter unter `/home/kallnik/tools/lib/libncurses.so` installiert. Diese Bibliothek muss anstelle der ebenfalls installierten Standardversion verwendet werden. Dazu kann das Verzeichnis, in dem die Bibliothek installiert ist, in die Umgebungsvariable `LD_LIBRARY_PATH` aufgenommen werden. Diese Variable sollte jedoch nur während der Ausführung des Programms syncaria gesetzt sein, da nicht bekannt ist, ob die Änderung negative Auswirkungen auf anderer Programme hat.

C.3 Benötigte Programme

Für die Auswertung der Logdateien mit Hilfe der Programme logfilesorter und logfileanalyser sowie zum Erstellen der API-Dokumentation werden einige externe Programme verwendet. Diese müssen jedoch nicht auf dem Roboter installiert sein, da sie nicht zum Aufzeichen der Logdateien benötigt werden.

Im einzelnen werden die folgenden Programme benötigt:

gnuplot gnuplot wird verwendet, um die Logdateien zu plotten. Mit Hilfe des Plottprogramms können die von den einzelnen Modulen bestimmten Roboterpositionen als Pfad geplottet werden oder einzelne Sensorwerte über die Zeit abgetragen werden. (Siehe auch Abschnitt 9.2.)

sort Das GNU-Textutility sort wird verwendet, um die Logdateien nach der Aufzeichnung nach dem Zeitstempel zu sortieren. (Siehe auch Abschnitt 9.1).

doxygen Das Tool doxygen wird zum Erstellen der API-Dokumentation benötigt. Es wandelt die Kommentare in den Sourcedateien in eine Dokumentation um.

dot Das Programm dot wird von doxygen benötigt, um Graphen zu erzeugen. Dieses Programm gehört zu dem graphviz-Paket.

LaTeX Um die API-Dokumentation im PDF-Format erzeugen zu können, ist eine LaTeX-Installation mit pdflatex nötig.

Die Programme sort, gnuplot, doxygen und LaTeX sind auf den Rechnern des RTR installiert, das Programm dot ist auf der CD erhalten und liegt auf den Rechnern des RTR im Verzeichnis `/home/kallnik/tools/bin/dot`.

C.4 Weitere benötigte Bibliotheken

Neben NCURSES und ARIA werden weitere Bibliotheken von syncaria benötigt. Die ARIA-Bibliothek verwendet zwei weitere Bibliotheken: libpthread, die die Threadverwaltung übernimmt, und libdl, die das dynamische Laden von Bibliotheken erlaubt. Beide Bibliotheken gehören unter Linux zu den Standardpaketen (glibc) und sind somit auf allen Linux-Systemen vorhanden.

Der Kalmanfilter verwendet zur Matrizenrechnung die GNU Scientific Library (gsl) und die Bibliothek gslwrapper, die einen objektorientierten Zugriff auf die gsl-Bibliothek erlaubt. Diese Bibliotheken sind auf dem Roboter und den Rechnern des RTR installiert sowie auf der CD enthalten.

Anhang D

Installation

D.1 syncaria

Das Synchronisationsprogramm syncaria muss auf dem Roboter installiert werden, zur Benutzung des Offlinemodus empfiehlt sich auch die Installation auf einem PC.

Das Programm ist auf der CD im Archiv `syncaria.tgz` enthalten. Es muss im Verzeichnis `~/code/` ausgepackt werden (`tar -xzf syncaria.tgz`). Anschließend wechselt man in das Verzeichnis `syncaria` und kompiliert das Programm mit den Befehlen `./configure` und `make`. Eine systemweite Installation ist als Benutzer root mit dem Befehl `make install` möglich, jedoch nicht notwendig.

Vor der Installation muss weiterhin sichergestellt werden, dass alle benötigten Programme und Bibliotheken installiert sind (siehe Anhang C).

D.2 Hilfsprogramme

Die Programme lopscpp und logfileanalyser werden wie das Programm syncaria installiert. Der logfilesorter ist lediglich ein Shell-Skript. Dieses muss in ein Verzeichnis kopiert werden, das im Pfad liegt.