Institut für Informatik VII
Robotik und Telematik

Bachelorarbeit

Implementierung eines Controllers zur Steuerung eines mobilen Roboters mit virtueller Spurstange

Sven Jörissen

September 2015

Erstgutachter: Prof. Dr. Andreas Nüchter
Betreuer: MSc. Dorit Borrmann

Zusammenfassung

Die vorliegende Arbeit befasst sich mit der Erarbeitung eines ROS Package zur Steuerung des mobilen Roboters ARTEMIS mit Ackermann Lenkung aus der *Volksbot* Serie des Fraunhofer Instituts für Intelligente Analyse- und Informationssysteme. Die Controller zur Ansteuerung der Motoren sowie die Motoren selber sind von der Firma *Maxon Motor AG* hergestellt. Die Steuerung besteht aus einer Schleife, in der für jeden neuen Fahr- und Lenkbefehl die notwendigen Parameter wie Lenkwinkel, Fahr- und Drehgeschwindigkeiten sowie Beschleunigung und Abbremsung der jeweiligen Räder neu berechnet werden. Intern werden diese in SI-Einheiten vorliegenden Parameter in maxon-spezifische Einheiten umgerechnet und an die Controller übergeben. Als Schnittstelle zwischen PC und Controller kommt ein USB-to-CAN Adapter der Firma *Peak System* zum Einsatz. Zur Steuerung stehen wahlweise ein 3-Achsen-Joystick oder ein Lenkrad mit Schaltknöpfen sowie Gas- und Bremspedal zur Verfügung. Der Roboter ist mit einem Laserscanner der Firma *SICK* ausgestattet, der über Ethernet mit dem PC verbunden ist. Damit können durch den Einsatz von *gmapping* und *amcl* Karten unbekannter Umgebungen erstellt werden und der Roboter kann sich in diesen lokalisieren. Durch den Einsatz von zwei Blei-Gel Akkus mit jeweils 12 V und 45 Ah ist gewährleistet, dass auch bei längeren Einsätzen die Akkus nicht nachgeladen werden müssen. Weiterhin steht so genug Energie zur Verfügung, um zusätzliche Sensoren wie Wärmebildkameras oder 3D-Scanner anzuschließen, ohne dass eine weitere, externe Stromversorgung notwendig ist. Durch die Implementierung einer Pfadverfolgung nach einem Regler von G. Indiveri und M. L. Corradini ist es auch möglich, den Roboter vorgegebene Pfade selbständig per Odometrie- oder Amcldaten nachfahren zu lassen.

Danksagung

- Zuerst möchte ich mich bei meinem Professor, **Andreas Nüchter**, für die Vergabe des Themas und die Unterstützung beim Tuning und Kauf von Zubehör bedanken.
- Der nächste Dank gilt meiner Betreuerin **Dorit Borrmann** für die vielen geopferten Stunden kostbarer Zeit, sei es zur Beantwortung von Fragen, zur Fehlersuche, zur Unterstützung bei Experimenten oder zum Korrekturlesen und Verbesserungsvorschläge schreiben.
- Für die Unterstützung bei der Verwendung von iSpace und dem Lösen des Odometrieproblems danke ich **Robin Hess**.
- Bei **Florian Kempf** bedanke ich mich für die Fehlersuche in der Odometrieberechnung sowie für den Nachschub an Nahrung und die Bereitstellung eines Gewindeschneiders.
- Ein besonderer Dank geht an **Bertram Koch** für die schwarzen Kaffeepausen und die Auflockerung der Arbeitsatmosphäre im Control Lab.
- Für das Korrekturlesen danke ich meiner Mutter, **Andrea Jörissen**, meiner Freundin, **Birte Balbinot**, sowie **Christoph Hagen** und **Bertram Koch**.

Inhaltsverzeichnis

Abbildungsverzeichnis

Symbolverzeichnis

α Lenkwinkel des virtuellen Rades

B_A Abstand der Radaufhängungen der Vorderachse

Δd Zurückgelegter Weg in einem Zeitintervall

h, γ Parameter des Reglers

$\kappa(s)$ Krümmung einer Kurve

L Abstand zwischen Vorder- und Hinterachse

$\Delta\Phi$ Änderung der Orientierung

R Radius des Kreises, den das virtuelle Rad fährt

R_H Abstand zwischen dem Kreismittelpunkt und dem virtuellen Rad auf der Hinterachse

S Spurbreite

Δt Zeitintervall

θ Orientierung des Roboters relativ zur x-Achse

Δv Änderung der Geschwindigkeit in einem Zeitintervall

v_0 Geschwindigkeit des virtuellen Rades

ω_0 Winkelgeschwindigkeit des virtuellen Rades

Die im Text verwendeten Indizes i und a bzw. l und r beziehen sich auf das kreisinnere und -äußere bzw. das linke und rechte Rad.

Kapitel 1

Einleitung

Roboter sind im 21. Jahrhundert ein nicht mehr wegzudenkender Teil des Lebens. Beispielhaft hierfür stehen stationäre Industrieroboter, die in Fabriken monotone, gleich bleibende Aufgaben präzise erfüllen und so die Produktivität und Qualität eines Betriebes deutlich erhöhen. Zudem können sie in für den Menschen kritischen und gefährlichen Umgebungen eingesetzt werden und gesundheitsgefährdende Aufgaben übernehmen. Im Gegensatz zu stationären Robotern stehen die mobilen Roboter. Im privaten Bereich nimmt das Interesse an Servicerobotern wie beispielsweise Staubsauger- oder Fensterputzrobotern in den letzten Jahren stetig zu. Eine weitere Kategorie, unter die auch der in dieser Arbeit verwendete Roboter fällt, sind die Forschungsroboter. Der in den letzten Jahren wohl berühmteste Vertreter ist der Mars-Rover „Curiosity“, der mit einer Vielzahl von Messgeräten zur Erkundung von Marsoberfläche und -atmosphäre sowie der Suche nach Leben ausgestattet ist. Ein anderes Einsatzgebiet mobiler Roboter ist die 2D-Kartierung oder 3D-Modellerstellung von einsturzgefährdeten archäologischen Funden, um diese für die Nachwelt zu sichern, ohne dabei den Menschen zu gefährden. Zu letzterer Kategorie zählen auch die beiden Roboter in Abbildung 1.1.

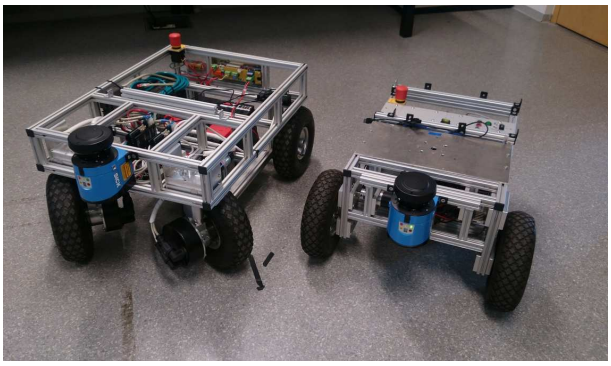


Abbildung 1.1: Volksbot Ackermann (links) und Volksbot RT3-2 (rechts) [Fra]

1.1 Zielsetzung und wissenschaftliche Relevanz

Ziel dieser Arbeit ist die Implementierung eines Controllers, der die Steuerung des *Volksbot Ackermann Lenkung* [Fra] ermöglicht und die Verwendung dessen im Zusammenspiel mit dem *Robot Operating System* [Wil]. Des Weiteren soll mit Hilfe eines Laserscanners und der Odometrie die Möglichkeit gegeben werden, eine anfangs unbekannte Umgebung zu kartieren, sich in dieser zu lokalisieren und aufgezeichnete Pfade selbstständig nachzufahren. Hierfür wird eine Implementierung eines Reglers von G. Indiveri und M. L. Corradini verwendet [IC04]. Durch den Einsatz der Ackermann Lenkung hat der Roboter, im Vergleich zu einem Volksbot RT3 mit Differentialantrieb, zwei entscheidende Vorteile. Einerseits ist die Stabilität bei Kurvenfahrten erhöht, da durch die Ackermann Geometrie ein ausbrechen der Räder verhindert wird. Andererseits werden die Schräglaufwinkel, die durch die Bewegung der Reifen entgegen ihrer Orientierung entstehen, und die daraus resultierenden Querkräfte auf die Reifen, vor allem bei engen Kurvenfahrten und hoher Gewichtsbelastung, minimiert. Die Dimensionierung des Fahrzeugs erlaubt außerdem den Einsatz stärkerer Batterien und bietet Platz für zusätzliche Nutzlast.

1.2 Aufbau der Thesis

Die Arbeit ist in sechs Hauptkapitel unterteilt.

Nach dieser *Einleitung* folgt im Kapitel *Mobile Roboter* eine allgemeine Einführung zu mobilen Robotern mit dem Versuch einer Definition und einer Klassifizierung von mobilen Robotern nach deren Antriebsart mit Beispielen.

Im Kapitel *Theoretische Grundlagen* wird zunächst das Volksbot Baukastensystem und insbesondere der Volksbot Ackermann eingeführt. Anschließend wird auf dessen Antrieb und die verwendeten Sensoren eingegangen. Danach wird das Robot Operating System (ROS) vorgestellt. Auch wird die Kinematik mobiler Roboter erörtert sowie die im Controller verwendeten Formeln zur Steuerung und Regelung des Roboters erklärt. Darauf folgt die Bestimmung der Odometrie sowie eine Einführung zum CAN-Bus. Die Vorstellung von GMapping, AMCL und *iSpace* sowie die Erläuterung des verwendeten Reglers zur Pfadverfolgung schließen das Kapitel ab.

Im Kapitel *Technische Ausarbeitung* wird zunächst die Schnittstelle zwischen dem PC und den EPOS Controllern beschrieben. Danach werden zwei unterschiedliche Systeme zur Steuerung des Roboters, wahlweise über einen Joystick oder ein Lenkrad erfolgt, erklärt. Anschließend wird das entwickelte ROS Package vorgestellt. Hier wird zunächst das Kalibrierungssystem besprochen, anschließend der Aufbau und die Struktur der Ackermann Library erläutert und zuletzt die Anbindung an ROS beschrieben.

Im Kapitel *Experimente* wird die Kartierung und Lokalisierung mit *gmapping* und *amcl* beschrieben. Anschließend wird die eine Simulation zur Berechnung der Odometriedaten vorgestellt. Die Experimente zur Pfaderzeugung- und verfolgung und eine Diskussion der Ergebnisse schließen das Kapitel ab.

Im letzten Kapitel werden die Ergebnisse zusammengefasst. Ein Ausblick auf mögliche Optimierungen des Prozesses und Verbesserungen der Qualität der Experimente gegeben sowie ein Fazit schließen die Arbeit ab.

Kapitel 2

Mobile Roboter

Durch die steigende Beliebtheit und zunehmende Verbreitung mobiler Roboter in Forschung und Lehre ist es notwendig, den Begriff der Robotik zu überdenken und anzupassen. Im Folgenden wird zunächst versucht, eine allgemeine Definition für mobile Roboter zu finden, danach werden diese anhand Ihrer Fortbewegungsart kategorisiert und Beispiele zu den jeweiligen Gruppen gegeben.

2.1 Versuch einer Definition

Der folgende Abschnitt orientiert sich am Buch „Mobile Roboter - Eine Einführung aus Sicht der Informatik“ von J. Hertzberg, K. Lingemann und A. Nüchter [HLN12], Kapitel 1.

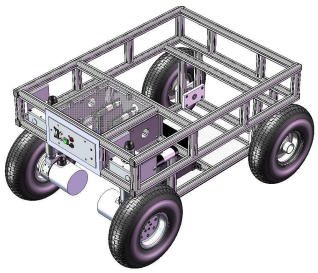
Die allgemeine Definition für Roboter kommt aus der VDI-Richtlinie 2860 von 1990:

> „Ein Roboter ist ein frei und wieder programmierbarer, multifunktionaler Manipulator mit mindestens drei unabhängigen Achsen, um Materialien, Teile, Werkzeuge oder spezielle Geräte auf programmierten, variablen Bahnen zu bewegen zur Erfüllung der verschiedensten Aufgaben.“ ([HLN12, S.2])

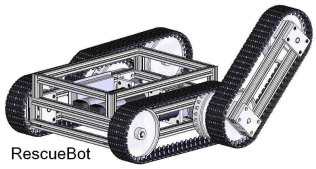
Diese Definition trifft vorrangig auf Industrieroboter zu, da diese in einem bekannten, kontrollierten und abgeschlossenen Umfeld auf „programmierbaren, variablen Bahnen“ arbeiten. Aufgrund dieser Tatsache kommt bei Industrierobotern eine offene Steuerung zum Einsatz, da diese keinen äußeren Störungen ausgesetzt sind und für jeden Produktionsschritt denselben Ablauf von Befehlen abarbeiten müssen.

Mobile Roboter unterscheiden sich von dieser Definition dadurch, dass diese sich in zumeist unbekannten Umgebungen fortbewegen und ihre Aktionen somit der Umgebung und äußeren Einflüssen anpassen müssen. Die Umgebungserfassung wird mit Hilfe verschiedenster Sensoren durchgeführt, deren Daten der Roboter dann auswertet und die Ausführung des Programms entsprechend anpasst. Hierzu wird im Unterschied zu den Industrierobotern eine geschlossene Regelung mit Rückkopplung verwendet.

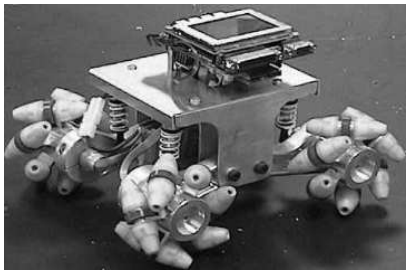
(a) Volksbot RT4 [Fra]



(b) Volksbot Ackermann [Fra]



(c) Volksbot Rescue Bot [Fra]



(d) Omni-2 [Bra06]

(e) Sony SDR-4X [FKID03]

Abbildung 2.1: Mobile Roboter mit unterschiedlichen Antrieben

Zur weiteren Charakterisierung wird die in [HLN12, S.3] eingeführte Definition für Roboter allgemein verwendet.

> „Unter einem Roboter verstehen wir [...] eine frei programmierbare Maschine, die auf Basis von Umgebungssensordaten in geschlossener Regelung in Umgebungen agiert, die zur Zeit der Programmierung nicht genau bekannt und/oder dynamisch und/oder nicht vollständig erfassbar sind."

Für einen *mobilen Roboter* gilt zusätzlich, dass er sich in der Umgebung, in der er agiert, fortbewegen kann. Wird ein Roboter als *autonomer* Roboter bezeichnet, bedeutet dies, dass er sich in seiner Umgebung unabhängig von Datenübertragung und externer Energiezufuhr bewegt. Alle Aktionen werden vom Roboter autonom als Ergebnis verschiedenster Algorithmen ausgeführt, solange die interne Energiequelle, wie beispielsweise Akkus oder Brennstoffzellen, genügend Energie liefert, das Programm nicht beendet wird und fehlerfrei läuft. Dieser Zusatz wird im Weiteren aufgrund der inkonsistenten Verwendung in der Literatur jedoch weggelassen, auch wenn er auf den für diese Arbeit verwendeten Roboter im Bezug auf die Pfadverfolgung zutrifft.

2.2 Klassifizierung mobiler Roboter

Wie bereits in der Einleitung angedeutet, lassen sich Roboter in verschiedene Kategorien einteilen und nach verschiedenen Kriterien klassifizieren. Den in Abbildung 2.1 dargestellten Robotern

liegt jeweils ein unterschiedliches Konzept der Fortbewegung zugrunde.

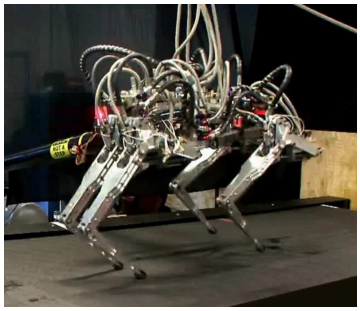
Abbildung 2.1(a) zeigt den Volksbot RT4 mit Differentialantrieb. Man spricht von einem Differentialantrieb, wenn der Roboter zwei unabhängig voneinander angetriebene Räder besitzt, die sich auf einer Achse befinden. Drehen beide Räder mit der selben Geschwindigkeit, fährt der Roboter eine gerade Linie. Eine Kurvenfahrt kommt dadurch zustande, dass die Räder mit unterschiedlichen Geschwindigkeiten drehen, die Differenz der Geschwindigkeiten nicht Null ist.

In Abbildung 2.1(b) ist ein CAD-Modell des Volksbot Ackermann zu sehen. Auf die Funktionsweise der Ackermann Lenkung wird in Kapitel 3.1 *Volksbot Ackermann Lenkung* eingegangen.

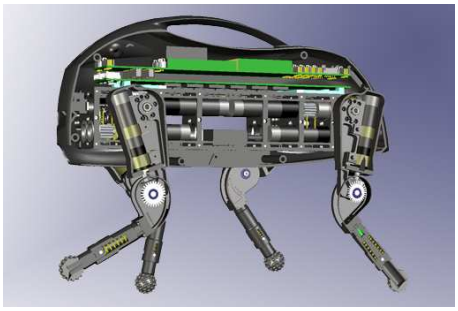
Abbildung 2.1(c) zeigt den *Volksbot Rescue Bot*. Die Räder sind hier, wie bei einem Panzer, durch Ketten verbunden. Ein Kettenantrieb verwendet dasselbe Prinzip wie ein Differentialantrieb. Diese Art des Antriebs kommt vorrangig im Gelände zum Einsatz, da durch die Ketten die Auflagefläche vergrößert und somit bei schweren Fahrzeugen der Druck auf den Boden besser verteilt wird. Dadurch wird der Untergrund geschont und das Fahrzeug kann bei weichem Untergrund nicht so sehr einsinken.

Abbildung 2.1(d) zeigt den Roboter Omni-2 der *University of Western Australia* mit omnidirektionalem Antrieb. Ein Roboter dieser Art kann sich, ohne eine Drehung durchführen zu müssen, in jede beliebige Richtung bewegen. Möglich ist dies durch den Einsatz von sogenannten Allseitenrädern, bei denen die Auflagefläche durch einen Ring mit Rollen, deren Längsachse nicht senkrecht zur Ringachse steht, ersetzt wird. Als bekanntester Vertreter gilt das in Abbildung 2.1(d) am Roboter angebrachte *Mechanum-Rad* des schwedischen Erfinders Bengt Ilon, bei dem die Rollen um 45 ° gedreht sind. Durch diese spezielle Anordnung wird beim Drehen der Räder die entstehende Kraft in zwei Kräfte aufgeteilt. Eine Kraft in Richtung der Drehrichung des Rades und die zweite Kraft in Richtung der Längsachse der Rollen. Ein großer Nachteil dieser Art der Räder liegt in der nicht trivialen Implementierung der Steuerung aufgrund des komplexen kinematischen Modells. Weitere Informationen zu Allseitenrädern finden sich in [HLN12], Kapitel 4.1.1 und [Bra06], Kapitel 8.

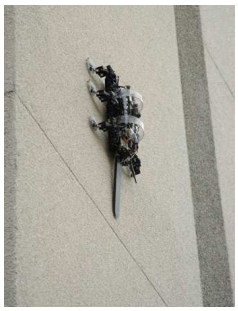
Der *Sony SDR-4X* aus Abbildung 2.1(e) gehört zur Gruppe der laufenden Roboter. Diese werden nach der Anzahl ihrer Beine spezifiziert. Die meisten dieser Roboter haben zwei (Mensch), vier (Hund), sechs (Insekten) oder acht (Spinnentiere) Beine. Wenige Modelle haben mehr Beine um beispielsweise einen Tausendfüßler zu imitieren. Der vierbeinige Roboter CHEETAH (Abbildung 2.2(a)) der Firma *Boston Dynamics* [Bos] ist in der Lage, sich mit einer Höchstgeschwindigkeit von 46,67 $\frac{km}{h}$ vorwärts zu bewegen. Der vierbeinige Roboter LittleDog (Abbildung 2.2(b)) wurde speziell für die Forschung an Fortbewegungsmöglichkeiten entwickelt und wird aktuell unter anderem am *Massachusetts Institute of Technology* (MIT) verwendet [Bos]. Der Roboter RiSE (Abbildung 2.2(c) ist dank der Mikrogripper an seinen Füßen in der Lage, senkrechte Wände oder Bäume hinaufzuklettern [Bos]. Der achtbeinige Roboter SCORPION (Abbildung 2.2(d)) des *Deutschen Forschungszentrums für Künstliche Intelligenz* wurde entwickelt, um sich auf einem extrem steilen und unstrukturierten Untergrund fortbewegen zu können [SK07]. Auch für den privaten Bereich werden zunehmends laufenden Roboter entwickelt, wie der in Abbildung 2.2(e) zu sehende Roboter Phoenix der Firma *Lynxmotion*. Einen großen Einfluss auf die Anzahl der Beine und die Konstruktionsart hat, wie die eben genannten Beispiele verdeutlichen, die Bionik.



(a) CHEETAH [Bos]



(b) LittleDog [Bos]



(c) RiSE [Bos]

(d) SCORPION [SK07]

(e) Phoenix 18 [Lyn]

Abbildung 2.2: Laufende Roboter mit unterschiedlicher Anzahl an Beinen

Bionik ist eine Wissenschaftsdisziplin, bei der biologische Phänomene auf eine technische Ebene übertragen werden, um sich diese zu Nutzen zu machen.

Neben den bereits erwähnten Antriebsarten gibt es eine ganze Reihe weiterer Formen, wie zum Beispiel Roboter, die in einem Ball eingeschlossen sind und sich durch Gewichtsverlagerung oder Schwungräder fortbewegen. Auf eine weitere Ausführung wird an dieser Stelle verzichtet und auf die ROS-Homepage [Wil] verwiesen, die eine umfangreiche Sammlung verschiedenster Robotertypen bietet.

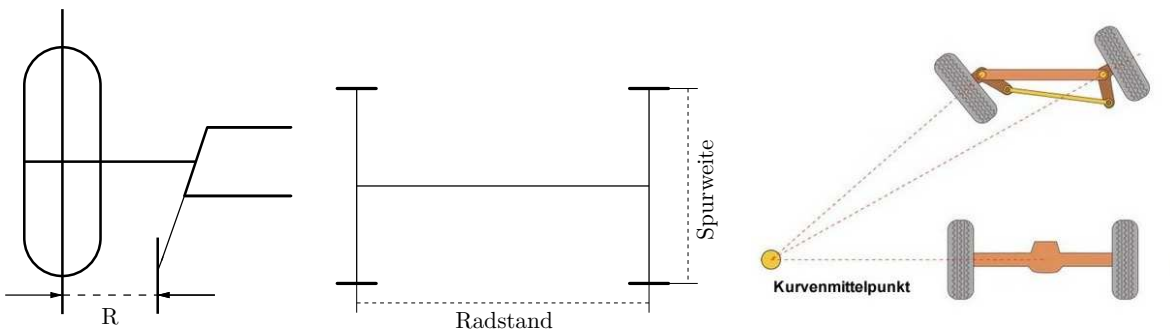
Kapitel 3

Theoretische Grundlagen

3.1 Volksbot Ackermann Lenkung

Volksbot ist ein vom *Fraunhofer Institut für Intelligente Analyse- und Informationssysteme* (IAIS) entwickeltes Baukastensystem für mobile Roboter [Fra]. Das Ziel dieses Projektes ist es, kostengünstig modulare, mobile Roboter zur Prototypen-Realisierung, Forschung und Entwicklung sowohl im universitären als auch im industriellen Bereich anzubieten. Volksbot bietet vorrangig Systeme mit Differentialantrieb an, da die Steuerung hierfür leicht zu implementieren ist. Eine Ausnahme im Programm bildet der *Volksbot Ackermann Lenkung*.

Die Ackermann- oder auch Achsschenkellenkung wurde 1816 nach der Idee von Georg Lankensperger durch Rudolf Ackermann patentiert. Die Schwierigkeit herbei besteht darin, die korrekten Lenkwinkel (oder Ackermann Winkel) für jedes Rad zu finden. Damit sich die Senkrechten auf die Räder im Mittelpunkt der Kurve schneiden, muss das kurveninnere Rad weiter eingelenkt werden als das kurvenäußere. Die klassische bei Fahrzeugen verwendete Lenkung realisiert die korrekte Einstellung der Winkel durch ein Lenktrapez (Abbildung 3.1(c)). Bei der für diese Arbeit ver-



(a) Lenkrollradius R (b) Radstand und Spurweite (c) Ackermann Lenkung

Abbildung 3.1: Grundbegriffe des Fahrwerks und Prinzip der Ackermann Lenkung

wendeten Lenkung werden die korrekten Lenkwinkel durch die Software berechnet. Diese hängen von Radstand, Spurweite und Lenkrollradius des Fahrzeugs ab (Abbildung 3.1). Somit kann das Fahrzeug nach Belieben dimensioniert und die Lenkwinkel deutlich größer gewählt werden. Für Autos liegt der maximalen Einschlagwinkel gewöhnlich im Bereich von 45 bis 50 °([Trz10], S. 446), mit physikalisch nicht verbundenen Rädern sind theoretisch beliebige Winkel möglich. Auf die Umsetzung in diesem Modell wird im Kapitel 4.2 *Steuerung des Roboters* genauer eingegangen. Der Fahrantrieb befindet sich auf der gelenkten Achse, um auch bei großen Lenkwinkeln gute Fahreigenschaften zu gewährleisten. Für den *Volksbot Ackermann Lenkung* beträgt der Radstand 560 mm, die Spurweite 436 mm und der Lenkrollradius 62,5 mm. [Fra]

3.1.1 Der Antrieb

Der Antrieb setzt sich aus dem Fahrantrieb und dem Lenkantrieb zusammen, diese bestehen jeweils aus:

1. Lenkantrieb
    - Maxon RE-max 29 Motor, 22 W
    - 2 Planeten- und Schneckengetriebe
    - Maxon EPOS 24/1 Motor Controller
2. Fahrantrieb
    - Maxon EC 90 Flat Motor, 90 W
    - Harmonic Drive Getriebe
    - Maxon EPOS 24/5 Motor Controller

Das *IAIS* sieht eine Stromversorgung durch zwei Blei-Gel-Akkus mit je 12 V und 9 Ah vor. Um die Reichweite des Roboters zu erhöhen kommen bei dem für diese Arbeit verwendeten Modell stärker dimensionierte Batterien mit 45 Ah zum Einsatz.

3.1.2 Die Sensoren

Für die Fahr- und Lenkfunktion stehen dem *Volksbot Ackermann Lenkung* verschiedene Sensoren zur Verfügung. Zur Ermittlung der Ausrichtung der Räder sind an den Motoren des Lenkantriebs **optische Inkrementalgeber** angebracht. Diese bestehen aus einer Lochscheibe, einer Lichtquelle und einem Fototransistor. Durch die Drehung des Motors wird die Lochscheibe gedreht, so werden auf dem Fototransistor abwechselnd Licht und Schatten detektiert. Durch den Einsatz von zwei Kanälen kann zusätzlich die Drehrichtung bestimmt werden. Da für eine höhere Genauigkeit meistens pro Kanal die steigende und fallende Flanke registriert wird, spricht man auch oft von einem Quadraturencoder. Die Detektierung einer Flankenänderung wird in der Literatur oft als „Tick“bezeichnet. Zur absoluten Positionsbestimmung werden von einem Indexkanal die kompletten Umdrehungen relativ zu einem definierten Nullpunkt gezählt. Zur Festlegung des

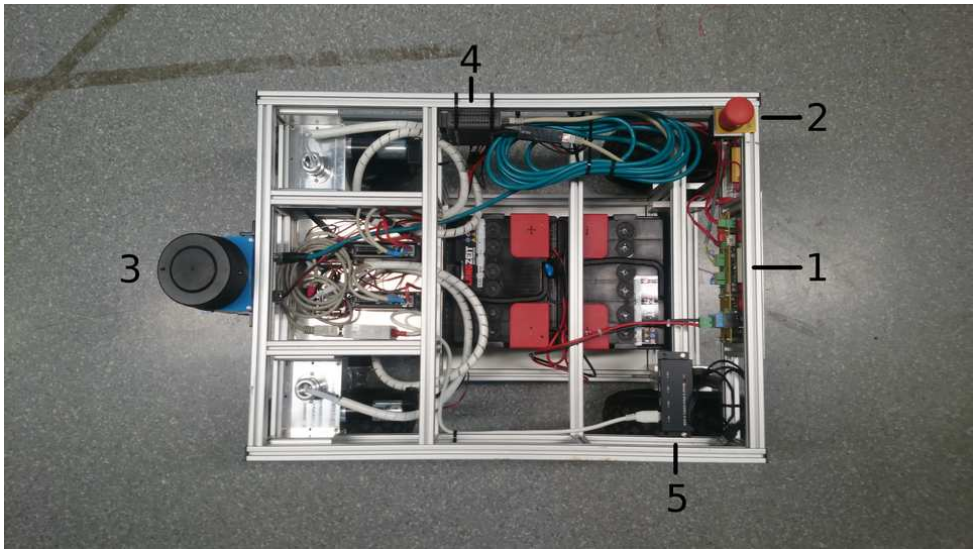


Abbildung 3.2: Der Roboter ARTEMIS

Nullpunkts werden **Drehpotentiometer** verwendet. Diese sind an der Lenkwelle des Roboters angebracht. Ein Drehpotentiometer besteht aus einem elektrischen Widerstand, dessen Widerstandswert durch Drehen einer Achse verändert werden kann. Die am Potentiometer gemessene Spannung ist proportional zu Kennlinie (linear oder logarithmisch). Zur absoluten Nullpunktsbestimmung muss das Potentiometer vermessen werden und die Nullpunktsspannung im Controller gespeichert werden. Diese wird dann beim Initialisieren des Roboters verwendet, um die Räder auszurichten. Die **Hallsensoren** an den Fahrantrieben werden zur Drehzahlregelung des Motors verwendet. Ein stromdurchflossener Hallsensor, auf den ein Magnetfeld wirkt, liefert eine Spannung, die proportional zur Stärke des Magnetfelds ist. Am bürstenlosen Gleichstrommotor des Fahrantriebs sind drei Hallsensoren jeweils um 120 ° versetzt angebracht. Diese Detektieren so beim Vorbeidrehen des Magnets des Rotors eine Hallspannung. Durch den Status der drei Sensoren (Spannung oder keine Spannung) sind sechs verschiedene Kombinationen möglich. Diese werden vom Controller genutzt, um die Geschwindigkeit des Motors zu regeln.

Um aus einem *Volksbot Ackermann Lenkung* den Roboter ARTEMIS (Ackermann Robot That Explores Mainly Indistinct Squares) werden zu lassen, waren die folgenden Schritte notwendig. Alle genannten Positionen beziehen sich auf Abbildung 3.2.

Die Volksbot SPU (Safety and Power Unit, Position 1) und der Notschalter (Position 2) wurden von vorne nach hinten versetzt, um dem Laserscanner LMS100 (Position 3) Platz zu machen. Der Notschalter kann durch die Versetzung bei Vorwärtsfahrten im Falle einer notwendigen Betätigung leichter erreicht werden. Des Weiteren wurde ein 5-Port 100MBit/s Netzwerk Switch (Position 4) sowie ein 4-Port USB 2.0 Hub (Position 5) installiert. Aktuell ist am Switch nur der Laserscanner, am USB Hub der Joystick und der USB-to-CAN Adapter angeschlossen. Durch die freien Ports der beiden Schnittstellen können nachträglich sehr einfach weitere Geräte wie beispielsweise Wärmebildkameras oder Gyroskope angeschlossen werden. Der Laserscanner LMS100

der Firma Sick besitzt eine Reichweite von 20 m bei einem Öffnungswinkel von maximal 270°. Es handelt sich um einen Sicherheitslaser der Klasse 1, er ist nicht schädlich für das menschliche Auge. Die Abtastfrequenz kann wahlweise auf 25 Hz oder 50 Hz eingestellt werden, die Leistungsaufnahme liegt maximal bei 10 W für 10,8 V bis 30 V Gleichspannung. Der Laserscanner ist nach IP65 gegen Staub und Spritzwasser geschützt. [SIC15]

3.2 Robot Operating System

Das *Robot Operating System* ROS [Wil] ist eine Sammlung von Libraries, Werkzeugen und Konventionen zur Entwicklung von Software für Roboter. Durch die Bereitstellung der Software und Pakete verschiedener, oftmals spezialisierter Entwickler, werden Austausch und gemeinschaftliche Entwicklung gefördert. ROS arbeitet mit so genannten *Nodes*, die über das Publizieren von *Messages* auf bestimmte *Topics* und das Abonnieren ebendieser miteinander kommunizieren können. Das Publizieren erfolgt in einem fest vorgegebenen Zeitabstand, das Abonnieren arbeitet nach dem Prinzip des *Interrupts*. Sobald neue Daten verfügbar sind, werden diese verarbeitet. Um nicht auf neue Daten warten zu müssen, können über *Services* Anfragen an einen *Node* geschickt werden. Eine *Message* besteht aus primitiven Datentypen wie *char, int, float, double, bool, ...* oder Feldern ebendieser. Auch kann eine *Message* weitere *Messages* enthalten. Ein *Service* besteht immer aus zwei *Messages*, einer für den *Request* und einer für den *Response*. Diese sind innerhalb der Datei durch eine Zeile mit der Zeichenfolge „—“ voneinander getrennt. Software für ROS kann in C++ oder Python geschrieben werden. *ROS* wurde für den Einsatz unter Ubuntu entwickelt, dementsprechend ist die Dokumentation zur Installation und Problemlösung sehr ausführlich und detailliert. Durch die immer größer werdende Popularität mobiler Roboter und der daraus resultierenden wachsenden Benutzergruppe von ROS, gibt es mittlerweile auch gute grundlegende Anleitungen zur Installation unter *OS X* oder anderen Linux Distributionen wie *Debian* oder *Arch Linux*. *ROS* bietet die Möglichkeit, verschiedene *Nodes* dezentral in einem Netzwerk zu verteilen. Jeder teilnehmende Rechner muss eine statische IP-Adresse besitzen und es muss eine komplette, bidirektionale Kommunikation zwischen allen Rechnern auf allen Ports möglich sein. Des Weiteren muss jeder Rechner durch seinen Namen eindeutig erkennbar sein und die Namen aller anderen Teilnehmer kennen. Der *ROS-Master*, der als zentrale Kommunikationsschnittstelle für alle *Nodes* dient, darf nur einmal gestartet werden und muss jedem Teilnehmer bekannt sein ([Wil], Tutorial *Multiple Machines*). Somit lässt sich bei kabelgebundener Steuerung das Eingabegerät an einem Rechner anschließen, der über WLAN mit dem sich auf dem Roboter befindlichen Rechner kommuniziert. Ein typischer ROS Workspace hat die in Abbildung 3.3 dargestellte Struktur und kann beliebig viele Pakete mit sinnvollerweise mindestens den aufgeführten Dateien und Ordner enthalten. Im Verzeichnis *include* werden sämtliche Header-Dateien abgelegt, das Verzeichnis *src* enthält die dazugehörigen Quelltext-Dateien. Weitere mögliche Verzeichnisse sind *launch* (Enthält Launch-Dateien zum einfachen Starten verschiedener Nodes und setzten der dazugehörigen Parameter), *srv* (Enthält Service-Dateien) und *msg* (Enthält Message-Dateien). Die *CMakeLists.txt* ist eine Skriptdatei, die als Input für das Programmierwerkzeug *CMake* dient. Sie enthält die Informationen zum Generieren und Installieren der Software aus dem Code. Hierbei entstehen *Executables*,

```
workspace/
   build/                        <-- wird von catkin erstellt
   devel/                        <-- wird von catkin erstellt.
   src/                          <-- enthält die eigenen Pakete
      CMakeLists.txt             <-- 'Toplevel' CMakeLists.txt,
                                      wird von catkin erstellt
      package_1/                 <-- Paket 1
         include/                <-- include Verzeichnis
         src/                    <-- src Verzeichnis
         CMakeLists.txt          <-- CMakeLists.txt für Paket 1
         package.xml             <-- package.xml für Paket 1
      ...
      package_n
         include/
         src/
         CMakeLists.txt
         package.xml
```

Abbildung 3.3: Typische Struktur eines ROS Workspace

eigenständige ausführbare Dateien, und *Libraries*, die gegen verschiedene *Executables* gelinkt werden können und standardmäßig dem gesamten System zur Verfügung gestellt werden. ROS wird über die Kommandozeile mit einer Reihe von Befehlen gesteuert. An dieser Stelle wird auf eine Beschreibung der verschiedenen Befehle und deren Parameter sowie Syntax verzichtet. Diese kann auf der Homepage von ROS [Wil] oder in komprimierter Form auf dem *ROS cheatsheet* [Ope15] nachgelesen werden.

3.3 Kinematik Mobiler Roboter

Der folgende Abschnitt orientiert sich stark an [Sch08], Kapitel 2.2.2.

Wichtig für die Bewegung des Roboters sind die Ackermann Winkel und die Drehgeschwindigkeiten der Räder. Die mathematischen Beziehungen hierfür werden im Folgenden aus Abbildung 3.4 hergeleitet. Die Berechnungen erfolgen mit Hilfe eines virtuellen, auf die zentrierte Längsachse des Roboters angebrachten Rades. Der Mittelpunkt der Kurve wird mit M, der Abstand zwischen Vorder- und Hinterachse mit L bezeichnet.

Für den Lenkwinkel α gilt

$$\cot \alpha = \frac{R_H}{L} \tag{3.1}$$

und folglich für den Abstand R_H zwischen dem virtuellen Rad und M

$$R_H = \cot \alpha \cdot L. \tag{3.2}$$

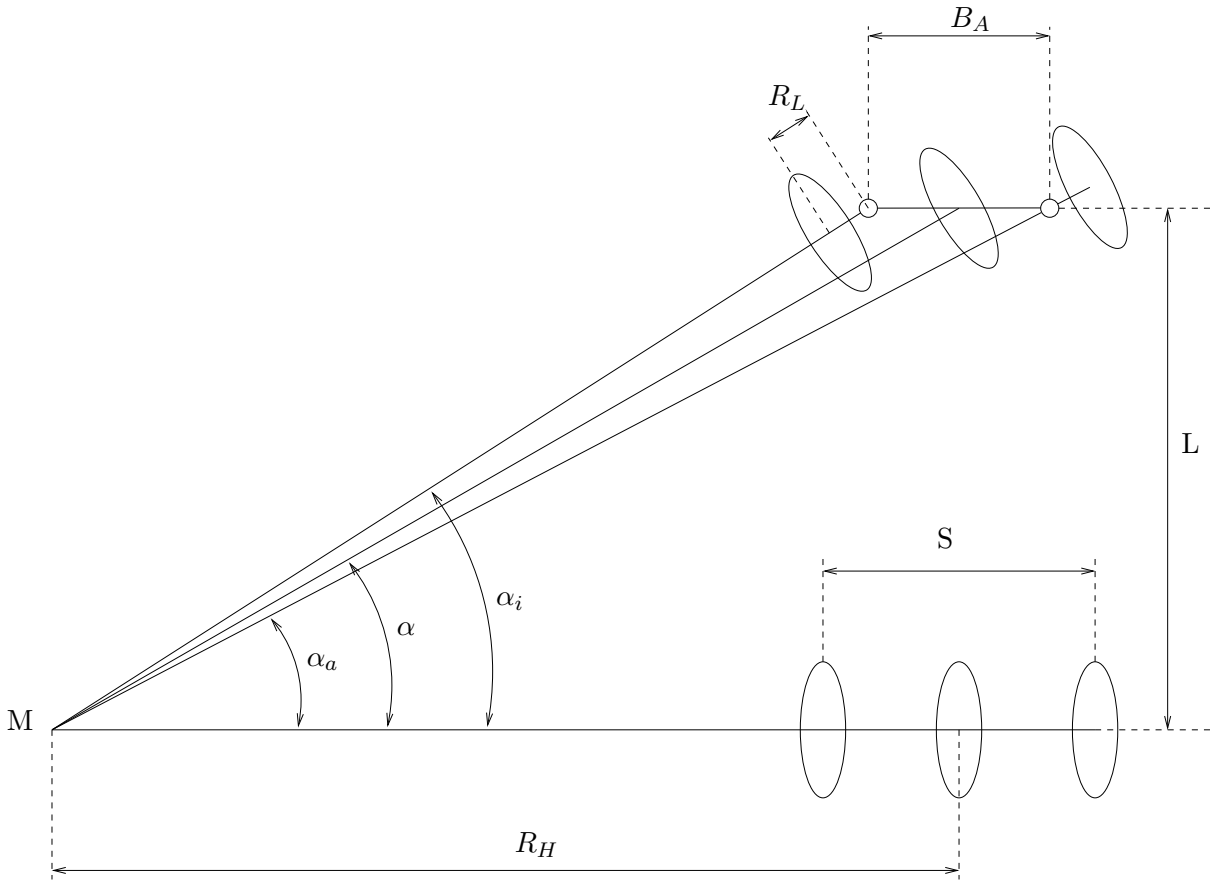


Abbildung 3.4: Herleitung der kinematischen Gleichungen für eine Ackermann Lenkung, Quelle: Eigene Darstellung nach [Dud05], Abb. 3.1, S.30

Somit folgt für den Winkel des äußeren Rades α_a

$$\cot \alpha_a = \frac{R_H + \frac{B_A}{2}}{L}. \tag{3.3}$$

Nach Einsetzen von Formel 3.2

$$\alpha_a = \operatorname{acot}(\cot \alpha + \frac{B_A}{2L}) \tag{3.4}$$

und analog für den Winkel des inneren Rades α_i

$$\alpha_i = \operatorname{acot}(\cot \alpha - \frac{B_A}{2L}). \tag{3.5}$$

B_A bezeichnet den Abstand der Drehpunkte der Vorderräder. Dieser lässt sich aus Spurbreite S und Lenkrollradius R_L nach

$$B_A = S - 2 \cdot R_L \tag{3.6}$$

berechnen. Allgemein gilt, dass der Lenkwinkel des kurveninneren Rades stets größer als der des kurvenäußeren ist. Mit diesen Formeln lassen sich bei gegebenem Lenkwinkel mit Hilfe der

Fahrzeugdimensionen die richtigen Ackermann Winkel (Lenkwinkel des inneren und äußeren Rades) für gleitungsfreies Rollen bestimmen.

Für die Geschwindigkeit v und die Winkelgeschwindigkeit ω des virtuellen Rades auf einer Kreisbahn mit festem Radius gilt

$$v = \omega \cdot R = \omega \cdot \frac{L}{\sin\alpha} \Longrightarrow \omega = v \cdot \frac{\sin\alpha}{L}. \tag{3.7}$$

Mit konstantem ω folgt für die Geschwindigkeit v_i des inneren Rades

$$v_i = \omega \cdot R_i = v \cdot \frac{\sin\alpha}{L} \cdot (\frac{L}{\sin\alpha_i} - R_L) \tag{3.8}$$

und analog für die Geschwindigkeit v_a des äußeren Rades

$$v_a = \omega \cdot R_a = v \cdot \frac{\sin\alpha}{L} \cdot (\frac{L}{\sin\alpha_a} + R_L). \tag{3.9}$$

Eine weitere zu bestimmende Größe ist die Beschleunigung, mit der die Räder eine Geschwindigkeitsänderung vornehmen. Eine konstante Beschleunigung a bei geradliniger Bewegung ist allgemein definiert als Änderung der Geschwindigkeit Δv in einem Zeitintervall Δt nach

$$a = \frac{\Delta v}{\Delta t}. \tag{3.10}$$

Die Änderung der Geschwindigkeit Δv berechnet sich nach

$$\Delta v = v(t_2) - v(t_1) = \omega(t_2) \cdot R - \omega(t_1) \cdot R = (\omega(t_2) - \omega(t_1)) \cdot R. \tag{3.11}$$

Somit folgt nach Divison der Änderungen Δv_i der Innen- und Δv_a Außengeschwindigkeiten

$$\frac{\Delta v_i}{\Delta v_a} = \frac{(\omega(t_2) - \omega(t_1) \cdot R_i}{(\omega(t_2) - \omega(t_1) \cdot R_a} = \frac{R_i}{R_a}. \tag{3.12}$$

Für den Beschleunigungsvorgang wird vorausgesetzt, dass die beiden Räder die Geschwindigkeitsänderung in derselben Zeitspanne absolvieren, Δt soll also für beide Räder gleich sein. Somit ergeben sich für die Beschleunigungen a_i des inneren und a_a des äußeren Rades

$$a_i = \frac{\Delta v_i}{\Delta t} \text{ und } a_a = \frac{\Delta v_a}{\Delta t} \tag{3.13}$$

und für deren Verhältnis nach Divison durcheinander

$$\frac{\Delta v_i}{\Delta v_a} = \frac{a_i}{a_a}. \tag{3.14}$$

Setzt man nun die Formeln 3.8 und 3.9 sowie 3.12 und 3.14 gleich, folgt

$$\frac{a_i}{a_a} = \frac{R_i}{R_a} = \frac{v_i}{v_a}. \tag{3.15}$$

Das Verhältnis der Beschleunigungen ist also, ebenso wie das Verhältnis der Geschwindigkeiten, gleich dem Verhältnis der Radien.

Das für die Beschleunigung angewandte Prinzip lässt sich auf die Berechnung der notwendigen Drehgeschwindigkeit zum Ausrichten der Räder übertragen. Die Winkelgeschwindigkeit ω_Φ, mit der die Räder eine Winkeländerung $\Delta\Phi$ vornehmen ist allgemein definiert nach

$$\omega_\Phi = \frac{\Delta\Phi}{\Delta t}. \tag{3.16}$$

Analog zu den Formeln 3.13 und 3.14 folgt für das Verhältnis der Winkelgeschwindigkeiten bei gleichem Δt

$$\frac{\omega_{\Phi_i}}{\omega_{\Phi_a}} = \frac{\Delta\Phi_i}{\Delta\Phi_a}. \tag{3.17}$$

Die hier vorgestellte Art zur Berechnung der Winkelgeschwindigkeiten ist lediglich eine Näherung. Um die korrekten Werte zu erhalten, müsste eigentlich die Funktion, die die Änderung der Winkel über die Zeit beschreibt, nach der Zeit abgeleitet werden. Da die verwendeten EPOS Controller jedoch nur konstante Werte und keine Funktionen zur Geschwindigkeitsregelung entgegen nehmen, sind die oben beschriebenen Formeln implementiert.

3.4 Odometrie eines Ackermann Fahrzeugs mit Vorderradantrieb

Zur Positionsbestimmung eines Roboters ist die aktuelle Pose $P_n = [x, y, \theta]^T$ entscheidend. Beginnend bei $P_0 = [0, 0, 0]^T$ lässt sich der gefahrene Pfad mit Hilfe der Inkrementalgeber nach einfachen geometrischen Formeln, die im weiteren Verlauf erklärt werden, bestimmen. Die Positionsbestimmung erfolgt also immer relativ zum Startpunkt. Zur Positionsbestimmung ist Odometrie für lange Strecken sehr unzuverlässig, da sich die Fehler, die bei den Berechnungen entstehen, aufsummieren. Man muss hier zwischen systematischen und zufälligen Fehlern unterscheiden. Zu den systematischen Fehlern zählen beispielsweise ungenau bestimmte Konstanten wie der Radumfang oder die Ticks, die der Inkrementalgeber pro Radumdrehung liefert. Diese können durch Experimente und genauere Messmethoden meist eliminiert werden. Zufällige Fehler, wie zum Beispiel das Durchdrehen eines Rades auf zu glatter Oberfläche oder das seitliche Wegrutschen des Roboters bei schneller Kurvenfahrt, werden vom System nicht erkannt und führen so zu ungenauen Ergebnissen. Als einfache Methode zur Positionsbestimmung ist die Odometrie trotz der einhergehenden Probleme weit verbreitet und wird in nahezu jedem Roboter eingesetzt, meist jedoch als Kombination mit weiteren Sensoren wie zum Beispiel einer IMU (Inertial Measurement Unit) oder Lokalisierungsmethoden wie *iSpace* oder AMCL, auf die im Kapitel 3.6 *Kartierung und Lokalisierung mit GMapping, AMCL und iSpace* genauer eingegangen wird.

Zur Odometriebestimmung werden die in [DL] eingeführten Formeln verwendet. Das verwendete Modell ist in Abbildung 3.5 zu sehen. θ_{n-1} bezeichnet die Orientierung des Roboters zur x-Achse zum Zeitpunkt $n-1$, Φ bezeichnet den Lenkwinkel, L ist der Abstand der Vorder- und

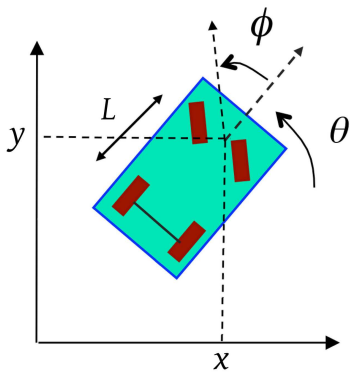


Abbildung 3.5: Modell für die Odometrieberechnung der Ackermann Lenkung [DL]

Hinterachse. Es wird die Annahme gemacht, dass der Roboter immer im Nullpunkt des globalen Koordinatensystems mit Orientierung der Längsachse in x-Richtung startet. Die Position des Roboters zum Zeitpunkt $t(n)$ berechnet sich nach

$$x_n = x_{n-1} + \Delta d \cdot cos\theta_{n-1} \cdot cos\Phi \tag{3.18}$$

und

$$y_n = y_{n-1} + \Delta d \cdot sin\theta_{n-1} \cdot cos\Phi \tag{3.19}$$

Für die Orientierung θ gilt

$$\theta_n = \theta_{n-1} + \frac{1}{L} \cdot sin\Phi. \tag{3.20}$$

Δd ist der im Mittelpunkt der Achse zurückgelegte Weg

$$\Delta d = \frac{\Delta r + \Delta l}{2}. \tag{3.21}$$

Die zurückgelegte Strecke des jeweiligen Rades berechnet sich nach

$$\Delta l = \frac{l \cdot \Omega}{\Theta} \text{ und } \Delta r = \frac{r \cdot \Omega}{\Theta} \tag{3.22}$$

mit Ω als Radumfang und Θ als Ticks pro Radumdrehung.

3.5 Der CAN Bus

Das Controller-Area-Network (CAN)-Protokoll wurde 1981 von Bosch und Intel entwickelt und war ursprünglich zum Ersetzen der immer größer und schwerer werdenden Kabelstrukturen in Fahrzeugen gedacht. In den letzten Jahren wird der CAN-Bus zunehmend in industriellen Anwendungen und als interner Kommunikationsbus für Maschinen und Anlagen eingesetzt. Im Folgenden werden kurz einige wichtige Leistungsmerkmale ([Ets94], Kapitel 1.6.5) des CAN-Protokolls genannt. Der Bustopologie ist linienförmig, hierarchische Netzstrukturen sind durch

Tabelle 3.1: Data Frame im CAN Protokoll, [Max06]

command specifier	index	subindex	data
1 Byte	2 Byte	3 Byte	4 Byte

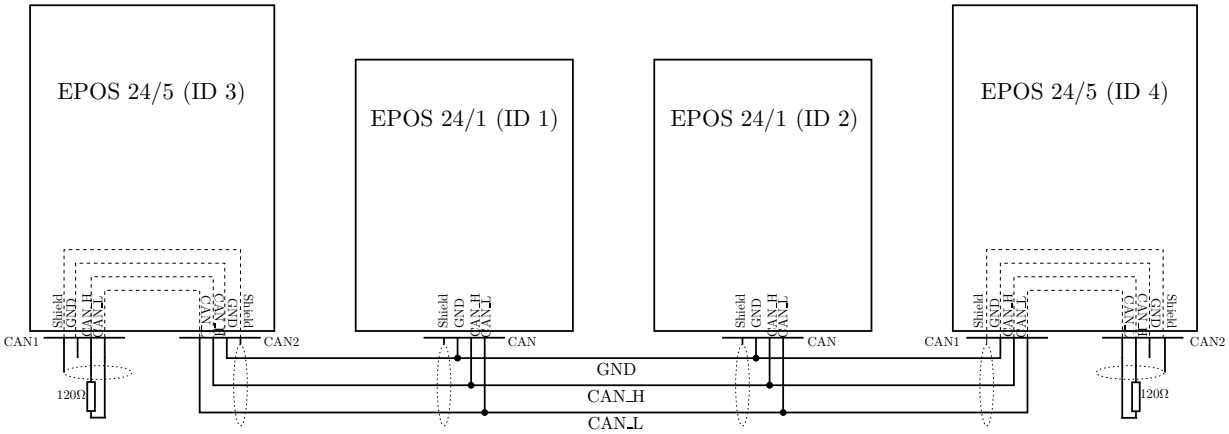


Abbildung 3.6: Netzwerk Struktur der EPOS Controller im Volksbot Ackermann

Knoten möglich. Jede Nachricht ist durch einen Identifier gekennzeichnet und priorisiert, die Blocklänge beträgt 8 Byte. Es handelt sich um ein Multi-Master-System, jeder Busteilnehmer kann senden, sofern der Bus frei ist. Die Busarbitrierung (Zuteilung der Zugriffe und Zugriffszeiten verschiedener Busteilnehmer) ist bitweise und verlustlos, die Latenzzeit ist durch Verwendung von CSMA/CA (*Carrier Sense Multiple Access/Collision Avoidance*) und der kurzen Nachrichtenlänge sehr gering. Zusätzlich beträgt die Restfehlerwahrscheinlichkeit bei der Datenübertragung lediglich 10^{-13}. Auf physikalischer Ebene betrachtet besteht ein CAN-Netzwerk aus zwei Busleitungen, *CAN_H* und *CAN_L*, GND und dem optionalen Shield. Die Ruhespannung für rezessive Bits liegt in einem 5 V-Betrieb für beide Busleitungen bei 2,5 V. Für dominante Bits ändert sich die Spannung für *CAN_H* auf 3,5 V und für *CAN_L* auf 1,5 V. Das Netzwerk wird vor dem ersten und nach dem letzten Knoten jeweils mit einem 120 Ω Widerstand terminiert. Die EPOS Controller verwenden das CAN High-Speed Protokoll bei einer Datenrate von 1 MBit/s, die EPOS 24/5 haben einen Kippschalter zur Aktivierung des 120 Ω Widerstandes integriert. Die Controller sind wie in Abbildung 3.6 miteinander verbunden. Der Adapter als Schnittstelle zum PC ist am *CAN1* des *EPOS 24/5 ID1* angeschlossen.

Ein Data-Frame, mit dem die Informationen zum Controller gesendet werden können, ist wie in Tabelle 3.1 zu sehen, aufgebaut. Der *command specifier* gibt an, wie viele Daten in *data* übertragen werden und ob es eine Lese- oder Schreibanforderung ist. Der *index* und der *subindex* beschreiben, welches Objekt angesprochen wird. Die Daten können maximal 32 Bit beinhalten. Zu beachten ist, dass in den Feldern *index* und *data* die Bytes im *Big-Endian* Format stehen müssen.

3.6 Kartierung und Lokalisierung mit GMapping, AMCL und iSpace

Zur Orientierung innerhalb einer Umgebung muss ein Roboter seine Pose anhand von Sensordaten wie Laserscannern oder Ultraschall bestimmen. Hierzu benötigt er allerdings eine Karte der Umgebung. Diese muss jedoch zuerst erstellt werden, indem der Roboter, ausgehend von seiner Pose, den Abstand zu Objekten und Hindernissen bestimmt. Eine Lokalisierung funktioniert also nur mit einer vorgegebenen Karte und eine Karte kann nur mit bekannter Pose erstellt werden. Die gleichzeitige Kartierung einer unbekannten Umgebung und Lokalisierung in dieser (SLAM, Simultaneous Localization And Mapping) ist ein Problem, das bei mobilen Robotern weit verbreitet ist. ROS bietet hierfür das Paket *gmapping* an, das anhand von Laserscanns eine Karte schrittweise erstellt und die Posedaten des Roboters nutzt, um die Lokalisierung vorzunehmen. Da die Posedaten mit zunehmender Dauer des Vorgangs schlechter werden, erkennt der Algorithmus Teile der bereits vermessenen Umgebung und korrigiert die Pose innerhalb der Karte. Zur Lokalisierung innerhalb einer bereits bekannten Umgebung stellt ROS das Paket *amcl* zur Verfügung. Dieses Paket ist eine Implementierung der Monte-Carlo-Lokalisierung. Für komplexe Probleme ist eine korrekte Berechnung der Lösung teilweise nur schwierig oder gar nicht machbar. Ist eine Schätzung der Lösung für ein Problem hinreichend, wird diese in der Informatik oftmals mit Hilfe von Monte-Carlo-Algorithmen und einer Reihe von Stichproben realisiert. Die hierzu verwendeten Stichproben, auch Partikel genannt, dienen als mögliche Pose der Roboters. Zu Beginn der Lokalisierung ist die Pose des Roboters unbestimmt. Anhand der Laserscans bei Bewegung durch die Umgebung vergleicht der Roboter die Messungen mit der Karte und kann so seine Pose bestimmen. Besteht eine Umgebung aus vielen ähnlichen Teilgebieten kann eine Lokalisierung unter Umständen sehr lange dauern. Da die Anzahl der möglichen Posen oder auch Partikel immer weiter reduziert wird, bezeichnet man AMCL auch als Partikelfilter. [TBF06]

iSpace ist ein Positionsbestimmungssystem der Firma Nikon Metrology. Das System besteht aus mehreren Transmittern, die Laserstrahlen bei einer eigenen charakteristischen Frequenz um 40 Hz emittieren. Ihr Öffnungswinkel ist auf 40 ° begrenzt, die Entfernung der Objekte darf zwischen 2 und 55 Metern liegen. Die Transmitter müssen um den Bereich, in dem gemessen werden soll, an den Wänden oder auf Stativen angebracht werden. Ein *iSpace* System besteht aus mindestens vier Transmittern, somit kann auch bei teilweiser Abschirmung des Sensors dieser in der gesamten Messumgebung detektiert werden. Ein Sensor besteht mindestens aus einer Photodiode mit einem horizontalen Öffnungswinkel von 360 ° und einem vertikalen Öffnungswinkel von 90 °. Mit Hilfe eines Sensors kann die Position im Raum in 3D bestimmt werden. Um eine 6D-Position (x, y, z, roll, pitch, yaw) zu erhalten, müssen zwei Sensoren als Stereosensor verbaut werden. Bevor das System einsatzfähig ist, muss es kalibriert werden. Hierzu wird ein speziell aufgebautes Sensorsystem verwendet, die mitgelieferte Software führt durch den Kalibrierungsprozess. Der Ursprung des Koordinatensystems kann hierbei beliebig im Raum positioniert werden. Die Messgenauigkeit liegt laut Hersteller unter optimalen Bedingungen (gleichmäßige Anordnung der Transmitter, keine Reflexionen durch Glas oder Spiegel, ...) bei [±0,25]mm. Die Berechnung der Pose eines Roboters über das *iSpace* System erfolgt, im Gegensatz zur Odometrie, absolut zu einem externen Bezugssystem, in diesem Fall einem globalen Koordinatensystem. [Nik14], [BHE+15]

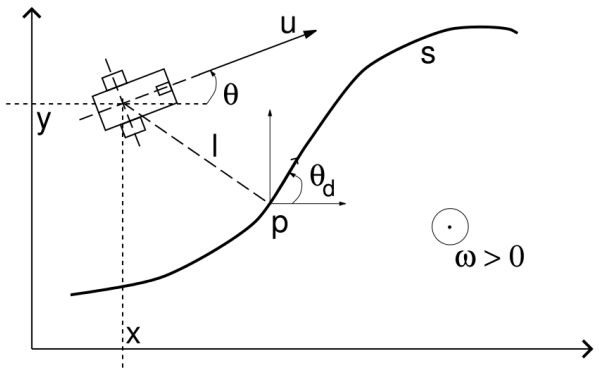


Abbildung 3.7: Modell für den Algorithmus zur Pfadverfolgung [IC04]

3.7 Pfadverfolgung

Zur Pfadverfolgung wird eine Implementierung des Reglers von G. Indiveri und M. L. Corradini verwendet. Der Algorithmus baut auf der Arbeit von Canudas de Wit et al. [CKSS93] auf und garantiert asymptotische Konvergenz auf den Pfad und asymptotisch stabile Fehlerdynamik nach Lyapunov für einen beschränkten, nicht-linearen Pfad. Dieser wird nun so erweitert, dass er sowohl den minimalen Wendekreis als auch die Maximalgeschwindigkeit des Roboters berücksichtigt. Des Weiteren werden die Parameter des Reglers in jedem Schritt neu berechnet, was zu einer schnelleren Konvergenz auf den Pfad führt. Als Grundlage der Berechnung dient nicht mehr der dem Roboter nächste Punkt auf dem Pfad, sondern seine orthogonale Projektion auf diesen. Das Prinzip des Reglers ist in Abbildung 3.7 als Modell dargestellt. Der Pfad wird durch eine Unterteilung in lineare Teilstücke genähert. l bezeichnet den Abstand zwischen dem Roboter und seiner orthogonalen Projektion auf den Pfad. θ ist der Winkel zwischen der x-Achse und der Ausrichtung des Roboters, θ_d der Winkel zwischen einer Tangente an den Pfad im Punkt p und der x-Achse. Somit ist der Winkel $\tilde{\theta}$ zwischen dem Roboter und der x-Achse definiert nach

$$\tilde{\theta} \equiv \theta - \theta_d. \tag{3.23}$$

Die Formel

$$\omega = \frac{u\,\kappa(s)\,cos\tilde{\theta}}{1 - l\,\kappa(s)} - h\,u\,l\,\frac{sin(\tilde{\theta})}{\tilde{\theta}} - \gamma\,\tilde{\theta} : h, \gamma > 0 \tag{3.24}$$

für die Winkelgeschwindigkeit ω mit u als Geschwindigkeit des Roboters, $\kappa(s)$ als Krümmung der Kurve und h, γ als Parameter des Reglers, vereinfacht sich für lineare Teilstücke mit $\kappa(s) = 0$ zu

$$\omega = -h\,u\,y\,\frac{sin(\tilde{\theta})}{\tilde{\theta}} - \gamma\,\tilde{\theta} : h, \gamma > 0, \tag{3.25}$$

wobei p nun der Ursprung. Durch eine Koordinatentransformation dient die x-Achse nun als Referenz und l wird durch y ersetzt. [IC04]

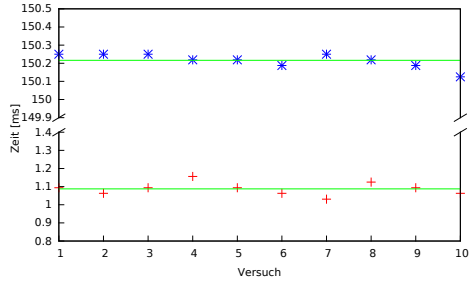
Kapitel 4

Technische Ausarbeitung

4.1 Schnittstelle zwischen PC und EPOS Controllern

Um Befehle an die Controller zu senden wird der *Volksbot Ackermann Lenkung* mit einem RS232-to-USB Adapter geliefert. Der CANopen Master ist für die Zuordnung der Nachrichten zu den jeweiligen Empfängern zuständig. Die Befehle, die im RS232 Protokoll empfangen werden, müssen von diesem ins CAN Protokoll umgewandelt werden, um sie dann auf den CAN-Bus zu schreiben. Hinzu kommt die mit 115.2 kBit/s deutlich geringere maximale Bitrate des RS232 gegenüber der maximalen Bitrate von 1 MBit/s des CAN-Bus [Max08]. Beim Testen der Library und versuchtem gleichzeitigen Starten der Motoren, war eine deutliche Verzögerung von mehreren hundert Millisekunden zwischen den jeweiligen Motoren sichtbar. Dieser limitierende Faktor kann durch den Einsatz eines USB-to-CAN Adapter behoben werden. Für diese Arbeit wird ein Modell der Firma *Peak System* [Pea] verwendet. Bei der Installation des hierzu erhältlichen Treibers ist es notwendig, den Support für die *netdev configuration* zu deaktivieren und so die Installation als *chardev configuration* zu aktivieren. Anderenfalls wäre lediglich das Schreiben auf den CAN-Bus, jedoch nicht das Lesen des CAN-Busses möglich [Pea]. Als Konsequenz dieser Umstellung musste die verwendete Library für die EPOS Controller von Grund auf neu geschrieben werden, um die Verwendung der PCAN Library zu ermöglichen.

Um den Geschwindigkeitsvorteil des CAN-Busses in Zahlen sichtbar zu machen, wurde folgender Versuch durchgeführt: Die vier Controller des Netzwerkes wurden jeweils über CAN und RS232 initialisiert und dabei die Dauer der Vorgänge mit Hilfe der Systemzeit gemessen. Bei der Initialisierung wurde zunächst der aktuelle Status des Controllers abgefragt (1), anschließend ein „Shutdown“(2) erzwungen. Nach dem Senden von „Switch On“(3) und „Enable Operation“(4) ist der Controller bereit, Befehle zu empfangen und es liegt Spannung am Motor an. Der Zustandsautomat der EPOS Controller ist in [Max10], Kapitel 8 detailliert erklärt. Bei vier zu initialisierenden Controllern und zwei Interaktionen pro Befehl/Abfrage (*sendMessage* und *readAnswer*) ergeben sich also $4 \cdot 4 \cdot 2$ Interaktionen. Dabei wurden die in Abbildung 4.1, links dargestellten Zeiten gemessen, die in Abbildung 4.1, rechts geplottet sind. Die Mittelwerte sind als grüne Linie dargestellt. Aus diesen Daten ergibt sich ein Verhältnis von 1:138. Der



Grafische Darstellung

CAN [ms]	CAN $[\frac{ms}{call}]$	RS232 [ms]	RS232 $[\frac{ms}{call}]$
35	1,09	4808	150,25
34	1,06	4808	150,25
35	1,09	4808	150,25
37	1,16	4807	150,22
35	1,09	4807	150,22
34	1,06	4806	150,19
33	1,03	4808	150,25
36	1,13	4807	150,22
35	1,09	4806	150,19
34	1,06	4804	150,13
34,8	1,09	4806,0	150,22

Messwerte und die jeweiligen Mittelwerte

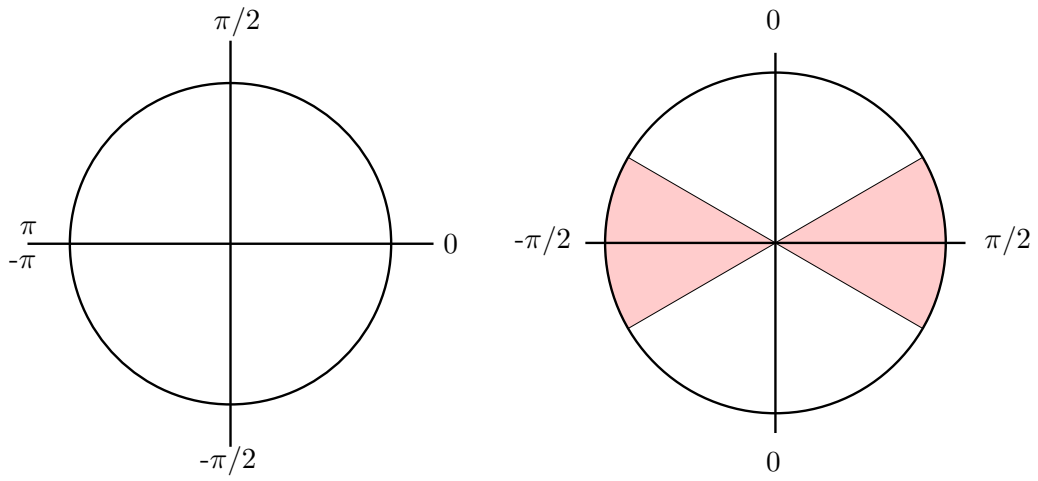
Abbildung 4.1: Vergleich zwischen RS232 und CAN

reine Geschwindigkeitsvorteil durch die Datenübertragungsrate beläuft sich auf ungefähr 1:8,5. Der restliche Faktor von 1:16 scheint durch die interne Umwandlung vom RS232- auf das CAN-Protokoll verursacht zu werden. Somit kann nun, trotz des seriellen Busses, von einem parallelen Ansprechen der Motoren gesprochen werden.

4.2 Steuerung des Roboters

Zur Steuerung des Roboters wurden zwei Varianten implementiert. Bei der Steuerung mit einem handelsüblichen Joystick mit mindestens zwei Achsen werden Winkel und Geschwindigkeit aus den x- und y-Koordinaten des Lenkknüppels berechnet. Bei der Steuerung mit einem Lenkrad wird der Winkel aus der Stellung des Lenkrads relativ zum Nullpunkt und die Geschwindigkeit aus der Stellung des Gaspedals berechnet. Im Folgenden werden die beiden Varianten erläutert.

Für die Ermittlung der notwendigen Fahrparameter sind bei der Eingabe durch einen Joystick mehrere Berechnungen und Transformationen notwendig. Für die Bestimmung des notwendigen Ackermann Winkels wurde das Koordinatensystem aus Abbildung 4.2(b) entwickelt. In diesem Koordinatensystem befinden sich positive Winkel rechts der y-Achse, negative Winkel links. Des Weiteren sind die Werte achsensymmetrisch zur x-Achse, um für positive wie negative Geschwindigkeiten gleiche Winkel zu gewährleisten. Zusätzlich soll der maximal einstellbare Ackermann Winkel auf 60 ° begrenzt werden (Rot markierter Bereich in Abbildung 4.2(b)). Dieser Wert gewährleistet ausreichend Manövrierbarkeit und verhindert, dass das Innenrad bei größeren Winkeln nahezu senkrecht zur Längsachse des Roboters steht. Um den Roboter mit einem Joystick oder Gamepad steuern zu können, musste eine Koordinatensystemtransformation vorgenommen werden. Solche Eingabegeräte liefern für die jeweiligen Achsen Werte im Bereich



(a) Joystick Koordinatensystem (b) Roboter Koordinatensystem

Abbildung 4.2: Transformation vom Joystickkoordinatensystem in das Roboterkoordinatensystem

[-32767,32767]. Um hiermit sinnvoll weiterrechnen zu können, werden die Skalen zunächst auf den Bereich [-1,1] skaliert. Die unter C++ verfügbare Funktion *atan2(y,x)* berechnet aus diesen Werten die in Abbildung 4.2(a) dargestellten Winkel. Für die Transformation waren folgende Schritte notwendig:

1. Negieren des Winkels und addieren von $\frac{\pi}{2}$ ergibt eine Spiegelung an der Achse y=x mit Fehler im III. Quadranten.
2. Subtraktion von 2π für Winkel größer π behebt Fehler im III. Quadranten.
3. Für Winkel größer $\frac{\pi}{2}$ werden diese von π subtrahiert, um den IV. Quadranten auf Werte zwischen 0 und $\frac{\pi}{2}$ zu begrenzen.
4. Analog wird dies für den III. Quadranten gemacht, allerdings wird dieser auf Werte zwischen 0 und $-\frac{\pi}{2}$ begrenzt.
5. Zur Begrenzung auf 60° werden alle Winkel größer $\frac{1}{3}\pi$ auf $\frac{1}{3}\pi$ gesetzt.
6. Analog wird dies für Winkel kleiner $-\frac{1}{3}\pi$ gemacht.

Zur Ermittlung der Geschwindigkeit werden zwei Parameter benötigt, der Abstand der aktuellen Position zum Mittelpunkt für den Betrag der Geschwindigkeit und das Vorzeichen der Position bezüglich der y-Achse für das Vorzeichen der Geschwindigkeit. Da der Joystick die Koordinaten als x- und y-Werte innerhalb eines Quadrats liefert, müssen diese in Polarkoordinaten auf dem Einheitskreis umgerechnet werden. Das Vorzeichen des y-Wertes wird mit einer modifizierten Signum-Funktion bestimmt, die für denn Fall $y = 0$ eins liefert, da der Roboter sonst stehen bleiben würde. Um eine Instabilität des Systems zu verhindern, wurde eine Hysterese in die Berechnung der Ackermann Winkel integriert. Nach jeder Neuberechnung überprüft diese, ob der Betrag der Differenz zwischen dem neuen und dem alten Winkel größer als fünf Grad ist.

Dies gewährleistet eine Lenkung, die fein genug auf gewünschte Änderungen der Richtung reagiert ohne dabei eine Instabilität des eingestellten Winkels zu erzeugen. Liegt der gewünschte Winkel über dem Schwellwert wird er an das System übergeben, andernfalls bleibt der alte Winkel im System gespeichert und es findet keine Änderung der Radorientierung statt. Mit weiteren Knöpfen kann die aktuelle Maximalgeschwindigkeit im Bereich von 20 % bis 100 % der Höchstgeschwindigkeit von 0.7 $\frac{m}{s}$ eingestellt werden, auch ist mit der Funktion *Quick-Stop* eine Art Vollbremsung implementiert.

Die Berechnung des Winkels und der Geschwindigkeit gestaltet sich bei einem Lenkrad mit Gas- und Bremspedal deutlich einfacher. Das Lenkrad liefert, ebenso wie der Joystick, Werte im Bereich von [-32767,32767], die zunächst normiert werden. Der notwendige Ackermann Winkel α berechnet sich dann nach

$$\alpha = l \cdot \frac{1}{3}\pi, \tag{4.1}$$

wobei l der aktuelle Wert des Lenkrads ist. Da dessen Betrag nie größer als eins wird, ist die Beschränkung auf 60 ° automatisch gegeben. Das Gaspedal liefert Werte im Bereich von [0,32767], das Bremspedal die dazugehörigen, negativen Werte. Die Werte des Gaspedals g werden ebenso normiert, um die notwendige Geschwindigkeit nach

$$v = g \cdot (i \cdot s), \tag{4.2}$$

wobei $i \cdot s$, wie bei der Steuerung mit einem Joystick, der aktuelle Prozentsatz i der Höchstgeschwindigkeit s ist. Erkennt das System eine Betätigung der Bremse, wird ein *Quick-Stop* initiiert.

Sowohl Joystick als auch Lenkrad können lokal am Rechner oder, wie in Kapitel 3.2 *Robot Operating System* beschrieben, ausgelagert über das Netzwerk an einem anderen Rechner betrieben werden. Durch die Kabelgebundenheit der meisten Eingabegeräte empfiehlt es sich, letztere Variante zu wählen, die deutlich mehr Flexibilität gewährleistet.

4.3 ROS Package

Das aus dieser Arbeit entstandene ROS Package orientiert sich zu großen Teilen an den aus Benjamin Schwakes Diplomarbeit [Sch08] hervorgegangenen Klassen und dem daraus entstandenen und vom IAIS geschriebenem ROS Package. Das Package besteht grundsätzlich aus zwei Teilen, der *executable* zur Kalibrierung des Lenksystems und setzten der Parameter für die integrierten Regler, sowie der *Ackermann Library* zur Ansteuerung und Lenkung des Roboters. Im Folgenden wird auf die beiden Teile genauer eingegangen.

4.3.1 Kalibrierung des Roboters

Zur Bestimmung der Lenkwinkel der Räder sind im *Volksbot Ackermann Lenkung* optische Inkrementalgeber und Stereodrehpotentiometer. Die optischen Inkrementalgeber werden beim Ein-

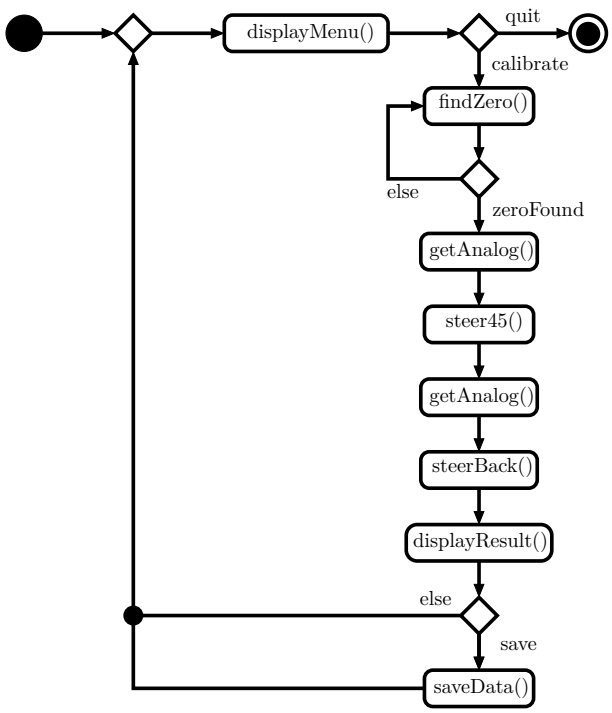


Abbildung 4.3: Aktivitätsdiagramm der Kalibrierungsfunktion

schalten auf Null initialisiert. Mit ihnen lässt sich der Drehwinkel in Rad relativ zum Nullpunkt nach

$$\alpha = 4 \cdot \mathrm{e} \cdot \mathrm{r} \;/\; (2 \cdot \pi)$$

bestimmen, wobei e ('„encoder“) die Auflösung des Inkrementalgebers und r („ratio“) die Untersetzung des Getriebes ist. Beim Volksbot Ackermann gilt i = 500 und r = 946.

Der Encoder liefert pro Umdrehung über zwei Kanäle jeweils zwei Flanken (*rising and falling edges*), daher muss seine Auflösung mit vier multipliziert werden. Zur Ermittlung des absoluten Nullpunkts wird wie in Grafik 4.3 zu sehen vorgegangen. Der Ablauf der Kalibrierung ist für das linke wie rechte Rad identisch. Zunächst muss das Rad manuell ausgerichtet werden, die Differenz zum absoluten Nullpunkt muss iterativ in vollen Grad eingegeben werden (*findZero()*). Hierbei ist ein großes Geodreieck und ein gutes Auge äußerst nützlich. Danach werden je 100 Werte der beiden Drehpotentiometer über die analogen Eingänge 1 und 2 ausgelesen und gemittelt, hierbei werden außerdem die beiden Werte verglichen (*getAnalog()*). Bei zu großer Differenz wird eine Warnung ausgegeben, da möglicherweise eine Fehlfunktion eines der beiden Potentiometer vorliegt. Danach wird das Rad um 45 Grad gedreht (*steer45()*) und wiederum die Sensoren ausgelesen und gemittelt (*getAnalog()*). Nach dem Zurückdrehen des Rades *steerBack()*) werden die ermittelten Parameter ausgegeben (*displayResult()*). Schließlich kann der Benutzer die ermittelten Werte im internen Speicher des Controllers ablegen lassen (*save()*) oder den Vorgang abbrechen. Anschließend beginnt die Schleife von vorne. Somit werden der absolute Nullpunkt und die Steigung der Kennlinie des Potentiometers bestimmt. Mit diesen Informationen kann

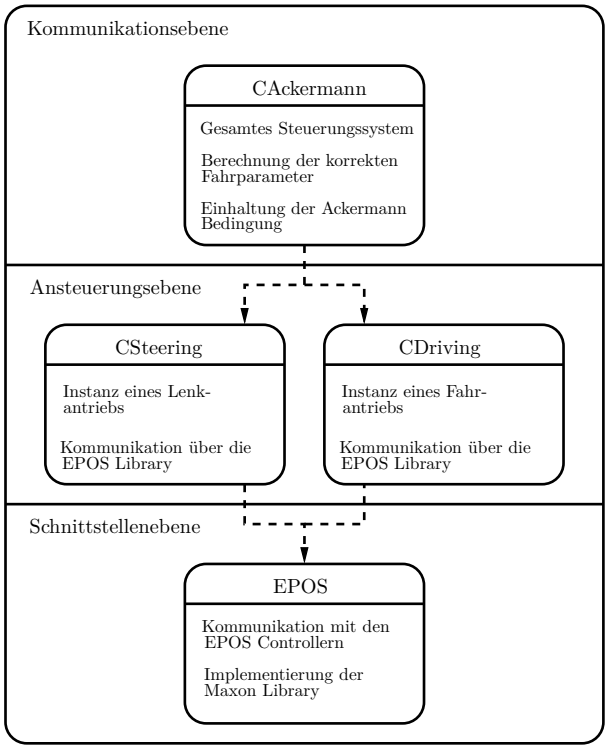


Abbildung 4.4: Strukturbild der Ackermann Library

jedes Rad beim Einschalten des Roboters und der Initialisierung des Controllers ausgerichtet werden.

Die in die EPOS-Controller integrierten Regler können ebenfalls über die *Executable* eingestellt werden. Nach der Auswahl des gewünschten Controllers werden die aktuell gespeicherten Parameter angezeigt. Anschließend müssen die neuen Werte für den PI-Regler der Spannung, den PI-Regler der Geschwindigkeit und den PID-Regler der Position eingegeben werden. Auch hier kann der Nutzer die ermittelten Parameter im internen Speicher ablegen lassen oder den Vorgang abbrechen. Die aktuell im Roboter abgelegten Parameter für die Regler wurden mit der Software *EPOS Studio* [Max] ermittelt.

4.3.2 Die Ackermann Library

Die Library besteht aus den in Abbildung 4.4 dargestellten Klassen. Dies entspricht dem in [Sch08] entwickelten Aufbau, der einen objektorientierten Ansatz verwendet. Als Library für die Kommunikation mit den EPOS Controllern dient die Klasse *Epos*. Sie ist eine Implementierung der durch den *EPOS Communication Guide* und der *EPOS Firmware Specification* bereitgestellten Objekte und Funktionen. Die Klassen *CSteering* und *CDriving* stehen für jeweils eine Instanz des Lenk- und Fahrantriebs. Mit Ihnen werden die Fahrparameter sowie die Profile gesetzt. Außerdem stellen sie Funktionen zum verbinden mit dem Controller sowie zur Initia-

lisierung, Aktivierung und Deaktivierung dessen zur Verfügung. Auch können die Sensordaten ausgelesen werden. Da die Antriebe für das linke und rechte Rad gespiegelt angeordnet sind, muss jeder Instanz beim Erstellen die Drehrichtung vorgegeben werden. Außerdem erfolgt hier die Umrechnung von SI-Einheiten in Maxon spezifische Einheiten. Als Schnittstelle für den Benutzer dient die in der Hierarchie oberste Klasse *CAckermann*. Diese steht symbolisch für den kompletten Antrieb und enthält je eine Instanz für den linken sowie rechten Fahr- und Lenkantrieb. Hier werden die korrekten Lenkwinkel und Geschwindigkeiten nach denen in Kapitel 3.3 *Kinematik Mobiler Roboter* vorgestellten Formeln berechnet. Als Input dienen die vom Steuergerät übermittelten Werte. Zur Fehlererkennung gibt es außerdem zwei unterschiedliche Exceptions, die von allen Klassen außer der *EPOS Library* geworfen werden können. Diese werden im Programmablauf der Oberklasse abgefangen und, wenn möglich, behandelt. Hierzu wird der Fehlerspeicher des betroffenen Controllers ausgelesen und die Fehlermeldung auf dem Display gezeigt.

4.3.3 Anbindung an ROS

Die Anbindung an ROS findet über die Klasse *Ackermann_steering* statt. In dieser wird eine Instanz von CAckermann initialisiert. Des Weiteren enthält sie einen *Publisher* für den *State*, der neben dem aktuellen Winkel und der aktuellen Geschwindigkeit auch die Ticks des linken und rechten Rades enthält, sowie einen *Subscriber* auf die Steuerung, der bei Änderungen den *callback* zur Übergabe der neuen Daten an die Benutzerschnittstelle aufruft. Die Controller halten für die Inkrementalgeber einen Speicher von 16 Bit bereit und liefern so Werte im Bereich von [0,65535]. Der Roboter bewegt sich ungefähr alle 47 Ticks um einen Zentimeter nach vorne. Eine 16 Bit Variable läuft also nach knapp 14 m über und wird zurückgesetzt. Um die Odometrie bei weiteren Strecken sinnvoll berechnen zu können, werden die Ticks auf einen 32 Bit Speicher erweitert. Diese Variable läuft bei 47 Ticks pro Zentimeter erst nach ungefähr 914 km über, was für die Einsatzgebiete des Roboters mehr als ausreichend ist. In jedem Schleifendurchlauf müssen nun drei unterschiedliche Fälle behandelt werden, um den Zähler, der die Überläufe zählt, zu aktualisieren und die Gesamtticks zu bestimmen. Ist die Differenz zwischen dem aktuellen Wert und dem im vorherigen Durchlauf gemessenen Wert kleiner als -32767, so dreht sich das Rad vorwärts und die Variable ist übergelaufen, der Zähler muss um eins hochgezählt werden. Ist die Differenz jedoch größer als 32767, dreht sich das Rad rückwärts und die Variable ist übergelaufen. Somit muss der Zähler um eins verringert werden. Für alle anderen Fälle wird auf den aktuellen Gesamtwert lediglich die Differenz des alten und aktuellen Wertes addiert. Am Ende jedes Schleifendurchlaufs werden die aktuellen Ticks archiviert. Eine weitere Funktion berechnet die notwendige Geschwindigkeit zum Ausrichten der Räder anhand der Differenz zwischen dem aktuell eingestellten Winkel und dem gewünschten Winkel. So wird eine zu hektische Bewegung des Systems bei kleinen Winkeländerungen vermieden.

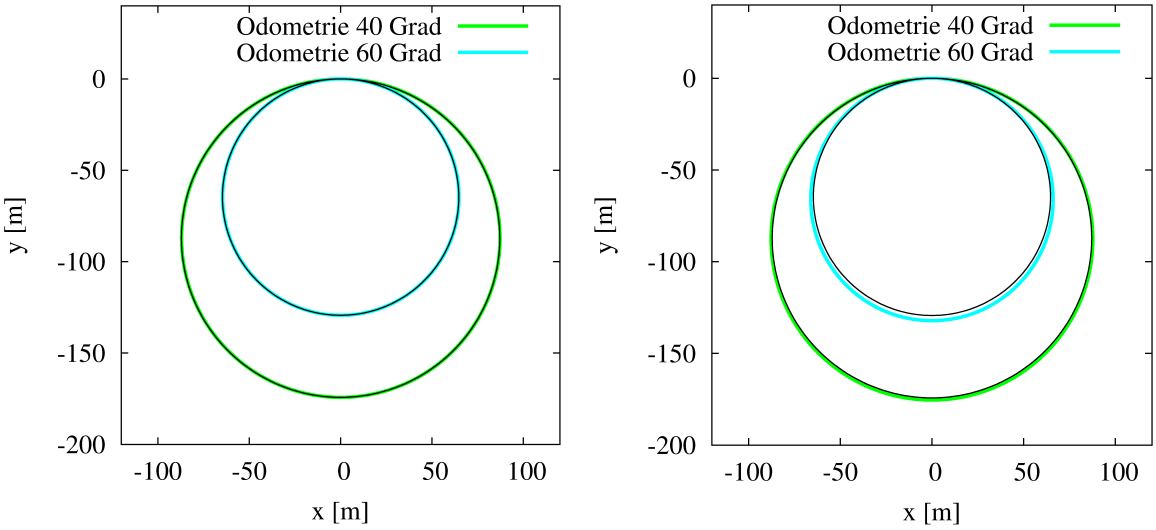
Kapitel 5

Experimente

Um die korrekte Funktionsweise des ROS Package zu überprüfen, wurden eine Reihe von Tests durchgeführt. Zuerst wird ein Simulator zum Testen der Odometrieberechnung erklärt. Weiterhin wird die Erstellung der Karte einer unbekannten Umgebung mit *gmapping* und die anschließende Lokalisierung durch Odometrie und AMCL beschrieben. Zuletzt wird die Pfaderzeugung und Pfadverfolgung an vier Beispielen diskutiert.

5.1 Simulation der Odometrieberechnung

Zum Testen der Odometrieberechnung wurde ein Simulator geschrieben, der diese für einen Kreis mit einem vorgegebenen Winkel und einer vorgegebenen Geschwindigkeit simuliert. Die notwendigen Ackermann Winkel des linken und rechten Rades sowie die jeweiligen Geschwindigkeiten werden nach den in Kapitel 3.3 *Kinematik Mobiler Roboter* eingeführten Formeln berechnet. Diese werden der Funktion zur Odometrieberechnung des Roboters übergeben, die für eine fest vorgegebene Anzahl an Schritten die Orientierung θ sowie die Koordinaten x und y der aktuellen Position berechnet und ausgibt. Die hier implementierten Formeln wurden in Kapitel 3.4 *Odometrie eines Ackermann Fahrzeugs mit Vorderradantrieb* eingeführt. Diese Werte können als Textdatei gespeichert und mit *Gnuplot* [Gnu] grafisch dargestellt werden. Eine Simulation für zwei Kreise mit $40\,^\circ$ und $60\,^\circ$ Lenkwinkel ist in Abbildung 5.1(a) zu sehen. Schwarz gestrichelt ist die Referenz der Kreise für die jeweiligen Lenkwinkel. Der Radius des Kreises lässt sich für einen gegebenen Lenkwinkel nach Formel 3.7 bestimmen. Da die Simulation die zu erwartenden Ergebnisse liefert, bestätigen sie das verwendete Modell. Ein anderer Ansatz zur Berechnung der Odometrie verwendet das Modell eines Differentialantriebs und berechnet die Spurbreite für jeden Einschlagwinkel neu. Die Formeln hierzu können in [HLN12], Kapitel 4.2.1 nachgelesen werden. Wird nun als Spurbreite die Differenz der Radien des linken und rechten Rades verwendet, kommt man zu dem in Abbildung 5.1(b) dargestellten Ergebnis. Wie zu erkennen ist, weicht die Berechnung für große Lenkwinkel mehr von der Realität ab als für kleine Lenkwinkel. Dies lässt sich darauf zurückführen, dass als tatsächliche Spurbreite nicht der Abstand der beiden Räder, sondern die Summe der Abstände der beiden Räder zur Verlängerung der



(a) Simulation zweier Kreise mit 40 ° und 60 ° Lenkwinkel

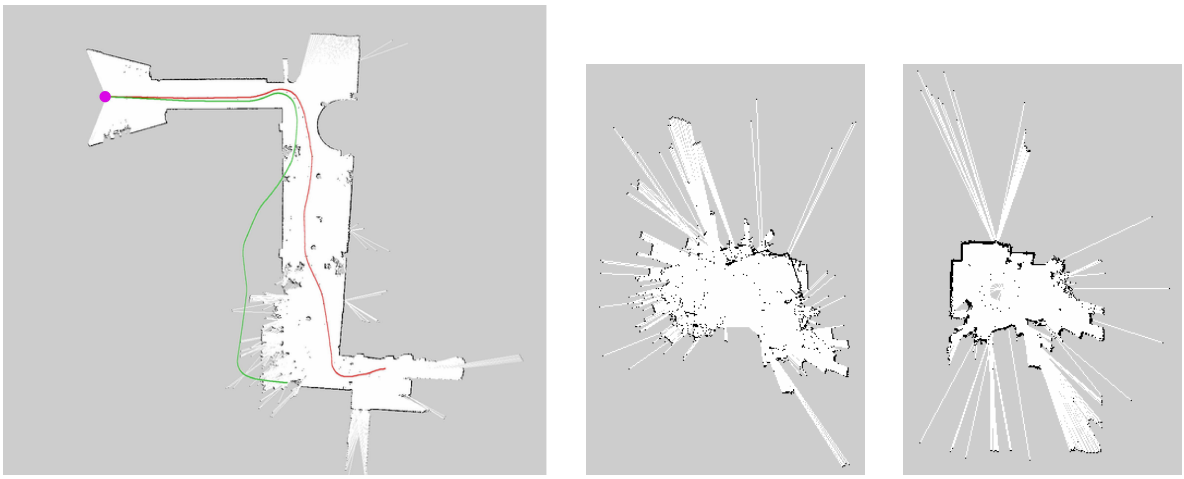
(b) Simulation zweier Kreise mit 40 ° und 60 ° Lenkwinkel durch das Modell des Differentialantriebs

Abbildung 5.1: Simulation der Odometrieberechnung

Orientierungsachse des virtuellen Rades eingesetzt werden müsste. Diese Distanz wird für große Winkel größer als der Abstand der gefahrenen Radien. Somit würde die Änderung des Winkels $\Delta\theta$ kleiner werden und folglich die x- und y Koordinaten der Odometrie näher am eigentlich zu fahrenden Kreis liegen. Eine Überprüfung des Modells durch Berechnung oder die Verwendung der Approximation über die Differenz der Kreisradien ist möglich, liefert aber gegenüber dem verwendeten Modell keine Vorteile.

5.2 Kartierung mit GMapping und Lokalisierung mit AMCL

Zur Erstellung einer zweidimensionalen Karte einer unbekannten Umgebung aus den aktuellen Posedaten und den Daten eines Laserscanners wird das Modul *GMapping* verwendet. Die benötigten Posedaten bekommt die ROS Paket *gmapping* über die Odometrie, die mit einer Frequenz von 30 Hz publiziert wird. Die Karte wird als PGM-Datei (*Portable Graymap*) erzeugt, zusätzliche Metadaten werden in einer YAML-Datei (*YAML Ain't Markup Language*) abgelegt. Im Experiment wurde zunächst eine Karte des Hangeschosses des Informatikgebäudes der Universität Würzburg erstellt. Anschließend wurde ein Pfad aufgezeichnet und die Odometrie- sowie AMCL-Daten bestimmt. Abbildung 5.2(a) zeigt eine Überlagerung der Karte mit den Daten. Dies veranschaulicht die Funktionsweise von SLAM. Der Ausgangspunkt der Aufnahme ist durch einen magentafarbenen Punkt markiert. AMCL und Odometrie stimmen hier noch überein. Würde die Erstellung der Karte ohne Korrektur der Odometriedaten und Abgleich mit dem bereits aufgenommenen Teil der Karte stattfinden, wäre der grüne Odometriepfad die Referenz zur Bestimmung der Abmessungen der Karte. Ab der ersten Kurve würde sich die Karte



(a) Hanggeschoss des Informatikgebäudes der Universität Würzburg mit überlagerten Odometrie- (grün) und AMCL-Daten (rot)

(b) Falscher Grundriss Control Lab Universität Würzburg

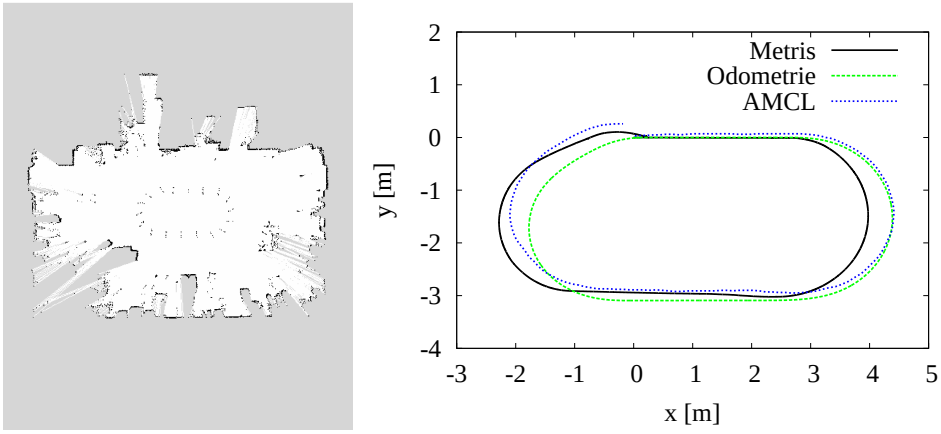
(c) Korrekter Grundriss Control Lab Universität Würzburg

Abbildung 5.2: Kartierung mit *gmapping*

so verschieben, dass die Odometrie immer innerhalb des Grundrisses liegt. Ein Beispiel für eine falsch erzeugte Karte durch fehlerhaft publizierte Odometriedaten findet sich in Abbildung 5.2(b), die korrekt erzeugte Karte ist in Abbildung 5.2(c) zu sehen. Die von AMCL erzeugte Trajektorie (Abbildung 5.2(a), rot) ist eine Schätzung der Position anhand der Daten des Laserscanners und der mit *gmapping* erzeugten Karte. Der Vorteil von AMCL gegenüber der Odometrie besteht darin, dass die Fehler sich nicht aufsummieren. Selbst bei theoretisch unendlich langer Laufzeit sind die von AMCL erzeugten Fehler nicht größer als die Fehler bei kurzer Laufzeit. Die Fehler in der Odometrie summieren sich bei längerer Laufzeit, somit wird die Qualität der Positionsbestimmung mit zunehmender Dauer des Vorgangs schlechter. AMCL benötigt eine komplette Karte des Gebiets, in dem es angewendet werden soll, die Berechnung der Trajektorie kann in einer unbekannten Umgebung also erst nachträglich erfolgen. Die Odometrie hingegen wird in Echtzeit bestimmt. Zusätzlich zu den genannten Informationen, die *amcl* und *gmapping* benötigen, ist auch eine sogenannte *tf-Transformation* als Eingabeparameter notwendig. *tf* ist ein Paket in ROS, welches die Verhältnisse zwischen sämtlichen verwendeten Koordinatensystemen und deren Transformation untereinander aufzeichnet und mit einem Zeitstempel versieht. Als einfachstes Beispiel kann die Transformation zwischen Weltsystem und Robotersystem genannt werden. Somit können zu beliebigen Zeitpunkten die verschiedenen Eingabeparameter zwischen sämtlichen Systemen transformiert werden.

5.3 Pfaderzeugung und -verfolgung

Die im Folgenden beschriebenen Experimente wurden in der Robotikhalle der Universität Würzburg durchgeführt. Die Halle ist mit dem iSpace-System *Metris* der Firma Nikon Metrology, bestehend



(a) Grundriss der Robotikhalle (b) AMCL und Odometrie im Vergleich mit Metris

Abbildung 5.3: Kartierung der Robotikhalle

aus sechs an den Wänden montierten Transmittern sowie verschiedenen Sensoren, ausgestattet. Ein Stereosensor zur 6D-Lokalisierung wurde mittig über der angetriebenen Achse auf dem Roboter angebracht, um so die tatsächlich gefahrene Trajektorie aufzeichnen zu können. Die von Metris aufgezeichneten Trajektorien sind in den folgenden Abbildungen stets als schwarze Linie dargestellt. Um auch die Genauigkeit von AMCL zu testen, wurden mit *gmapping* zunächst eine Karte der Robotikhalle (Abbildung 5.3(a)) erstellt. In Abbildung 5.3(b) sind die Trajektorien der Odometrie- und AMCL-Daten im Vergleich mit der tatsächlichen Trajektorie nach Metris dargestellt. Der in Kapitel 5.2 *Kartierung mit GMapping und Lokalisierung mit AMCL* bereits erwähnte, kontinuierliche Fehler von AMCL ist besonders gut auf den drei Metern vom Nullpunkt aus rechts erkennbar. Diese Abweichung von ungefähr 7,5 cm in y-Richtung ist auf dieser Strecke konstant. Typisch für eine Lokalisierung durch AMCL ist auch die Unruhe der Bahn. Die Pose wird nach genügend großer zeitlicher und räumlicher Änderung anhand der Odometrie- und Laserdaten neu geschätzt, jedoch wird kein Zusammenhang zu den vorherigen Werten hergestellt und diese folglich nicht korrigiert.

Um die Qualität der Odometrieberechnung besser darstellen zu können, wurden die Metrisdaten mit den Odometriedaten zeitlich abgeglichen und wie in Abbildung 5.4 zu sehen dargestellt. Die Farben der einzelnen Punkte stehen für ihre jeweilige Abweichung zum korrespondieren Messwert der Metrisdaten. Die besonders im unteren Bereich der Bahn teilweise hohen Abweichungen von bis zu einem Meter lassen vermuten, dass die Fahrwerksparameter, die für die Berechnung der Odometrie verwendet werden, nicht korrekt sind. Zufällige Fehler wie seitliches Rutschen des Roboters oder Durchdrehen der Räder können als unwahrscheinlich angesehen werden, da die grundsätzliche Form der Odometrietrajektorie mit der wirklichen Bahn übereinstimmt und auch der Endpunkt nur wenige Zentimeter vom wahren Punkt abweicht. Durchschnittlich beträgt die Abweichung auf der gesamten Strecke ungefähr 46 cm.

Als nächstes wurde der Einfluss des getrennten Fahr- und Lenkvorgangs auf die Odometrieberechnung getestet. An den in Abbildung 5.5 ersichtlichen Kanten in der von Metris bestimmten

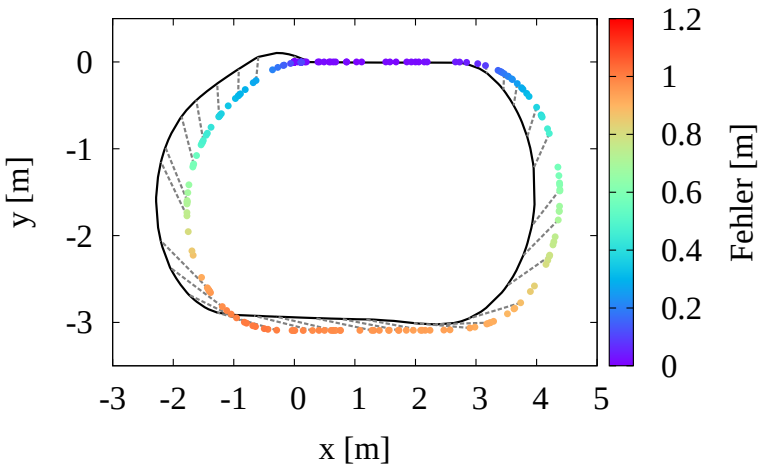


Abbildung 5.4: Abweichung der Odometriedaten von der wahren Trajektorie

Trajektorie wurde der Roboter angehalten, ein neuer Lenkwinkel wurde eingestellt und die Fahrt danach fortgesetzt. Auch ein großer Lenkwinkel wie an der durch einen blauen Kreis markierten Stelle wird nicht registriert. Diese Kanten tauchen in der Odometriekurve nicht auf, da die Änderung der Ticks, aus denen die Odometrie berechnet wird, nicht schnell genug erfolgt. Dies führt zu einer starken Verfälschung der Messung, die Abweichung der berechneten Endposition von der tatsächlichen Position beträgt ungefähr 1,8 m.

Ein großes Problem für die Genauigkeit der Odometrieberechnung bei Differentialantrieben ist die Geschwindigkeit. Bei höheren Geschwindigkeiten und engeren Kurven neigen die Räder zum Durchdrehen und Wegrutschen. Um zu testen ob dies auch auf die Ackermann Lenkung zutrifft, wurde versucht, zwei ähnliche Trajektorien mit niedriger und hoher Geschwindigkeit zu fahren. Hierbei wurden die in Abbildung 5.6 dargestellten Ergebnisse erzielt. Gut zu erkennen sind wieder die systematischen Fehler, die wie bei der Kartierung zu teilweise erheblichen Abweichungen führen. Bekräftigt wird die Annahme, dass es sich nicht um zufällige Fehler handelt, durch die Farbgebung des Plots an den magenta markierten Stellen. Hier heben sich die Fehler nach Fahren einer Rechts- und Linkskurve nahezu auf, eine solche Korrektur wäre bei zufälligen Fehlern sehr unwahrscheinlich. Da die Qualität der Odometrieberechnung für beide Geschwindigkeiten ähnlich ist und die durchschnittliche Abweichung bei niedriger Geschwindigkeit 58 cm bzw. bei hoher Geschwindigkeit 53 cm beträgt, kann gesagt werden, dass die Geschwindigkeit keine Auswirkung auf die Odometrieberechnung hat.

Abschließend wurde der verwendete Regler getestet. Der Regler benötigt als Eingabeparameter die aktuelle Pose des Roboters und berechnet im Zusammenhang mit den Koordinaten des abzufahrenden Pfades die notwendige Orientierungsänderung $\Delta\Phi$. Zum Abfahren der Trajektorie wird eine konstante Geschwindigkeit sowie eine konstante Beschleunigung zur Änderung der tatsächlichen Geschwindigkeiten der Räder vorgegeben. Die Geschwindigkeit zum Ausrichten

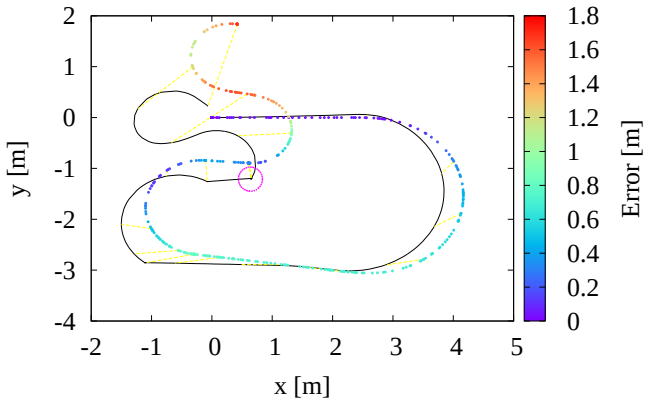
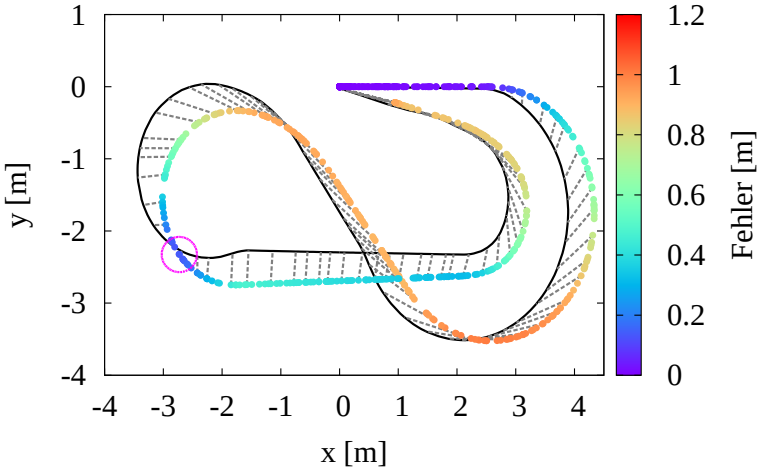


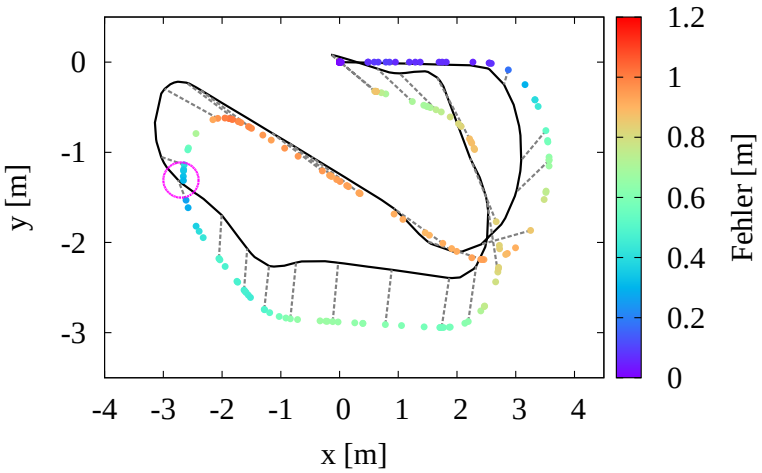
Abbildung 5.5: Odometrieberechnung bei getrenntem Fahr- und Lenkvorgang

der Räder wird weiterhin über die Schnittstelle zu ROS bestimmt. Der Versuch wurde zweimal durchgeführt. Bei den in Abbildung 5.7(a) aufgenommenen Daten reagierte der Roboter auf jede Winkeländerung, die der Regler berechnete. Abbildung 5.7(b) zeigt das Ergebnis für eine Hysterese von 0,2 Rad, sämtliche Winkeländerungen unter diesem Schwellwert wurden also ignoriert, erst bei einer größeren Änderung reagierte der Roboter. Durch die verzögerte Reaktion entsteht die teilweise vorhandene Oszillation. Diese Einschränkung führt zu einer ungenaueren Odometriebestimmung, im Mittel weichen die Berechnungen für eine Pfadverfolgung ohne Hysterese um 53 cm ab, mit Hysterese steigt dieser Wert auf 62 cm.

Auffällig bei sämtlichen Experimenten ist, dass die reale Trajektorie stets um einige Zentimeter in negativer x-Richtung verschoben ist. Dies bestätigt die Vermutung, dass die Fahrwerksparameter nicht korrekt sind und es sich um systematische Fehler handelt. Zufällige Fehler würden sich nicht in Form einer immer ähnlichen Verschiebung der Trajektorie ausdrücken, sondern durch beispielsweise einen nicht nachvollziehbaren Knick innerhalb der tatsächlichen Trajektorie sichtbar werden. Die allgemeine Form der Trajektorien würde sich grundlegend von denen in der Odometrie berechneten unterscheiden. Die Experimente lassen allgemein den Schluss zu, dass die Odometrieberechnung des Roboters funktioniert. Die berechneten Pfade stimmen in ihrer Form mit den tatsächlich gefahrenen überein, die Odometrieberechnungen und ihre Fehler verhalten sich so, wie man es erwarten würde.

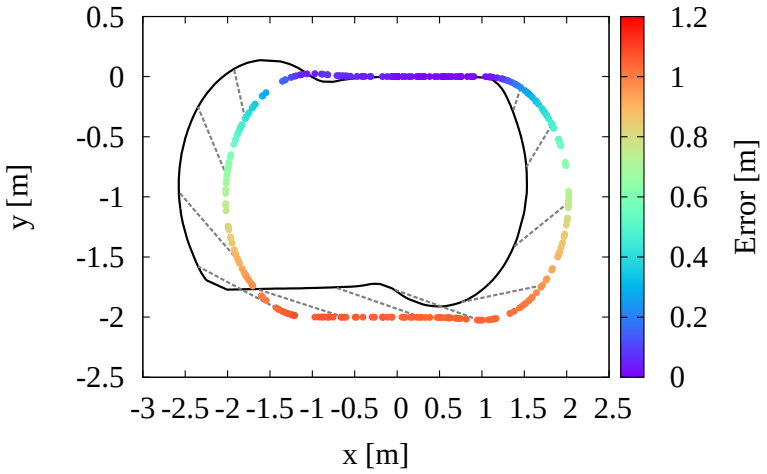


(a) Niedrige Geschwindigkeit

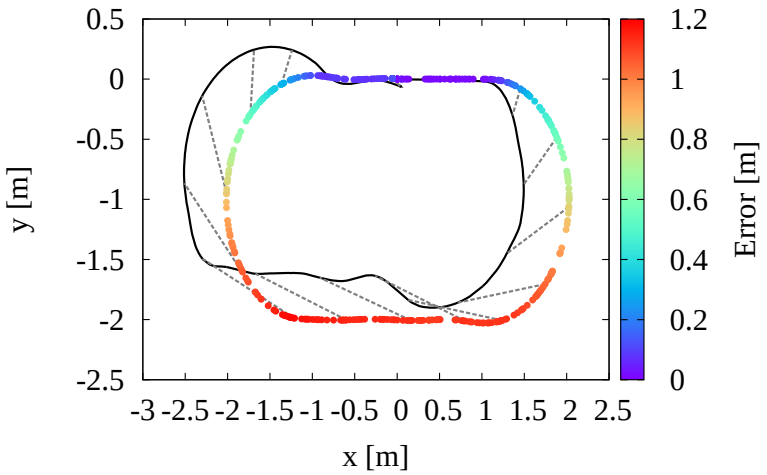


(b) Hohe Geschwindigkeit

Abbildung 5.6: Vergleich der Odometriegenauigkeit bei unterschiedlichen Geschwindigkeiten



(a) Keine Hysterese



(b) 0,2 Rad Hysterese

Abbildung 5.7: Vergleich der Odometriegenauigkeit bei der Pfadverfolgung mit unterschiedlichen Hysteresen bei der Einstellung der Lenkwinkel

Kapitel 6

Schlussbetrachtung

Die zu Beginn der Arbeit formulierten Ziele wurden erfüllt. Mit dem implementierten ROS Package kann der Roboter über zwei unterschiedliche Eingabesysteme gesteuert werden. Zur Erstellung von 2D Karten ist die Unterstützung für *GMapping* integriert, zur Lokalisation stehen wahlweise Odometriedaten oder AMCL-Daten zur Verfügung. Die Systeme wurden auf Funktionalität geprüft und lieferten die zu erwartenden Ergebnisse. Pfadverfolgung ist über die verwendete Implementierung des Algorithmus von G. Indiveri problemlos möglich.

Zur Verbesserung der Qualität der Odometriebestimmung und Optimierung der Fahreigenschaften können in Zukunft noch einige, im Folgenden kurz angesprochene Schritte vorgenommen werden.

Die Parameter für die PI- und PID-Regler, die wie in Kapitel 4.3.1 angesprochen bestimmt wurden, stellen lediglich eine erste Annäherung an die optimalen Parameter dar. Besonders auffällig ist dies im laufenden Betrieb bei einem Beschleunigungsvorgang. Die gewünschte Geschwindigkeit wird zunächst überschossen und der anschließende Regelungsvorgang benötigt einige Sekunden. Optimal können die Regler mit verschiedenen Methoden, beispielsweise durch iterative Parameterbestimmung, eingestellt werden.

Ebenso kann durch weitere Experimente die Odometrieberechnung verbessert werden. Da die Formeln, wie die Simulation zeigt, korrekt sind, können, durch experimentelle Bestimmung der tatsächlichen Fahrwerksparameter, die systematischen Fehler in der Berechnung minimiert werden. Hierzu zählen beispielsweise die realen Untersetzungen der Getriebe, die von den aus den Datenblättern berechneten Werten durchaus abweichen können, oder die Abmessungen des Fahrzeugs wie Spurbreite und Radstand. Zur Optimierung der Odometrieberechnung können die in [LCY10] vorgestellten Methoden ein erster Anhaltspunkt sein.

Da die Positionsinformationen, die *Metris* liefert, häufiger als die Odometrie des Roboters publiziert werden, mussten die Daten abgeglichen werden. Hierzu findet eine Überprüfung der Zeitstempel der jeweiligen Daten statt. Ist ein Zeitstempel von *Metris* näher als 20ms an dem der betreffenden Odometriedaten, so werden die Daten extrahiert und in eine gesonderte Datei geschrieben. Diese Methode ist eine schnelle und einfache Variante, die Daten zu syn-

chronisieren. Da die von *Metris* gesendeten Daten jedoch nicht regelmäßig, sondern, durch die Verzögerung über WLAN bedingt, in Blöcken ankommen und weichen so die ROS Zeitstempel von dem tatsächlichen Zeitpunkt der Aufnahme ab. Zur korrekten Darstellung ist eine Linearisierung der Metrisdaten mit Zeitsynchronisation zwischen den Metris-Zeitstempeln und den ROS-Zeitstempeln. Folglich sind die im Kapitel 5.3 *Pfaderzeugung und -verfolgung* dargestellten Plots der Messwerte in Wirklichkeit etwas besser.

Die Implementierung des verwendeten Algorithmus zur Pfadverfolgung ist für differentialgetriebene Roboter optimiert. Wird an die Qualität der Pfadverfolgung eine höhere Anforderung gestellt, als mit der Methode, die in dieser Arbeit erklärt wurde gewährleistet werden kann ist eine Überarbeitung der Implementierung notwendig.

Für Einsätze in schwierig zu befahrenden Umgebungen beim Einsatz zur 3D-Modellerstellung müssten weitere Tests vorgenommen werden, um das Verhalten und die Qualität der Lokalisierungsmethoden zu überprüfen. Die Reichweite der Batterien ist ein weiterer Punkt, der noch nicht getestet wurde.

Abschließend kann gesagt werden, dass der Roboter problemlos zur Forschung verwendet werden kann und so die Einsatzmöglichkeiten mobiler Roboter an der Universität Würzburg erweitert.

Literaturverzeichnis

[BHE+15] Borrmann, Dorit; Hess, Robin; Eck, Daniel; Houshiar, Hamidreza; Nüchter, Andreas; Schilling, Klaus: Evaluation of Methods for Robotic Mapping of Cultural Heritage Sites. In: *Proceedings of the 2th IFAC conference on Embedded Systems, Computer Intelligence and Telematics (CESCIT '15)* (2015). – Maribor, Slovenien

[Bos] Boston Dynamics: *Dedicated to the Science and Art of How Things Move.* `http://www.bostondynamics.com/`, . – Zugriff am 7. September 2015

[Bra06] Braunl, Thomas: *Embedded Robotics.* Second Edition. Springer-Verlag Berlin Heidelberg, 2006

[CKSS93] Canudas de Wit, Carlos; Khennouf, H.; Samson, Claude; Sordalen, Ole J.: Nonlinear control design for mobile robots. In: *Recent Trends in Mobile Robots*, Yuan F. Zheng, 1993

[DL] De Luca, Alessandro: *Robotics 1 - Wheeled Mobile Robots - Introduction and Kinematic Modeling.* `http://www.dis.uniroma1.it/~deluca/rob1_en/16_MobileRobotsKinematics.pdf`, . – Sapienza Università di Roma

[Dud05] Dudziński, Piotr: *Lenksysteme für Nutzfahrzeuge.* Springer-Verlag Berlin Heidelberg, 2005

[Ets94] Etschberger, Konrad: *CAN - Controller-Area-Network - Grundlagen, Protokolle, Bausteine, Anwendungen.* Carl Hanser Verlag München Wien, 1994

[FKID03] Fujita, Masahiro; Kuroki, Yoshihiroand; Ishida, Tatsuzo; Doi, Toshi T.: Autonomous Behavior Control Architecture of Entertainment Humanoid Robot SDR-4X . In: *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems* (2003), Oktober. – Las Vegas, Nevada

[Fra] Fraunhofer Institut für Intelligente Analyse- und Informationssysteme: *Volksbot Robotersysteme.* `http://www.volksbot.de`, . – Zugriff am 7. September 2015

[Gnu] Gnuplot: *Gnuplot Copyright.* `http://www.gnuplot.info/`, . – Zugriff am 7. September 2015

[HLN12] Hertzberg, Joachim; Lingemann, Kai; Nüchter, Andreas: *Mobile Roboter.* Springer-Verlag Berlin Heidelberg, 2012

[IC04] Indiveri, Giovanni; Corradini, Maria L.: Switching Linear Path Following for Bounded Curvature Car-Like Vehicles. In: *Proceedings of the 5th Symposium on Intelligent Autonomous Vehicles* (2004). – Lissabon

[LCY10] Lee, Kooktae; Chung, Woojin; Yoo, Kwanghyun: Kinematic parameter calibration of a car-like mobile robot to improve odometry accuracy. In: *Elsevier - Mechatronics* (2010)

[Lyn] Lynxmotion: *Lynxmotion Robot Kits*. http://www.lynxmotion.com/, . – Zugriff am 7. September 2015

[Max] Maxon Motor AG: *Epos Studio*. www.maxonmotor.com, . – Zugriff am 7. September 2015

[Max06] Maxon Motor AG (Hrsg.): *EPOS Communication Guide*. CH-6072 Sachseln: Maxon Motor AG, Oktober 2006

[Max08] Maxon Motor AG (Hrsg.): *Application-Note "RS232 to CANopen Gateway*. CH-6072 Sachseln: Maxon Motor AG, Mai 2008

[Max10] Maxon Motor AG (Hrsg.): *EPOS Firmware Specification*. CH-6072 Sachseln: Maxon Motor AG, April 2010

[Nik14] Nikon Metrology: *iSpace – Portable Metrology System User Manual and Startup Guide*. http://www.nikonmetrology.com, July 2014

[Ope15] Open Source Robotics Foundation: *ROS-cheatsheet-catkin*. https://github.com/ros/cheatsheet/releases/tag/0.0.1, 2015. – Zugriff am 7. September 2015

[Pea] PEAK-System Technik GmbH: *Peak System*. http://www.peak-system.com/PCAN-USB.199.0.html?&L=1, . – Zugriff am 7. September 2015

[Sch08] Schwake, Benjamin: *Integration einer Achsschenkellenkung in das VolksBot Roboterbaukastensystem*, Fachhochschule Bonn-Rhein-Sieg, Diplomarbeit, 2008

[SIC15] SICK AG (Hrsg.): *LMS1xx Laser Measurement Sensors Operating Instructions*. Waldkirch: SICK AG, Juli 2015

[SK07] Spenneberg, Dirk; Kirchner, Frank: The Bio-Inspired SCORPION Robot: Design,Control & Lessons Learned, Climbing and Walking Robots: towards New Applications. In: *Climbing and Walking Robots: towards New Applications* (2007)

[TBF06] Thrun, Sebastian; Burgard, Wolfram; Fox, Dieter: *Probabilistic Robotics*. The MIT Press, 2006

[Trz10] Trzesniowski, Michael: *Rennwagentechnik - Grundlagen, Konstruktion, Komponenten, Systeme*. 2. Auflage. Vieweg+ Teubner Verlag, 2010

[Wil] Willow Garage: *Robot Operating System*. http://www.ros.org/, . – Zugriff am 7. September 2015

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Würzburg, September 2015