Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Ein automatischer Parkassistent auf Basis einer Laserscanner-Abstandserfassung für ein fahrerloses Transportsystem

Ning Liu

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Ein automatischer Parkassistent auf Basis einer Laserscanner-Abstandserfassung für ein fahrerloses Transportsystem

Ning Liu

Masterarbeit eingereicht im Rahmen der Masterprüfung
Verteilte Systeme im Studiengang Technische Informatik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. -Ing. Bernd Schwarz
Zweitgutachter : Prof. Dr. -Ing. Andreas Meisel

abgegeben am: 17. Juli 2008

Ning Liu

Thema der Masterarbeit

Ein automatischer Parkassistent auf Basis einer Laserscanner-Abstandserfassung für ein fahrerloses Transportsystem

Stichworte

Fahrerassistenzsystem, Einparkassistent, Abstandsregelung, kinematisches Fahrzeugmodell, Odometrie, reaktive ereignisgesteuerte Systeme, POSIX-Threads, Harel-Automaten, FAUST, ARM7TDMI, GEME 2000, Pentium, Linux, Sick-Laserscanner

Kurzzusammenfassung

Bei dem fahrerlosen Transport System handelt es sich um ein verteiltes Echtzeitsystem, in dem die Steuergeräte und Rechner unterschiedliche Aufgaben und Steuerungen übernehmen. Diese Arbeit dokumentiert die Entwicklung eines Einparkassistenten, der die Umgebung eines fahrerlosen Transportsystems mit einem Laserscanner erfasst. Der Abstandregler besteht aus einem PD-Regler. Bei der Modellierung des Einparkassistenten wird die Harel-Automaten-Notation verwendet und in einer Laufzeitumgebung implementiert, welche Datenstrukturen für Automaten und ein Reaktionsverhalten gemäß der Harel-Notation bietet. Anschließend wird die Portierung des Einparkassistenten auf einem Microcontroller dokumentiert.

Ning Liu

Title of the paper

An automatic parking assistant based on laser scanner-distance coverage for a driverless transport system

Keywords

Driver Assistance System, parking, distance control, kinematic vehicle model, odometry, reactive system, POSIX threads, Harel infinite state machines, FAUST, ARM7TDMI, GEME 2000, Pentium, Linux, Sick laser scanner

Abstract

The driverless transportation system is a distributed real-time system, in which embedded systems lead different tasks and controls. This thesis documents the development of an automatic parking system, which scans the environment of a transport system with a laser scanner. The distance control consists of a PD control. The modeling of the automatic parking system is designed with the Harel infinite state machines notation and implemented within a run-time environment, which includes data structures for the states and a reaction algorithm correlating to the Harel notation. The automatic parking system will be ported into a microcontroller..

Für meine Eltern

Inhaltsverzeichnis

1. Einleitung

Der Schwerpunkt des Informatik-Masterstudiengangs der HAW Hamburg ist "Verteilte HW/SW-Systeme". Begleitend zu den Vorlesungen werden in zwei Semestern Projekt- und Seminarkurse angeboten, die eine kontinuierliche Vertiefung im Bereich der eingebetteten Systeme für Echtzeitanwendungen verschaffen. Dort werden die Grundlagen sukzessive erarbeitet und im Anschluss in den fahrerlosen autonomen Systemen (FAUST) Fahrerassistenzsysteme entwickelt. Das übergeordnete Thema FAUST bietet mehrere Plattformen für die Implementierung. Die Masterstudenten wenden die für die Industrie relevante Grundlage in den Bereichen der Sensorik, Bussysteme, und konzeptionelle Entwurfsmuster für Echtzeitsysteme an. Im fachübergreifenden HAWKS Racing Team werden im Informatik Department, Sensorik und Telemetrielösungen an einem Rennfahrzeug entwickelt. Die Aufgabenstellung im Intelli Truck Projekt besteht darin, vorgegebene Kurse autonom abzufahren. Beim Sensor Controlled Vehicle (SCV) werden Fahrerassistenzsysteme für ein Flurförderfahrzeug entwickelt, die konzeptionell aus der Kfz-Industrie stammen.

Neben dem technischen Anspruch hat die Einführung von Fahrerassistenzsystemen eine wirtschaftliche Relevanz, denn die Untersuchung einer Unternehmensberatung gibt an, dass ein Wachstum von 14% bis zum Jahre 2010 im Markt für Fahrerassistenzsysteme zu erwarten ist [29]. Die höhere Sicherheit von Automobilen, ist nach deren Untersuchung einer der wichtigsten Verkaufsgründe für Fahrerassistenzsysteme. Die Funktionalitäten von Fahrerassistenzsystemen wie Tempomat, Fahrspurwechsel- oder Einparkassistenten, tragen dazu bei, den Fahrer zu entlasten und kritische Fahrsituationen schneller zu erkennen. In alltäglichen Routinemanövern wie Abstand halten, Fahrspurwechsel oder Einparken werden die Sinne des Fahrers beansprucht. Durch menschliches Versagen werden Situationen falsch eingeschätzt, oder auf diese zu spät reagiert. Fahrerassistenzsysteme schaffen hier Abhilfe, im Gegensatz zum Fahrer, sind sie in der Lage Dimensionen Millimeter genau abzumessen und innerhalb von einem Bruchteil einer Sekunde zu reagieren. Qualitativ lässt sich diese Aussage durch das Sinken der Unfallzahlen mit den Statistiken des Bundesamtes nachweisen [9].

Diese Masterarbeit konzentriert sich auf die Entwicklung eines Einparkassistenten. Dieser Einparkassistent soll mit einer lasergestützten Abstandserfassung, die autonome Fahrfunktion das seitlichen Einparkens übernehmen. Der Einparkassistent führt mit der Einstellung des Lenkwinkels, das SCV bei einer konstanten Geschwindigkeit von 0,5 m/s entlang einer Trajektorie in die Parklücke. Mit der Odometrie, die auf einem nichtlinearen kinematischen Einspurfahrzeugmodells beruht, wird die Bewegung des SCVs in die Parklücke modelliert. Für die Implementierung der Odometrie müssen die Gleichungen des kinematischen Modells mit dem Runge-Kutta-Verfahren online numerisch integriert werden.

Als Versuchsplattform wird das SCV des Departments Informatik verwendet, welches als Entwicklungsplattform für aktive und passive Fahrerassistenzsysteme dient. (vgl. Abb. 1). In den vorausgegangen Masterarbeiten, wurden ein Brems-[7] und zwei Ausweichassistenten [20] [23], sowie ein zeitgesteuertes und verteiltes SW-Konzept entwickelt [24]. Die Fahrerassistenzsysteme verwenden, wie der Einparkassistent, die Abstandserfassung durch einen Laserscanner und greifen aktiv in die Steuerung des SCVs ein.

In einer vorausgegangenen Master Projektarbeit [26] wurden die Grundlagen für eine Weiterentwicklung gelegt. Das SCV lässt sich über einen Leitstand fernsteuern, da für die interne und externe Kommunikation ein WLAN-Router, ein CAN-Bus und ein Kommunikationsrechner in das Fahrzeug installiert wurde. Zusätzlich ist im Laufe der Projektarbeit des Masterkurses im Wintersemester

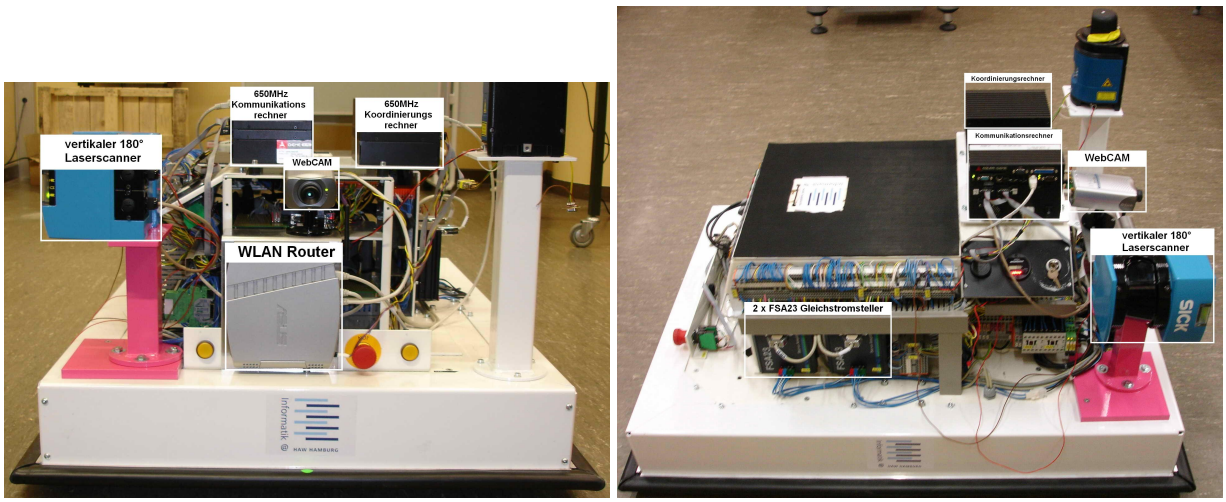

Abb. 1: Front- und Aufsicht des SCVs

06/07 das SCV um einen 180° Laserscanner erweitert worden. Die Hardwarekomponenten des SCV lassen sich in drei Bereiche aufteilen. (vgl. Abb. 2)

- Die Antriebseinheit regelt die Geschwindigkeit des Fahrantriebes und die Einstellung des Lenkwinkels für das Vorder- und Hinterrad.
- Die Fahreinheit übersetzt die Befehle des Kommunikationsrechners für die jeweiligen Räder die von der Bedieneinheit übertragen werden.
- Mit der Bedieneinheit wird das SCV über einen Leitstand gesteuert und misst die Abstände über einen angeschlossenen Laserscanner.

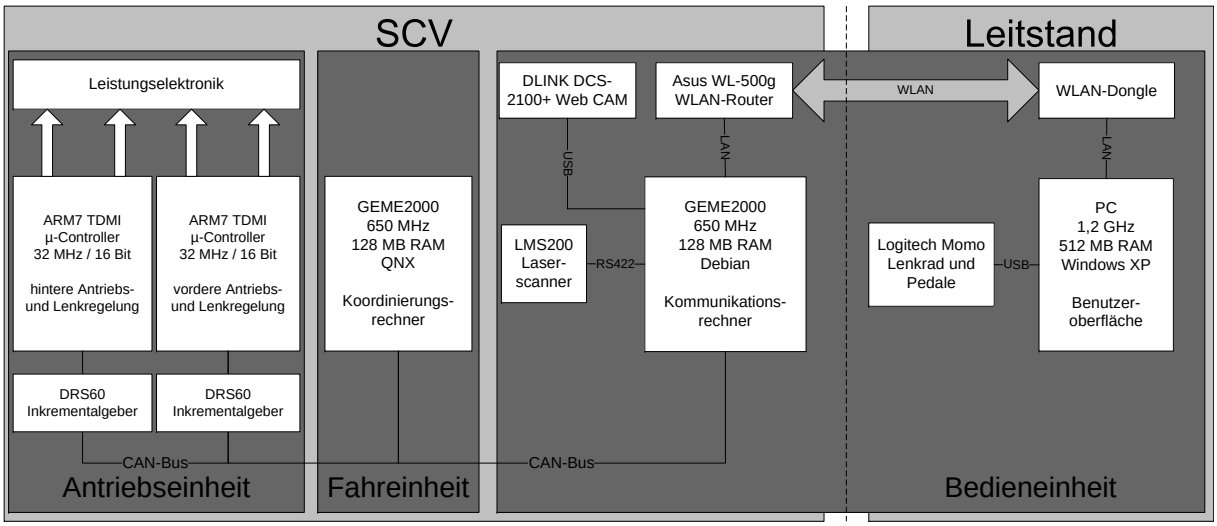

Abb. 2: Übersicht zu den SCV Hardwarekomponenten

Es wurden zwei GEME-Rechner mit 650 MHz installiert, um den Bedarf an Rechnerressourcen für zukünftige Erweiterungen zu erfüllen. Die Rechner können durch die PC104 Steckkarten modular erweitert werden, so kann das SCV je nach Bedarf mit zusätzlichen Sensoren für zusätzliche Fahrerassistenzsystemen ausgestattet werden. Mit der Installation von Linux verringert sich der Entwicklungssaufwand, da nicht proprietäre Treiber für die CAN und den Sick Laserscanner verwendet werden und diese Treiber auf andere Embedded Systems portiert werden können, die in ANSI C entwickelt wurden.

In der Antriebseinheit wird die Leistungselektronik beider Räder, mit jeweils einem ARM7 TDMI μ-Controller gesteuert. Die dort realisierten digitalen Regler erzeugen die Stellgrößen für Antrieb und Lenkwinkel über zwei Servos, des Weiteren lesen zwei Inkrementalgeber die Geschwindigkeit bzw. den Lenkwinkel aus. Je nachdem welcher Fahrmodus des SCVs eingestellt ist, wird der Lenkwinkel des Vorder- oder Hinterrads geregelt. Die Fahreinheit koordiniert die Regelung beider Räder. Diese nimmt die Nachrichten der Soll-Wert für Geschwindigkeit und Lenkwinkel vom Kommunikationsrechner über den CAN-Bus entgegen, bereitet sie für die Servos auf und leitet sie mittels CAN an die jeweiligen ARM7 Mikrocontroller weiter. Auf Seiten des SCVs wird ein Kommunikationsrechners eingesetzt, an der zwei Sensorik- und zwei Kommunikationskomponente angeschlossen sind:

- LMS200, vertikaler 180° Laserscanner mit 75Hz Scanfrequenz und serieller RS232/422 Schnittstelle
- Web Cam mit einer Auflösung von 320 x 240 Pixel angeschlossen über USB
- 801.11g WLAN-Router
- CAN-Bus

Der Leitstand bildet mit der grafischen Oberfläche die Schnittstelle zwischen Mensch und Maschine. Er besteht aus einem PC, an dem ein Lenkrad mit Fußpedalen und ein WLAN-Dongle angeschlossen sind. Mit dem Lenkrad und den Pedalen wird das SCV gesteuert und mit den Knöpfen des Lenkrades der Einparkassistent gestartet. Um die vom Leitstand übertragenden Steuerungsbefehle auf den ARM-μ-Controller auszuführen, müssen die Nachrichten von WLAN auf CAN durch den Kommunikationsrechners übersetzt werden. Die erzeugten Soll- und die tatsächlichen Ist-Werte über Geschwindigkeit und Lenkwinkel, sowie das Aufnahmebild der Web CAM, werden über die mit Labview implementierte, graphische Oberfläche ausgegeben.

Bei der Entwicklung ist das Konzept eines in der Fahrzeugindustrie vorhandenen Einparkassistenten übernommen worden. Zum Aufgabenfeld gehören folgende Aspekte:

- Auswertung der Abstandswerte vom 180° Laserscanner für die Parklückenidentifikation und während des Einparkvorgangs.
- Integration des 180° Laserscanners, der Antriebs- und Lenkungsregelung, des Leitstands und der Schnittstellen WLAN und CAN.
- Software Design für das reaktive System als Harel-Automaten auf dem Kommunikationsrechner
- Implementierung der Software mit Pthreads für die parallele Ausführung der Automaten
- Entwurf eines kinematischen Einspurfahrzeugmodells zur Bestimmung der Position während der Fahrt.
- Realisierung der numerischen Integration mit ANSI C zur online Berechnung der Fahrzeugposition
- Entscheidungsalgorithmus, der den Übergang zwischen lasergestützter und virtueller Positionserfassung ermittelt.
- Simulation der virtuellen Deichsel als impliziten Regler in Dymola, zur Analyse der Genauigkeit der Odometrieimplementierung
- Messtechnische Analyse einer Abstandsreglung mit einem digitalen PD-Regler
- Integration des Einparkassistenten auf einem μ-Controller

Es wurden die Zeitkennwerte T_{LS} und T_{CAN} ermittelt, die in der Masterarbeit Verwendung finden:

T	Einheit	Beschreibung
13,32	ms	T_{LS} = Scanzyklus des vertikalen, 180° Laserscanners
50	ms	T_{CAN} = Sendezyklus der Ist-Werte von der Antriebseinheit

Damit der Regler das SCV parallel zur seitlichen Parklückenbegrenzung ausrichten kann, muss die Parklücke 1,2 m tief und 2,4 lang sein. Dabei wird bei der Parklückensuche der vertikale Laserscanner verwendet. Dieser misst in zeitdiskreten Abständen T_{CAN} die seitliche Begrenzung der Parklücke. Dazu wird der Laserscanner in der Abtastauflösung parametrisiert, damit in Abhängigkeit der Rechnergeschwindigkeit die Abstandswerte innerhalb des periodischen Scanzyklus T_{LS} verarbeitet werden können.

Bei dem Einparkassistenten handelt es sich um ein reaktives System. Deswegen wird die Software des ereignisgesteuerten verteilten Systems mit den Harel-Automaten modelliert. Zur Verringerung des Implementierungsaufwandes für die Automaten wird die AIRA-Laufzeitumgebung verwendet [15]. Die Laufzeitumgebung bietet Schablonen für Datenstrukturen, Zustände, Automaten, Nachrichten für die Kommunikation unter den Automaten, Schnittstellen für zustandserhaltende und zustandsverändernde Reaktionen und ein Reaktionsalgorithmus an. Zusätzlich werden die Automaten nebenläufig ausgeführt, da die Auswertung der Parklücke innerhalb von Mikrosekunden und der Empfang der Abstandswerte vom Laserscanner in Millisekunden Bereich durchgeführt werden. Während die Messdaten vom Laserscanner empfangen werden, können die restlichen Automaten parallel ausgeführt werden.

Die Einbindung der Antriebs- und Lenkungsregelung und des Leitstandes in die Softwarearchitektur des Einparkassistenten ist ein weiterer Schwerpunkt. Bei der Parklückensuche und der geregelten Fahrt in die Parklücke, werden verschiedene Automaten zur Ansteuerung der Hardwarekomponenten aktiviert. Während der Parklückensuche werden die Lasermessdaten mit dem Verfahrweg des SCVs zusammen ausgewertet. Aufgrund der Ausrichtung des Laserscanners können nur punktuelle Abstände in der Horizontalen erfasst werden. Da die Odometrie die Fahrt des SCVs in die Parklücke modelliert, können zusammen mit den gemessenen Abständen, Aussagen darüber getroffen werden, ob die seitliche Begrenzung der Parklücke, oder andere für den Einparkvorgang nicht relevante Abstände erfasst wurden. Kann der Laserscanner die seitliche Parklückenbegrenzung nicht erfassen, werden die Koordinaten des SCVs aus der Odometrie, die parallel mitberechnet werden, dem Regler zugeführt. Es werden Kriterien für den Übergang zwischen lasergestützter und virtueller Abstandsregelung diskutiert.

Die aus der Odometrie ermittelten X- und Y-Koordinaten des Vorder- und Hinterrades, sowie des Achswinkels werden in einer virtuellen Deichsel [2] verarbeitet, welche aus der Position des Fahrzeugs relativ zum Zielpunkt einen einzuschlagenden Soll-Lenkwinkel für die Fahrt in die Parkbucht berechnet. Überprüft wird das kinematische Einspurfahrzeugmodell in Verbindung mit dem impliziten Regler der virtuellen Deichsel in einer Dymolasimulation, die daraus resultierende Trajektorie für das Einparken, zeigt den Verlauf des Einparkvorgangs in die Parklücke. Für die C-Implementierung der Odometrie werden die nichtlinearen Gleichungen mit dem Runge-Kutta-Verfahren integriert, wobei die Koordinaten des Fahrzeugs aus der Dymola-Simulation als Referenz für die Genauigkeit gelten.

Neben der impliziten Regelung durch die virtuelle Deichsel wird ein digitaler PD-Regler eingesetzt, der in einer Masterarbeit für einen Ausweichassistenten [23] entwickelt wurde. Da durch die Einführung der Abstandserfassung mit der virtuellen und lasergestützten Positionserfassung Sprünge

während des Übergangs zu erwarten sind, soll durch den PD-Regler den Abstand zur Wand geregelt werden. Mit einer messtechnischen Analyse anhand eines Prototyps für den Regler wird die Ausgangsgröße der Lenkwinkel Soll-Vorgabe ausgewertet, und für den Einparkvorgang in die Parklücke parametrisiert.

Es wurde mit der Timing-Analyse der Ausführungszeiten für die Threads festgestellt, dass im Kommunikationsrechner die CPU nicht ausgelastet ist, und der Einparkassistent auf einem Mikrocontroller mit einem 32 MHz CPU ausgeführt werden kann. Deswegen wird ein Konzept für die Portierung des Einparkassistenten auf einem Mikrocontroller aufgestellt. Da eine nebenläufige Ausführung auf den μ-Controller mangels Betriebssystem nicht unterstützt wird, muss das Softwaredesign der nebenläufigen Ausführung der Threads geändert, und durch eine periodische, sequentielle Ausführung ersetzt werden.

Es werden im zweiten Kapitel, zwei Beispiele für vorhandene Einparkassistenten beschrieben. Darauf aufbauend, wird ein Konzept für die Eigenentwicklung erstellt und ein Szenario bestehend aus zwei Phasen dokumentiert.

Die Aufstellung der Gleichungen für die Odometrie und der Regler, wird in Kapitel drei beschrieben. Es werden die numerischen Verfahren vorgestellt, die zur Berechnung der Position des SCVs online, während des Einparkvorgangs, im zeitdiskreten Bereich angewendet werden. Anhand einer Simulation werden die Verstärkungsfaktoren der virtuellen Deichsel parametrisiert, damit die Trajektorie der Einparkkurve die Bedingungen der Parklückengrenzen von 1,2 m Tiefe und 2,4 m Breite erfüllt. Die Positionsdaten der Odometrie aus der Simulation, werden als Referenz für die Genauigkeit genommen.

Im vierten Kapitel werden die einzelnen Hardwarekomponenten des SCVs dokumentiert. Es werden die Steuergeräte, sowie die Sensoren und die Antriebs-, Bedien-, Fahreinheit vorgestellt.

Im fünften Kapitel wird die Modellierung der Software mittels Harel-Automaten beschrieben. Es werden Automatenmodelle und Sequenzdiagramme vorgestellt, die das Zusammenspiel der Automaten untereinander und die Ausführungsfolge dokumentieren.

Die nebenläufige Ausführung der Automaten durch das POSIX-Framework und die Entwicklung der Automaten in der AIRA-Laufzeitumgebung, wird in Kapitel sechs dokumentiert.

Die Ergebnisse der messtechnischen Analyse werden im Kapitel sieben diskutiert. Die Ausführungszeiten für Threads werden anhand von Timing-Messungen dargestellt, sowie die Odometrie-Werte in der Simulation und in der Implementierung miteinander verglichen.

In Kapitel acht, wird die Portierung des Einparkassistenten auf dem ARM7 Mikrocontroller beschrieben. Die in Kapitel sieben ermittelten Ausführungszeiten zeigen, dass der Einparkassistent auf dem 32 MHz Mikrocontroller portiert werden kann.

Im Anhang werden ergänzende Oszilloskop Schnappschüsse der Ausführungszeiten, die Pin-Belegung des PhyCORE Extension-Board, Pthreads Zustände, sowie Code Auszüge zur Ansteuerung der parallelen Schnittstelle, AIRA-Laufzeitumgebung, Odometrie in C/Dymola und der AIRA Laserscanner Automat dokumentiert.

2. Konzepte des Einparkassistenten

In diesem Kapitel werden zwei Einparkassistenten aus der Industrie vorgestellt. Für die Marke Prius und Touran von Volkswagen und Toyota existieren bereits Einparkassistenten. Beide besitzen dieselbe Funktionalität, unterscheiden sich jedoch in der Bedienung. Es werden beide Einparkassistenten beschrieben und analysiert, um eine Grundlage für die Entwicklung eines vergleichbaren Assistenten auf den SCV zu schaffen.

Anschließend wird ein Szenario der zwei Phasen des Einparkassistenten beschrieben. In der ersten Phase wird die Suche nach einer Parklücke mittels einem vertikalen 180° Scanner und der Vermessung des Verfahrwegs dargestellt. In der zweiten Phase wird die Fahrt des SCVs in die Parklücke mit einem nichtlinearen kinematischen Einspurfahrzeugmodell und einem Regler dokumentiert.

Des Weiterem wird die Bedienung des Einparkassistenten über dem Leitstand während der zwei Phasen erläutert.

2.1. Beispiele für Einparkassistenten in Kfz-Anwendungen

Beim Intelligent Park Assist des Prius handelt es sich um einen Einparkassistenten, der die Lenkung des Fahrzeugs, während der Fahrt in die Parklücke übernimmt. Allerdings, muss der Fahrer bei der Parklückensuche über einem Touchscreen selber einschätzen, ob die Parklücke für das Fahrzeug ausreichend groß ist.

Die Aktivierung des IPA erfolgt, sobald der Fahrer den Rückwärtsgang einlegt und den IPA Knopf drückt. Das Bild einer Kamera, die in der Heckklappe des Prius montiert ist, erscheint auf dem Touchscreen. Auf dem Touchscreen des DVD-Navigationssystems, erscheinen mehrere Funktionspfeile, mit denen ein grünes Rechteck verschoben werden kann. (vgl. Abb. 3) Das Rechteck entspricht den Maßen des Fahrzeugs und stellt die gewünschte Endposition des Prius nach dem Einparkvorgang dar.

Abb. 3: Bestimmung der Endposition des Einparkvorgangs im IPA, durch Platzierung des grünen Quadrates mit den Pfeilen auf dem Touchscreen [27].

Der Parkvorgang wird durch zweimaliges Drücken auf dem Touchscreen gestartet, damit bestätigt der Fahrer zuerst die Zielposition und schließt anschließend mit dem Hinweis, dass der IPA die Steuerung übernimmt. Durch das leichte Anheben der Bremse, kontrolliert der Fahrer die Geschwindigkeit des Einparkvorgangs. Ist die Geschwindigkeit zu hoch, wird der Fahrer gewarnt. Reagiert er nicht auf den Hinweis, wird der Vorgang abgebrochen. Beim Einparkvorgang erscheinen zum grünen Rechteck zusätzlich drei horizontale Linien, mit denen lässt sich der Abstand des Hecks messen. (vgl. Abb. 4)Die rote Linie ist 10 cm und die erste gelbe 20 cm vom Heck des Prius entfernt. Die vertikalen Linien stellen die seitlichen Begrenzungen des Fahrzeugs dar. Je nach eingeschlagenem Lenkwinkel, biegen sich die Linien in die entsprechende Richtung und der Fahrer kann den Verlauf des Einparkvorgangs auf dem Bildschirm verfolgen.

Abb. 4: Einparkvorgang mit Linien zur Abstandsüberwachung. Die horizontalen Linien stellen die Entfernung vom Heck (rot 10 cm, gelbe 20 cm) und die senkrechten Linien die seitliche Begrenzung des Fahrzeugs dar. Ein Warnhinweis erscheint, wenn die Geschwindigkeit zu hoch ist [27].

Im Gegensatz zum Prius übernimmt der Park Assist vom VW Touran nicht nur die Steuerung während des Einparkvorgangs, sondern auch die Parklückenanalyse. Das Sensorennetzwerk besteht aus zehn Ultraschallsensoren. Zwei Sensoren links und rechts messen die Tiefe der Parklücke aus, acht sind vorne und hinten platziert, diese messen den Abstand zum vorderen und hinteren Hindernis. Der Einparkassistent wird mit dem Knopf auf dem Bedienfeld aktiviert. Mit der Verwendung des Blinkers wählt der Fahrer aus, ob der Touran links oder rechts einparken soll. Wenn der Touran 30 km/h unterschreitet, schalten sich die Ultraschallsensoren auf der linken oder rechten Seite ein und die Suche für eine Parklücke wird initialisiert. (vgl. Abb. 5) Die Parklücke muss mindestens 5,8 m lang sein und ist damit 1,4 m länger als das Fahrzeug selbst.

Abb. 5: Auf der rechten der Fahrzeugseite wurde eine Parklücke gefunden, die für den Einparkvorgang verwendet werden kann. [28]

Hat der Park Assist eine Parklücke für das Einparken gefunden, muss der Benutzer anhalten. Nach dem Schalten in den Rückwärtsgang, kontrolliert der Fahrer mit dem Gaspedal die Geschwindigkeit des Einparkens, während die Lenkung vom Park Assist übernommen wird.(vgl. Abb. 6) Wenn der Fahrer in das Lenkrad eingreift oder die Geschwindigkeit 7 km/h überschreitet, wird der Parkvorgang abgebrochen. Bordsteinkanten, Personen und Fahrbahnbegrenzungspfähler werden mit den Ultraschallsensoren nicht erfasst, mit der Folge, dass der Fahrer während des Einparkens die Umgebung im Auge behalten muss.

Abb. 6: Nach dem Schalten in den Rückwärtsgang übernimmt Park Assist die Lenkung für den Einparkvorgang. [28]

Der Park Assist des Tourans benötigt für die Fahrt in die Parklücke 15 Sekunden und maximal 2 Züge. Der zweite Zug wird für das Geradeziehen benötigt, falls der erste Zug den Touran nicht parallel zur seitlichen Fahrbahnbegrenzung geführt hat. Dazu muss der Fahrer den Gang wieder auf Vorwärtsfahrt schalten.

Der in dieser Masterarbeit entwickelte Einparkassistent orientiert sich konzeptionell am Park Assist des VW Tourans. Dabei wird bei der Parklückenidentifikation ein vertikaler 180° Laserscanner eingesetzt. Aus den ermittelten Abständen zum seitlichen Hindernis soll der in dieser Masterarbeit entwickelte Einparkassistent, das SCV in die Parklücke führen. Im Folgenden wird das Szenario für den entwickelten Einparkassistenten dargestellt.

2.2. Szenarien für den Einparkvorgang

Der folgende Abschnitt zeigt die Ausführung des Einparkassistenten. Es wird ein Szenario vorgestellt, in dem der Einparkassistent die Tiefe und Länge der Parklücke mit dem Laserscanner, und dem gefahrenen Weg ausmisst und anschließend einen Einparkvorgang mittels Odometrie oder PD-Regler einleitet. Für den Aufbau des Szenarios steht ein Raum von 30m^2 des Departments Informatik der HAW Hamburg zur Verfügung. Die Parklücke wird durch die Wand und zwei Hindernisse begrenzt. Bei der Parklückensuche soll das SCV parallel zur Wand fahren. Mit der Wand wird die seitliche Parklückenbegrenzung simuliert, während die Kisten stehende Fahrzeuge repräsentieren. Der Einparkassistent lässt sich in zwei Phasen aufteilen. In der ersten Phase wird nach einer Parklücke gesucht, während in der zweiten Phase der Einparkassistent die Lenkung über das SCV beim Einparkvorgang übernimmt. Solange die Parklückensuche stattfindet, behält der Fahrer die Kontrolle über das SCV.

Die Entwicklung des Einparkvorgangs wird in zwei Schritten durchgeführt. Beim Einparkvorgang werden die Lenkwinkel im Einparkassistenten durch einen Regler vorgegeben. Zur Ermittlung der

Position des SCVs, während des Einparkvorgangs, wird zunächst ein Ansatz ohne Zuführung der Abstandswerte angewendet. Im ersten Entwurf des Einparkassistenten aktualisiert alle T_{CAN} die Odometrie mit den Eingangsgrößen Lenkwinkel und Geschwindigkeit die Position des Fahrzeugs. Im zweiten Ansatz fließen die Abstandswerte zur seitlichen Parklückenbegrenzung mit ein und die Odometrie wird mit dem Ist-Lenkwinkel des SCV gekoppelt, die alle T_{CAN} empfangen werden. Während des Einparkvorgangs kann der Fahrer die Fortführung oder Unterbrechung mit dem Fußpedal bestimmen.

2.2.1. Parklückensuche mit den 180° Laserscanner und der Messung des zurückgelegten Weges

Die Parklückenlänge wird mit dem zurückgelegten Weg gemessen, die von den μ-Controllern im Fahrzeug gesendet werden. Zur Ausmessung der Parklückentiefe wird ein seitlich angebrachter, vertikaler, 180° Laserscanner verwendet, in Abb. 7 mit einem S dargestellt. Die Parklücke befindet sich im Bild zwischen zwei stehenden Hindernissen. Eine für den Einparkvorgang verwendbare Parklücke misst 2,4m in der Länge und 1,2m in der Tiefe. Während der parallelen Fahrt zur Wand mit einer vom Fahrer vorgegebenen Geschwindigkeit v, misst der Laserscanner mit einem Winkel von 90° zum Boden, zyklisch den Abstand zwischen SCV und der seitlichen Begrenzung. Erfasst der Laserscanner innerhalb der Strecke von 2,4m einen Abstand der kleiner als 1,2m ist, wird die Parklücke verworfen und die Suche startet erneut, indem der zurückgelegte Weg wieder auf 0 gesetzt wird. Die folgende Sequenz soll den Vorgang der Parklückensuche verdeutlichen. (vgl. Abb. 7)

- In Position 1 ist das SCV auf derselben Höhe wie das hintere Fahrzeug und der gemessene Abstand A_1 ist für den Einparkvorgang nicht ausreichend. Aus diesem Grund wird der Verfahrweg B nicht gemessen. Der Verfahrweg wird erst ausgewertet, wenn der Abstand zur Wand mindestens 1,2 m beträgt.
- In der 2. Position ist der Abstand A_2 größer als 1,2 m und der Verfahrweg B_2 gemessen. Sobald der Verfahrweg den Schwellenwert 2,4 m überschreitet, ist die Parklückensuche abgeschlossen und der kürzeste Abstand wird als Parklückentiefe gepeichert. Verringert sich der Abstand während der Parklückensuche unter 1,2m wird der Verfahrweg verworfen und erst wieder ausgewertet, wenn sich der Abstand wieder auf 1,2m vergrößert hat.
- In Position 3 hat das SCV den Weg B_3 hinterlegt und misst einen Abstand A_3, der für eine Parklücke nicht ausreicht. Zwar ist der Abstand A_3 unter 1,2m, jedoch wurde schon eine Parklücke detektiert, die für einen Einparkvorgang verwendet werden kann. Jede Verletzung der Abstandsregel wird jetzt ignoriert.

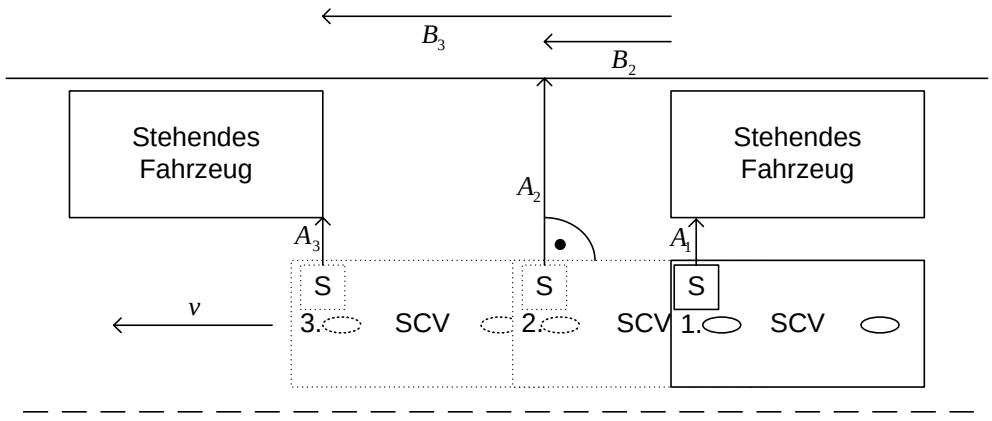

Abb. 7: Szenario zur Erkennung einer Parklücke. Tiefenmessung Ax mit einem Laserscanner S und parallele Verfahrwegerfassung Bx zwischen den Parklückenbegrenzungen.

2.2.2. Einparkvorgang nach berechneter Soll-Wert Trajektorie, ohne Abstandserfassung

Sobald eine Parklücke identifiziert wurde, kann der Einparkvorgang eingeleitet werden. Da der Laserscanner für den Einparkvorgang im ersten Ansatz deaktiviert wird, muss für die Positionsermittlung ein Fahrzeugbeobachter eingeführt werden, welche die lasergestützte Abstandserfassung ersetzt, dies wird mit der Odometrie berechnet.

Das Konzept der virtuellen Deichsel [2] ist zum Ausweichen für Hindernisse entwickelt worden. Sie benötigt für die Vorgabe des Lenkwinkels, die Eigengeschwindigkeit und Abstandswerte relativ zum Hindernis. Es handelt sich hier um einen impliziten Regler, der die Abweichung der Distanz zwischen virtueller Deichsel und Fahrzeug, durch die Vorgabe der Stellgröße Lenkwinkel regelt. Die Deichsel kann aber auch, durch das Platzieren der virtuellen Deichsel in die Parklücke, für den Einparkvorgang verwendet werden. Die virtuelle Deichsel nimmt die Position 1. in Abb. 8 ein.

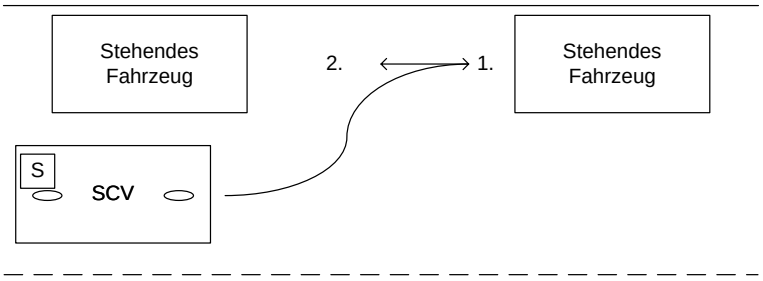

Abb. 8: Szenario des Einparkvorgangs. Rückwärtsfahrt mit anschließender Ausrichtung mittig in der Parklücke.

In der Implementierung des Szenarios werden die Tiefe und Länge der Parklücke als virtuelle Grenzen für die Parklücke aufgestellt und die Koordinaten für das Vorder- und Hinterrad mittels Odometrie berechnet. Aus den X- und Y-Koordinaten der Räder und dem Achswinkel lässt sich bestimmen, wann das SCV die Positionen 1 und 2 erreicht.

Der Einparkassistent führt folgende Schritte während des Einparkvorgangs aus:

- Aus der Tiefe und Länge der Parklücke wird vor der Fahrt der gesamte Einparkvorgang mit einer Abtastrate von T_{CAN}, die Positionen der Räder und die Vorgabe des Lenkwinkels mittels der Odometrie und der virtuellen Deichsel berechnet.

- Zunächst fährt das SCV rückwärts in einer longitudinalen Bewegung in die Parklücke zur Position 1, um sich dann mittig in der Position 2 in einer geraden Fahrt auszurichten.
- Mit dem Abschluss des Parkvorgangs in der Position 2 befindet sich das Fahrzeug mittig in der Parklücke.
- Während die Lenkwinkel vom Einparkassistenten vorgegeben werden, wird die Geschwindigkeit vom Fahrer durch das Betätigen des Gaspedals bestimmt. Sobald das Pedal den Schwellwert von 0,005 m/s überschreitet, fährt das SCV mit einer konstanten Geschwindigkeit von 0,5 m/s. Wird der Schwellwert unterschritten, unterbricht das SCV den Einparkvorgang und führt ihn erst wieder fort, wenn der Schwellwert überschritten wird.
- Um den Parkvorgang abzubrechen, muss der Fahrer den Knopf für den Abbruch des Einparkvorgangs auf dem Lenkrad drücken. Daraufhin erhält der Fahrer wieder die Kontrolle über das SCV.

2.2.3. Trajektorie-Berechnung mit der virtuellen Deichsel

Aus der Tiefe und Länge der Parklücke werden die Grenzen für einen virtuellen Raum in der Ebene aufgestellt und die Deichsel auf die Position 1 gesetzt. Mit dem Abstand zwischen Deichsel und Hinterrad des SCVs, lässt sich ein Lenkwinkel berechnen. Bewegt sich das SCV in einer longitudinalen Bewegung auf die Deichsel zu, verringert sich der relative Abstand und der einzuschlagende Lenkwinkel wird kleiner. So entsteht bei der zyklischen Neuberechnung des Lenkwinkels ein Satz von Vorgaben, die, wenn sie ausgeführt werden, eine Trajektorie ergibt. (vgl. Abb. 9)

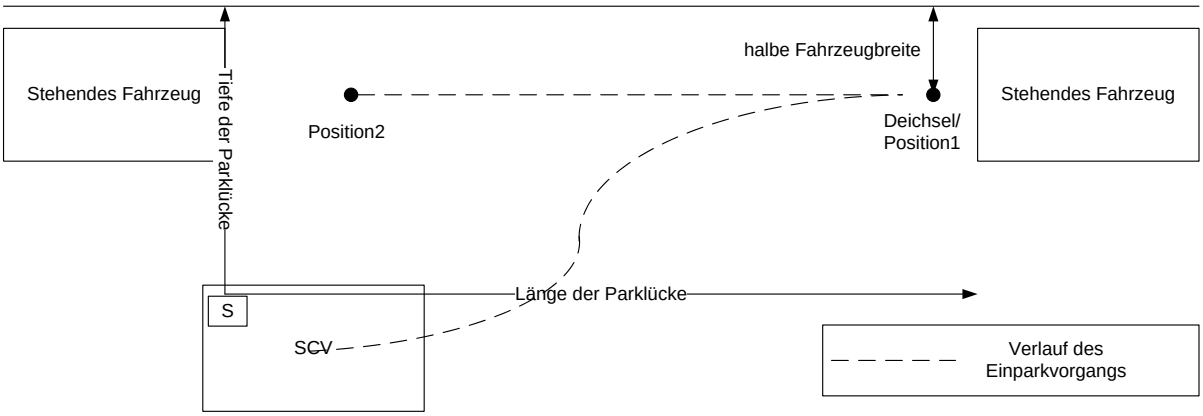

Abb. 9: Die virtuelle Deichsel wird auf die Position 1 gesetzt. Aus dem Abstand zum Vorderrad kann ein Lenkwinkel berechnet werden.

Der Nullpunkt des kartesischen Graphen befindet sich in der Position, an der die Parklückensuche abgeschlossen wurde. Die Tiefe der Parklücke wird als Y-Achse und die Länge als X-Achse bezeichnet. Mit dem Abstand zwischen Vorderrad und Deichsel wird der Lenkwinkel berechnet, der einzuschlagen ist, um die Position der Deichsel zu erreichen. Vor dem Start des Einparkvorgangs, wird zunächst die gesamte Einparkstrecke in Abständen von T_{CAN} mit der Odometrie und der virtuellen Deichsel berechnet. Nach dem Start des Einparkvorgangs, werden die gespeicherten Lenkwinkel in Abhängigkeit der Zeit an das Fahrzeug geschickt.

In Abb. 10 wird die Position der Deichsel mit y_{dl}, x_{dl} und die des Vorderrades mit x_p, y_p beschrieben.

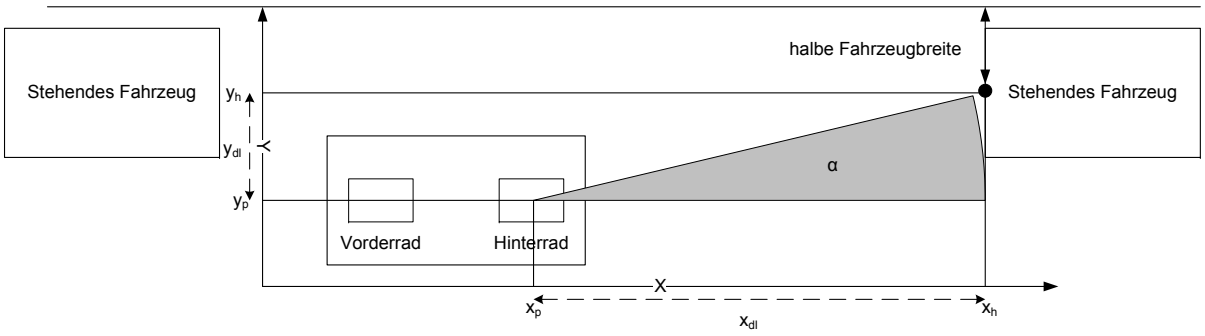

Abb. 10: Berechnung des Lenkwinkels α aus dem Abstand zwischen Hinterrad und Deichsel

Der einzuschlagende Lenkwinkel α berechnet sich aus dem Arkustangens der Differenz der Koordinaten vom Vorderrad und Deichsel.

$$\alpha = \arctan(\frac{y_{dl} - y_p}{x_{dl} - x_p}) \tag{1}$$

In einer Dymolasimulation wird das Verhalten der Deichsel untersucht. Es werden mehrere Szenarien aufgestellt, in denen das SCV mit Abständen von 1,2 bis 2,4m zur seitlichen Begrenzung den Einparkvorgang startet. Damit das SCV die Einparkpositionen in der vorgegebenen Länge Y erreichen kann, fließt ein Verstärkungsfaktor im Zähler der Formel 1 mit dem Wert 4 ein. Aus der Dimolasimulation ergibt sich die S-Kurve des SCVs für den Einparkvorgang. Auf die Odometrie des Fahrzeugmodells und die Messtechnische Analyse, sowie der Simulation der virtuellen Deichsel, wird in den Kapitel 3, 3.3, 7 näher eingegangen.

2.2.4. Einparkvorgang mit Abstandsregelung

Im zweiten Ansatz wird die Vorgabe des Lenkwinkels während des Einparkvorgangs mit den Abstandswerten vom Laserscanner zur seitlichen Begrenzung online berechnet. Durch die Verwendung des vertikalen 180° Laserscanners ist die Erfassung der seitlichen Begrenzung in der Horizontalen auf einen Punkt begrenzt, mit dem Ergebnis, dass beim Einlenken nicht mehr der Abstand zur Wand, sondern der Abstand zum vorderen Fahrzeug erfasst wird, wenn das SCV rückwärts in die Parklücke fährt.

Mit dem kinematischen Modell und den Abstandswerten vom Laserscanner kann eine Aussage gemacht werden, ob es sich bei dem gescannten Objekt, um die seitliche Begrenzung oder ein anderes Hindernis handelt. Hierzu wird der zurückgelegte Weg in Parklückenlängsrichtung mit dem Abstandsdreieck verglichen. (vgl. Abb. 11) Das Abstandsdreieck besteht aus dem Achswinkel des Fahrzeugs und dem ermittelten Abstandswert. Die Gegenkathete beschreibt die Tiefe des gemessenen Objekts in der Parklücke und die Ankathete den gefahrenen Weg in Längsrichtung.

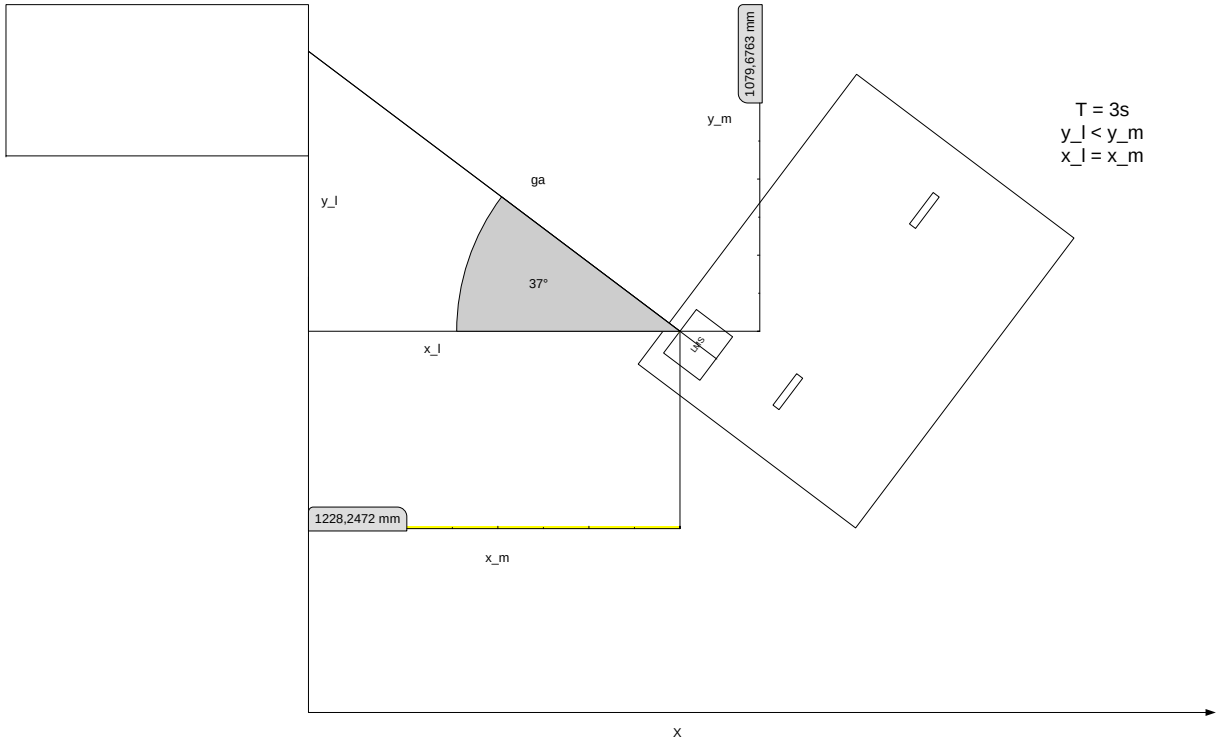

Abb. 11: Odometrie Koordinaten: x_m = Längstrichtung ,y_m = Longitudinale Richtung, θ = Achswinkel vom SCV, Abstandsdreieck berechnet, aus dem Abstand zur Wand: ga = Abstand zur Wand/Fahrzeug, x_l = berechnete Längstrichtung, y_l = berechnete Longitudinalen Richtung, Koordinaten der virtuellen Deichsel: x_h, y_h

2.3. Bedienung und Repräsentation der Einparkphasen auf dem Leitstand

Der Leitstand besteht aus einem PC mit Lenkrad und Pedalen und einer Bedienoberfläche. Der Start und Abbruch des Assistenten wird durch zwei Knöpfe auf dem Lenkrad ausgelöst. Auf der Bedienoberfläche des Einparkassistenten können dabei die tatsächlichen Zustände in den zwei Phasen des Einparkassistenten abgelesen werden, zusätzlich werden die Ist- und Soll-Werte für die Geschwindigkeit und des Lenkwinkels abgebildet. Für genauere Informationen bezüglich des Leitstandes, wird hier auf das Kapitel 5 verwiesen. Das Zusammenspiel der Softwaremodule bei der Parklückensuche, sowie die Fahrt in die Parklücke, werden in Kapitel 5 mit Sequenzdiagrammen dargestellt.

Um den Einparkassistenten zu starten, muss der Benutzer den Parkwunsch dem SCV durch das Drücken des Startknopfs auf dem Lenkrad mitteilen. Als Antwort leuchtet auf dem Leitstand das LED Licht zur Parklückensuche auf. (vgl. Abb. 12) Damit wird die erste Phase der Parklückensuche eingeleitet.

Abb. 12: Die LEDs des Leitstandes

Wenn eine Parklücke gefunden wird, teilt der SCV dem Benutzer mit, dass der Parkvorgang eingeleitet werden kann, dies wird durch das Aufleuchten der LED Parklücke angezeigt. Sobald die Parklücke über das Lenkrad mit demselben Knopf, wie dem Start des Einparkassistenten angenommen wird, wechselt der SCV in den Einparkvorgang des Einparkassistenten. Wird sie abgelehnt, beginnt die Parklückensuche wieder von vorne. Wird die Parklücke angenommen starten die Berechnungen für die Fahrt in die Parklücke.

Sobald der Benutzer die vom SCV angebotene Parklücke annimmt, leuchtet die LED Einparkvorgang auf und der Einparkassistent übernimmt die Lenkung. Durch das Loslassen des Pedals wird der Einparkvorgang unterbrochen. Abgebrochen wird der Einparkvorgang durch das Drücken des Abbruch-Knopfes. In diesem Fall wird der Einparkassistent deaktiviert und der Fahrer erhält wieder die Kontrolle über Lenkrad und Gaspedal.

3. Odometrie mit einem kinematischen Modell des SCVs

In diesem Kapitel wird die Odometrie des nichtlinearen kinematischen Einspurfahrzeugmodells vorgestellt. Bei der Odometrie handelt es sich um einen Beobachter, der die Eingangsgrößen des Ist-Lenkwinkels und Geschwindigkeit des Vorderrades zur Positionsermittlung zugeführt bekommt und der in beiden Ansätzen mit und ohne lasergestützte Abstandserfassung angewendet wird:

- Positionsermittlung ohne Abstandserfassung: Ermittlung der Positionen der einzelnen Räder, während der Fahrt in die Parklücke, ohne Abstandserfassung vom Laserscanner. Das SCV fährt in einem virtuellen Raum mit 2,4m Länge und die vom Laserscanner ermittelten Tiefe.
- Positionsermittlung mit Abstandserfassung: Ermittlung der Positionen der einzelnen Räder, während der Fahrt in die Parklücke mit Abstandserfassung. Zusammen mit dem Abstandswerten wird festgestellt, ob die seitliche Parklückenbegrenzung erfasst wird. Die Y-Koordinate des Vorderrads wird durch die Abstandserfassung ersetzt.

Anhand des Radabstandes SCVs, der eingestellten Geschwindigkeit und des Lenkwinkels werden Rückschlüsse über die Bewegung im Raum geschätzt. Es wird zyklisch zu jeden Zeitpunkt des Einparkvorgangs die Position des Achswinkels, des Vorder- und Hinterrades neu berechnet. Die ermittelten Größen werden später in Kapitel 3.3 dem Regler, zwecks Abstandsregelung zugeführt. Andere wissenschaftliche Veröffentlichungen [4], [21], [12], verwenden für automatisches Einparken für die Berechnung der Trajektorie ebenfalls eine Odometrie.

Zur Anwendung der Odometrie im zeit diskreten Bereich wird das Runge-Kutta-Verfahren zur Lösung des Grenzwert Problems erläutert. Mit Simulationen werden die Lenkwinkel Vorgaben beider Regler untersucht.

Der Übergang von virtueller zur lasergestützten Positionsermittlung wird anhand der Odometrie und den Lasermesswerten festgestellt. Da während der Fahrt in die Parklücke das vordere Hindernis statt der seitlichen Parklückenbegrenzung erfasst werden kann, muss eine Auswertungslogik eingeführt werden, die anhand der Odometrie und den Lasermesswerten feststellt, ob die seitliche Parklückenbegrenzung oder ein Hindernis erfasst wird. Ein Abschnitt stellt anhand von Zeichnungen die Übergangskriterien auf.

3.1. Berechnung des Achswinkels

In Bezug zum Koordinatensystem beschreibt der Achswinkel θ die Lage der Fahrzeuglängsachse. Die Veränderung des Achswinkels über die Zeit wird über die Eingangsgrößen der konstanten Geschwindigkeit v_m des Vorderrades von 0,5 m/s und des Lenkwinkels α bestimmt. Der zurückgelegte Weg liegt auf der Kreisbahn s_m eines Einheitskreises. Es ergibt sich der Schnittpunkt S des Einheitskreises, um das sich das SCV bei einem eingeschlagenen Lenkwinkel α dreht. Die Radien des Einheitskreises werden im folgenden Bild 13 als r_m für das Vorder- und r_p für das Hinterrad bezeichnet. Mit v_m und r_m resultiert eine Winkelgeschwindigkeit ω, die für beide Räder gilt und ist damit äquivalent zur Änderung des Achswinkels θ über die Zeit. (vgl. Abb. 13)

Bei einer Änderung von α ändert sich die Winkelgeschwindigkeit θ, der Radius r_m des Einheitskreises und damit der Bezugspunkt S, bei Annahme einer konstanten Geschwindigkeit v_m. Unter den

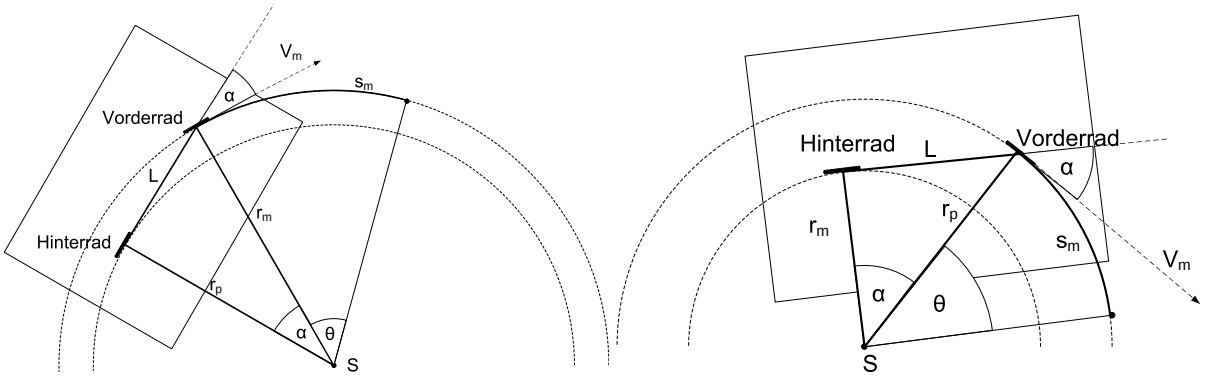

Abb. 13: Nichtlineares kinematisches Einspurfahrzeugmodell im Einheitskreis. Achswinkel θ, Lenkwinkel α und Geschwindigkeit des Vorderrades v_m.

geometrischen Annahmen aus der Abb. 13 können folgende Gleichungen aufgestellt werden. Für die Winkelgeschwindigkeit ω im Bogenmaß gilt:

$$\omega = \frac{v_p}{r_p} = \frac{v_m}{r_m} \Rightarrow \omega = \frac{v_m}{r_m} \cdot \frac{360^\circ}{2\pi} \tag{2}$$

Aus dem Radabstand L und des Lenkwinkels α, wird der unbekannte Radius r_m substituiert.

$$r_m = \frac{L}{\sin(\alpha)} \Rightarrow \omega = \frac{v_m}{L} \cdot \sin(\alpha) \cdot \frac{360^\circ}{2\pi} \tag{3}$$

Die Änderung des Achswinkels über die Zeit $\frac{d\theta}{dt}$ ist äquivalent zur Winkelgeschwindigkeit ω, somit gilt:

$$\frac{d\theta}{dt} = \omega \Rightarrow \frac{d\theta}{dt} = \frac{v_m}{L} \cdot \sin(\alpha) \cdot \frac{360^\circ}{2\pi} \tag{4}$$

In der Gleichung 5 ist die einzig nicht konstante Variable, der vorgegebene Lenkwinkel α.

$$\theta(t) = \frac{v_m \cdot 360^\circ}{L \cdot 2\pi} \int_0^T \sin(\alpha) dt \tag{5}$$

3.2. Berechnung der Koordinaten für das Vorder- und Hinterrad

Zusammen mit Achswinkel, Lenkwinkel und Geschwindigkeit des Vorderrades lassen sich die Veränderung der Positionskoordinaten des Hinter- und Vorderrades, sowie die Geschwindigkeit des Hinterrades bestimmen. Bei der Ermittlung des Achswinkels, werden die Koordinaten für das Vorder- und Hinterrad und die Geschwindigkeit des Hinterrades aus den Eingangsgrößen des Vorderrades abgeleitet.

Um die Bewegung des starren Hinterrads zu berechnen, muss die Geschwindigkeit des Hinterrades v_p bekannt sein. Diese ergibt sich aus der Geschwindigkeit des Vorderrades v_m und dem Lenkwinkel α. (vgl. Abb. fig:vp)

v_p wird mit dem Geschwindigkeitsdreieck bestimmt. Sie besteht aus dem Lenkwinkel α und der Geschwindigkeit des Vorderrades v_m, vgl. Abb. 14:

$$v_p(t) = v_m(t) \cdot \cos(\alpha) \tag{6}$$

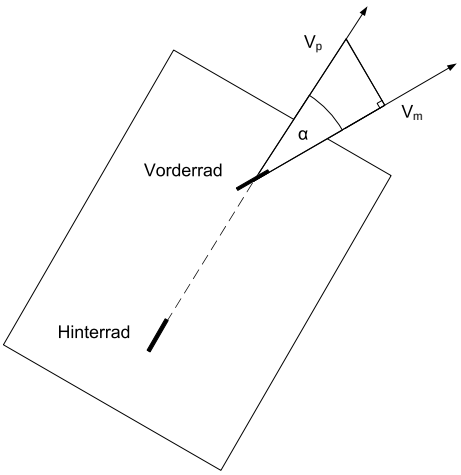

Abb. 14: Berechnung der Längsgeschwindigkeit v_p mit dem Lenkwinkel α und der Geschwindigkeit des Vorderrads v_m.

Mit der Geschwindigkeit des Vorderrads v_p aus Gl. 6 und des Achswinkels θ aus Gl. 5 können die X- und Y-Koordinanten des Hinterrads berechnet werden. (vgl. Abb. 15)

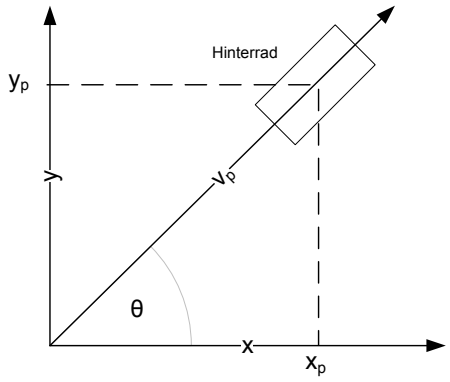

Abb. 15: Die Veränderung der X- und Y-Koordinaten des Hinterrades in Abhängigkeit von der Geschwindigkeit v_p und des Achswinkels θ.

Die X- und Y-Koordinaten des Hinterrades x_p und y_p ergeben sich aus den Komponenten des Geschwindigkeitsvektors v_m des Vorderrades:

$$\frac{dx_p}{dt} = v_p(t) \cdot \cos(\theta) \Rightarrow x_p(t) = \int_0^T v_p \cdot \ cos(\theta) dt \tag{7}$$

$$\frac{dy_p}{dt} = v_p(t) \cdot \sin(\theta) \Rightarrow y_p(t) = \int_0^T v_p \cdot \sin(\theta) dt \tag{8}$$

Der Lenkwinkel α beschreibt den Winkel des Vorderrads relativ zum Achswinkel θ. Bei der Positionsermittlung des Vorderrads wird der eingeschlagene Winkel α vom tatsächlichen Achswinkel θ abgezogen. (vgl. Abb. 16) Aus dem resultierenden Winkel $\theta - \alpha$, kann über die Geschwindigkeit des Vorderrades v_m, die neue Position des Vorderrades in X- und Y-Richtung aus dem Dreieck bestimmt werden.

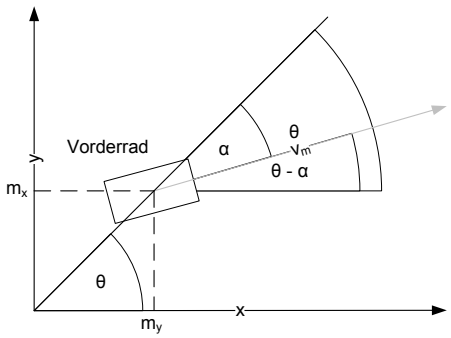

Abb. 16: Die Veränderung der X- und Y-Koordinaten des Vorderrades in Abhängigkeit der Geschwindigkeit v_m und des Achswinkels θ.

Demnach gelten für die X- und Y-Koordinaten des Vorderrades nach Abzug des Lenkwinkels α vom Achswinkel θ mit der Geschwindigkeit v_m:

$$\frac{dx_m}{dt} = v_m \cdot \cos(\theta - \alpha) \Rightarrow x_m(t) = v_m \int_0^T \cos(\theta - \alpha) dt \tag{9}$$

$$\frac{dy_m}{dt} = v_m(t) \cdot \sin(\theta - \alpha) \Rightarrow y_m(t) = v_m \int_0^T \sin(\theta - \alpha) dt \tag{10}$$

Aus den über Odometrie aufgestellten Gleichungen des SCVs, kann die Bewegung im virtuellen Raum verfolgt werden. Mit den Eingangsgrößen Lenkwinkel α und Geschwindigkeit des Vorderrades v_m, wird die Position des Vorderrads(x_m,y_m), Hinterrads (x_p,y_p) und der Achswinkel θ bestimmt.(vgl. Abb. 17) Im nächsten Abschnitt wird die Einstellung des Lenkwinkels α vorgestellt. Die hier aufgestellten Gleichungen für das Fahrzeugmodell werden in der Dymola-Simulation verwendet. Zusammengefasst gilt für das gesamte SCV Fahrzeugmodell:

$$\theta(t) = \frac{v_m \cdot 360^\circ}{L \cdot 2\pi} \int_0^T \sin(\alpha) dt \tag{11}$$

$$v_p(t) = v_m(t) \cdot \cos(\alpha) \tag{12}$$

$$x_m(t) = v_m \int_0^T \cos(\theta - \alpha) dt \tag{13}$$

$$y_m(t) = v_m \int_0^T \sin(\theta - \alpha) dt \tag{14}$$

$$x_p(t) = \int_0^T v_p \cdot \ cos(\theta) dt \tag{15}$$

$$y_p(t) = \int_0^T v_p \cdot \sin(\theta) dt \tag{16}$$

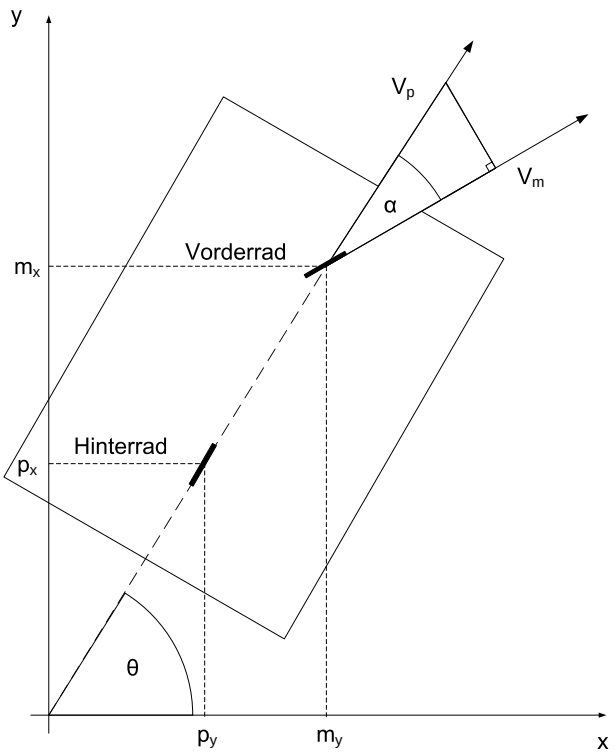

Abb. 17: Das gesamte nichtlineare kinematische Fahrzeugmodell für das SCV. Mit den X- und Y-Koordinaten für das Vorderrad (x_m,y_m), Hinterrad (x_p,y_p), dem Achswinkel θ, dem Lenkwinkel α und die Geschwindigkeitsvektoren für beide Räder (v_p,v_m).

3.3. Regelung der Lenkwinkel Soll-Vorgabe durch einen Abstandsregler

In diesem Abschnitt werden beide Regler zur Lenkwinkelregelung vorgestellt. Aus der Entfernung zwischen SCV und Bezugspunkt lässt sich der Lenkwinkel α bestimmen, der einzuschlagen ist, um das SCV auf die longitudinale Tiefe der Parklücke zu bewegen. Der Sprung oder die zu regelnde Größe bezieht sich auf den relativen Abstand von Bezugspunkt und Vorderrad. Bei der zeitdiskreten Berechnung des einzuschlagenden Lenkwinkels fließen die veränderten Koordinaten aus der Odometrie mit ein, so dass sich der Lenkwinkel ständig aktualisiert.

3.3.1. Abstandsregelung mit der virtuellen Deichsel

Auf der Basis des nichtlinearen kinematischen Einspurfahrzeugmodells, wird mit der virtuellen Deichsel in zeitdiskreten Abständen der Lenkwinkel α berechnet. Aus der Entfernung zwischen Deichsel und Hinterrad [2] in X- und Y-Richtung wird über dem Arkustangens der Winkel berechnet, der einzuschlagen ist, um das SCV auf die longitudinalen Tiefe der Parklücke zu bewegen. (vgl. Abb. 18)

Zur Berechnung des Lenkwinkels α gilt:

$$\alpha = \arctan \frac{y_{dl}}{x_{dl}} \tag{17}$$

und:

$$-\pi \leq \alpha \leq \pi \tag{18}$$

Bei einem positiven α, lenkt der Motor das Vorderrad nach rechts, bei einem negativen α, steuert das SCV nach links.

Die Distanz zwischen Deichsel und Hinterrad:

$$y_{dl} = y_h - Halbefahrzeugbreite - y_p \quad (19)$$

$$x_{dl} = x_h - x_p \quad (20)$$

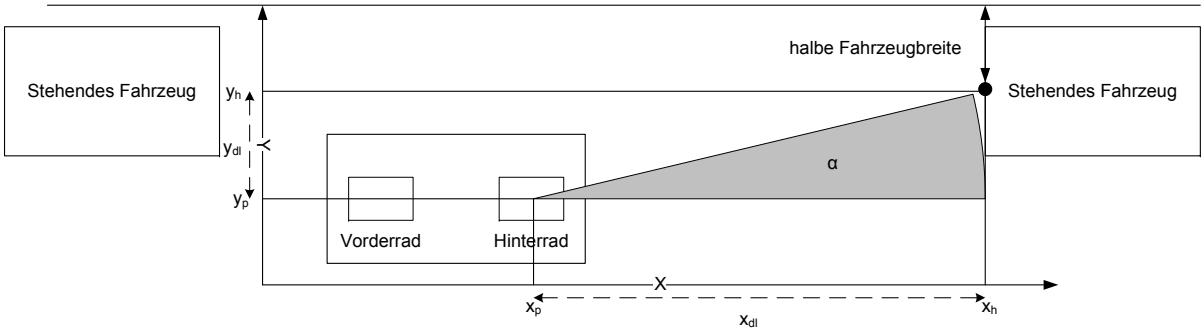

Abb. 18: Abstandsregelung mit der virtuellen Deichsel. Ausgangsgröße: Lenkwinkel α; Eingangsgrößen Koordinaten des Hinterrades: x_p, y_p, Deichsel: x_h, y_h, Distanz: x_{dl}, y_{dl}

3.3.2. Abstandsregelung mit einem PD-Regler

Als Alternative zur virtuellen Deichsel wird ein digitaler PD-Regler [23] angewendet. Der Regler wurde für einen Ausweichassistenten entworfen und musste für den Einparkassistenten parametrisiert werden. Die Übertragungsfunktion sowie die Führungsübertragungsfunktion wurden bereits aufgestellt und können aufgrund der identischen Odometrie übernommen werden.

Der Regler regelt den Abstand der Y-Koordinate des Hinterrads. Aus der Führungsgröße y_h und dem Abstand y_m ergibt sich die Regeldifferenz e. Diese Differenz wird durch die Vorgabe des Soll-Lenkwinkels α vom Regler geregelt und der Odometrie zugeführt. Die Rückwirkung durch die Messeinrichtung des Laserscanners ist abhängig von der Auswertungslogik. Wird ein stehendes Hindernis erfasst, können die Abstandswerte nicht verwendet, werden und die Rückführgröße besteht aus der Koordinate des Hinterrads y_m. (vgl. Abb. 19) Die Regeldifferenz besteht aus dem

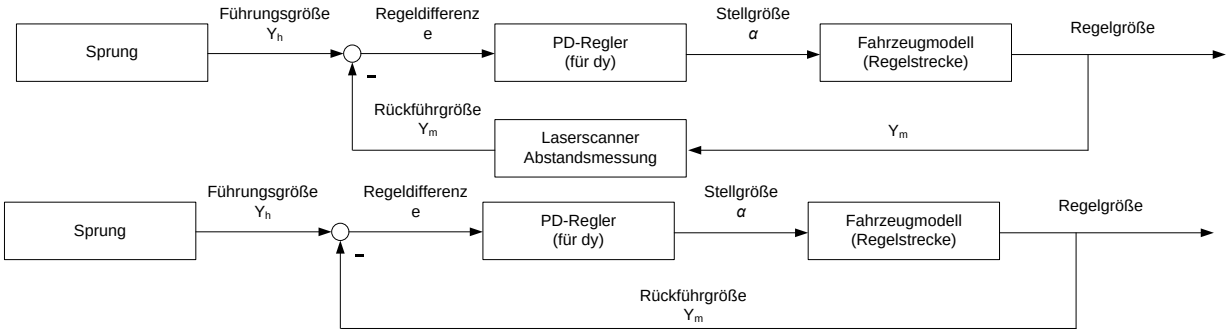

Abb. 19: Regelkreis mit und ohne Abstandserfassung

absoluten Y-Abstand des Hinterrads zur seitlichen Parklückenbegrenzung, abzüglich der halben Fahrzeugbreite von 0,45m.

$$e = y_h - Halbefahrzeugbreite - y_p \quad (21)$$

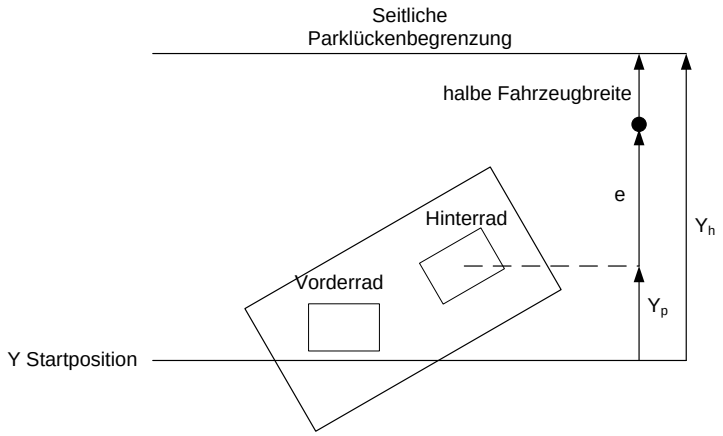

Abb. 20: Regeldifferenz e, Y-Koordinate Hinterrad y_p, Abstand Startposition und seitliche Parklückenbegrenzung y_h

Die Regelparameter musste modifiziert werden. Zwar weist der digitale PD-Regler ein ausreichend stationäres Verhalten aus, der D-Anteil musste jedoch angepasst werden. Ohne Anpassung des D-Anteils waren die Soll-Vorgaben zu niedrig, um das SCV in der geforderten Länge von 2,4 m einzuparken.

3.4. C-Modellierung des nichtlinearen kinematischen Einspurfahrzeugmodells

Mit dem Aufstellen des kinematischen Modells im vorherigen Abschnitt, wird in diesem Abschnitt ein Verfahren zur Berechnung der Änderung der Koordinaten im diskreten Bereich vorgestellt. Die aufgestellten Gleichungen in Abschnitt 3 bis 3.2 werden mit dem Runge-Kutta-Verfahren [19] linearisiert, um in zeitlichen Abständen von T_{CAN} die Veränderungen der Koordinaten und Winkeln berechnen zu können.

Mathematisch gesehen handelt es sich um einen Satz von N Gleichungen 1. Ordnung für die Funktion $y_i, i = 1, \cdots, N$, wo die Funktion f_i bekannt ist, in folgender Form:

$$\frac{dy_i(x)}{dx} = f_i(x, y_1, \cdots, y_N), i = 1, \cdots, N \tag{22}$$

Das numerische Lösen dieser Gleichung wird als Grenzwert-Problem bezeichnet. Allgemein können diese für diskrete spezifische Punkte angenähert werden, halten aber zwischen den Punkten nicht stand. Grenzwert-Probleme lassen sich in zwei Kategorien aufteilen:

- Im Anfangswert-Problem sind für alle y_i im Punkt x_s die Startwerte bekannt und es wird nach dem Wert von y_i zu dem Punkt x_f gesucht, oder nach einer diskreten Liste von Punkten, mit festgelegten Abständen.
- Beim Zwei-Punkt Grenzwert Problem werden die Grenzwerte zu mehreren x spezifiziert. Typischerweise werden die Grenzwerte zum Punkt x_s festglegt, und die restlichen zum Zeitpunkt x_f.

In diesem Fall handelt es sich um ein Anfangswertproblem. Vom Startpunkt des Einparkvorgangs sind die X- und Y-Koordinaten für den virtuellen Raum bekannt.

3.4.1. Euler-Verfahren

Das Konzept für jedes Verfahren, das sich mit dem Anfangswertproblem beschäftigt, ist gleich. Es werden die dys und die dxs in Gleichung 22 in endliche Schritte von Δx und Δy umgeformt und die Gleichungen mit Δx multipliziert. Dabei werden die Werte der Funktionen mit der unabhängigen Variable x, mit den Schritt Δx berechnet. Je kleiner diese Schritte gewählt werden, desto genauer werden die Differentialgleichungen angenähert. Als anschauliches Verhalten dient das Euler-Verfahren, welches nicht für praktische Anwendungen empfohlen wird. (vgl. Abb. 21) Jedoch ist das Euler-Verfahren im konzeptionellen Sinne interessant, denn das vorgestellte Runge-Kutta-Verfahren orientiert sich daran. Die Addition des Werts y_n und der Funktion multipliziert mit der Schrittgröße h.

$$y_{n+1} = y_n + hf(x_n, y_n) \tag{23}$$

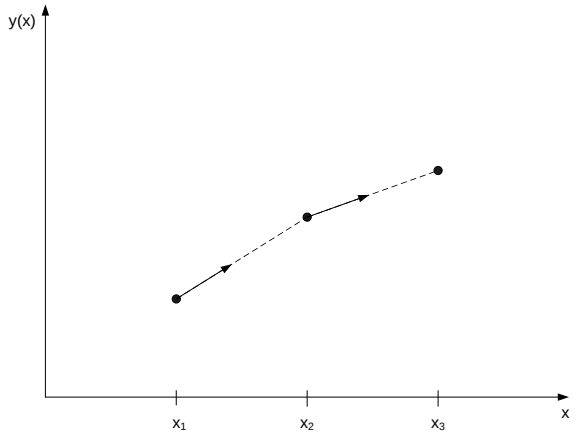

Abb. 21: Euler-Verfahren zur Integration einer DGL 1.Ordnung. Ableitung zum Startpunkt wird extrapoliert, zur Berechnung des nächsten Funktionswerts.

3.4.2. Runge-Kutta-Verfahren

Das Euler-Verfahren ist aus einem Grund für den praktischen Einsatz in der Form nicht zu empfehlen: Im Vergleich zu den anderen Verfahren mit gleicher Schrittlänge ist es ungenau. Das Runge-Kutta-Verfahren verwendet das Euler-Verfahren, um die Genauigkeit zu skalieren. Es verbindet die Werte aus mehreren Euler-Schritten, jeder Schritt fließt in die Berechnung der Funktion f mit ein. Dieses Verfahren wird verwendet, wenn die zu berechende Funktion f simpel und eine moderate Genauigkeit von $\leq 10^-5$ benötigt wird, im Falle der Bewegung im virtuellen Raum ergibt sich eine Abweichung in der Simulation und Implementierung von +/- 2,4 cm bei einer Abtastperiode T_{CAN}. Beim zweistufigen Runge-Kutta-Verfahren wird zunächst ein halber Schritt $\frac{h}{2}$ über das Intervall getätigt (1), um die dabei berechneten Werte für x und y im Mittelpunkt(2), als Berechnungsgrundlage für das ganze Intervall zu verwenden(3). (vgl. Abb. 26) Das Runge-Kutta-Verfahren wird auch als Mittelpunkt-Methode bezeichnet.

$$k_1 = hf(x_n, y_n) \tag{24}$$

$$k_2 = hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1) \tag{25}$$

$$y_{n+1} = y_n + k_2 \tag{26}$$

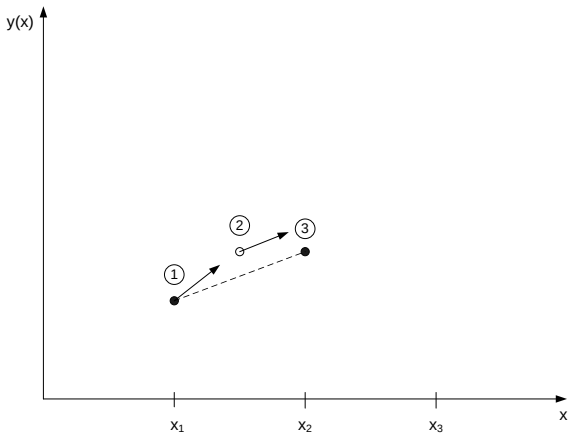

Abb. 22: Runge-Kutta-Verfahren zur Integration einer DGL 1.Ordnung. Ableitung, zum Startpunkt, um einen halben Schritt über das Intervall zu berechnen. Mittelpunkt-Wert als Ableitung für den ganzen Schritt. Gefüllte Kreise 1 und 3:Endwerte, Ungefüllte Kreise 2:Zwischenschritt der Berechnung.

Um eine höhere Genauigkeit zu erzielen, lässt sich die Schrittgröße der Mittelpunkt-Methode beliebig skalieren. Dabei steigt der Rechenaufwand proportional zu der Anzahl der berechneten Schritte. Da die Gleichungen der Odometrie, Kosinus und Sinusfunktionen berechnet werden, muss der Rechner Fließkommazahlen verarbeiten. Es kann eine Verletzung der Zeitbedingung eintreten, so dass ein Echtzeitverhalten nicht mehr garantiert werden kann. Damit ist die Anzahl der Rechenschritte durch die Berechnungsgeschwindigkeit der CPU eingegrenzt.

3.5. Dymola-Simulation des kinematischen Einspurfahrzeugmodells und der virtuellen Deichsel

In diesem Kapitel wird die Simulation des nichtlinearen kinematischen Einspurfahrzeugmodells und der virtuellen Deichsel mit dem Simulationsprogramm Dymola [1] vorgestellt. Es wird die Trajektorie evaluiert, die unter der Verwendung der virtuellen Deichsel und des Fahrzeugmodells berechnet wird und die Fahrt visualisiert. Mit diesem Programm werden physikalische Systeme simuliert. Dazu bietet es eine Reihe von wieder verwendbaren Modelica Bibliotheksfunktionen, oder die direkte Integration der Gleichungen für das Fahrzeugmodell und der virtuellen Deichsel.

Die Analyse der Kurven für die Trajektorie hat ergeben, dass die Gleichung 17 parametrisiert werden muss, damit das SCV innerhalb der Parklückenlänge von 2,4m einparken kann. Die Simulation für die Fahrt in die Parklücke wird in Dymola geplottet. Mit den Werten für die Position und den Winkeln wird festgestellt, ob während der Fahrt eine Kollision gegen die seitliche Begrenzung stattfindet. Die Schrittgröße Δt lässt sich dabei stufenlos parametrisieren und das Intervall T_{CAN} eingestellt.

In der Simulation der Trajektorie wird der Lenkwinkel auf 0° eingestellt, wenn beide Räder denselben Abstand zur Wand besitzen. Die Anwendung der virtuellen Deichsel hat den Nebeneffekt, dass das Vorderrad überschwingt, während es versucht die Y-Position der Deichsel zu erreichen. Würde der Eingriff nicht geschehen, benötigte das SCV mehr als 2,4m, bevor die stationäre Y-Position des Vorderrades erreicht wird. Es werden mehrere Einparkvorgänge mit unterschiedlichen Abständen zur Wand simuliert, mit dem Ergebnis, dass zwar die Y-Position der Deichsel nicht erreicht wird, aber eine konstante Abweichung von 100mm zur Soll-Position zu allen Startbedingungen besteht, sobald das SCV parallel zur seitlichen Begrenzung steht.

3.5.1. Modellierung des SCVs und der virtuellen Deichsel in Dymola

Bei der Modellierung des SCVs kann das System mit Modelica Blockdiagrammen beschrieben werden. Die Transformation der Mathematik in das Blockdiagramm übernimmt Modelica. Eine andere Möglichkeit besteht darin, die Gleichungen direkt als Code zu schreiben. Dabei können die Differentialgleichungen ohne Umformung aufgestellt werden, in der Simulation übernimmt Dymola die Integration. In den Abschnitten 3 und 3.3 sind die Gleichungen für das Fahrzeugmodell und die virtuelle Deichsel aufgestellt, welche in der Dymola-Simulation übernommen wurden. In der Visualisierung und der Analyse wird der Verlauf der einzelnen Parameter, wie Position des Hinterrades, des Vorderrades, Lenkwinkel und Achswinkel über einen Zeitraum von 10s geplottet. Dymola bietet eine Reihe von Linearisierungsmethoden an. Es wird der rkfix2 ausgewählt, da in der Implementierung auch das Runge-Kutta-Verfahren verwendet wird.

3.5.2. Parametrisierung der Verstärkungsfaktoren für die Y-Koordinate der virtuellen Deichsel und des einzuschlagenden Lenkwinkels

Damit der Einparkvorgang unter den Randbedingungen ausgeführt werden kann, müssen die Gleichungen 17 und 20 mit Verstärkungsfaktoren multipliziert werden. Die Randbedingungen sind Folgende:

- 1. Der Parkvorgang muss innerhalb der Parklückenlänge von 2,4m abgeschlossen sein.
- 2. Das SCV muss in der Parklücke parallel zur seitlichen Abgrenzung stehen, mit einem maximalen Sicherheitsabstand von 0,1-0,2 m

Ohne Parametrisierung der Gleichung 17, benötigt die Simulation des Einparkvorgangs mehr als die Parklückenlänge von 2,4m, um die Y-Koordinate der virtuellen Deichsel zu erreichen. Der Verstärkungsfaktor 2 wird mit der Y-Koordinate der virtuellen Deichsel multipliziert. So dass folgende Gleichung gilt:

$$\alpha = \arctan \frac{Verstaerkungsfaktor2 \cdot y_{dl}}{x_{dl}} \tag{27}$$

Der Lenkwinkel wird größer, wenn die Werte für den Verstärkungsfaktor 2 größer als eins sind. Je größer der Lenkwinkel α eingestellt wird, desto größer die Veränderung des Achswinkels θ über die Zeit. Damit wird weniger Weg in X-Richtung benötigt, um die Y-Koordinate der virtuellen Deichsel zur erreichen. Aus Versuchen mit unterschiedlichen Startpositionen der Abstände zur Wand, hat sich für den Verstärkungsfaktor 2 ein Wert von 4 ergeben. Dies ist ein Kompromiss zwischen der Übersteuerung des Vorderrades und des zurückgelegten Weges in Längsrichtung. Wird der Wert zu groß eingestellt, neigt Gleichung 27 dazu zu übersteuern oder auch zu große α einzustellen. Ist dagegen der Wert zu klein eingestellt, wird die erste Randbedingung verletzt.

Die Berechnung der Y-Koordinate der virtuellen Deichsel muss ebenfalls modifiziert werden, siehe Gleichung 20. Damit die 2. Randbedingung erfüllt werden kann, muss der Verstärkungsfaktor 1 mit der halben Fahrzeugbreite multipliziert werden.

$$y_{dl} = y_h - (Verstaerkungsfaktor1 \cdot Halbefahrzeugbreite) - y_p \tag{28}$$

$$x_{dl} = x_h - x_p \tag{29}$$

Für den Verstärkungsfaktor 1 hat sich ein optimaler Wert von 0,5 ergeben. Hier wurden ebenfalls mehrere Versuche mit unterschiedlichen Startpositionen simuliert, um den Sicherheitsabstand von 0,1 - 0,2m zu erreichen.

3.5.3. Abschluss der longitudinalen Fahrt in die Parklücke mit dem Einstellen des Lenkwinkels auf 0°

Bei der Simulation der Trajektorie ist ein Überschwingen des Vorderrades zu beobachten, mit der Folge, dass, wenn kein Eingriff erfolgt, das SCV gegen die seitliche Begrenzung stößt. Die von der virtuellen Deichsel generierten Lenkwinkel versuchen den stationären Soll-Wert mit dem Hinterrad zu regeln, siehe Gl. 29. Da das Hinterrad starr ist, muss das Vorderrad ständig die Abweichung des Soll-Werts in Y-Richtung ausgleichen. Aus diesem Grund ist die Überschwingung des Vorderrades zu erklären.

Als Lösung dieses Problems kann die Parametrisierung der Gleichung 29 angepasst werden. Jedoch ist die Amplitude des Überschwingens für die unterschiedlichen Startbedingungen dynamisch, d.h. für jeden Abstand muss ein anderer Verstärkungsfaktor 2 eingestellt werden. Eine andere Möglichkeit ist das Einstellen des Lenkwinkel α auf 0°, sobald beide Vorderräder dieselbe Position in Y-Richtung aufweisen, d.h. der Achswinkel θ bei annährend 0° liegt. Die Simulationen mit den fixen Verstärkungsfaktoren und variablen Startbedingungen von 1,2m bis 2,1m in Schrittgrößen von 0,05m haben ergeben, dass bei allen Versuchen im Punkt θ annährend 0°, die Differenz beider Räder zum Soll-Wert in Y-Richtung 0,1m beträgt. (vgl. Abb. 23) Aus diesem Grund wird diese Möglichkeit als das Abschlusskriterium der longitudinalen Fahrt in die Parklücke genommen.

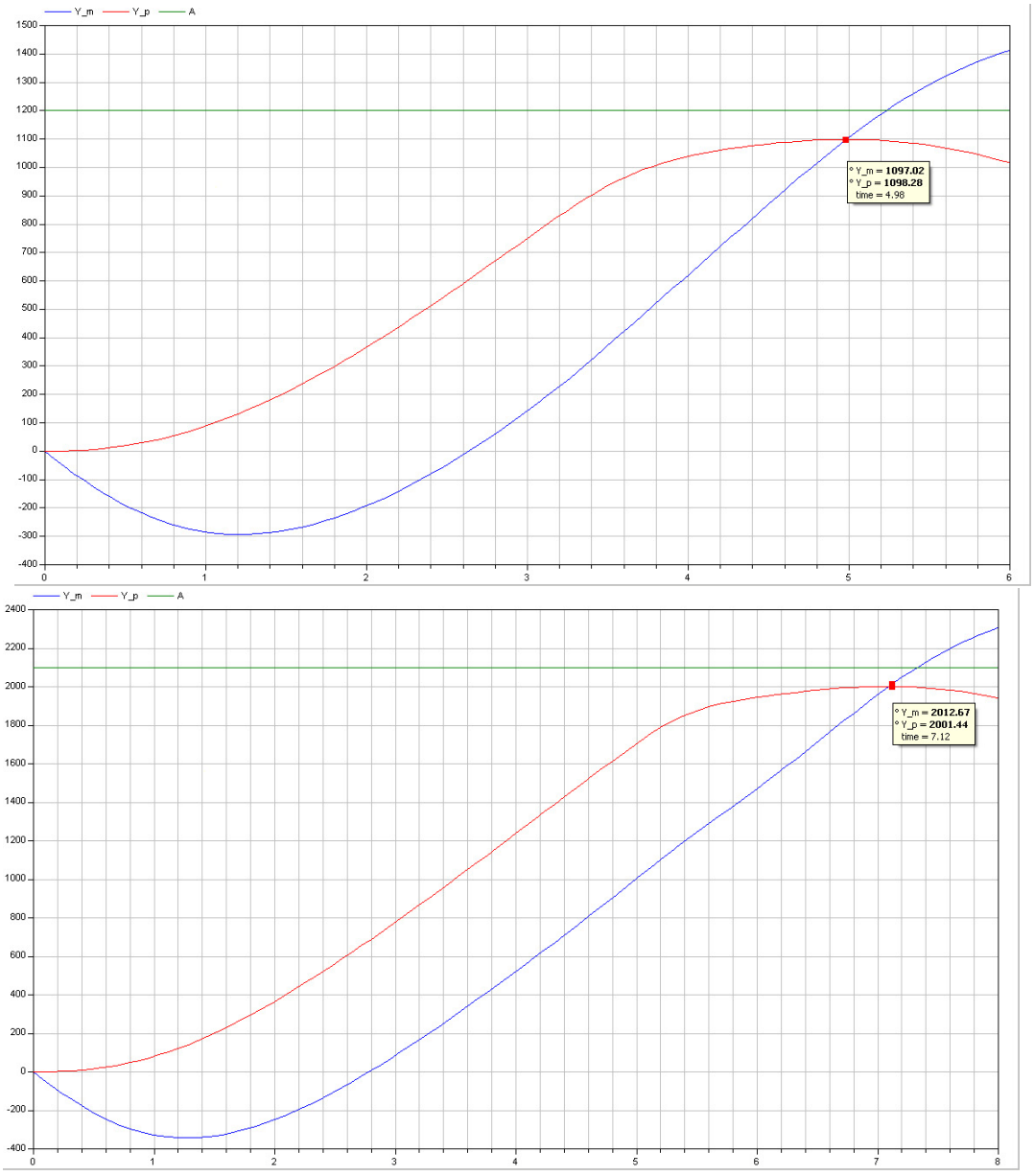

Abb. 23: longitudinalen Fahrt in die Parklücke. Verlauf Y-Koordinaten Hinter- und Vorderrad(Y_p,Y_m) mit Startabstand: 1,2m und 2,1m(A)

3.6. Übergang von virtueller zur lasergestützten Positionsermittlung

Der Übergang zwischen virtueller und lasergestützter Berechnung des kinematischen Modells und der Trajektorie lässt sich mit einem Vergleich zwischen Abstandsdreieck und der Position des SCVs in Längsrichtung bestimmen.

Beim Einparkvorgang in die Parklücke dreht sich das SCV um die eigene Achse. Mit dem Ergebnis, dass zunächst die Abstände zum vorderen Hindernis gemessen wird. Im späteren Verlauf, wenn sich das SCV zur seitlichen Parklückenbegrenzung ausrichtet, wird die seitliche Parklückenbegrenzung mit dem Laserscanner erfasst. (vgl. Abb. 24)

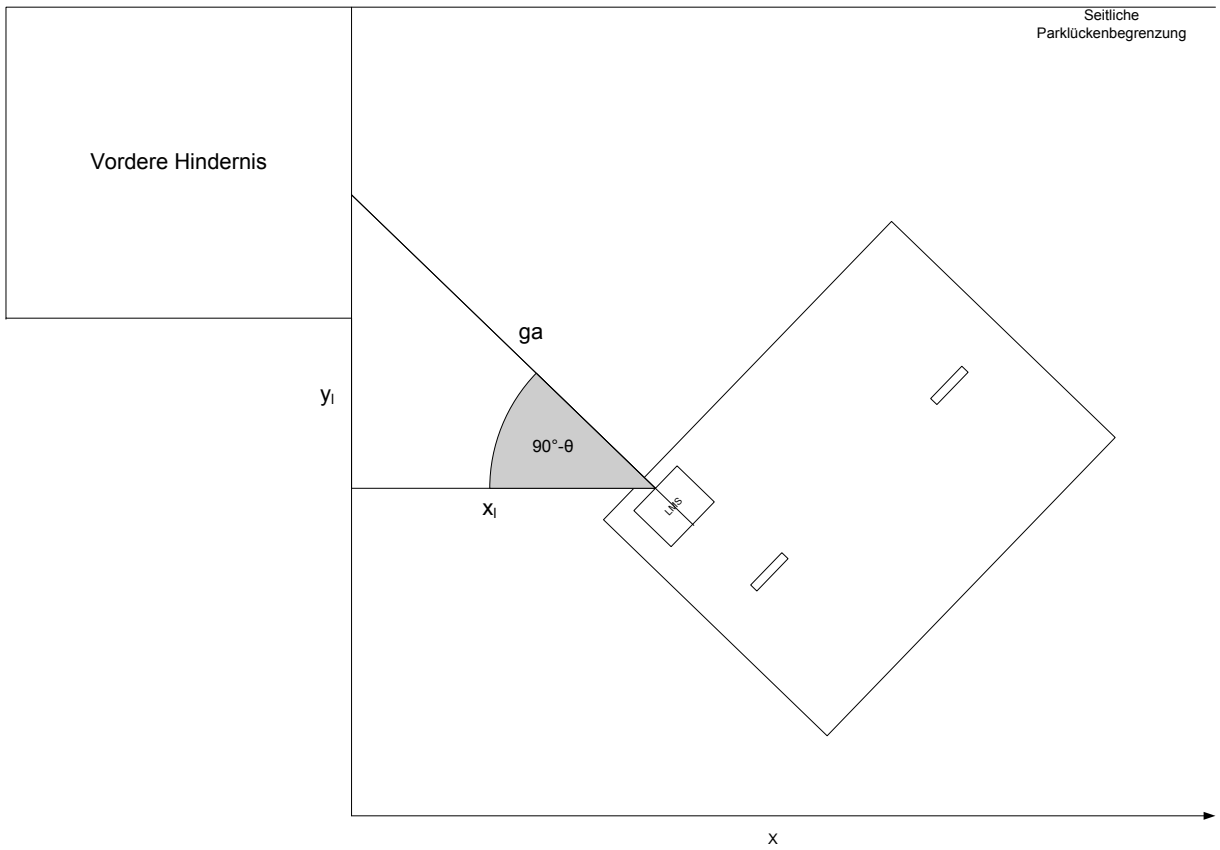

Abb. 24: Abstandserfassung ga mit vertikalen Laserscannear. Abstandsdreieck: x_l in Parklückenlängsrichtung, y_l in Parklückentiefenrichtung

Aufgrund der Ausrichtung des Laserscanners gibt es keine Möglichkeit, den tatsächlichen Ist-Abstand zur seitlichen Parklückenbegrenzung zu messen, wenn das vordere Hindernis erfasst wird. In diesem Fall dürfen die gemessenen Abstände nicht in die Positionsermittlung und der Bestimmung des Lenkwinkels mit einfließen. Erst wenn die seitliche Parklückenbegrenzung erfasst wird, können die Messwerte in den Berechnungen verwendet werden.

Als Vorausbedingung für den Übergang zwischen virtueller und lasergestützter Positionsermittlung, gilt das Setzen des Nullpunktes der Längsrichtung. Nach der Annahme der Parklücke durch den Fahrer, bewegt sich das SCV rückwärts mit einem eingeschlagenen Lenkwinkel von 0°. Sobald der Laserscanner dann eine Parklückentiefe von mindestens 1,2m erfasst, wird diese Position als Nullpunkt für X und Y im Koordinatensystem gesetzt.

Das Abstandsdreieck besteht aus den Katheten in Längsrichtung als x_l und Parklückentieferichtung y_l in Bezug zum Achswinkel θ.

$$x_l = gemessenerAbstand \cdot \cos(90^\circ - \theta) \quad (30)$$

$$y_l = gemessenerAbstand \cdot \sin(90^\circ - \theta) \quad (31)$$

Die berechneten Katheten werden mit dem kinematischen Modell berechneten Positionen des Laserscanners x_m und y_m verglichen. In der Abb. 25 sind die Abstandsdreiecke dargestellt, wenn der Laserscanner das vordere Hindernis oder die seitliche Parklückenbegrenzung erfasst. In dem Fall, dass der Laserscanner den Abstand zur horizontalen Begrenzung des vorderen Hindernises erfasst, ist die Kathete x_l größer als der zurückgelegte Weg in Längstrichtung x_m. Wird der Abstand zur vertikalen Begrenzung des stehenden Hindernisses gemessen, ist x_l gleich x_m. Sobald jedoch der Abstand zur seitlichen Parklückenbegrenzung erfasst wird, ist x_l kleiner als x_m

Aus diesen Beobachtungen kann der Übergang von virtueller zur lasergestützter Positionserfassung definiert werden.

Diese Annahmen gelten jedoch nur unter der Bedingung, dass die vertikale Begrenzung des vorderen Hindernisses keine Unebenheiten aufweist und mit einem rechten Winkel zur seitlichen Parklückenbegrenzung steht. Wird diese Bedingung verletzt, greifen die Bedingungen nicht. Dann sollte zusätzlich die Tiefenkathete y_l mit denen der berechneten oder gemessenen Parklückentiefe y_m verglichen werden. Dies setzt wiederum voraus, dass die seitliche Parklückenbegrenzung keine Unebenheiten aufweist. Wenn sowohl die vertikale und horizontale Begrenzung des Hindernisses als auch die seitliche Parklückenbegrenzung Unebenheiten aufweist, ist die Positionsermittlung mit einem vertikalen Laserscanner nicht möglich.

Abb. 25: Kriterium: Übergang von virtueller zur lasergestützter Positionsermittlung. Vergleich Abstandsdreieckwerte: x_l und y_l und Position vom Startpunkt des Einparkvorgangs: x_m und y_m. Vorderes Hindernis wird erfasst: $y_l < y_m$; $x_l = x_m$. Seitliche Parklückenbegrenzung wird erfasst: $x_l < x_m$; $y_l = y_m$.

4. Übersicht zur Rechnerplattform

In diesem Kapitel werden die Funktionen der unterschiedlichen Hardkomponenten des SCV vorgestellt. Das SCV wird als Versuchsplattform für die Entwicklung des Einparkassistenten verwendet. Im vorausgegangen Masterkurs wurde das SCV entwickelt, so dass es sich über einen Leitstand fernsteuern lässt. Dabei lässt sich die Hardware des SCV in drei Bereiche aufteilen.

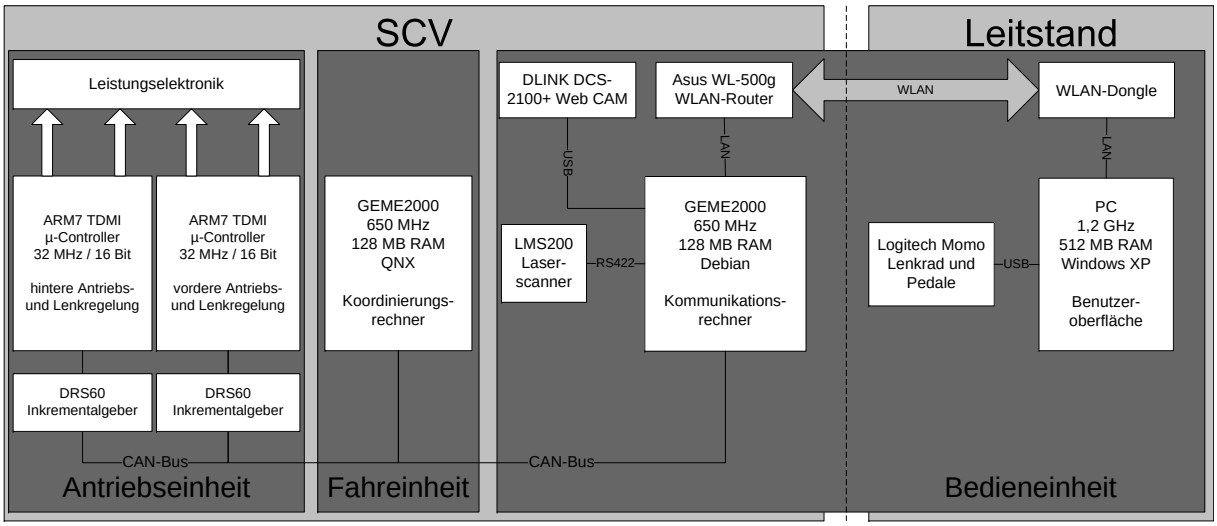

Abb. 26: Übersicht zu den SCV Hardwarekomponenten. Antriebseinheit: Regelung des Antriebs und des Lenkwinkels, Vorder- und Hinterrades. Fahreinheit: Übersetzung der Befehle vom Kommunikationsrechner auf die jeweiligen Räder. Bedieneinheit: Steuerung des SCVs über einen Leitstand und Messung der Parklücke über den Laserscanner.

In diesem Kapitel wird das Zusammenspiel der drei Einheiten sowie deren Funktionalität erläutert. Der Einparkassistent wird auf dem Kommunikationsrechner entwickelt, der aus einem GEME-Rechner besteht. (vgl. Abb.26). Die Verwendung von zwei GEME-Rechnern hat folgenden Hintergrund:

- Erweiterungen von zukünftigen Fahrerassistenzsystemen, die parallel auf einem Rechner ausgeführt werden sollen.
- Modulare Ergänzung von PC104 Steckkarten zur Integration von zusätzlicher Hardware.
- Installation von Linux zur Unterstützung von nebenläufigen Threads, um mehrere Fahrerassistenzsysteme parallel zu verarbeiten sowie nicht proprietären CAN- und Laserscanner-Treiber im ANSI C Format zu verwenden.

4.1. Antriebseinheit

Die Antriebsmotoren des SCVs werden von zwei Mikrocontrollern geregelt. Es handelt sich hier um zwei ARM7 TDMI 32 MHz/16Bit μ-Controller mit 4 Megabyte Flash und 2 Megabyte SRAM. Um mit dem CAN-Bus zu kommunizieren, verwenden beide den 82C900 CAN-Kommunikationskontroller. Da die Ein- und Ausgänge der FSA23 Servos für den Antrieb und die Lenkung eine Spannung von +/-10V für die Ansteuerung der Motoren nutzen, müssen über einen Operationsverstärker die Aus- und Eingangssignale der μ-Controller von 0 bis 3,3V umgewandelt werden. Zwei Inkrementalgeber geben die tatsächliche Geschwindigkeit und den Lenkwinkel beider Räder aus. Diese liefern

über zwei um 90° phasenverschobene digitale Ausgänge Signale, aus denen die Fahrtrichtung, der Lenkwinkel und die Geschwindigkeit der beiden Räder bestimmt werden können.

4.2. Fahreinheit

Der Koordinierungsrechner dient als Schnittstelle/Übersetzer zwischen dem Kommunikationsrechner und der Antriebseinheit. Für die Koordinierung wird ein GEME2000 Rechner mit 650MHz, 128 MByte RAM auf einem QNX Echtzeitbetriebssystem verwendet. Die Bedieneinheit gibt generische Befehle zur Ansteuerung des SCVs vor, welche unabhängig vom Fahrmodus ist. Deshalb werden die Nachrichten auf den CAN-Bus bestehend aus den Soll-Wert-Vorgaben für die Geschwindigkeit und den Lenkwinkel vom Koordinierungsrechner für das Vorder- und Hinterrad übersetzt. Es sind beim SCV drei Fahrmodi möglich:

- lenkbares Vorderrad, starres Hinterrad
- starres Vorderrad, lenkbares Hinterrad
- synchrone Vorder- und Hinterradlenkung

Die Soll-Werte bezüglich Geschwindigkeit und Lenkwinkel werden vom Koordinierungsrechner für das Vorder- und/oder Hinterrad übersetzt. Umgekehrt werden die tatsächlichen Ist-Werte beider Räder für den Leitstand umgerechnet. Da die Fahrbefehle vom Leitstand über ein UDP Socket übertragen werden, kann es zu Aussetzern in der Nachrichtenübertragung kommen. Damit der Koordinierungsrechner einen Verbindungsabbruch detektiert, erwartet die Antriebseinheit zyklische Soll-Wert-Vorgaben. Bleiben diese aus, führt der Koordinierungsrechner das SCV in einen sicheren Zustand und stoppt das Fahrzeug. Bei kurzzeitigen Verbindungsabbrüchen werden die ausbleibenden Soll-Werte vom Koordinierungsrechner bis zu einer halben Sekunde generiert. Soll-Werte die nicht im gültigen Bereich liegen, werden bis zu drei Sekunden kompensiert. Werden innerhalb dieser zeitlichen Toleranzgrenzen keine gültigen Soll-Werte empfangen, stoppt der Koordinierungsrechner das SCV.

4.3. Bedieneinheit

Die Leitstandkomponente stellt die Schnittstelle des Menschen zum SCV dar. Das SCV wird über dem Leitstand mittels Lenkrad und Pedale ferngesteuert, dabei werden die Informationen bezüglich der Soll/Ist-Geschwindigkeit und Lenkwinkel des SCVs über eine graphische Oberfläche dem Fahrer angezeigt. Zusätzlich sieht der Fahrer auf dieser Oberfläche das Bild der WebCam mittels eines Videostreams. Der Datenaustausch erfolgt zwischen Leitstand und Kommunikationsrechner mittels WLAN. Auf dem SCV bereitet der Kommunikationsrechner die WLAN-Fahrbefehle für das interne CAN-Bus Protokoll vor und wertet die Lasermesswerte über die RSS422 Schnittstelle für den Einparkassistenten aus. Es werden folgende Systemzustände über CAN an die graphische Oberfläche des Leitstands übermittelt:

- Suche nach einer Parklücke.
- Erkennung einer geeigneten Parklücke für den Einparkvorgang.
- SCV befindet sich im Einparkvorgang

Für eine vollständige Liste der CAN-Nachrichten wird auf [14]verwiesen.

Mit den Pedalen wird die Fahrtrichtung signalisiert und die Geschwindigkeit des SCVs gesetzt:

- mit dem rechten Pedal, die Vorfährtsfahrt
- mit dem linken Pedal, die Rückwärtsfahrt

Von den Knöpfen des Lenkrads werden folgende Befehle entgegengenommen (vgl. Abb.27):

- Start des Einparkassistenten
- Annahme einer vom Einparkassistenten gefundenen Parklücke
- Ablehnen einer vom Einparkassistenten gefundenen Parklücke
- Abbruch des Einparkassistenten

Abb. 27: Lenkrad mit drei Köpfen zum Starten und Abbrechen des Einparkassistenten und zur Annahme oder Ablehnung der angebotenen Parklücke

4.3.1. Kommunikationsrechner

Für die Kommunikation zwischen Leitstand und SCV wird ein separater GEME2000 Rechner mit 650MHz, 128 MByte RAM verwendet. Es wird ein Linux-Betriebssystem auf dem Rechner mit einer Festplatte installiert. Die Aufgaben des Kommunikationsrechners bestehen aus:

- Umwandlung der Soll- und Ist-Werte von WLAN zu CAN.
- Übernahme der Steuerungsfunktion vom Leitstand für den Einparkassistenten
- Ansteuerung des vertikalen 180° Laserscanners
- Server für den Videostream der WebCam

Zwischen dem SCV und des Leitstandes werden die Soll- und Ist-Werte über WLAN übertragen. Dabei werden während der Fahrt die Soll-Werte vom Leitstand über WLAN auf das interne CAN-Bus umgewandelt und an den Koordinierungsrechner weitergeleitet. In der Entgegengesetzten Richtung werden die vom Koordinierungsrechner gelieferten Ist-Werte für die Geschwindigkeit und den Lenkwinkel von CAN auf WLAN übersetzt. Ist der Einparkassistent aktiviert, benötigt dieser beim Einparkvorgang Zugriff auf die Antriebs- und Lenkungsregelung. Dabei filtert der Kommunikationsrechner in der Einparkphase die Befehle vom Leitstand und übernimmt die Soll-Wert

Generierung der Geschwindigkeit und des Lenkwinkels. Des Weiteren benötigt der Einparkassistent die Daten vom vertikalen 180° Laserscanner, welches mit einer RS422 Schnittstelle an dem Kommunikationsrechner angeschlossen ist. Zusätzlich wurde eine WebCam mittels USB an den Kommunikationsserver angebunden, der nicht für den Einparkassistenten verwendet wird.

Da der Kommunikationsserver eine zentrale Schnittstelle für den Zugriff auf interne und externe Hardwarekomponenten darstellt, wurde der Einparkassistent auf diesem entwickelt. Zwei Gründe sprechen für die Wahl des Kommunikationsrechners:

- Die Laserscanner API arbeitet zuverlässiger unter Linux
- Begrenzung der Nachrichtenlast auf dem CAN-Bus durch Übernahme der Leitstandfunktion

Auf dem Kommunikationsrechner ist Linux bereits installiert und bei der Entwicklung des Treibers wird eine API [30] als Basis verwendet, welche speziell für Linux entwickelt worden ist. Ein weiteres Argument ist die Funktion des Kommunikationsserver als Nachrichtenparser von WLAN zu CAN und zurück. In der Einparkphase sollen die Soll-Werte für den Lenkwinkel vom Leitstand gefiltert werden, wenn der Kommunikationsrechner die Vorgabe der Lenkwinkel übernimmt, wird die Nachrichtenweiterleitung vom Kommunikationsserver zum Koordinierungsrechner unterbunden und damit die Nachrichtenlast vom Leitstand auf den CAN-Bus entfernt.

4.4. Laserscanner im SCV

Es wird ein vertikaler 180° LMS200 Laserscanner mit einer Scanfrequenz von 75Hz verwendet. (vgl. Abb. 28) Angeschlossen wird der LMS200 über einen DB9 Stecker der die Kommunikationsstandards RS232/422 unterstützt. Für den Kommunikationsrechner wurde eine RS422 Karte eingebaut, denn die bereits vorhandene RS232 Schnittstelle kann nicht verwendet werden. Die maximale Datensatzrate der RS232 Schnittstelle von 38,4 kbit/s hätte dazu geführt, dass die Scannerdaten nicht schnell genug abgeliefert worden wären. Berechnungen zu Folge [14], benötigt RS232 bei der maximalen Transferrate 55,2 ms, um das Datenpaket für einen kompletten Laserscanzyklus abzuliefern. Der komplette Scanzyklus des LMS200 beläuft sich auf T_{LS}. Durch die Verwendung der RS422 Schnittstelle verringert sich die Übertragungszeit auf 4,24 ms, für einen kompletten Scanzyklus bei einer Baudrate von 500 kbit/s.

Abb. 28: Sick LMS200, 75Hz Scanfrequenz, 180°Sichtfeld, RS232/422 Interface

5. Softwaremodellierung mit einem Harel-Automatenmodell

Bei der Entwicklung der Software des Einparkassistenten wird eine Harel-Automatenmodellierung zur Beschreibung der Systemzustände angewendet. In diesem Kapitel wird die Modellierung der einzelnen Automaten *Laserscanner*, *Einparken*, *Steuer_Parkassistent* und *Parkluecke* erläutert.

Es wird das Design der Zustandsdiagramme erklärt und die Funktionalitäten der einzelnen Automaten beschrieben. Die Automaten kommunizieren mit Nachrichten der AIRA-Laufzeitumgebung untereinander, mit Sequenzdiagrammen wird das Zusammenspiel der Automaten. während der Parklückensuche und des Einparkvorgangs verdeutlicht.

5.1. AIRA-Automaten

Im Rahmen einer vorausgegangenen Projektarbeit wurde jedem Teilnehmer die Aufgabe zur Entwicklung eines Automats zugewiesen. Für die Implementierung der Funktionalitäten, das Zusammenspiel der Automaten und den Austausch der Nachrichten, hat man sich auf die AIRA-Modellierung festgelegt [15], siehe Kapitel 6. Das Verhalten der Zustände ist von der Analyse der eingetroffenen Nachrichten abhängig. Sie kann aus einer Kombination von Zustandswechsel, das Aufrufen einer Funktionalität, das Auslösen von einem Ereignis oder der Versand einer weiteren Nachricht bestehen. Dabei gilt, dass jeder Automat in die Nachrichten Empfangsqueue der anderen Automaten schreiben kann. Die Nachrichten werden nach dem FIFO-Prinzip in den einzelnen Automaten gespeichert. Jeder Automat besitzt dafür eine dedizierte Nachrichten-Empfangsqueue.

Die Automaten verfügen über folgende Funktionalitäten:

- *Comm_Server*: Die Bidirektionale Umwandlung der WLAN, CAN-Nachrichten, die Weiterleitung des Verfahrwegs und der Geschwindigkeit zum Steuerparkassistent und das Senden des Ist-Lenkwinkels zum Automaten *Einparken*, sowie die Generierung von CAN-Nachricht zur Vorgabe des Lenkwinkels und der Geschwindigkeit.
- *Steuer_Parkassistent*: Schnittstelle zwischen AIRA und der Funktionalitäten des *Comm_Server*.
- *Parkluecke*: Auswertung der Parklücke durch den Empfang der Abstandswerte vom 180° Laserscanner und des Verfahrwegs vom Steuerparkassistent.
- *Laserscanner*: Initialisierung, Parametrisierung und Weiterleitung der Abstandswerte von der RS232/422 Schnittstelle zum Automat *Parkluecke*.
- *Einparken*: Berechnung der Odometrie und Abstandsregelung mit und ohne Abstandswerte vom Laserscanner, sowie die Soll-Vorgabe von Geschwindigkeit und Lenkwinkel.

In den zwei Phasen des Einparkassistenten kommunizieren unterschiedliche Automaten miteinander. Während der Parklückensuche tauschen die Automaten *Parkluecke*, *Laserscanner*, *Comm_Server* und *Steuer_Parkassistent* Nachrichten aus, beim Einparkvorgang die Automaten *Einparken*, *Laserscanner* und *Steuer_Parkassistent*.

Beim Start des Einparkassistenten wird zunächst der Laserscanner aktiviert. Nach der Initialisierung, der Parametrisierung der Übertragungsgeschwindigkeit und Auflösung des Laserscanners durch den Automat *Laserscanner*, wird die Auswertung der Parklücke gestartet. Dabei übermittelt der Automat *Laserscanner* die Lasermesswerte an den Automaten *Parkluecke*. Um die

Analyse der Parklücke zu vervollständigen benötigt der Automat den zurückgelegten Weg. Diese Information erhält der Automat vom *Steuer_Parkassistent* durch die Kapslung der CAN-Nachricht vom *Comm_Server*. Wenn die analysierten Daten für eine Parklücke ausreichen, d.h. die Parklücke ist 2,4m breit und durchgehend mindestens 1,2m tief ist, wird dies dem Fahrer über dem Leitstand mitgeteilt. Der Benutzer kann den Einparkvorgang einleiten oder ablehnen. Wird der Einparkvorgang eingeleitet wird, wird *Comm_Server* in einen Zustand versetzt, wo *Steuer_Parkassistent* die Soll-Vorgaben für die Geschwindigkeit und des Lenkwinkels generiert. Die WLAN-Steuerungsbefehle werden vom *Comm_Server* nicht mehr in das CAN-Format übersetzt, so dass *Steuer_Parkassistent* die Kontrolle über die Steuerung des SCVs erhält.

Nun berechnet der Automat *Einparken* die Soll-Vorgaben für den Lenkwinkel und leitet in T_{CAN} Abständen Lenkwinkel- und Geschwindigkeits-Soll-Werte an *Steuer_Parkassistent* weiter. Unterbrechungen des Einparkvorgangs werden vom Fahrer durch das Loslassen des Gaspedals auf dem Leitstand signalisiert. (vgl. Kapitel 2.2.2) Der *Steuer_Parkassistent* erkennt dies und setzt das Einpark Automat in einen Stand-By Modus. Sobald der Fahrer das Gaspedal wieder betätigt, übernimmt der Automat *Einparken* wieder seine ursprüngliche Funktionalität und der Einparkvorgang wird fortgesetzt.

5.1.1. Nachrichtenfluss im SCV

Alle Automaten die im Einparkassistenten untereinander kommunizieren verwenden die internen AIRA-Nachrichten-Queues der Laufzeitumgebung. In Abbildung 29 werden die einzelnen Automaten, bestehend aus *Comm_Server*, *Steuer_Parkassistent*, *Parkluecke*, *Einparken* und *Laserscanner* aufgezeigt.

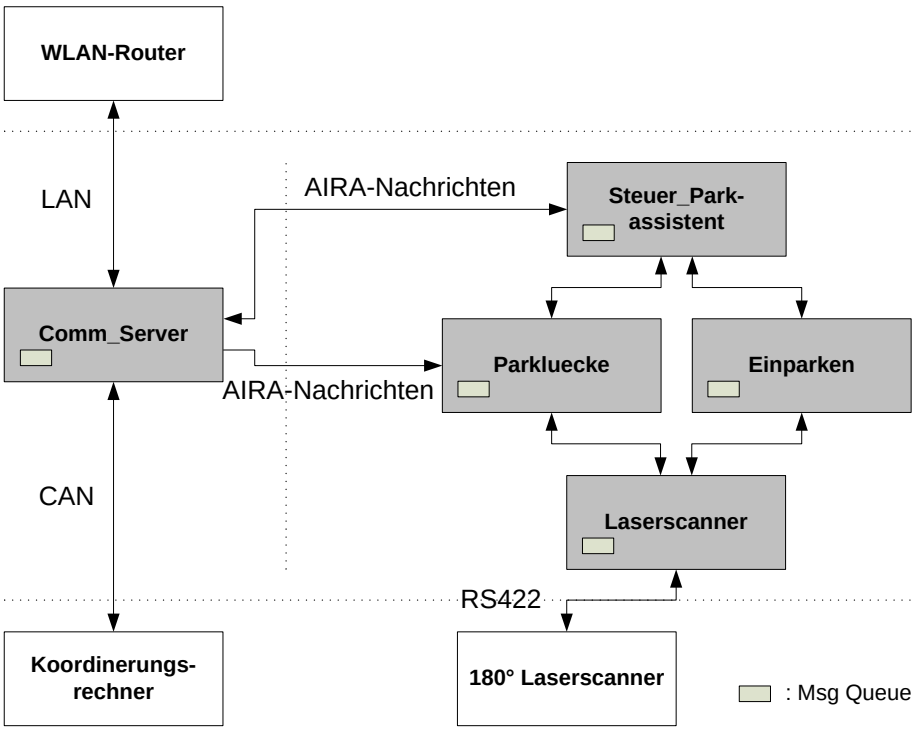

Abb. 29: Kommunikationskanäle der Automaten

In der AIRA-Laufzeitumgebung werden zwei Arten von Nachrichten unterschieden. Die N-Nachrichten enthalten Daten, während die S-Nachrichten Signale zu einem Ereignis darstellen. Bei den N-Nachrichten können eigene Datenstrukturen definiert werden, damit kann die Größe skaliert und das Datenformat festgelegt werden. Der zweite Parameter enthält den Empfänger der Nachricht dabei stehen:

- LMS für Automat *Laserscanner*
- PL für Automat *Parkluecke*
- STP für Automat *Steuer_Parkassistent*
- EP für Automat *Einparken*
- LST für *Leitstand*

Während der dritte und alle nachfolgende Parameter aus einem selbst definierten Namen besteht. Die Nachrichtennamen werden in der globalen Headerdatei global_setting.h festgelegt. (vgl. Abb. 30)

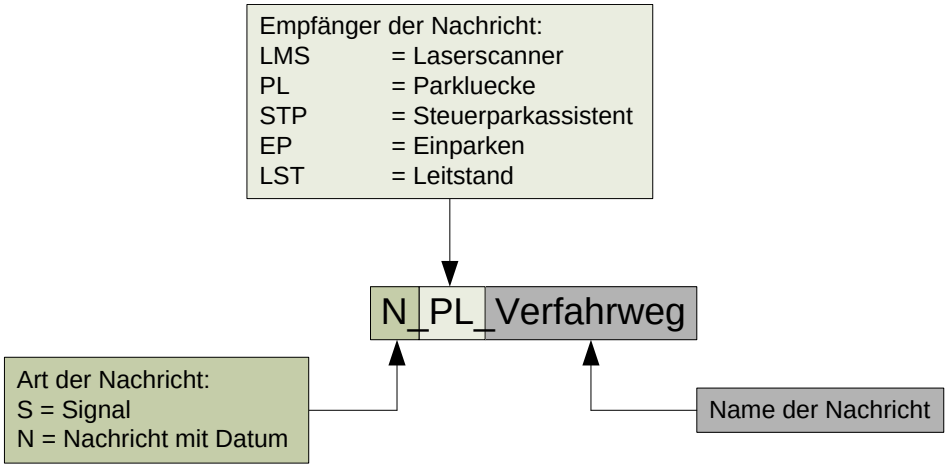

Abb. 30: Beispiel einer Nachricht nach der definierten Notation

5.1.2. Phase der Parklückensuche mit dem 180° Laserscanner

Die Abstände zur Wand werden vom Laserscanner in einem Ringpuffer gespeichert, die der Automat *Parkluecke* zusammen mit dem Verfahrweg vom Schrittmotor auswertet. Der *Steuer_Parkassistent* signalisiert *Leitstand* die Suche und die Erkennung einer Parklücke und nimmt die Befehle vom *Leitstand* bezüglich des Starts des Assistenten, sowie Ablehnen und Annahme der Parklücke durch den Fahrer entgegen.

Im Automaten *Parkluecke* gibt es für die Initialisierung des Laserscanners einen Zustand. Dieser wird verlassen sobald die Initialisierung des Laserscanners abgeschlossen wurde. Wenn die Parklückensuche abgeschlossen oder abgebrochen wird, muss der Laserscanner vorübergehend gestoppt werden. Für die Reinitialisierung des Laserscanners werden fünf bis zehn Sekunden benötigt. Damit der Bediener nicht auf den Neustart des Laserscanners warten muss, wird der Scanzyklus nie gestoppt, sondern die Übertragung der Messdaten an den Automat *Parkluecke* unterdrückt. Die bereits erfolgte Initialisierung des Laserscanners wird im Automat *Parkluecke* mit einem Pufferzustand Standby berücksichtigt. Es wird die Parklückensuche nicht mehr durch das

Verlassen der Initialisierungszustände des Automaten *Parkluecke* gestartet, sondern durch das Verlassen des Standby-Zustands.

Automat Laserscanner

Der LMS200 verfügt über ein eigenes Nachrichtenaustauschformat [25]. Während die Dokumentation über die Nachrichtentelegramme öffentlich zugänglich ist, musste der Treiber selber entwickelt werden. Eine Antwort auf Anfragen für einen Treiber gab es von der Firma Sick nicht. Der Entwicklungsaufwand für eine vollständige Treiber-Bibliothek wurde als zu hoch eingestuft. Als Folge wurde im Internet ein Demonstrationsprogramm für eine Linux Umgebung gefunden [30]. Dieser Treiber wird in diesem entwickelten Automaten zur Parametrisierung und Auslesen der Messwerte von der RS422 Schnittstelle verwendet.

Der Laserscanner benötigt für einen Scanzyklus T_{LS}, hier wird das Abtasttheorem von Nyquist angewandt und alle 6 ms stößt der Tick Timer das Auslesen der Abstandswerte an. Da der Treiber keine Interrupts verarbeitet, wird das Auslesen der Abstandswerte aus dem File-Deskriptor vom Tick-Timer ausgelöst. Die dabei aufgerufene Linux read()-Funktion liest die Abstände byteweise von der RS422 Schnittstelle aus. Eine Abstandsnachricht vom LMS200 besteht aus 14 Bytes, so dass die Funktion 14-mal aufgerufen wird. (vgl. Abb. 31)

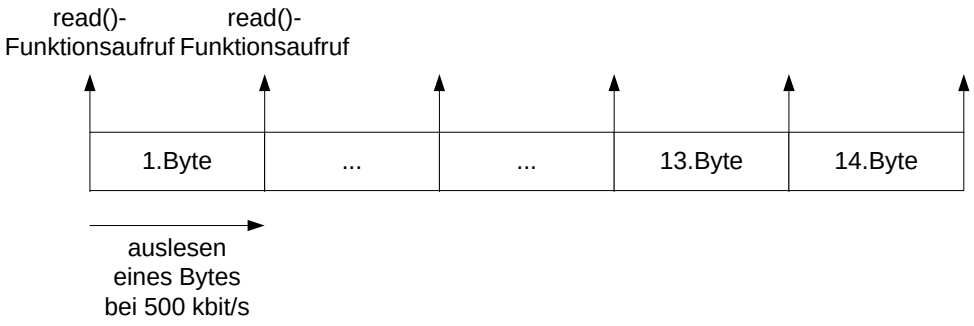

Abb. 31: Auslesen einer 14 Byte großen Abstandsnachricht vom LMS200

Nach dem Einschalten des SCVs befindet sich der Automat im Startzustand Z_LMS_IDLE. Sobald dieser Zustand verlassen wurde, wird er erst wieder betreten, wenn beim Laserscanner ein Fehler aufgetreten ist. Im Z_LMS_INIT Zustand wird der vertikale 180° Laserscanner für die Parklückensuche parametrisiert, dabei wird eine Messauflösung von 1° und eine Übertragungsrate von 500 kbit/s eingestellt. Im Z_LMS_SCAN Zustand werden die Lasermesswerte der seitlichen Abständen in einen Ringpuffer geschrieben. Mit der boolsche-Variable LMS_Surpressing wird die Übertragung von Abstandwerten unterdrückt. Die Variable wird durch zwei Nachrichten gesetzt oder gelöscht. Der Zustand Z_LMS_SCAN_PARKEN leitet die Abstandswerte an dem Automaten *Einparken* weiter. (vgl. Abb. 32)

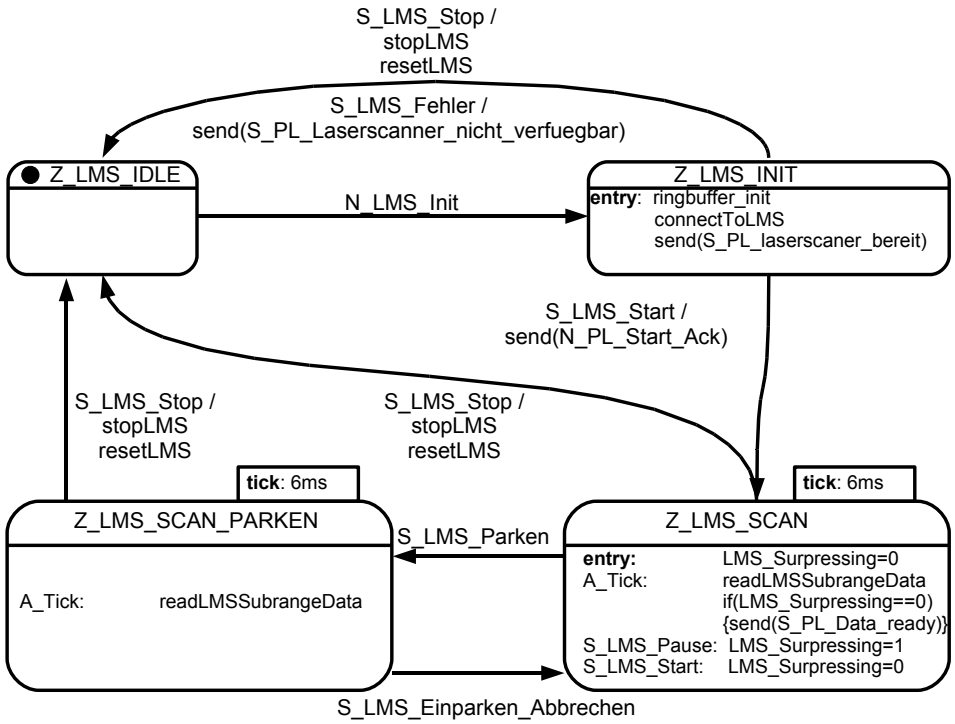

Abb. 32: Zustandsdiagramm Automat *Laserscanner*

Parkluecke

Dieser Automat ist in der Phase der Parklückensuche aktiv. Er analysiert die Abstandswerte und den Verfahrweg, prüft ob die Parklücke 2,4m lang und durchgehend mindestens 1,2m tief ist. Die Variable **tiefe** stellt den Abstandwert dar, während **laenge** den zurückgelegten Verfahrweg speichert. Aus der Position bei der die Parklücke erkannt wurde, startet der Einparkvorgang.

Der Automat enthält die Zustände Z_PL_IDLE, Z_PL_INIT_SCANNER und Z_PL_SUCHEN, im Automat *Laserscanner* sind die Zustände identisch. Bei der Parklückensuche werden dieselben kollaborierenden Zustände eingenommen. Im Zustand Z_PL_INIT_SCANNER initialisiert *Parkluecke* den Laserscanner. Ist nach dem dritten Versuch die Initialisierung des Laserscanners fehlgeschlagen, wird der Idle Zustand wieder hergestellt. Nach der Einstellung des Laserscanners wechselt der Automat in den Z_PL_SUCHEN Zustand, dort werden die Messwerte der Abstände mit den Werten des Verfahrwegs ausgewertet, die er vom *Steuer_Parkassistent* erhält. Nach der Identifikation einer für den Einparkvorgang verwendbaren Parklücke, wird dies *Steuer_Parkassistent* mitgeteilt und die Messwerte vom Laserscanner unterdrückt. Im Zustand Z_PL_ANBIETEN wird der Verfahrweg zur Startposition mitgezählt. Erst mit der Annahme der Parklücke durch den Fahrer, wird die Messung des Verfahrwegs gestoppt und an *Steuer_Parkassistent* übertragen. (vgl. Abb. 33)

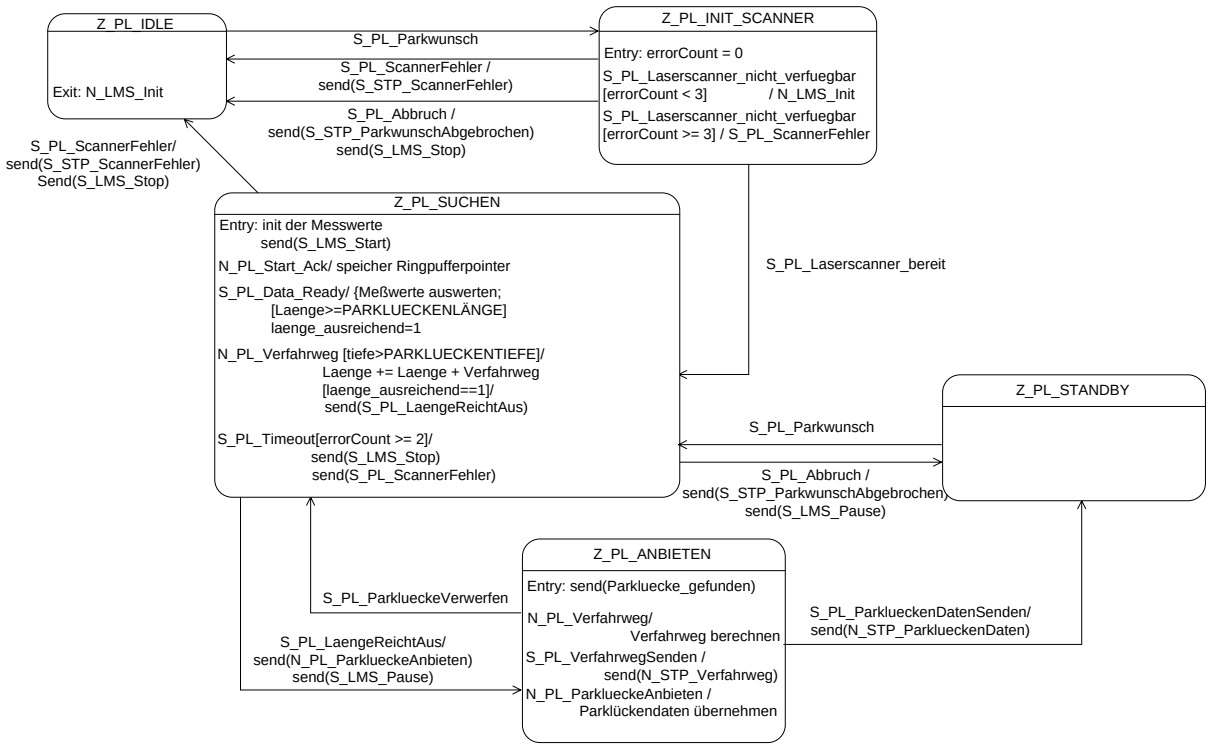

Abb. 33: Zustandsdiagramm für das Automat *Parkluecke*

5.1.3. Lenkwinkel Soll-Wert Vorausberechnung mit den Abstandsreglern

Der Automat *Einparken* beinhaltet sämtliche Berechnungen der Odometrie, des PD-Reglers und der virtuellen Deichsel. Die Aktualisierung der Odometrie und der Vorgabe des Lenkwinkel Soll-Werts geschieht mit der Synchronisation der CAN-Nachricht für den Ist-Lenkwinkel α, die alle T_{CAN} empfangen wird. Die berechnete Geschwindigkeit und der Lenkwinkel Soll-Wert werden *Steuer_Parkassistent* zugeführt und an *Comm_Server* weitergeleitet. Zusätzlich registriert *Steuer_Parkassistent*, ob der Fahrer den Einparkvorgang startet, abbricht, unterbricht oder fortführt, da er diese Nachrichten vom *Comm_Server* erhält.

Die Unterbrechung des Einparkvorgangs nach Loslassen des Gaspedals wird im Automat *Einparken* durch Standby-Zustände repräsentiert, dort muss die Geschwindigkeit +/- 0,5 m/s vor dem Eintritt gespeichert werden, da im Laufe des Einparkvorgangs das SCV vor- oder rückwärts fährt.

Im *Steuer_Parkassistent* sind Pufferzustände, diese wurden für die Übergänge zwischen Parklückensuche, Einparkvorgang und Abschluss des Einparkvorgangs eingeführt. Beim Wechsel von Parklückensuche zu Einparkvorgang muss das Gaspedal betätigt werden, während der Parkvorgang erst abgeschlossen wird, wenn das Gaspedal losgelassen wird. Ohne Pufferzustände würde der Fahrer nicht wissen, wann er die Steuerung wieder übernehmen kann, und ob der Einparkvorgang gestartet oder beendet wurde.

Automat Einparken

Zunächst muss sich das SCV zur Startposition begeben, dazu erhält der Automat den Verfahrweg. Erst nachdem diese Position erreicht wird, kann der Einparkvorgang gestartet werden. Dazu fordert

der Automat die Länge und die Tiefe der Parklücke vom *Steuer_Parkassistent* an. Mit der lasergestützten Abstandsregelung, die in den Kapitel 3 und 3.3 dokumentiert wurden, fährt das SCV mit einer konstanten Geschwindigkeit von 0,5 m/s in die Parklücke.

Mit dem Eintritt in den Zustand Z_EP_X_POS_ERREICHEN wird der Verfahrweg bis zur Startposition des Einparkvorgangs angefordert, und das SCV bewegt sich mit einer Geschwindigkeit von +/- 0,5 m/s zur Startposition. Sobald die Position erreicht wurde, wird die Tiefe der Parklücke angefordert und der Zustand Z_EP_EINPARKEN angenommen. In diesem Zustand empfängt er die Abstandswerte vom Laserscanner und lässt durch die Entscheidungslogik bestimmen, ob ein Hindernis oder die seitliche Parklückenbegrenzung erfasst wird. (vgl. Kapitel 3.6) Nach dem Empfang des Lenkwinkel Ist-Werts, wird die Odometrie und die Lenkwinkel Soll-Vorgabe berechnet und an *Steuer_Parkassistent* weitergeleitet. Sobald der Einparkvorgang abgeschlossen ist, wird der Startzustand wieder betreten. (vgl. Abb. 34)

Beide Standby-Zustände werden betreten, wenn der Fahrer das Gaspedal loslässt. Dabei wird die Geschwindigkeit, die vor der Unterbrechung eingestellt wurde mit einer Nachricht übergeben, damit bei einer Fortsetzung des Einparkvorgangs die ursprüngliche Geschwindigkeit wieder eingestellt wird.

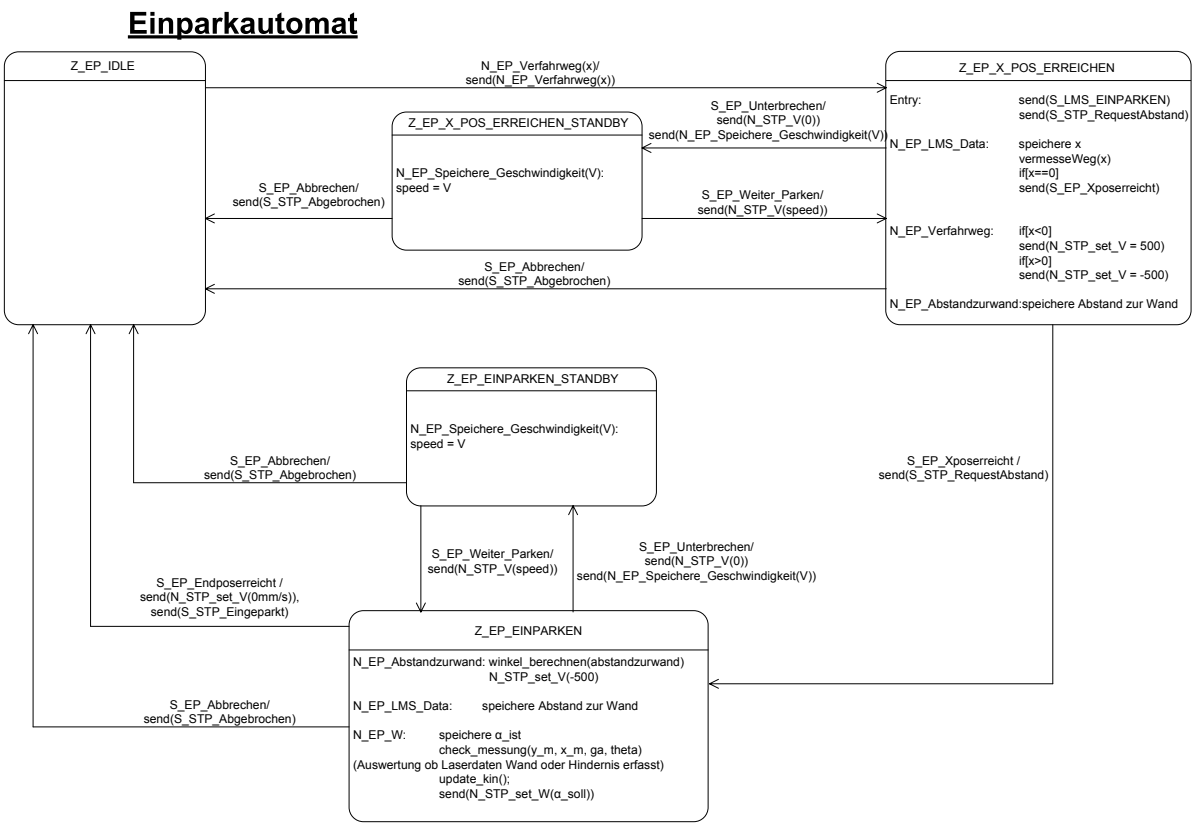

Abb. 34: Zustandsdiagramm für den Automat *Einparken*

Automat Steuer_Parkassistent

Der *Steuer_Parkassistent* dient als bidirektionale Schnittstelle zwischen WLAN und CAN. Er startet den Einparkassistenten und aktiviert die beteiligten Automaten *Einparken* und *Parkluecke*. Da zwischen diesen Automaten keine Nachrichten ausgetauscht werden, muss *Steuer_Parkassistent* während des Übergangs der Parklückensuche und dem Einparkvorgang, die Tiefe und Länge vom Automat *Parkluecke* zwischenspeichern und diese dem Automat *Einparken* übermitteln.

Während der Parklückensuche kapselt er die Verfahrweg-Nachricht vom CAN-Bus und leitet sie dem Automat *Parkluecke* weiter. Beim Einparkvorgang übernimmt *Steuer_Parkassistent* die Vorgabe des Lenkwinkels und der Geschwindigkeit und überprüft gleichzeitig, ob das Gaspedal losgelassen oder der Knopf zum Abbruch des Einparkassistenten betätigt wurde. Er empfängt die Nachrichten zum Start und Abschluss der Parklückensuche, sowie des Einparkvorgangs und signalisiert der graphischen Oberfläche auf dem Leitstand die Zustände mit den entsprechenden Nachrichten.

Für die Phasen Parklückensuche und Einparken gibt es einen Zustand: In Z_STP_PARKLUECKE_SUCHE startet *Steuer_Parkassistent* den Automat *Parkluecke* und übermittelt ihm den Verfahrweg. Wird eine Parklücke gefunden, soll das dem Fahrer über *Leitstand* mitgeteilt werden. Beim Einparkvorgang Z_STP_EINPARKEN_TICK übernimmt *Steuer_Parkassistent* den Empfang der Soll-Vorgaben für Geschwindigkeit und Lenkwinkel vom Automat *Einparken* und leitet sie an *Comm_Server* weiter. Gleichzeitig überprüft er das Gaspedal. Sobald dieser losgelassen wird, unterbricht *Steuer_Parkassistent* den Einparkvorgang. Mit Z_STP_WARTE_AUF_GAS wird verhindert, dass während des Vorgangs der Parklückensuche der Einparkvorgang eingeleitet wird, und mit Z_STP_WARTE_AUF_GAS_NULL nach Abschluss des Einparkvorgangs mit gedrücktem Gaspedal das SCV in das vordere Hindernis fährt. (vgl. Abb. 35)

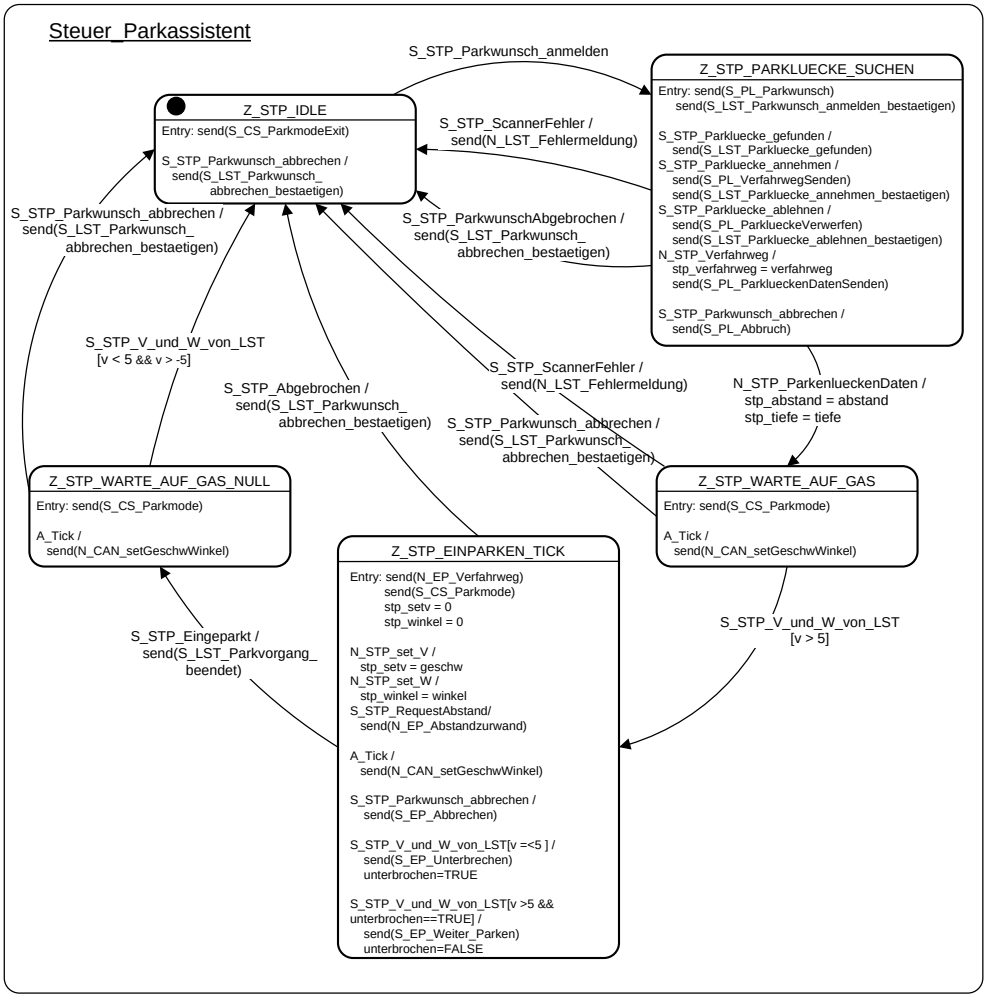

Abb. 35: Zustandsdiagramm für den Automat *Steuer_Parkassistent*

5.2. Sequenzdiagramme für den Einparkassistenten

In diesem Abschnitt sollen durch Sequenzdiagramme das Zusammenspiel und der Ablauf der Automaten des Einparkassistenten vervollständigt werden. Dieses Kapitel wird in zwei Abschnitte gegliedert, um dort die Sequenzen der beteiligten Automaten, während der Parklückensuche und des Einparkvorgangs zu beschreiben. Im Automat *Steuer_Parkassistent* gibt es für die Parklückensuche und Einparkvorgang einen einzelnen Zustand. Aus den Sequenzdiagrammen soll ein Überblick verschafft werden, welche Zustände Nachrichten miteinander austauschen. Dieser Überblick dokumentiert weniger die Zeitkennwerte, sondern den sequentiellen Ablauf des Nachrichtenversands, des Nachrichtentyps und der Zustandsübergänge.

5.2.1. Parklückensuche

In der Phase der Parklückensuche nehmen die Automaten *Leitstand*, *Parkluecke*, *Laserscanner* und *Steuer_Parkassistent* teil. Die Sequenzdiagramme zur Parklückensuche beinhalten die Initialisierung des Laserscanners, die Auswertung der Parklücke, die Speicherung des Verfahrwegs bis zur Startposition des Einparkvorgangs und den Abschluss der Parklückensuche.

Der Start des Einparkassistenten wird auf dem Lenkrad durch das Drücken des Startknopfs angeregt. Die dabei weitergeleiteten Nachrichten führen die Automaten *Parkluecke* und *Laserscanner* aus den Startzuständen in die Zustände der Initialisierung des Laserscanners. Nach der Initialisierung des Laserscanners wird die Parklückensuche gestartet, dabei kombiniert der Automat *Parkluecke* den zurückgelegten Verfahrweg mit den Abständen zum seitlichen Hindernis. Dabei wird der Verfahrweg durch *Steuer_Parkassistent* Automaten bereitgestellt, der in zyklischen Abständen von T_{CAN} von der Antriebseinheit verschickt wird, während die Abstände zur Wand vom Automat *Laserscanner* in einem Ringpuffer geschrieben werden.

Wenn die Parklückensuche abgeschlossen wurde, wird im Zustand Z_Parkluecke_anbieten von *Parkluecke*, ein von der Parklückenanalyse unabhängiger Verfahrweg bis zur Startposition des Einparkvorgangs gespeichert. Erst mit der Annahme der Parklücke durch den Fahrer wird der Verfahrweg bis zur Startposition an *Steuer_Parkassistent* gesendet und die Parklückensuche beendet.

Initialisierung des Laserscanners
Durch das Drücken des Startknopfes auf dem Lenkrad wird die S_STP_Parkwunsch_anmelden Nachricht an den Automat *Steuer_Parkassistent* gesendet. Diese Nachricht ist der Anstoß zum Start der Parklückensuche, in den Automaten *Parkluecke* und *Laserscanner* werden jetzt die Zustände eingenommen, die für die Initialisierung des Laserscanners zuständig sind. Dazu wird ausgehend vom *Steuer_Parkassistent* die Nachricht S_PL_Parkwunsch an den Automat *Parkluecke* geschickt. *Parkluecke* wechselt in den Initialisierungszustand Z_Parkluecke_Init_Scanner. Es wird die Nachricht N_LMS_Init erstellt, mit Auflösungs- und Übertragungsgeschwindigkeitsparametern an den *Laserscanner* weitergeleitet. Die Nachricht führt das Automat *Laserscanner* in den Initialisierungszustand Z_LMS_INIT, mit den Parametern wird der Laserscanner auf eine Auflösung von 1° pro Sample und einer Übertragungsrate von 500 kbit/s konfiguriert, der Ringpuffer initialisiert und der Pointer gespeichert. Sobald der Laserscanner und der Ringpuffer einsatzbereit sind, bestätigt der Automat *Laserscanner* der *Parkluecke* den Abschluss der Initialisierung mit der Nachricht S_PL_Laserscanner_bereit.

Nach dem Empfang der Nachricht wechselt der Automat *Parkluecke* in den Zustand der Parklückenanalyse Z_Parkluecke_suchen. Der Automat ist bereit, die vom vertikalen 180° Laserscanner ermittelten Abstände zur Wand mit dem Verfahrweg auszuwerten. Zwar ist der Laser initialisiert, liefert zu diesem Zeitpunkt jedoch noch keine Abstände zum seitlichen Hindernis. Erst mit dem Versand der Nachricht S_LMS_Start von *Parkluecke* und der anschließenden Antwort N_PL_Start_Ack vom *Laserscanner*, welche den Pointer auf den Ringpuffer enthält, können die Abstände in den Ringpuffer geschrieben werden. Der *Laserscanner* führt einen Zustandswechsel von Z_LMS_INIT zu Z_LMS_SCAN aus und schreibt die Abstände in den Ringpuffer. Damit sind beide Automaten in den Zuständen der Parklückenanalyse.

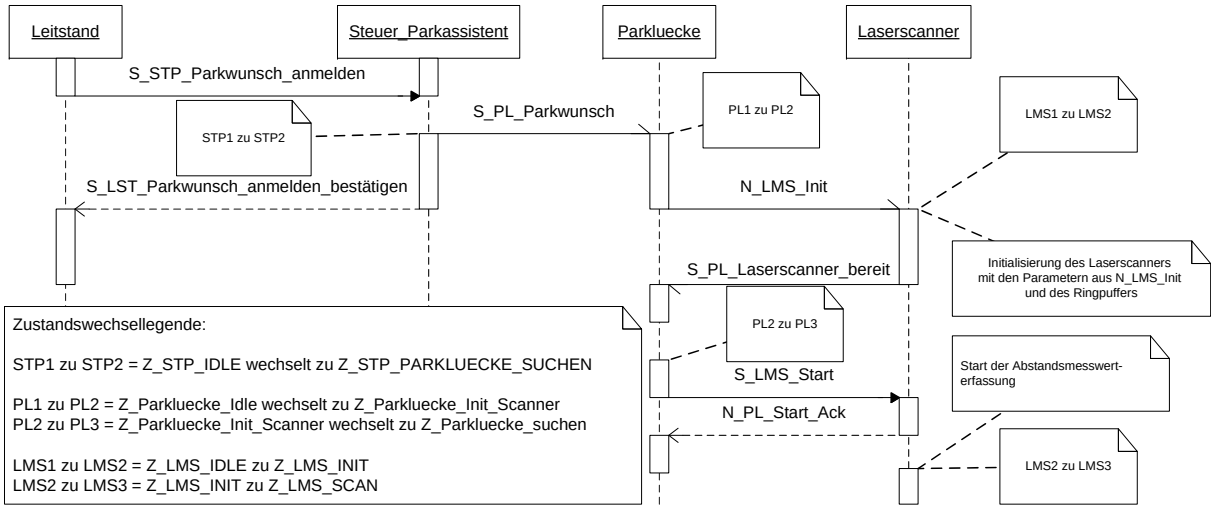

Abb. 36: Sequenzdiagramm zur Initialisierung des Laserscanners.

Auswertung der Parklückentiefe und des Verfahrweges

Zur Analyse der Parklücke wird der Verfahrweg und die Abstände zum seitlichen Hindernis benötigt. Dazu wird die Nachricht N_PL_Verfahrweg mit dem zurückgelegten Weg zyklisch an dem Automat *Steuer_Parkassistent* geschickt. Da es sich um eine CAN-Nachricht handelt, beträgt der Zyklus für das Versenden der Verfahrwegnachricht an Automat *Parkluecke* T_{CAN}. Der Zyklus für einen kompletten 180° Scan beträgt T_{LS}. Aus diesem Grund wird Nyquists Abtasttheorem angewandt und der Automat *Laserscanner* überprüft mit einer doppelten Abtastrate von 6ms, ob der Scanzyklus abgeschlossen ist. Nachdem der Automat die Abstände in den Ringpuffer geschrieben hat, signalisiert er dem Automat *Parkluecke* mit der Nachricht S_PL_Data_ready, dass neue Daten zur Analyse bereit stehen. Das Versenden dieser Nachricht kann durch das Setzen der Variable LMS_surpressing unterdrückt werden, wenn der Automat S_LMS_Pause empfängt.

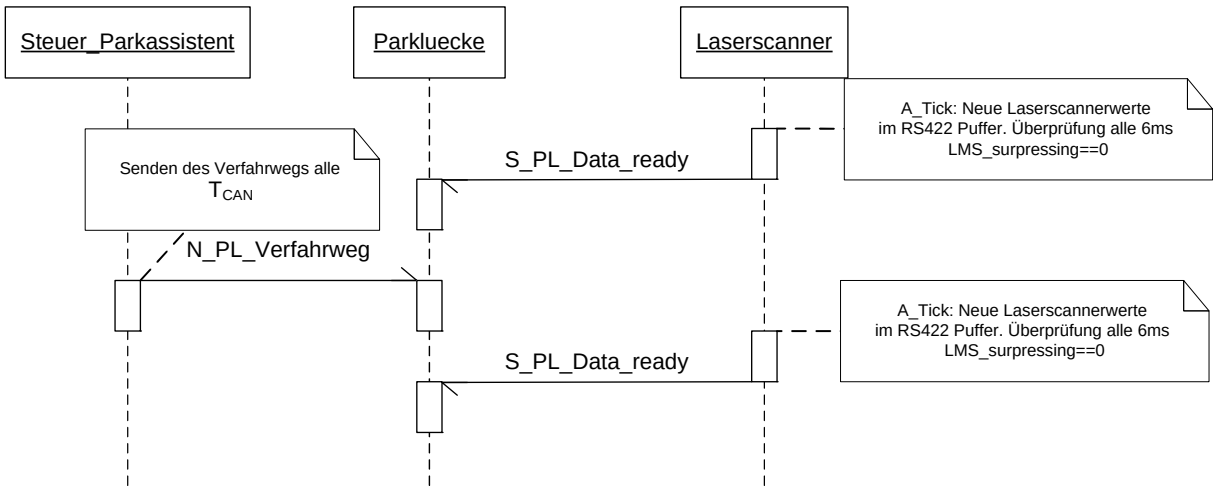

Abb. 37: Sequenzdiagramm zur Analyse der Parklücke mit den Abständen zur Wand vom Laserscanner und dem Verfahrweg vom *Steuer_Parkassistent*.

Abschluss der Parklückensuche

Sobald der Automat *Parkluecke* eine Parklücke gefunden hat, die durchgehend mindestens 1,2 m tief und 2,4 m lang ist, schickt es sich selbst eine S_PL_LaengeReichtAus-Nachricht. Damit ist die Suche nach der Parklücke abgeschlossen. Im weiteren Verlauf werden Nachrichten generiert, die die beteiligten Automaten aus den Zuständen der Parklückensuche führen.

Erst mit dem Neustart der Parklückensuche wird mit dem Versenden der S_LMS_Start-Nachricht, die Variable LMS_surpressing wieder aufgehoben und der Automat *Laserscanner* versendet wieder die Abstände. Das Sequenzdiagramm in Abb. 38 beschreibt den Fall, dass der Fahrer die Parklücke annimmt. Da es keine globalen Variablen im Automat *Parkluecke* gibt, wird die Tiefe und die Breite der Parklücke mit der Nachricht N_PL_Parkluecke_anbieten an sich selbst geschickt, so werden vor dem Zustandswechsel die Parklückendaten übernommen. Mit dem Versenden der N_PL_Parkluecke_anbieten Nachricht endet der Zustand Z_Parkluecke_suchen und wechselt in den Zustand Z_Parkluecke_anbieten. Dort wartet der Automat auf eine Nachricht, die die Annahme oder Ablehnung der Parklücke durch den Fahrer signalisiert. Mit der Nachricht S_STP_Parkluecke_gefunden wird dem *Steuer_Parkassistent* mitgeteilt, dass eine Parklücke gefunden wurde.

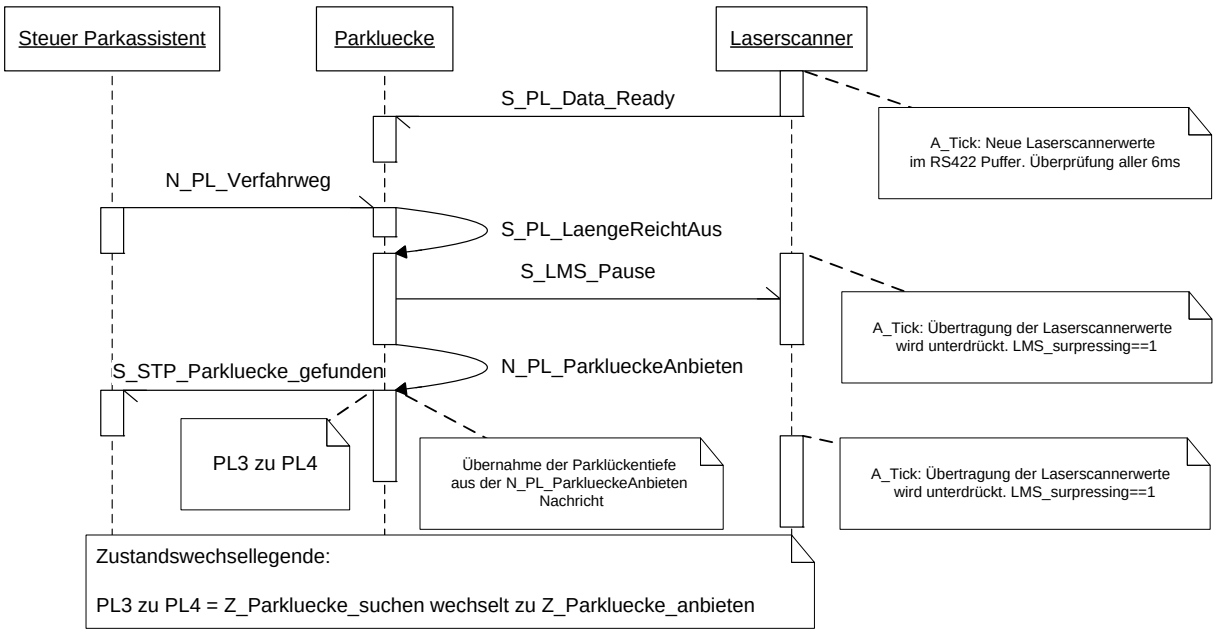

Abb. 38: Sequenzdiagramm zum Abschluss der Parklückensuche.

Speicherung des Verfahrwegs bis zur Startposition des Einparkvorgangs
Nachdem die Parklückensuche abgeschlossen ist, muss der Fahrer die gefundene Parklücke annehmen. Während der Fahrer auf die Annahme der Parklücke antwortet, hat sich das SCV von der Startposition entfernt, an der die Parklücke detektiert wurde. Entscheidet er sich für die Einleitung des Einparkvorgangs, werden dem *Steuer_Parkassistent* die Tiefe der Parklücke geschickt und der Weg, der zwischenzeitlich zurückgelegt wurde.

Zwischen den Automaten *Einparken* und *Parkluecke*, findet kein direkter Datenaustausch statt. Deswegen muss der zurückgelegte Weg vom *Steuer_Parkassistent* an den Automat *Einparken* weitergeleitet werden. Nach der Annahme der Parklücke wird vom *Leitstand* die Nachricht S_STP_Parkluecke_annehmen geschickt und durch die entsprechende Gegenantwort bestätigt. Der zurückgelegte Weg bis zum Startpunkt des Einparkvorgangs, wird vom Automat *Parkluecke* durch die Nachricht S_PL_VerfahrwegSenden angefordert. Die Antwort N_STP_Verfahrweg enthält den Weg zum Startpunkt des Einparkvorgangs.

Mit der Nachricht S_PL_ParklueckenDatenSenden wird die Tiefe der Parklücke angefordert. Nach Erhalt ist die Phase der Parklückensuche abgeschlossen, und der Einparkvorgang kann beginnen. Dazu wechselt *Steuer_Parkassistent* erst in einen Pufferzustand Z_STP_WARTE_AUF_GAS, der dafür sorgt, dass der Zustandsübergang in den Einparkvorgang erst durch das Betätigen des Gaspedals geschieht.

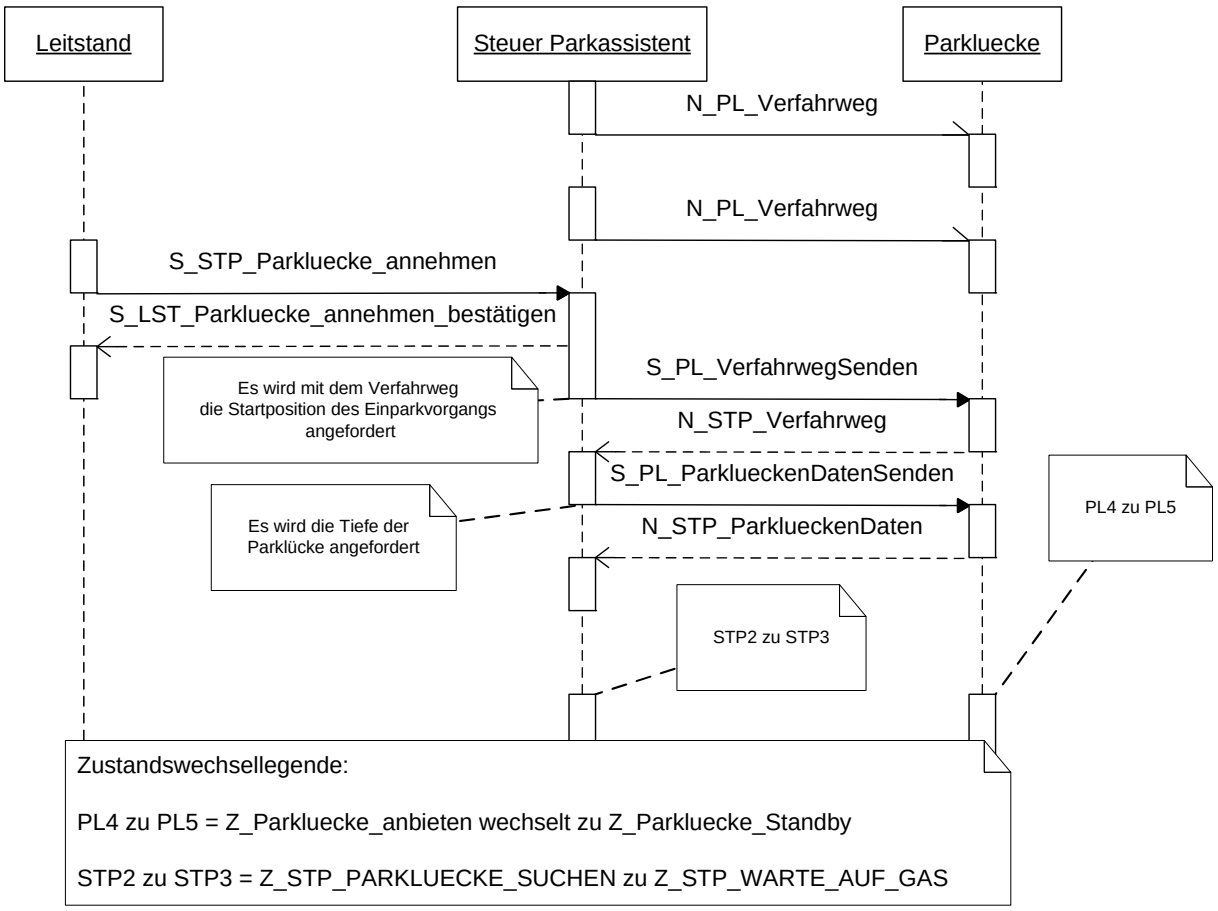

Abb. 39: Sequenzdiagramm zum Abschluss der Parklückensuche und der Speicherung des Verfahrwegs

5.2.2. Einparkvorgang

Nachdem die Parklückensuche abgeschlossen ist, folgt der Einparkvorgang mit den Automaten *Steuer_Parkassistent*, *Laserscanner* und *Einparken*. Zunächst muss der Automat *Einparken* jedoch, das SCV zur Startposition bewegen. Die benötigten Werte der Abstände zur Wand, sowie der Verfahrweg bis zur Startposition werden in der Phase der Parklückensuche gespeichert und vom *Steuer_Parkassistent* bereitgestellt.

Positionierung des SCVs zum Startpunkt des Einparkvorgangs
Mit dem Versand der Nachricht S_CS_Parkmode wird *Comm_Server* mitgeteilt, dass er die Steuerungsbefehle des Lenkrades und des Pedals vom Leitstand nicht mehr in das CAN-Format umwandeln soll. Die Soll-Vorgaben für Geschwindigkeit und Lenkwinkel werden vom *Steuer_Parkassistent* übernommen. Damit die Fahrt des SCVs zur Startposition beginnen kann, übermittelt *Steuer_Parkassistent* den noch verbleibenden Weg an den Automaten *Einparken* mit N_EP_Verfahrweg. Der Empfang erzeugt einen Zustandswechsel von Z_EP_IDLE zu Z_EP_X_POS_ERREICHEN.

Beim Eintritt wird mit dem *Laserscanner* ein zweites Mal überprüft, ob die Parklücke die benötigte Tiefe besitzt, und dann wird ein Zustandwechsel in den Einparkvorgang ausgeführt. Damit der Verfahrweg während des Zustandswechsels erhalten bleibt, schickt der Automat die N_EP_Verfahrweg-Nachricht an sich selbst. Mit dem Empfang der Nachricht im neuen Zustand,

beginnt die Fahrt zur Startposition des Einparkvorgangs. Zuerst wird anhand des vorzeichenbehafteten Verfahrwegs ausgewertet, ob sich die Startposition vor oder hinter dem SCV befindet. Dann wird in Abhängigkeit der Startposition mit der Nachricht N_STP_set_V die Geschwindigkeit des SCVs bei einem Lenkwinkel von 0° eingestellt. Nach dem Start der Bewegung des SCVs wird auf Seiten des Automaten *Einparken* in zyklischen Abständen von T_{LS} mit der Methode vermesseWeg() überprüft, ob die Startposition erreicht wurde.

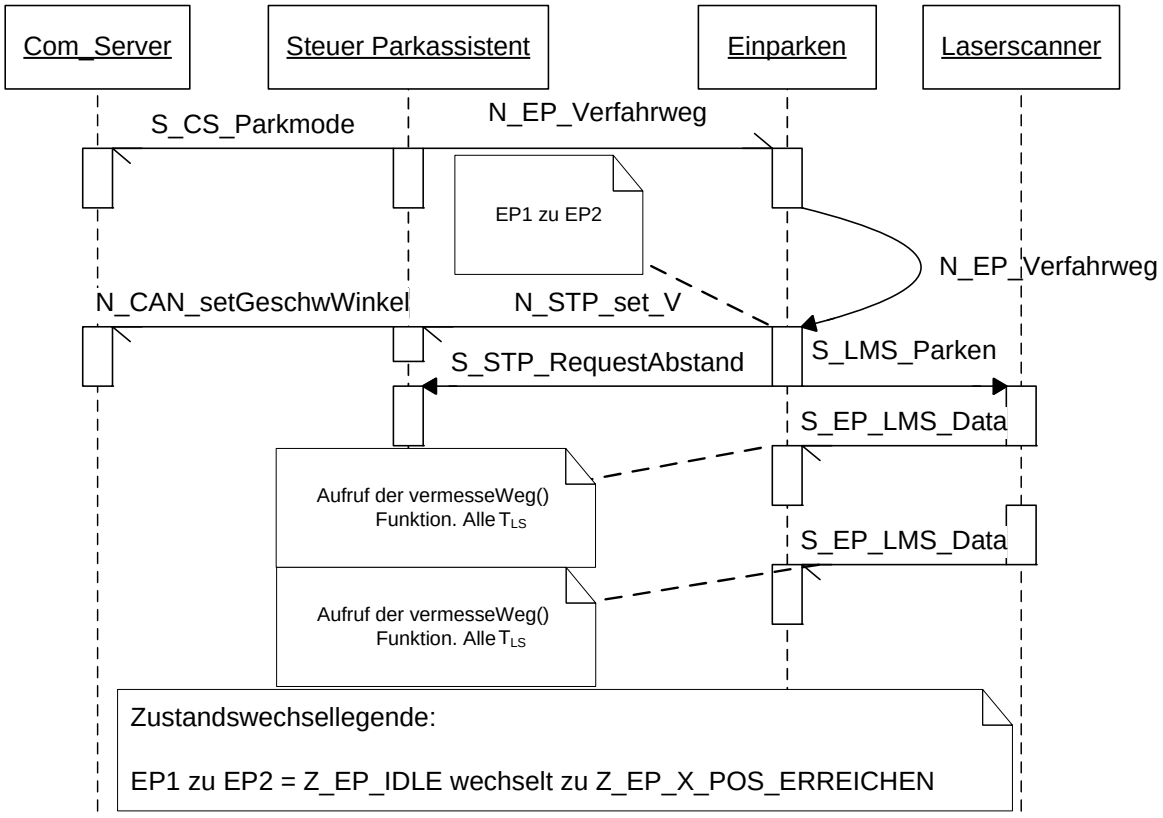

Abb. 40: Sequenzdiagramm zur Fahrt des SCVs zur Startposition mit dem Empfang des Verfahrwegs und der Abstandswerte.

Übergang von X-Position erreichen zur Fahrt in die Parklücke

Wenn sich das SCV in der Startposition befindet, wird dies mit der Nachricht S_EP_Xposerreicht signalisiert. Die Abstandsregelung kann durchgeführt werden, sobald die Anforderungsnachricht S_STP_Request_Abstand mit der Tiefe der Parklücke beantwortet wird. Anhand des Startabstands zur Wand, wird die Odometrie initialisiert, um dann mit der Nachricht N_STP_set_V und dem Argument -500 mm/s den Einparkvorgang zu starten.

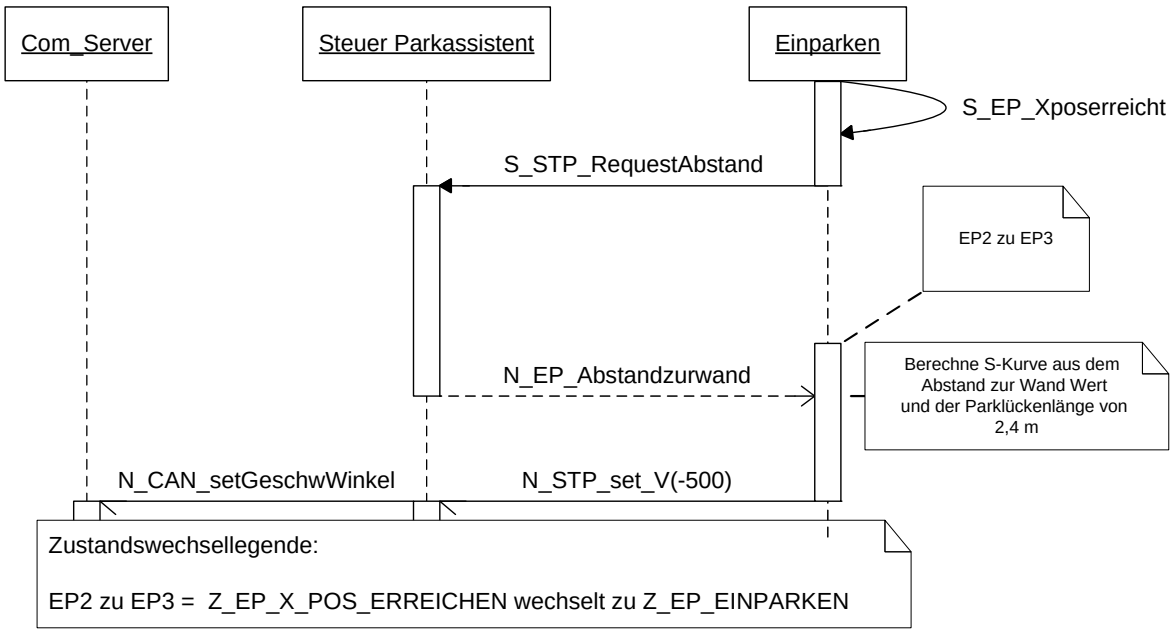

Abb. 41: Sequenzdiagramm zur Berechnung der Trajektorie unter Berücksichtigung der Tiefe der Parklücke. Es sind die Automaten *Steuer_Parkassistent*, *Einparken* und *Comm_Server* beteiligt.

Einparkvorgang mit Abstandregelung

Die Abstandwerte vom Laserscanner werden in zyklischen Abständen von T_{LS} gesendet und der Automat *Einparken* speichert sie. Die Odometrie wird in Abständen von T_{CAN} mit dem Ist-Lenkwinkel aktualisiert und ruft die zuletzt gespeicherten Abstände ab, dabei entscheidet die Auswertungslogik, ob es sich um die Wand oder ein anderes Hindernis erfasst wird. Der durch den Regler berechnete Lenkwinkel wird an *Steuer_Parkassistent* mit der Nachricht N_STP_set_W gesendet. Sobald das SCV sich parallel zur Wand befindet, muss es sich in der Parklücke durch eine Vorwärtsfahrt mittig ausrichten. Dazu wird das Fahrzeug gestoppt, der Lenkwinkel auf 0° gesetzt und die Geschwindigkeit wieder auf 500 mm/s eingestellt.

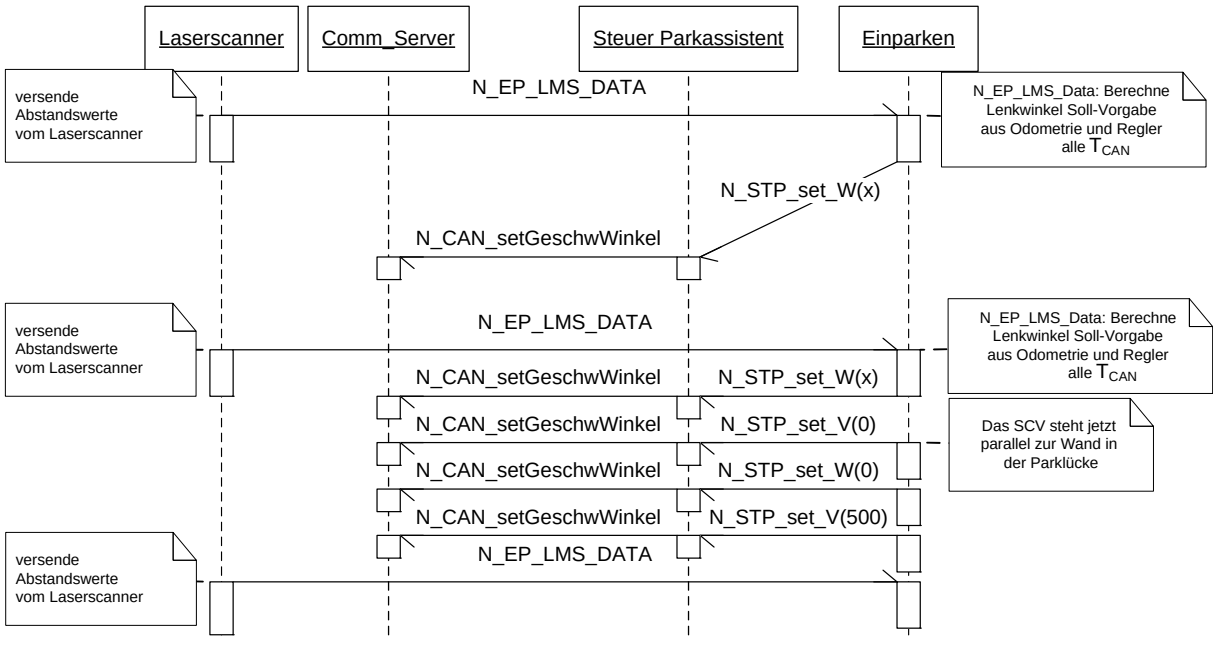

Abb. 42: Sequenzdiagramm des Einparkvorgangs. Berechnung der Odometrie und des Abstandsreglers mit der Vorgabe des Lenkwinkels.

Abschluss des Einparkvorgangs

Mit der Nachricht S_EP_Endposerreicht signalisiert der Automat *Einparken* sich selbst, dass der Einparkvorgang abgeschlossen ist und die Endposition erreicht wurde, dies wird *Steuer_Parkassistent* signalisiert. Als Folge signalisiert *Steuer_Parkassistent Leitstand* mit S_LST_Parkvorgang_beendet, dass der Parkvorgang abgeschlossen wurde, und wechselt in den Z_STP_WARTE_AUF_GAS_NULL Pufferzustand. Parallel erlischt auf der graphischen Oberfläche des Leitstands die LED Lampe *Parkvorgang*. Nun muss der Fahrer das Gaspedal loslassen. In zyklischen Abständen von T_{CAN} wird überprüft, ob das Gaspedal den Schwellenwert von 5 mm/s mit der Nachricht S_STP_V_und_W_ von_LST unterschritten wurde. Erst mit dem Unterschreiten dieses Schwellwerts wechselt *Steuer_Parkassistent* in den Startzustand und der Kommunikationsrechner übersetzt wieder die Befehle vom *Leitstand* auf dem CAN.

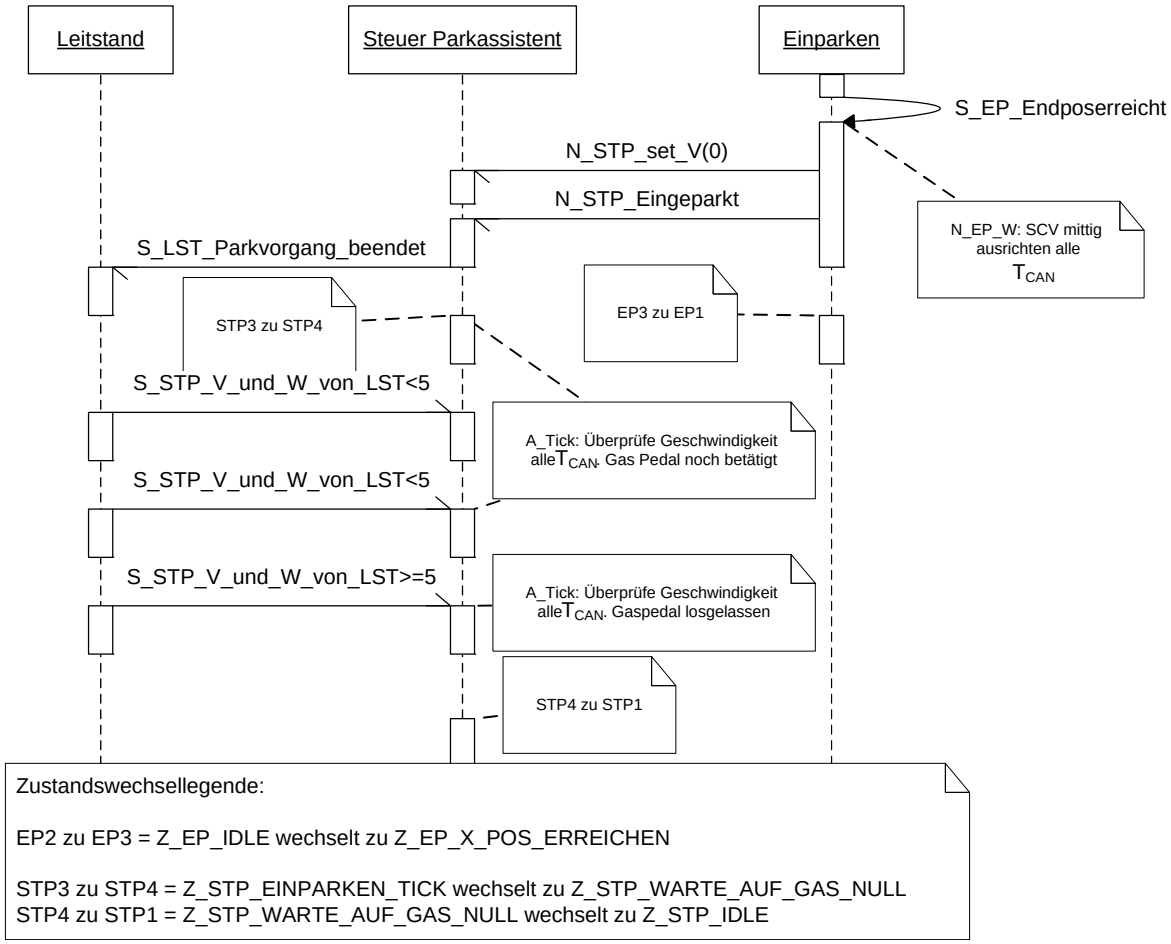

Abb. 43: Sequenzdiagramm zum Abschluss des Parkvorgangs.

6. Aufbau des Einparkassistenten mit dem POSIX-Thread Framework und der AIRA-Laufzeitumgebung

Der in dieser Masterarbeit vorgestellte Einparkassistent wird auf dem Kommunikationsrechner unter dem Linux Betriebssystem ausgeführt. Dieses Kapitel stellt den Aufbau des Einparkassistenten dar, vgl. Abb. 44 und beschreibt die Funktionen, die von Linux zur Verfügung gestellt werden, die POSIX-Threads und die AIRA-Laufzeitumgebung.

Der Hauptgrund für den Einsatz des Einparkassistenten auf dem Kommunikationsrechner, ist die Reduzierung der CAN-Buslast.

- Da der Kommunikationsrechner die Steuerungsbefehle vom Leitstand über WLAN- in das CAN-Format umwandelt und diese im Einparkvorgang vom Einparkassistenten übernommen wird, können die Nachrichten gefiltert werden. Dadurch sinkt die Buslast auf dem CAN-Bus.
- Wäre der Laserscanner an den Koordinierungsrechner angeschlossen, müssten die Lasermesswerte auf dem CAN-Bus übertragen werden. Eine CAN-Nachricht kann maximal 8 Bytes übertragen, zusätzlich müssen die 8 Bytes um Arbitrierungs-, Daten- und Steuerungsfelder erweitert werden. Der Lasermesswert für 1° ist 2 Bytes groß.
- Linux ist weitgehend POSIX konform, so dass für die Unterstützung des Multithreadings auf Bibliotheken zugegriffen werden kann.
- Es wurde im Internet ein Treiber für den LMS200 [30] für Linux recherchiert, welche die Basis für die Entwicklung des Treibers bildet.
- Zu Beginn basierte der SCV auf eine FlexRay Lösung, und durch eine Optimierung wurde der Koordinierungsrechner nicht benötigt.

Mit der Modellierung in der Harel-Automaten-Notation wird die Semantik des Einparkassistenten nach dem Auftreten von Ereignissen beschrieben. Bei dem Einparkassistenten handelt es sich um ein reaktives System [11]. Die vier Merkmale, die ein reaktives System beschreiben, sind im Einparkassistenten erfüllt. Sie beinhaltet Nebenläufigkeit, Einhaltung von Zeitbedingungen und Determinismus. Da der Einparkassistent auf eingehende Ereignisse Seitens des Benutzers und des Einparkassistenten reagiert, dienen hierarchische Automaten, als adäquate Beschreibung des Verhaltens, eines reaktiven Systems. Für eine detaillierte Modellierung des Einparkassistenten in der Harel-Automaten-Notation wird auf Kapitel 5 verwiesen.

Bei der Implementierung der Harel-Automaten wird die AIRA-Laufzeitumgebung [18] verwendet. Sie stellt Folgendes zur Verfügung:

- Datenstrukturen zur Definition der Automaten, Zustände und Reaktionen
- Suche und Ausführung der Zustandsreaktionen unter Auswertung der empfangenen Nachrichten
- Das Verknüpfen von min-,max- und tick-Timern an Zuständen für die Implementierung von temporaler Logik

Die AIRA-Laufzeitumgebung bietet Datenstrukturen, mit denen Reaktionen an Zustände gekoppelt werden können. Bei den hierarchischen Automaten ist die Reihenfolge der Reaktionen festgelegt und ein Reaktionsalgorithmus sucht nach der Reaktion, die auf ein auftretendes Ereignis reagiert. Mit diesen Datenstrukturen können Zustände und weitere Subautomaten definiert werden. Anhand

der definierten Reaktionen, die mit den Zuständen verknüpft werden, kann eine Reaktion den Zustand erhalten oder verändern. Optional kann zusätzlich eine Reaktion vor Eintritt und/oder nach dem Austritt aus einem Zustand definiert werden. Die Art der Reaktion wird durch die Auswertung der Nachrichten in der Queue der jeweiligen Automaten bestimmt, diese ist eine Linked List und funktioniert nach dem FIFO-Prinzip. Der Adressbereich der Nachrichtenqueues in den Threads wird geteilt, damit können die Automaten untereinander Nachrichten austauschen und jeder Automat kann über einen Zeiger in die Queues der anderen Automaten schreiben.

Des Weiteren können in der AIRA-Laufzeitumgebung Zustände an Timern gebunden werden. Mit dem max-Timer kann festgelegt werden, wie viele Millisekunden ein Automat sich maximal in einem Zustand befinden darf, und mit dem min-Timer, wie viele Millisekunden in einem Zustand verblieben werden muss, bevor ein Zustandswechsel erfolgen kann. Der tick-Timer wird benötigt, um das Auslesen der Bytes in zyklischen Abständen die Daten vom Laserscanner an zu triggern.

Die POSIX-API unter Linux bietet:

- die Möglichkeit Automaten, die logisch unabhängig voneinander sind, in Threads aufzuteilen.
- die Einstellung des Scheduler-Verfahrens und die Parametrisierung der Priorität, um I/O gebundene Automaten bei der Zuteilung des Prozessor den CPU intensiven Threads zu bevorzugen.
- Synchronisationsmechanismen, wie Mutexe und Bedingungsvariablen für den gegenseitigen Ausschluss des Zugriffs auf die Nachrichtenqueues der AIRA-Laufzeitumgebung und die Überführung der Automaten in einen blockierten Zustand, wenn die Queue leer ist, bzw. das Aufwecken wenn sie gefüllt ist.

POSIX dient als Schnittstelle zwischen der AIRA-Laufzeitumgebung und dem Betriebssystem. Sie wurde von der IEEE standardisiert mit dem Ziel, den Code portabler zu gestalten. Mit den Pthreads, spezifiziert als IEEE 1003.1c, werden Threads verwaltet, synchronisiert und gesteuert. Die einzelnen Funktionalitäten des Einparkassistenten, wie die Messwerterfassung oder Berechnung der Trajektorie für den Einparkvorgang, werden in Automaten gekapselt. Da sie logisch voneinander entkoppelt sind, werden sie aufgeteilt und als einzelne Pthreads ausgeführt. Für eine zeitnahe Verarbeitung der Messwerte muss der Laserscannerautomat vom Scheduler bevorzugt werden, dies wird durch die Parametrisierung der Priorität im Scheduler und die Wahl des Scheduler-Verfahrens erreicht. In Abschnitt 6.2.2 wird auf die Wahl der Prioritäten, das Scheduling-Verfahren und die damit verbundenen Wechselwirkungen eingegangen. Als Scheduler-Verfahren stehen zu Verfügung, FIFO, Round Robin und das Linux Scheduling-Verfahren [5]. Für den gegenseitigen Ausschluss bei der Interthread-Kommunikation mit den Nachrichtenqueues, werden in der Pthread API auf Synchronisationsmechanismen wie Mutexe und Bedingungsvariablen verwendet.

Für die Entwicklung des Einparkassistenten bietet das Betriebssystem Linux:

- eine POSIX-API für die nebenläufige Ausführung von Threads
- einen File Deskriptor zum Zwischenspeichern der Lasermesswert-Daten von der RS422 Schnittstelle
- die nanosleep() Funktion, um die Threads für eine Zeitspanne in einen blockierten Zustand zu versetzen.

Linux ist weitgehend POSIX konform und bietet eine API für Pthread. Für die min-,max- und tick-Timer ruft die AIRA-Laufzeitumgebung, die von Linux zur Verfügung gestellte Bibliotheksfunktion

nanosleep() auf. Mit dieser Funktion wird ein Thread in der Ausführung unterbrochen und fortgesetzt, wenn 1 ms abgelaufen ist. Bei der Übertragung der Messwerte vom Laserscanner durch die RS422 Schnittstelle werden die Daten in einem File Deskriptor zwischengespeichert. Mit einem blockierenden read()-Funktionsaufruf, der ebenfalls aus der Klassenbibliothek von Linux stammt, werden diese aus dem Puffer des File Deskriptors gelesen werden. (vgl. Abb. 44)

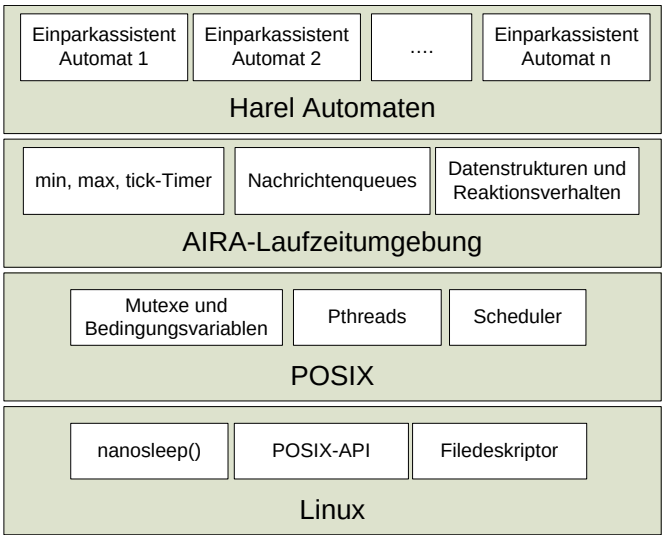

Abb. 44: Softwarearchitektur des Einparkassistenten.

6.1. AIRA-Laufzeitumgebung

Die AIRA-Laufzeitumgebung stellt dem Programmierer ein Werkzeug zur Verfügung, das es ihm erlaubt, das Design der Software mit dem Harel-Automaten zu modellieren. Mit Datenstrukturen für Automaten und Zustände, können Reaktionsmethoden definiert werden, die mit den Zuständen verknüpft sind. Es können Subautomaten in die Zustände eingebettet werden und damit Automaten beliebig tief verschachtelt werden. Durch das Design des Einparkassistenten werden keine benötigt Subautomaten benötigt.

Die AIRA-Laufzeitumgebung bestimmt auf ein auftretendes Ereignis hin, mit einem Reaktionsalgorithmus, die passenden Reaktionen, indem sie rekursiv durch die Automaten auf der Suche nach einer passenden Reaktionsmethode iteriert. Dabei lässt der Algorithmus den Zustand in der tiefsten Hierarchiestufe reagieren, wobei zustandserhaltende- vor zustandsverändernde-Reaktionen ausgeführt werden.

Eine Reaktion erfolgt nach der Auswertung der asynchronen Nachricht durch die Reaktionsmethode, die entweder aus der Nachrichtenqueue entnommen oder direkt von einem Timer gesendet wird. Die Nachrichtenqueue ist eine Linked List mit einem Zeiger auf ein typenloses Element, einem Zeiger auf das nächste Element, und funktioniert nach dem FIFO-Prinzip.

Mit den Timern in der AIRA-Laufzeitumgebung können temporale Logiken an Zustände gekoppelt werden. Mit den min, max und tick-Timern können Verweilzeiten in Zuständen festgelegt oder zyklische Signale generiert werden. Um das Pollen der Messwertdaten vom vertikalen Laserscanner anzustoßen, wird letzteres benötigt.

6.1.1. Datenstrukturen für die Automaten und Zustände

Die oberste Datenstruktur in der AIRA-Laufzeitumgebung ist die T_aktivitaet, sie beschreibt einen Automaten mit einer Liste von Zuständen und enthält einen Index auf den aktiven Zustand.

```
struct T_aktivitaet{
  int      akt_zustand;
  T_List   *zustaende;}
```

Listing 1: Datenstruktur für einen Automaten mit index auf dem aktuellen Zustand und einer Liste von Zuständen

Das Reaktionsverhalten der Zustände wird mit einem Zeiger auf eine Reaktionsmethode festgelegt. In der Datenstruktur des Zustands gibt es drei Platzhalter, die das Reaktionsverhalten beschreiben, siehe Listing 2:

- zustandserhaltende, interne Reaktion
- zustandsverändernde, externe Reaktion
- Reaktionen die nach der Beendigung eines Subautomaten ausgeführt wird, lambda Reaktion

Die Platzhalter haben nur Einfluss auf die Reihenfolge des Reaktionsverhaltens. Bei einem erwünschten Zustandsübergang muss der Programmierer in der externen Reaktion die Übergänge implizit mit einer Methode aufrufen. Der Wechsel der Zustände geschieht in der AIRA-Laufzeitumgebung nicht automatisch, dagegen werden von der Laufzeitumgebung die entry- und exit-Aktionen beim Zustandsübergang automatisch aufgerufen.

```
struct T_zustand{
  T_zustandsart    zustandsart;
  T_List                *aktivitaeten;
  FT_aktion        *entry_aktion;
  FT_aktion        *exit_aktion;
  FT_reaktionen    *int_reaktionen;
  FT_reaktionen    *ext_reaktionen;
  FT_reaktionen    *lambda_reaktionen;
  T_Timer               *max_timer;
  T_Timer               *min_timer;
  T_Timer          *tick_timer;}
```

Listing 2: Datenstruktur für einen Zustand, mit Platzhaltern für weitere optionale Subautomaten, interne Reaktionen, externe Reaktionen, entry- exit- Aktionen und den min-, max- und tick-Timer

6.1.2. AIRA-Timer

Die min-,max- und tick-Timer werden in einer Timer Liste verwaltet. Die Funktionen timer_tick() und timer_execute() verwenden diese Liste, um den Zähler zu dekrementieren oder eine Reaktion nach Ablauf des Timers zu schalten. Sobald ein max- oder tick-Timer mit der eingestellten Zeit abgelaufen ist, wird eine Timeout-Nachricht an den gekoppelten Zustand verschickt. Die Timeout-Nachricht wird direkt von der Reaktionsmethode des Zustandes ausgeführt und wird nicht in die Nachrichtenqueue des Automaten eingetragen. Beim Eintritt in den Zustand werden die an den Zustand

gekoppelten Timer in die Timer-Liste eingetragen, von der Laufzeitumgebung initialisiert und gestartet. Beim Verlassen eines Zustandes werden die Timer wieder aus der Timer-Liste entfernt. In der Timer-List können maximal drei Timer-Elemente vorhanden sein.

An einem Zustand können drei min-, max- und tick-Timer gekoppelt werden:

- Damit kann festgelegt werden, wie lange ein Automat sich mindestens in einem Zustand zu befinden hat, bevor ein Zustandswechsel erfolgen kann. (min-Timer)
- wie lange ein Automat sich maximal in einem Zustand befinden kann, bevor ein Timeout generiert wird und der Zustand verlassen wird. (max-Timer)
- Die Generierung einer tick-Nachricht in periodischen Abständen, die durch den Zustand selbst empfangen wird. (tick-Timer)

Im Falle des vertikalen Laserscanners werden zyklisch T_{LS} Nachrichtenpakete mit den gemessenen Abständen zur Wand geliefert. Der tick-Timer des Automaten *Laserscanner* wird so eingestellt, dass er alle 6 ms die 14 read()-Funktionsaufrufe startet.

Die Datenstruktur in Listing 3 enthält die Periode, den aktuellen Zähler des Timers in Millisekunden und seinen Typ (min, max oder tick). Damit die Timer eindeutig den Automaten und Zuständen zugeordnet werden können, sind in der Datenstruktur des Timers zwei Platzhalter dafür vorgesehen. Anhand der Zuordnung kann die Timeout-Nachricht an den jeweils gekoppelten Zustand verschickt werden. Beim min-Timer wird nach Ablauf des Timers keine implizite Timeout-Nachricht generiert, sondern timer_expired auf *wahr* gesetzt. Anhand dieses Flags kann der Reaktionsalgorithmus feststellen, ob der min-Timer abgelaufen ist und ein Zustandswechsel erfolgen darf.

```
struct T_Timer{
int              count;        /*Momentanwert des Zaehlers*/
int              start_count;/*Anfangswert des Zaehlers*/
T_Timertype      type;         /*min, max oder tick*/
T_aktivitaet*    aktivitaet;  /*Aktivitaet die auf timeout reagiert*/
T_zustand*       zustand;      /*Zustand der auf timeout reagiert*/
REF_T_Nachricht  timeout_nachricht;
int              timer_expired;}   /*set, when timer is expired*/
```

Listing 3: Datenstruktur für einen Timer, mit Platzhaltern für Zählervariablen, Typ, Zuordnung zur Aktivität, Zustand , Timeout-Nachricht, die generiert werden soll, und Flag für den Ablauf des Timers

6.1.3. Reaktionsalgorithmus der AIRA-Laufzeitumgebung

Die Automaten werden in einer Dauerschleife ausgeführt, welche aus zwei Funktionsaufrufen bestehen. Zunächst wird in der Nachrichtenqueue mit einer Bedingungsvariablen blockierend auf dem Empfang einer Nachricht gewartet, um anschließend diese Nachricht mit dem Reaktionsalgorithmus zu verbrauchen. Das Warten auf Nachrichten versetzt die Automaten in einen blockierten Zustand, welcher durch den Empfang einer Nachricht wieder aufgehoben wird.

Der Algorithmus sucht immer nach einer Reaktion in der untersten Automatenebene. Eine Reaktion erfolgt nach dem Aufruf der aktivitaet_reagieren() Methode, anhand des Rückgabewerts wird festgestellt, ob ein Zustand reagiert hat. Sie arbeitet nach folgendem Schema. (vgl. Abb. 45):

- 1. Suche nach Sub-Automaten im aktuellen Zustand, falls Sub-Automat vorhanden rekursiver Aufruf der aktivität_reagieren() Methode für den Sub-Automaten. Falls kein Sub-Automat vorhanden ist oder wenn der Sub-Automat nicht reagiert hat, mit Schritt 2 fortfahren.
- 2. Auswertung der zustandserhaltenden Reaktionsmethode, ob eine interne Reaktion stattgefunden hat. Wenn es eine gegeben hat, wird diese ausgeführt, die Aktivität hat die Nachricht verbraucht, der Algorithmus beendet sich und gibt einen entsprechenden Wert zurück. Gibt es keine Reaktion, wird mit Schritt 3 fortgefahren.
- 3. Auswertung der zustandsverändernden Reaktionsmethode, ob eine externe Reaktion stattgefunden hat. Im Fall einer Reaktion wird die exit-Aktion des aktuellen Zustands und die entry-Aktion des neuen Zustands ausgeführt. Der Reaktionsalgorithmus für die aktuelle Ebene ist beendet, wenn die Aktivität nicht reagiert hat. Dann wird in der nächsthöheren Ebene mit Schritt 1. das Schema wiederholt.

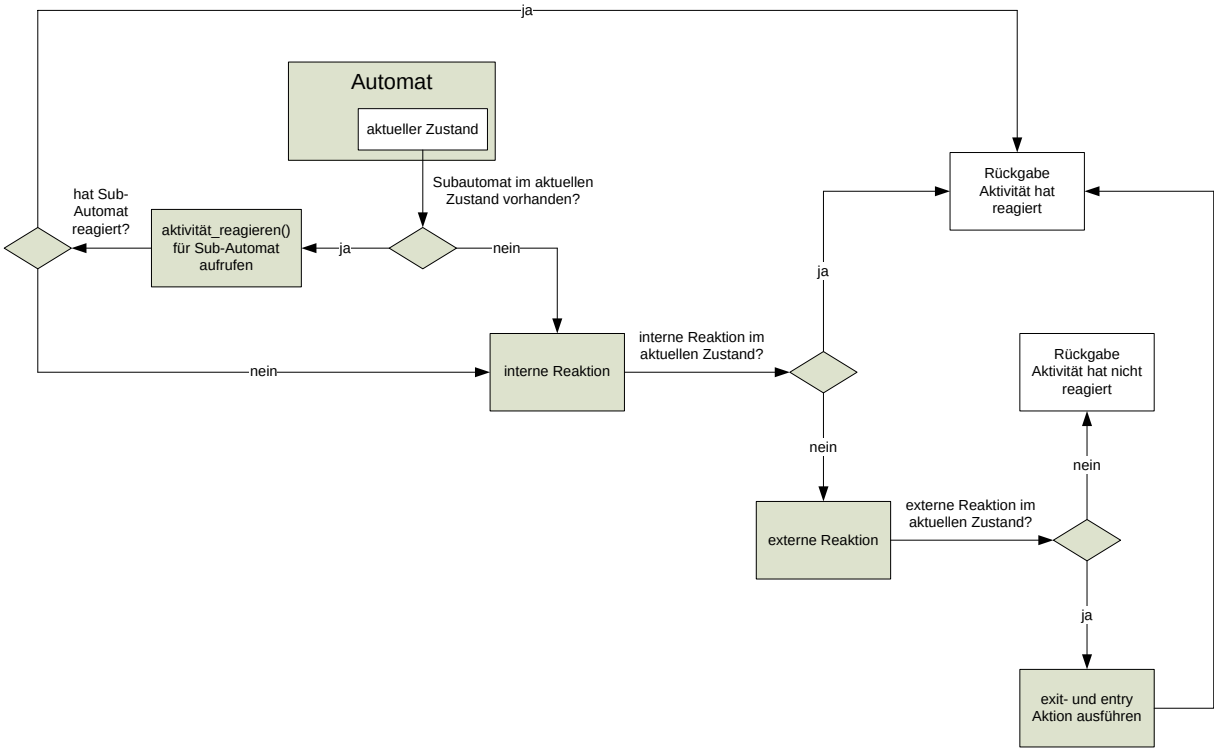

Abb. 45: Reihenfolge der Suche nach einer Reaktion mit aktivität_reagiern(), nach Empfang einer Nachricht

Die Reaktion wird durch den Typ der empfangenen Nachricht festgelegt. In der AIRA-Laufzeitumgebung wird bei der Signatur der Reaktionsmethode, die Nachricht als Zeiger übergeben, damit ist es möglich eigene Nachrichtentypen zu definieren. Die Auswertung des Nachrichtentyps erfolgt über eine switch-, case-Anweisung, dort kann für jeden Nachrichtentyp eine Reaktion definiert werden. Der Rückgabewert der Methode sagt aus, ob eine Reaktion stattgefunden hat.

6.2. Implementierung der AIRA-Automaten als Pthreads

Die AIRA-Automaten im Einparkassistenten werden als eigenständige Pthreads gestartet. In diesem Kapitel werden die Gründe für die Verwendung der Pthreads aufgezählt. [5].

Es wird die Einstellung der Prioritäten und die Wahl der Scheduling-Policy des Laserscanners Automaten dokumentiert, denn der I/O gebundene Automat $Laserscanner$ ist zeitkritisch und muss gegenüber den anderen CPU intensiven Automaten bevorzugt werden. Die Pthreads können in Hinsicht auf Prioritäten und der Auswahl des Scheduling-Verfahrens parametrisiert werden, so dass andere Pthreads mit einer geringeren Priorität von der Ausführung der CPU verdrängt werden. Es werden die drei einstellbaren Scheduling-Verfahren in ihrer Ausfürung erläutert.

Der Automat für den Laserscanner liest die Datenpakete über die RS422 Schnittstelle byteweise aus und benötigt einen Timer, der das Auslesen der Messwerte in Abständen von T_{LS} triggert. Aus diesem Grund wird die Implementierung der Timer in der AIRA-Laufzeitumgebung beschrieben. Ein Automat der mindestens einen Timer (min., max. oder tick) verwendet, startet zwei weitere Pthreads, die den Zähler dekrementieren und den Reaktionsalgorithmus starten, wenn der Zähler abgelaufen ist.

Werden keine Nachrichten ausgetauscht, blockieren die Automaten und es stehen in der Zwischenzeit die CPU für andere Pthreads zur Verfügung. Die Einstellung der Priorität und des Scheduling-Verfahrens zeigt keine signifikante Änderung des Systemverhaltens vom Einparkassistenten, bei dem 650 MHz Rechner. Des Weiteren wird die Auswirkung des AIRA-Timers auf die Ausführung der Pthreads erläutert.

6.2.1. Softwaremodellierung mit Pthreads

Für die Verwendung von Threads sollten bestimmte Eigenschaften der Software erfüllt sein [5]:

- 1. Die Ausführung der Threads muss unabhängig voneinander sein
- 2. Threads reagieren auf asynchrone Nachrichten
- 3. Threads verbringen eine längere Zeit im blockierten Zustand
- 4. Threads müssen anderen Gegenüber bevorzugt werden

Die AIRA-Automaten sind in der Ausführung zu einem großen Teil unabhängig voneinander. Insgesamt werden im Einparkassistenten 5 AIRA-Automaten verwendet, die jeweils unterschiedliche Funktionalitäten erfüllen. Bis auf die Synchronisation durch den Austausch der Nachrichten, können die Automaten unabhängig voneinander ausgeführt werden.

Der Austausch der Nachrichten erfolgt bei den AIRA-Automaten asynchron. Es können a priori keine Aussagen darüber gemacht werden, wann die Parklücke gefunden, das Gaspedal betätigt, losgelassen oder der Einparkassistent gestartet werden soll. Beim Auftreten eines Ereignisses versenden und empfangen die Automaten Nachrichten in die jeweiligen Queues und erzeugen dadurch ggf. Zustandsübergänge. Die Ausnahme bildet hier der Automat für den Laserscanner, dessen Messwerte alle T_{LS} bereitstehen und die Vorgabe des Soll-Lenkwinkels in periodischen Abständen von T_{CAN} zur Steuerung des Einparkassistenten während der Fahrt in die Parklücke.

Der Automat für den Empfang der Lasermesswerte muss den anderen Automaten gegenüber bevorzugt werden. Falls ein anderer Automat von der CPU ausgeführt wird und ein Byte auf der RS422 Schnittstelle bereitsteht, muss der aktuelle Automat der die CPU beansprucht, durch den Automat $Laserscanner$ verdrängt werden. Dieses Verhalten wird bei den Pthreads durch die Parametrisierung der Priorität und der Scheduling-policy erreicht.

6.2.2. Priorität und die Scheduling-policy der Pthreads

Damit möglichst in einer kurzen Latenzzeit die Lasermesswerte verarbeitet werden können, muss der Automat für den Laserscanner, die anderen Automaten verdrängen. Dabei können drei Scheduling-policys eingestellt werden:

- FIFO(SCHED_FIFO) Der Automat mit dieser Scheduling-policy wird so lange ausgeführt, bis er blockiert, sich beendet oder von einem anderen Pthread mit höherer Priorität verdrängt wird. Tritt dieser Fall auf, wird er ans Ende der Warteschlange eingetragen. Dabei gibt es so viele Warteschlangen, wie es eingestellte Prioritäten gibt, d.h. Pthreads mit der Priorität 10 werden auch in eine Warteschlange mit der Priorität 10 eingetragen. Der Scheduler teilt dann immer den Thread mit der höchsten Priorität den Prozessor zu.
- Round Robin(SCHED_RR) Ein Pthread verfügt über eine Zeitscheibe für die Nutzung der CPU. Wird diese Zeitscheibe verbraucht, wird er ans Ende der Warteschlange eingetragen und der Nächste in der Warteschlange ausgeführt.
- Linux-Scheduler(SCHED_OTHER) Diese Scheduling-policy weist den Pthreads ebenfalls Zeitscheiben zu. Allerdings ändert der Scheduler die Priorität der Pthreads dynamisch zur Laufzeit, indem er den Zeitscheibenverbrauch auswertet. Pthreads, welche die Zeitscheibe komplett verbrauchen, werden als CPU intensiv vom Scheduler kategorisiert. Verbraucht ein Pthread seine Zeitscheibe nicht komplett und wechselt vor Ablauf in den blockierten Zustand, wird er als I/O gebunden klassifiziert. [5] [8] I/O gebundene Threads erhalten eine höhere Priorität als CPU intensive.

Die SCHED_FIFO und SCHED_RR sind Real-Time Scheduling-Verfahren und werden vom Scheduler dem letzten gegenüber bevorzugt [16].

Die Werte für die Einstellung der Priorität reichen von 1 bis 99, wobei 99 die höchste Priorität repräsentiert. Der Automat *Laserscanner* Pthread wird mit einer Priorität von 20, die restlichen Automaten mit 1, und dem FIFO Scheduling-Verfahren parmetrisiert. Damit ist gewährleistet, dass *Laserscanner* die restlichen Automaten von der Ausführung verdrängen kann.

6.2.3. Verwendung von Mutexe und Bedingungsvariablen für die Nachrichtenqueue

Eine Reaktion im AIRA-Automaten wird durch den Empfang einer asynchronen Nachricht ausgelöst, dafür hat jeder AIRA-Automat eine eigene Nachrichtenqueue, welche nach dem FIFO-Prinzip arbeitet. Der Zugriff auf die Nachrichtenqueue erfolgt über einen Zeiger auf die Linked List. Da theoretisch zwei Automaten gleichzeitig in eine Nachrichtenqueue schreiben können, muss diese mit einem Mutex geschützt werden, welche von der Pthread API zur Verfügung gestellt werden.

Damit der Automat die Nachrichtenqueue nicht immer pollen muss, wird zusätzlich eine Bedingungsvariable für das Schreiben und Lesen eingeführt. Enthält die Nachrichtenqueue keine Nachricht, sorgt die Bedingungsvariable dafür, dass der Pthread blockiert wird. Dabei wird der Mutex, der die Nachrichtenqueue schützt, wieder freigegeben, damit die anderen Automaten in die Queue schreiben können. Andernfalls würde ein Deadlock auftreten.

```
void *msg_dequeue (const T_MsgQueue * queue){
void* p;
pthread_mutex_lock(queue->msg_mutex);     //Sperre der Queue mit der Mutex
if(queue->li->size == 0){                 //Falls die Queue Leer ist
```

```
        pthread_cond_wait(queue->msg_condvar,queue->msg_mutex);
}                                   //Warte auf die Bedingungsvariable
//und gebe den Mutex wieder frei
p = list_dequeue(queue->li);                //Verbrauche die Nachricht
pthread_mutex_unlock(queue->msg_mutex); //Gebe den Mutex wieder frei
return p; }                 //Rückgabe Zeiger auf Element
```

Listing 4: Methode zum Auslesen einer Nachricht aus der Nachrichtenqueue.

Wird eine Nachricht in die Queue geschrieben, wird die Bedingungsvariable gesetzt und der Nachrichtenqueue signalisiert, dass die Queue gefüllt ist, und der Automat wird aufgeweckt.

6.2.4. Implementierung der Timer mit den Pthreads

Benötigt ein Zustand einen min-,max- und tick-Timer, müssen parallel zu dem Automaten zwei weitere Pthreads erzeugt werden, welche die timer_tick() und timer_execute()-Funktionen ausführen. Beide Funktionen greifen in der Timer-Liste auf die Timer Datenstruktur, die im vorherigen Abschnitt 6.1.2 beschrieben wird.

- timer_tick(): Dekrementiert in periodischen Abständen von 1 ms die Zählervariable von jeden Timer in der Timer-Liste. Durch die Ausführung der Linux nanosleep() Funktion wird der Automat in einen blockierten Zustand versetzt und wird wieder behoben, wenn 1ms abgelaufen ist.
- timer_execute(): Überprüft in Abständen von 1ms die Zählervariable. Erreicht sie den Wert 0, wird eine Timeout-Nachricht an aktivität_reagieren() des gekoppelten Automaten geschickt.

Würde der tick-Thread auch die Reaktion ausführen, hätte das zur Folge, dass die nächsten Timer in der Liste um die Zeit verzögert werden, die der Reaktionsalgorithmus und die Reaktionsmethode benötigten. Aus diesem Grund werden beide Funktionen als unabhängige Pthreads gestartet.

6.2.5. Bidirektionale Umwandlung der WLAN-Steuerungsnachrichten und CAN SCV Fahrzeuginformationen auf dem Kommunikationsrechner

Bei dem CAN-Protokoll geschieht der Buszugriff nach dem Carrier-Sense-Multiple-Access / Collision-Avoidance-Verfahren. Bei einem gleichzeitigen Sendevorgang erhält die CAN-Nachricht mit der höheren Priorität den Zugriff auf dem Bus. Im Falle des Einparkassistenten bedeutet dies, dass die CAN- die WLAN-Nachrichten bei gleichzeitigem Zugriff verdrängen, da sie eine höhere Priorität besitzen. Trotzdem können Jitter vorkommen, da die Steuerungsbefehle vom Leitstand ereignisgesteuert auftreten und ein bereits erfolgter Sendevorgang nicht abgebrochen werden kann. Damit können die ereignisgesteuerten Steuerungsbefehle vom Leitstand, die zeitgesteuerten Ist-Werte von den ARM7 verzögern. Aus diesem Grund filtert der *Comm_Server* die Steuerungsbefehle vom Leitstand, während des Einparkvorgangs.

Die Fahrbefehle die vom Leitstand über WLAN durch den Fahrer generiert werden, müssen für die Regelung der Lenkungs- und Antriebsmotoren im ARM7TDMI μ-Controller in das CAN-Format umgewandelt werden. Die bidirektionale Umwandlung der WLAN- und CAN-Nachrichten wird im Kommunikationsrechner durch den *Comm_Server* Automaten mit zwei Threads durchgeführt. (vgl. Abb. 46) Diese nennen sich comm_server_thread und comm_server_dequeue_thread. Der comm_server_thread übernimmt die bidirektionale von Umwandlung CAN- und WLAN-

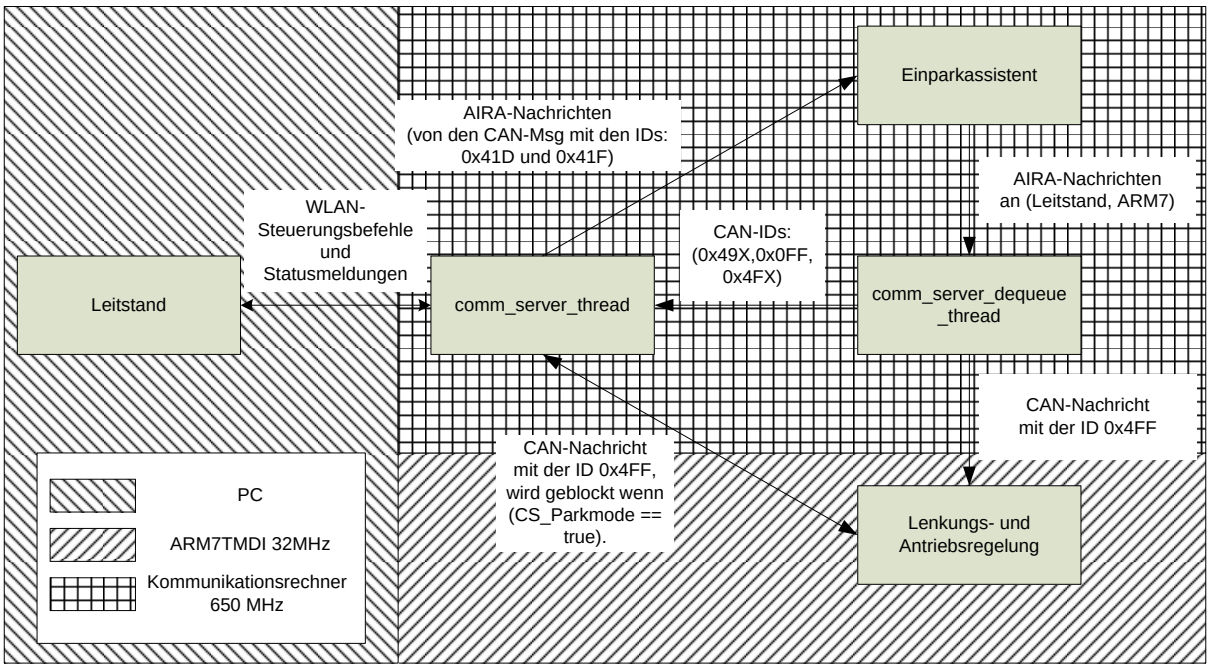

Abb. 46: Austausch der CAN- und WLAN-Nachrichten, Leitstand, μ-Controller und Kommunikationsrechner. Austausch der Nachrichten comm_server_thread, comm_server_dequeue_thread [14], comm_server_thread und Lenkungs- und Antriebsregelung [26]

Nachrichten. Er beinhaltet eine boolsche-Variable die vom comm_server_dequeue_thread über CAN gesetzt wird. Diese signalisiert, ob sich der Einparkassistent in der Einparkphase befindet. Ist dies der Fall, wird die Nachricht für die Vorgabe des Lenkwinkels und der Geschwindigkeit des SCVs vom Leitstand mit der CAN-ID 0x4FF nicht an den ARM7 weitergeleitet. Die Vorgabe beider Werte wird in der Phase vom Automat $Einparken$ des Einparkassistenten übernommen. Der comm_server_thread verarbeitet folgendes:

- Er wandelt die CAN-Nachrichten die vom comm_server_dequeue_thread und dem ARM7 empfangen werden, zu WLAN-Nachrichten um.
- Er wandelt die WLAN-Nachrichten des Leitstandes in das CAN-Format um, wobei die Nachricht mit der CAN-ID 0x4FF gefiltert wird, wenn sich der Einparkassistent im Einparkvorgang befindet.
- Er schreibt den Verfahrweg (CAN-ID 0x41D) und den Ist-Lenkwinkel (CAN-ID 0x41F) vom ARM7, die alle T_{CAN} empfangen werden, in die Nachrichten-Queues der Automaten $Parkluecke$, sowie $Einparken$.

Der comm_server_dequeue_thread dient dem Einparkassistenten als Treiber zum CAN-Bus. AIRA-Nachrichten, die an dem Leitstand oder dem ARM7 gerichtet sind, werden von dem Thread in CAN-Nachrichten umgewandelt und auf dem Bus verschickt. Diese Nachrichten sind unter anderem Bestätigungen der Zustände des Einparkvorgangs für die Repräsentation auf dem Leitstand, die unter Kapitel 4.3 beschrieben werden. Für eine detaillierte Liste der CAN-Nachrichten wird auf [14] verwiesen. Das Versenden der CAN-Nachricht mit der ID 0x4FF vom comm_server_thread zum ARM7, wird während des Einparkvorgangs vom comm_server_dequeue_thread übernommen.

- Generierung der Soll-Vorgaben für Geschwindigkeit und Lenkwinkel an die Antriebseinheit mit der CAN-ID 0x4FF.
- Senden der Befehlsbestätigungen vom Leitstand (CAN-ID: 0x4FX, 0x49X, 0xFF)

7. Messtechnische Analyse des Einparkassistenten

Dieses Kapitel dokumentiert das zeitliche Verhalten des Einparkassistenten unter Einfluss der Pthreads, der AIRA-Laufzeitumgebung und des Betriebssystems. Es werden die Ausführungszeiten der Automaten gemessen, dabei werden die Pins auf der parallelen Schnittstelle auf High, getrieben sobald die Reaktionsmethode betreten wird, und vor dem Verlassen der Methode wieder auf Low.

Des Weiteren wird die Implementierung der Odometrie mit der Dymola-Simulation verglichen, um die Abweichungen quantifizieren zu können. Es werden die Codeabschnitte der Odometrie und des Reglers aus dem Einparkassistenten übernommen. Mit dem Code in ANSI C werden die Koordinaten der Räder mit dem CHScite-Tool berechnet und mit denen aus der Dymola-Simulation in einer Excel Tabelle miteinander verglichen.

7.1. Timing-Verhalten des Einparkassistenten

Neben der Ausführungszeit der Reaktionsmethoden in der AIRA-Laufzeitumgebung haben folgende Komponenten in der Softwarearchitektur Einfluss auf das zeitliche Verhalten des Einparkassistenten.

- POSIX: Bedingungsvariable, Kontextwechsel
- AIRA-Laufzeitumgebung: tick-Timer, Reaktionsmethoden, Suchalgorithmus.
- Linux Betriebssystem: read()-Funktion, Scheduler

Die Automaten werden mit den Bedingungsvariablen aus und in dem blockierten Zustand geführt. Sobald eine Nachricht in die Queue eingetragen wird, signalisiert die Bedingungsvariable Linux, dass für diesen Automaten in der Scheduler-Warteliste, die Betriebsmittel für die Ausführung bereitstehen, und führt ihn vom blockierten in den bereiten Zustand. Dort wartet der Automat auf die Zuteilung der Prozessorzeit. Wurde die Nachricht verbraucht und die Nachrichtenqueue ist leer, führt die Bedingungsvariable den Automaten in den blockierten Zustand. (vgl. Kapitel 6.2.3)

Neben den Bedingungsvariablen wird der Kontextwechsel durch die blockierende read()-Funktion im Automaten $Laserscanner$ herbeigeführt. Die Abstände vom Laserscanner werden T_{LS} auf der RS422 Schnittstellte übertragen. Es wird das Abtasttheorem von Nyquist angewandt. Dabei löst der tick-Timer alle 6 ms das Pollen der Bytes mit der read()-Funktion auf dem File Deskriptor aus. Wenn keine Bytes auf der RS422 Schnittstelle anliegen, blockiert die read()-Funktion den Automaten $Laserscanner$. Im Anschluss trägt der Scheduler den Automaten in eine Thread Warteliste. Dort wartet der Automat auf die Bytes der RS422 Schnittstelle und die Zuteilung des Prozessors.

Der Automat $Comm\_Server$, der die WLAN- und CAN-Nachrichten empfängt, enthält keinen tick-Timer. Im Gegensatz zum Automaten $Laserscanner$ beinhaltet der $Comm\_Server$ zwei dedizierte Threads. Der comm_server_dequeue_thread sendet, und der comm_server_thread empfängt in einer $while$-Schleife. (vgl. Kapitel 6.2.5) Der CAN-Kommunikationskontroller führt den comm_server_thread aus dem blockierten Zustand, wenn eine CAN-Nachricht empfangen wurde. Da die Lenkwinkel dem Automaten $Einparken$ gesendet werden, wird durch die Periodizität der CAN-Nachricht von T_CAN der tick-Timer substituiert und es wird kein tick-Timer, wie beim $Laserscanner$ benötigt.

7.1.1. Auslösen eines Kontextwechsels durch den Scheduler

Bei einem Kontextwechsel der Threads wird die Zuteilung des Prozessors durch den Scheduler mit einer Warteliste verwaltet. Durch Parametrisierungen der Priorität können Threads mit niedriger Priorität von denen mit höherer preemptiv verdrängt werden. (siehe Kapitel 6.2.2) Nach der Ausführung des Automaten oder die Verdrängung durch einen anderen, wird der aktuelle Automat in die Warteliste eingetragen. Die Ausführungszeit der Zuteilung des Prozessors oder das Eintragen in die Warteliste wird nicht gemessen, da diese an das POSIX-Framework gekoppelt ist. Der Aufwand wird als zu hoch eingestuft. Bei einer Modifizierung des POSIX-Quellcodes mit den Treibern der parallelen Schnittstelle, müsste der Kernel neu kompiliert werden.

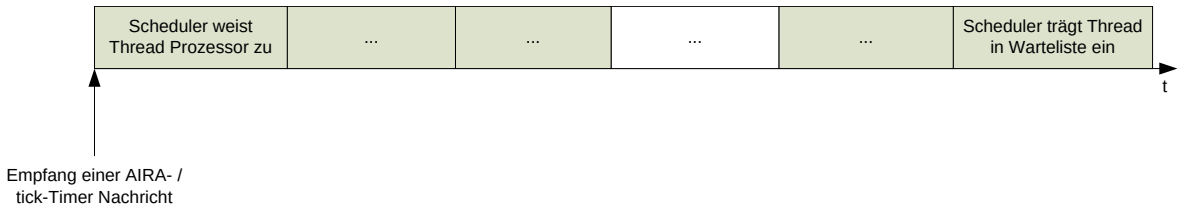

Abb. 47: Nicht messbare Ausführungszeit des Kontextwechsels

7.1.2. Start und Blockierung der Automaten durch die Bedingungsvariablen

Mit den Bedingungsvariablen der POSIX-Threadverwaltung werden die Automaten blockiert oder gestartet. (vgl. Kapitel 6.2.3) Die Ausführungszeit zum Setzen und Löschen der Bedingungsvariablen wird nicht gemessen, diese ist ebenfalls an das POSIX-Framework gekoppelt. (vgl. Abb. 48)

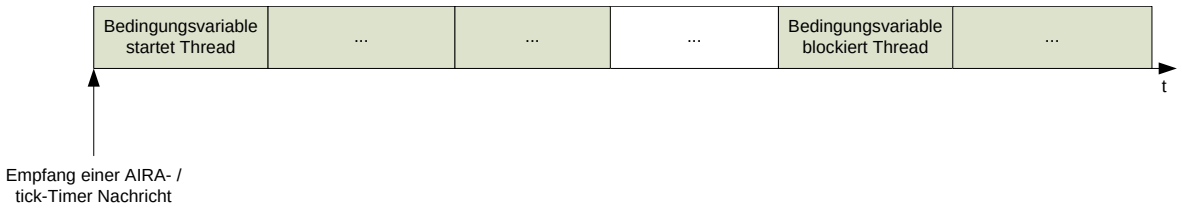

Abb. 48: Nicht messbare Ausführungszeit des Setzens und Löschens der Bedingungsvariable

7.1.3. Die Messung der Ausführungszeit der Reaktionsmethode

Im Einparkassistenten werden die Ausführungszeiten der Automaten mit und ohne tick-Timer gemessen. Bei den Automaten ohne tick-Timer werden die Reaktionsmethoden ohne Unterbrechungen in der Ausführung gemessen. (vgl. Abb. 49)

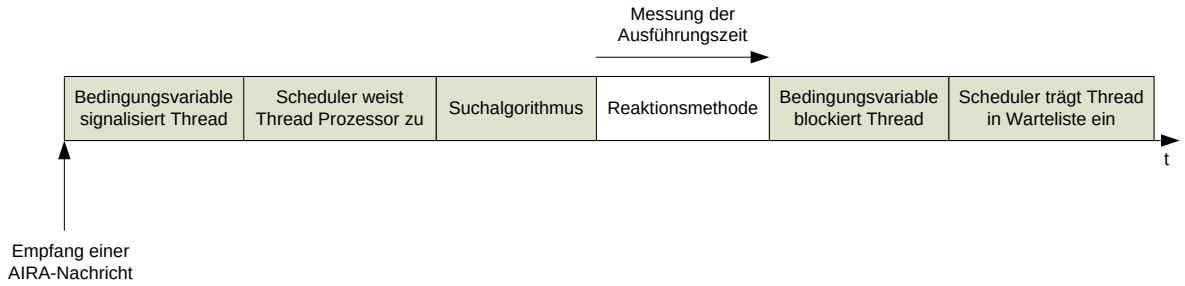

Abb. 49: Die Ausführungszeit der Reaktionsmethode ohne Unterbrechung

Der Automat *Laserscanner* ist an einem tick-Timer gekoppelt, da der Treiber des Laserscanners keine HW Interrupts nach Ablauf eines Scanzyklus von T_{LS} erzeugt, werden die Bytes von der RS422 Schnittstelle einzeln gepollt. (vgl. Kapitel 5.1.2) Dabei versendet der tick-Timer alle 6 ms eine Nachricht an den Automaten und löst damit den Suchalgorithmus und die Reaktionsmethode aus. Die Reaktionsmethode beinhaltet 14 read()-Funktionsaufrufe. (vgl. Kapitel 6.1.3) Die Ausführung der Reaktionsmethode wird mehrmals unterbrochen. Als Konsequenz wird neben der Ausführungszeit der Reaktionsmethode, auf einem separaten Kanal die Ausführungszeit der read()-Funktion, ohne Kontextwechsel über die RS422 Schnittstelle gemessen. (vgl. Abb. 50)

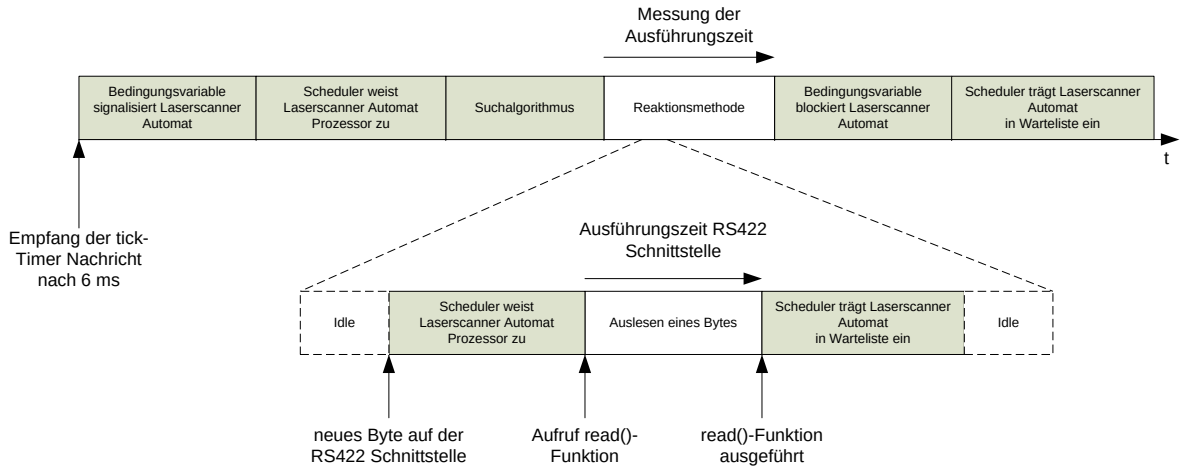

Abb. 50: Messung Ausführungszeit Reaktionsmethode mit Unterbrechung

7.2. Messung der Ausführungszeit der Automaten

Es wurden Messungen durchgeführt und die Analyseergebnisse werden hier dargestellt. Auf Kanal 1 ist die Ausführungszeit der Reaktionsmethode für den Automat *Laserscanner* dargestellt. Kanal 2 stellt die Ausführungszeit der für die einzelnen read()-Funktionen dar. Für Abb. 51 auf Kanal 1 gilt:

- Start Reaktionsmethode Pin=High.
- Ende Reaktionsmethode Pin=Low.

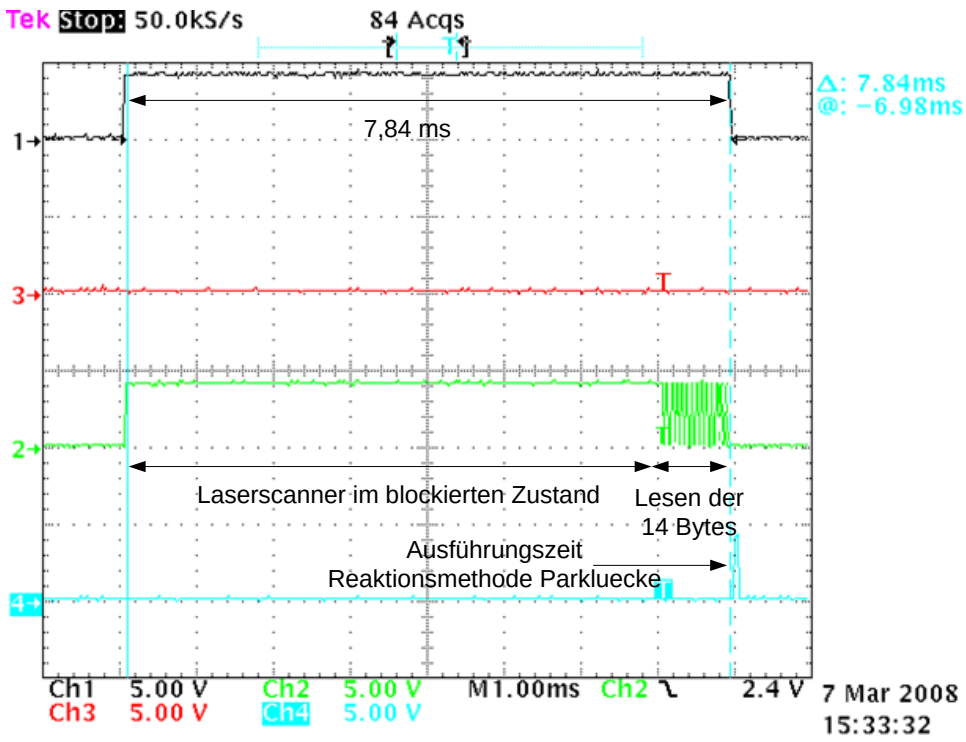

Abb. 51: Ausführungszeit, Auflösung 1ms/Raster und 5V/Raster. Ch1=Reaktionsmethode *Laserscanner* Ch2=read()-Funktion Ch4=Automat *Parkluecke*

Die Messungen zeigen, dass die Reaktionsmethode ungefähr 7,84 ms benötigt. Dabei ist die High-phase des ersten Bytes im Vergleich zu den restlichen 13 Bytes länger. Es geschieht folgendes:

- Der Pin auf Kanal 2 wird nach 6 ms durch den tick-Timer mit dem Aufruf der read()-Funktion auf High getrieben, es liegt jedoch kein Byte zum auslesen an.
- Folge: Die read()-Funktion blockiert die Ausführung des Threads.
- Der Scheduler trägt den Thread in die Warteliste ein und der Pin auf Kanal 2 der beim Eintritt der read()-Funktion auf High getrieben wurde, verbleibt auf High.
- Sobald T_{LS} abgelaufen sind und die 14 Bytes zum Auslesen auf dem File Deskriptor bereitstehen, kann das erste Byte ausgelesen werden.
- Nachdem das Byte ausgelesen wurde, kann die read()-Funktion abschließen und der Pin Kanal 2 wird wieder auf Low getrieben.
- Da jetzt die restlichen 13 Bytes in den File Deskriptor geschrieben werden, sind die folgenden read()-Funktionsaufrufe im Vergleich zur ersten kürzer.

Für eine Zeitspanne von 6 ms verweilt der Automat *Laserscanner* in einem Idle Zustand. Nach dem Aufruf der read()-Funktion verbringt der Automat 7,32ms in einem blockierten Zustand. Die Messung der Ausführungszeit im Automat *Parkluecke* auf Kanal 4 benötigt zum Vergleich 50 ms für die Reaktionsmethode.

Es wird eine feinere Zeitauflösung gewählt, es gilt für Abb. 52 auf Kanal 2:

– Start read()-Funktion, auslesen eines Bytes Pin=High.

– Ende read()-Funktion, auslesen eines Bytes Pin=Low.

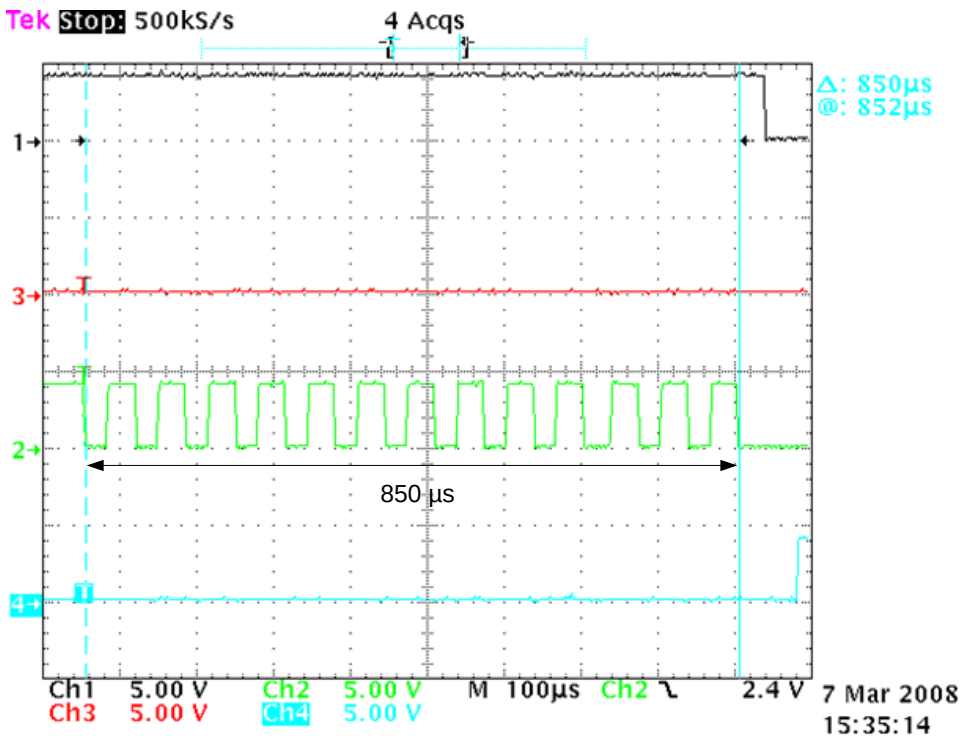

Abb. 52: Feinere Zeitauflösung der Abb. 51, Auflösung 100μs/Raster und 5V/Kasten. Ch1=Reaktionsmethode *Laserscanner* Ch2=read()-Funktion Ch4=Automat *Parkluecke*

Bei der Messung des zweiten Kanals wird festgestellt, dass die Ausführungszeit für das Auslesen eines Bytes 37,2μs und für die restlichen 13 Bytes insgesamt 850 μs beträgt. Die Low-Phasen sind 25 μs lang. In den Low-Phasen findet ein Kontextwechsel statt und der Scheduler teilt dem Automaten den Prozessor zu. (vgl. Abschnitt 7.1) Aus den gemessen High- und Low-Phasen muss eine Differenz von 28,8μs abgezogen werden, diese Zeit wird benötigt, um die steigende und fallende Flanke auf dem parallelen Port zu treiben.

7.3. Vergleich der erzeugten Trajektorie durch den Regler

In diesem Abschnitt werden die Lenkwinkel Soll-Vorgaben, die Odometrie und die von dem PD-Regler erzeugte Trajektorie mit der von der virtuellen Deichsel, während des Einparkvorgangs, miteinander verglichen.(siehe Anhang A.4) Vor der Implementierung des Reglers und der Odometrie wurde ein C Modell mit dem CHScite-Tool geschrieben, diese bietet die Möglichkeit die Ergebnisse in einem Diagramm darzustellen. Das erzeugte Diagramm wurde mit der Dymola-Simulation der virtuellen Deichsel verglichen. (siehe Anhang A.3)

Die messtechnische Analyse des SCVs während des Einparkvorgangs geschieht über die Konsolenausgabe. Es werden die Werte der Odometrie und des Reglers ausgegeben und in eine Excel-Tabelle gespeichert.

Um die Abweichungen von Dymola und der Simulation zu bestimmen, werden die Achswinkel, Koordinaten von Hinter- und Vorderrad in X- sowie Y-Richtung miteinander in einer Excel-Tabelle verglichen. In den Abb. 53 und 54 werden die Y-Koordinaten für das Vorder- und Hinterrad zusammen dargestellt. Die größte Abweichung von maximal 0,024 m weist y_p bei einem Startabstand von 1,2 m zur Wand.auf. (vgl. Abb. 53 und 54)

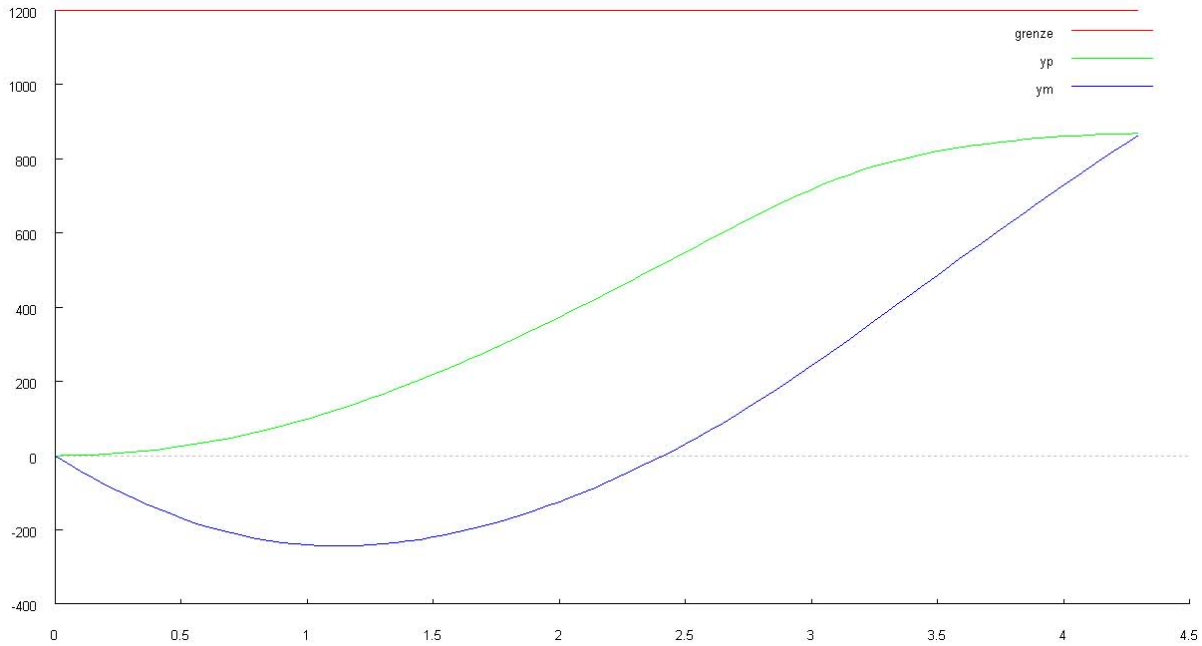

Abb. 53: C Modell Diagramm, ym; yp=Y-Koordinate des Hinter- und Vorderrades, Grenze=Startabstand

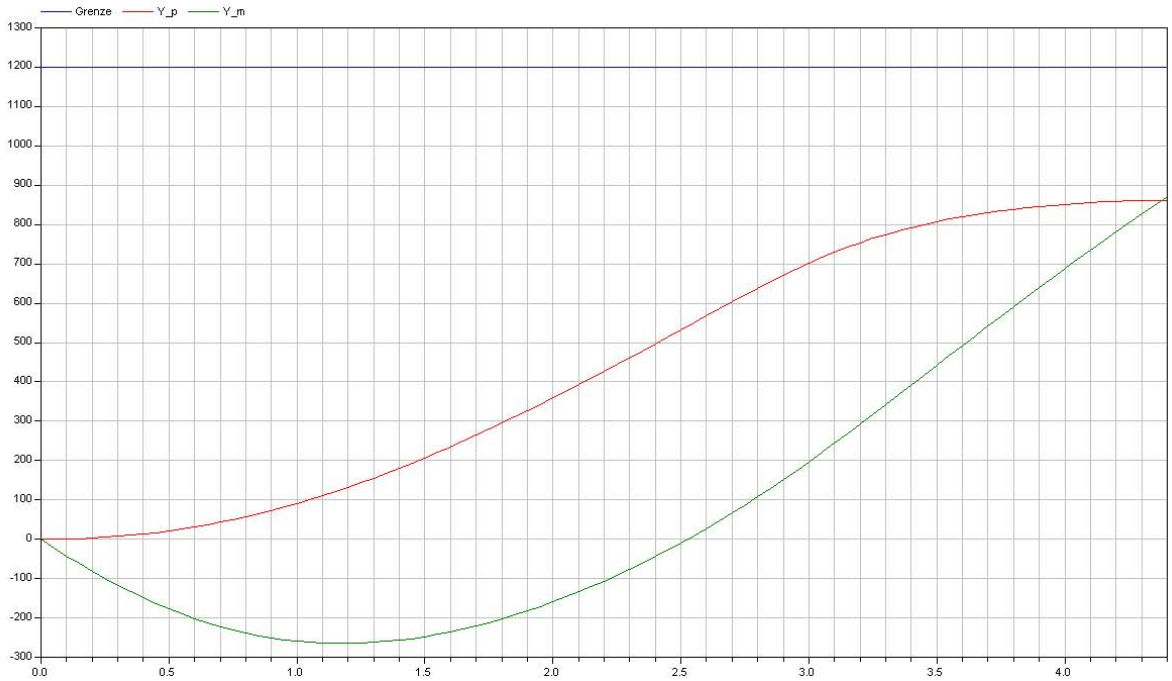

Abb. 54: Dymola-Diagramm, Y_m; Y_p=Y-Koordinate des Hinter- und Vorderrades, Grenze=Startabstand

Im Vergleich der virtuellen Deichsel und des PD-Reglers ergeben sich in der Excel-Tabelle größere Abweichungen, dabei weist die größte Differenz die Position des Vorderrades in X-Richtung auf. Sie beträgt 0,22 m bei einem Startabstand von 1,2 m zur Wand. Dieser um den Faktor 10 große Unterschied ist darauf zurückzuführen, da es sich um zwei verschiedene Regler handelt. (vgl. Abb. 55 und 56)

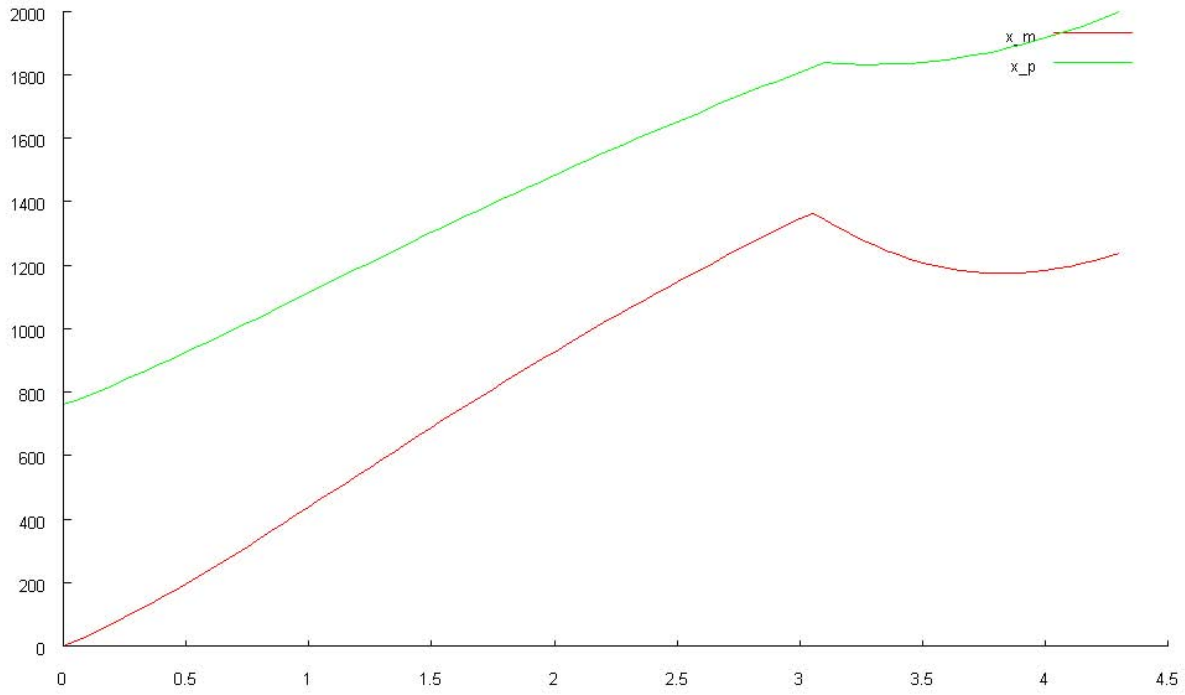

Abb. 55: C Modell PD-Regler, xm; xp=X-Koordinate des Hinter- und Vorderrades

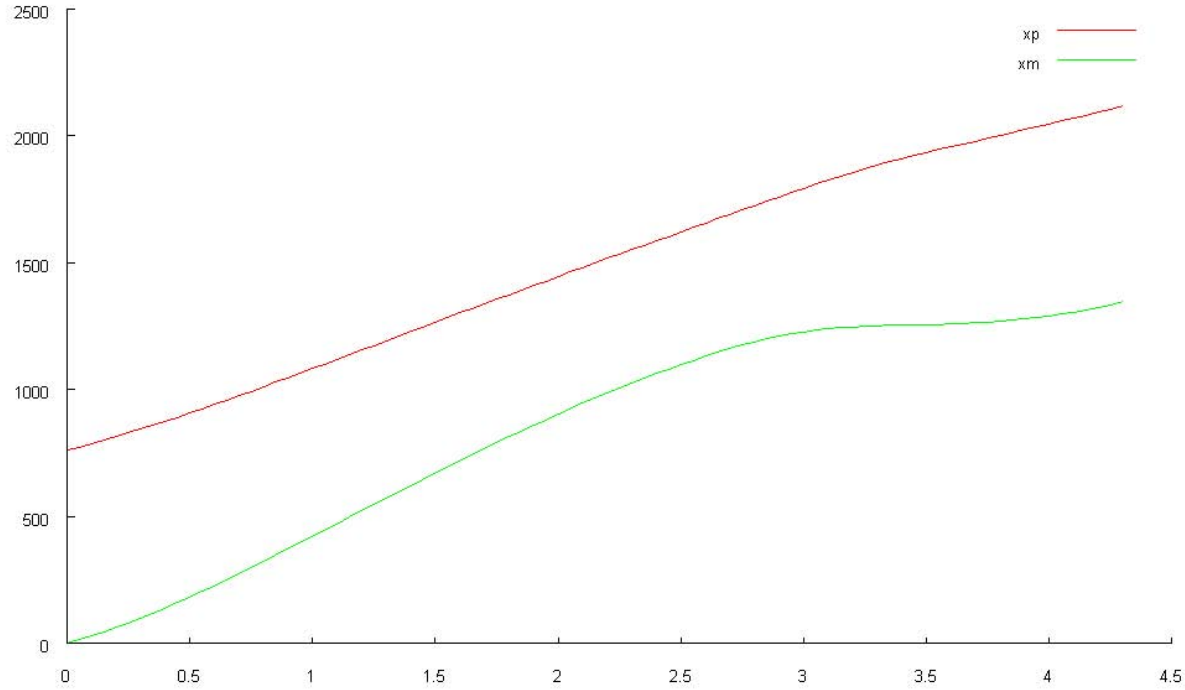

Abb. 56: C Modell virtuelle Deichsel, xm; xp=X-Koordinate des Hinter- und Vorderrades

Die Werte mit aktiviertem Entscheidungsmodul, welcher die Messwerte auswertet und feststellt, ob ein Hindernis oder die seitliche Parklückenbegrenzung erfasst wurde, werden ebenfalls analysiert. Bei einem Startabstand von 1,2 m von der Wand, wird in der ersten Sekunde die Wand erfasst, die nachfolgenden 2,5 Sekunden misst der Laserscanner das vordere Hindernis und in der Schlussphase die Wand, wenn das Fahrzeug annähernd parallel steht. Der Anteil während der longitidudalen Fahrt in die Parklücke beträgt 50% zu 50%. Bei einem Startabstand von 2 m wird der

Anteil der gemessen Abstände zur Wand geringer und das Verhältnis beträgt 63,63% zu 36,36%. Damit wird der Entscheidungsalgorithmus für den Übergang der virtuellen oder lasergestützten Positionsermittlung verifiziert, dass bei steigendem Startabstand das SCV öfter ein Hindernis erfasst, als die Wand. (vgl. Abb. 57)

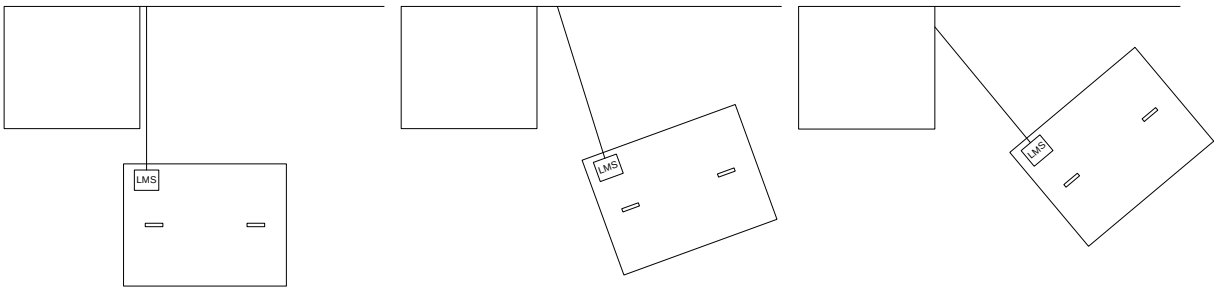

Abb. 57: Erfassung der Wand oder vorderes Hindernis beim Einparken

Die Implementierung der Odometrie in ANSI C ist ausreichend, wenn das Dymola als Referenz für die Genauigkeit gilt, eine Optimierung nicht benötigt. Der Vergleich virtuelle Deichsel und PD-Regler zeigt, dass beim Einparkvorgang durch den Einsatz des PD-Reglers eine um 0,22 m kürzere Parklücke in Längsrichtung benötigt wird. Die Messergebnisse des Entscheidungsalgorithmus für die virtuelle und lasergestützte Positionsermittlung korrelieren mit den Überlegungen, die zu dem Konzept des Algorithmus geführt haben und wurden somit verifiziert.

7.4. Auswertung der messtechnischen Analyse

Aus den Messungen lassen sich folgende Rückschlüsse ziehen:

- Der Aufruf der vom Linux zur Verfügung gestellten read()-Funktion, synchronisiert den Automaten mit dem Laserscanner. Beim Abrufen der Lasermesswert in Abständen von 6ms, wird nicht wie dokumentiert gepollt [14], sondern blockierend gewartet. Dabei wird der Lesevorgang vom tick-Timer im Automaten $Laserscanner$ angestoßen, so dass die Aufrufe der read()-Funktion auf dem File Deskriptor den Automaten blockieren, bis ein komplettes Datenpaket von 14 Bytes [25] ausgelesen wird. Damit wird die Ausführungszeit der Reaktionsmethode des Automaten um 7,84 ms verlängert und so auf T_{LS} synchronisiert, die der Laserscanner für einen kompletten Scanzyklus benötigt. Im Vergleich dazu benötigt der Automat $Parkluecke$ auf Kanal 4, der nach dem Empfang der AIRA-Nachricht aktiviert wird, 50μs für die Reaktionsmethode (vgl. Abb. 51). Die Ausführungszeit des Automaten $Laserscanner$ ist 150-mal länger, als die der anderen Automaten. Eine Reduzierung der Ausführungszeit ist nicht möglich, da auf die RS422 gewartet werden muss.
- Da die Automaten auf Ereignisse reagieren und zwischen den Wartezeiten auf das Auftreten dieser Ereignisse blockieren, ist die Dauer der Inaktivität der Automaten im Vergleich zu den tatsächlichen Ausführungszeiten der Reaktionsmethoden groß. Die 650MHz CPU ist überdimensioniert und der Einparkassistent könnte auf einen μ-Controller mit einer geringeren Taktfrequenz ausgeführt werden. Die längste, ununterbrochene Ausführungszeit für einen Automaten beträgt 150μs. Geht man von dem Automat $Laserscanner$ mit der kleinsten Periode T_{LS} aus, können in der Zwischenzeit 90 andere Automaten reagieren. Vernachlässigt man das Timing-Verhalten für ein anderes Betriebssystem auf dem μ-Controller kann die Taktfrequenz der CPU des μ-Controller um 1/20 des GEME-Rechners betragen. Mit einer

32,5MHz CPU besteht zwischen der Periode T_{LS} eine Zeitspanne für die Ausführung von 4 bis 5 Automaten.

- Die Einstellung der Scheduling-Parameter hat nur einen kleinen Einfluss auf die Ausführung der Automaten. Im Falle eines Worst-Case-Scenarios ohne Parametrisierung der Priorität der Automaten, würden nach Ablauf T_{LS} vom Laserscanner, die vier anderen Automaten vor dem Automaten $Laserscanner$ ausgeführt werden. Die Verzögerung würde max. 600μs betragen und beträgt 1/22 von T_{LS}, welches im Rahmen einer zeitnahen Ausführung liegt. Die Parametrisierung der Priorität und des Scheduling-Verfahrens wäre vonnöten, wenn die CPU-Taktrate gedrosselt wird. Eine Reduzierung der Oszillator Frequenz um den Faktor 20, würde die Verzögerungszeit in einem Worst-Case-Scenario auf 12ms steigen.
- Das Sicherheitskonzept des Linux-Betriebssystems schreibt vor, dass bei einem Zugriff auf die RS422-Schnittstelle während der Ausführung des Einparkassistenten vom User- in den Kernel Space gewechselt wird. Damit ist ein nicht zu entfernender Overhead verbunden, die das Betriebssystem für die Verabeitung der Low-Level-Daten benötigt. Bei einer Portierung des Einparkassistenten auf eine andere Rechnerplattform kann, in Hinblick auf einen langsameren Oszillator, auf das Betriebssystem verzichtet werden.
- Dazu kommt, dass die nebenläufige Ausführung der Automaten keine Auswirkung auf die Ausführungszeiten haben, da die Verarbeitung der Signale sequentiell erfolgt. Wie man aus den Sequenzdiagrammen in Kapitel 5.2.1 sehen kann, werden die Nachrichten erst verschickt, nachdem ein Signal komplett verarbeitet wurde. Ein Wechsel der Threads während der Ausführung, bringt keine erkennbaren Vorteile.
- Bei der Implementierung der Odometrie ergeben sich Abweichungen von 2 % zur Dymolasimulation und liegen damit im einstelligen Zentimeter-Bereich. Für das Einparken sind diese Abweichungen ausreichend und es wird keine weitere Optimierung benötigt.
- Der PD-Regler würde durch eine Erhöhung des D-Anteils höhere Lenkwinkel α vorgeben und damit die benötigte Parklückenlänge weiter verkürzen. Die Erhöhung des D-Anteils wird aber durch die physikalisch einstellbaren Lenkwinkel des Vorderrades begrenzt. Zusätzlich wird der PD-Regler ab einem Grenzwert unstabil und die Stellgröße oszilliert.

Zusammengefasst ergeben sich folgende Zeitkennwerte:

T	Einheit	Beschreibung
6	ms	tick_Timer des Laserscannerautomaten
7,84	ms	Ausführungszeit der Reaktionsmethode des Laserscanner Automaten
37,2	μs	Ausführungszeit einer read()-Funktion der die RS422 Schnittstelle ausliest
25	μs	Ausführungszeit des Kontextwechsels zwischen den read()-Funktionsaufrufen
887,2	μs	Ausführungszeit der 14-mal aufgerufenen read()-Funktion der die RS422 Schnittstelle ausliest
50	μs	kürzeste Ausführungszeit eines Automaten, ausschließlich Laserscanner
150	μs	längste Ausführungszeit eines Automaten, ausschließlich Laserscanner
28,8	μs	Ausführungszeit um die Pins der parallelen Schnittstelle von Low auf High zu treiben und High auf Low

8. Implementierung des Einparkassistenten auf einem 32 Bit Mikrocontroller

Aus dem Kapitel 7 wurde durch die Messung der Ausführungszeiten der Automaten festgestellt, dass der Einparkassistent auf einem μ-Controller mit einer 32 MHz Taktfrequenz lauffähig wäre. Dabei wäre eine nebenläufige Ausführung der Automaten nicht zwingend erforderlich, da die Automaten sequentiell verarbeitet werden. In diesem Kapitel wird die Implementierung des Einparkassistenten auf einem μ-Controller dokumentiert.

Der Einparkassistent wird auf einen ARM7TDMI μ-Controller mit 32 MHz portiert, der mit einem Twin CAN-Controller, sowie einer RS232 Schnittstelle ausgestattet ist. Es handelt sich um denselben μ-Controller, der die Antriebs- und Lenkungsmotoren des SCVs regelt. (vgl. Kapitel 4.1)

Da die AIRA-Laufzeitumgebung und die Modellierung der Automaten in ANSI C implementiert ist, wird der Code des Kommunikationsrechners übernommen und angepasst. Auf die Verwendung von POSIX zur Verwaltung der nebenläufigen Ausführung der Automaten wird verzichtet, die damit verbundene Ausführungszeiten für Kontextwechsel und Scheduling fallen weg. Sämtliche Timeraufrufe von Linux und die Einbindungen der POSIX-Bibliotheksfunktionen zur Threadverwaltung werden entfernt. Die Linux read()-Funktion der RS422 Schnittstelle wird ersetzt, da auf dem μ-Controller keine RS422 Schnittstelle vorhanden ist, wurde eine Methode entwickelt, die die RS232 Schnittstelle pollt.

Als Ersatz für die nebenläufige Ausführung der Automaten wird ein neues Softwarekonzept zur sequentiellen Verarbeitung der einzelnen Automaten eingeführt. Der Laserscanner, der die kleinste Bearbeitungsperiode im SCV besitzt, startet eine Verarbeitungskette der Automaten, sobald eine Laserscanner-Nachricht von der RS232 Schnittstellte ausgelesen wird.

Mit dem im Vergleich des GEME-Rechners um den Faktor 20 langsameren Oszillator des μ-Controllers, werden die Ausführungszeiten der Automaten erneut untersucht. Die messtechnische Auswertung wird mit Timing-Diagrammen dokumentiert, die Einhaltung der Echtzeitanforderung des Einparkassistenten validiert.

Im letzten Abschnitt werden Vorschläge zur Erweiterung, sowie Änderungen des Einparkassistenten auf dem μ-Controller besprochen. Es wird neben der konzeptionellen und programmiertechnischen Erweiterung, die Umstellung auf ein Time-Triggered-Softwarekonzept, sowie Alternativen zur Reduzierung der Ausführungszeiten für die Fließkommazahlenberechnung vorgestellt.

8.1. Der PhyCORE AT91M55800A Microcontroller

Es wird ein 32-Bit Microcontroller mit einem ARM7 Chipsatz, 32 MHz Oszillator, RS232 Schnittstelle und ein Twin CAN 82C900 Kommunikationscontroller verwendet. (vgl. Abb. 58)

Da eine RS422 Schnittstelle auf dem μ-Controller fehlt, wird die interne RS232 Schnittstelle verwendet und folgendermaßen parametrisiert:

- maximale Übertragungsrate von 38,4 kbit/s
- da kein Clocksignal von dem LMS generiert wird, ist der asynchrone Übertragungsmodus eingestellt
- 1 Startbit + 8 Bit Daten + 1 Stopbit = 10 Bit

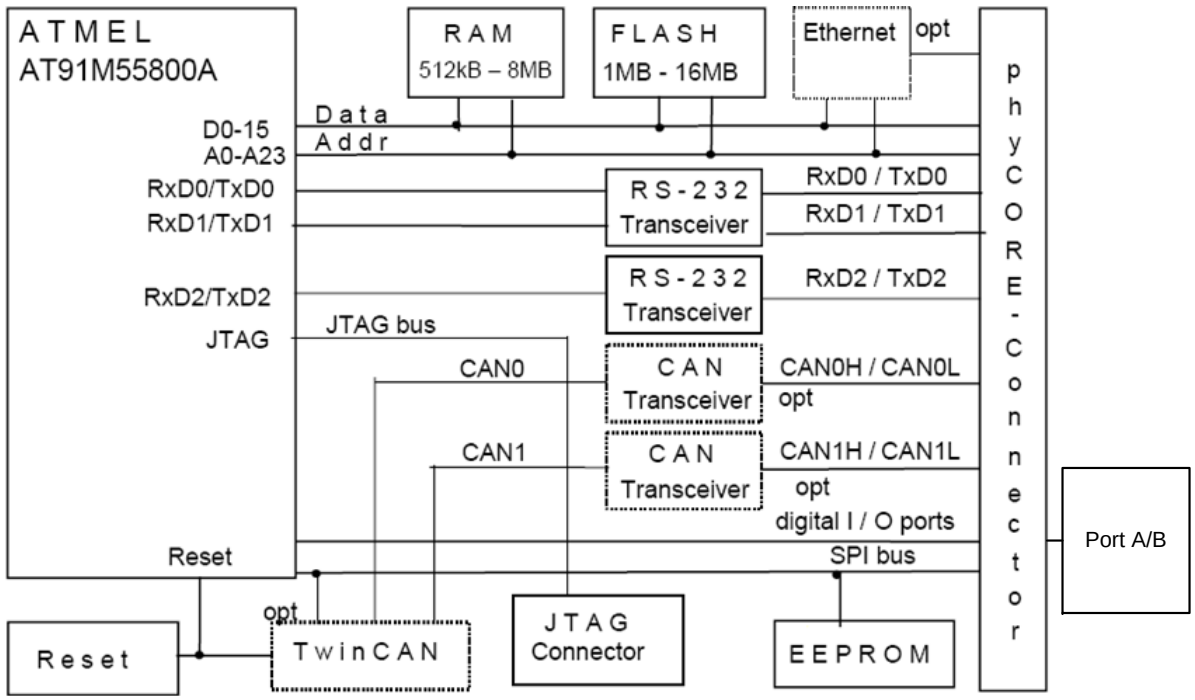

Abb. 58: Blockdiagrann PhyCORE AT91M55800A μ-Controller-Board [10] mit 82C251 Twin CAN- Controller, RS232 Schnittstelle und CS8900A Cirrus Logic Ethernet Controller.

Damit dauert das Auslesen einer Laserscanner-Nachricht mit 14 Bytes 3,646 ms über RS232. Die RS422 benötigt 887,2 μs.

Der Twin CAN 82C900 Controller besitzt zwei Kanäle, die jeweils an einem 82C251 CAN-Transceiver angeschlossen sind und 32 Nachrichtenpuffer für die Definition von Empfangs und Sendenachrichten.

Zusätzlich ist ein CS8900A Ethernet Controller angeschlossen. Dieser ist mit dem IEEE 802.3 Standard kompatibel und unterstützt den Full-Duplex-Modus bei einer Übertragungsgeschwindigkeit von 10 MBit/s mit Twisted-Pair-Kabel.

Damit kann der AT91M55800A, abgesehen von dem WebCAM Interface, sämtliche Funktionalitäten des GEME-Kommunikationsrechners ersetzen.

Für die messtechnische Analyse werden die Pins auf dem Port B getrieben. Die Signale werden bis zum Erweiterungsboard durchgeschliffen. Laut Benutzerhandbuch [10] weicht die Pin-Belegung auf dem Erweiterungsboard vom Schaltplan des Mainboards ab. (siehe Anhang A.1)

8.2. Das modifizierte Softwaredesign des Einparkassistenten

Ohne die Thread-Verwaltung benötigt das Softwarekonzept des Einparkassistenten eine Ergänzung für die sequentielle Ausführung der Automaten. Bei dem modifizierten Softwareentwurf wird die Ausführungskette anhand des Automaten *Laserscanner* mit der kürzesten Ausführungsperiode T_{LS} synchronisiert. Da die Automaten erst nach dem Empfang der AIRA-Nachrichten die Reaktionsmethoden ausführen, und im Anschluss Nachrichten zur weiteren Reaktionen versendet werden, beinhaltet das neue Konzept die sequentielle Ausführung von Automaten in einer Kette. Die Ausführungskette wird durch den *Laserscanner* nach erfolgtem Scanzyklus gestartet. Nachdem der letzte Automat *Einparken* seine Nachricht verarbeitet hat, pollt der Automat *Laserscanner*

auf der RS232 nach einem neuen Byte. Mit diesem Konzept synchronisiert sich der Einparkassistent auf die Zeitbedingung des Laserscanners T_{LS}, und der tick-Timer wird in der Form (vgl. Kapitel 6.1.2) nicht mehr benötigt.

Die AIRA-Laufzeitumgebung ist in C implementiert und wird mit Anpassungen in den μ-Controller portiert. Die modellierten Automaten können, mit der Ausnahme des Laserscanners, ohne Änderungen übernommen werden. Damit die Modellierung der AIRA-Automaten beibehalten wird, muss der tick-Timer entfernt und durch eine Methode ersetzt werden. Da die Automaten erst nach dem Empfang einer Nachricht ausgeführt werden, muss zum Auslösen der Pollingfunktion des Automaten *Laserscanner* eine tick-Nachricht generiert werden. (vgl. Kapitel 6.1.2) Die Generierung der tick-Timer-Nachricht wird durch den Aufruf einer Funktion erzeugt, nachdem der letzte Automat *Einparken* ausgeführt wurde. Als Folge verarbeitet die Reaktionsmethode des Automaten *Laserscanner* die tick-Nachricht und löst damit das Auslesen von 14 Bytes von der RS232 Schnittstelle aus. (vgl. Abb. 59)

Im Einparkassistenten wird die Aktualisierung der Odometrie und die Lenkwinkel Soll-Berechnung nach dem Empfang des Lenkwinkelist-Werts vom CAN-Bus angestoßen. Die CAN- Anbindung des μ-Controller ist noch nicht implementiert. Bei dieser temporären Lösung wird die Aktualisierung und Berechnung des Lenkwinkel Soll-Werts nicht wie im ursprünglichen Konzept alle T_{CAN}, sondern nach dem Empfang der Lasermesswerte ausgelöst.

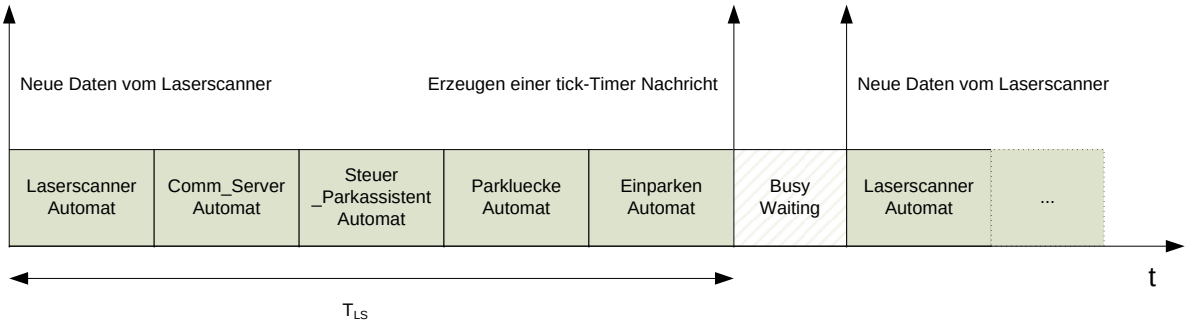

Abb. 59: Ausführungsreihenfolge der Automaten im neuen Softwarekonzept, Generierung des tick-Timers nach der Verarbeitung des letzten Automaten *Einparken*.

8.2.1. Messtechnische Analyse der Ausführungszeiten

Die modifizierte Implementierung auf dem μ-Controller enthält sämtliche Funktionalitäten des ursprünglichen Einparkassistenten, wie Parklückenauswertung, Odometrie Berechnung, Lenkwinkel Soll-Vorgabe durch den PD-Regler, WLAN/CAN-Kommunikation. Die CPU-Geschwindigkeit wurde um den Faktor 20 reduziert. In diesem Abschnitt wird die Einhaltung der Echtzeitanforderung dokumentiert.

Im SCV sind zwei Zeitzyklen der Hardware zu verarbeiten:

- T_{LS} Scanzyklus vom Laserscanner
- T_{CAN} Sendezyklus der Lenkwinkel Ist-Werte

Die Ausführungszeit T_{uC} für die Aktualisierung der Odometrie und der Vorgabe der Lenkwinkel durch den PD-Regler beträgt zusammen mit dem Auslesen des Lasermesswerts 40 ms. (vgl. Abb 60) Die gemessenen Zeiten sind mit der Berechnung der Fließkommazahlen zu erklären. Bei der Fließkommazahlen-Arithmetik werden in der CPU Exceptions geworfen, die er auffangen muss, mit

der Folge, dass im Kernel ein Kontextwechsel erzwungen wird [3]. Deswegen sind Berechnungen der Fließkommazahlen im Vergleich zu Ganzzahlen 10-mal langsamer in der Ausführungszeit.

Nach T_{uC} wird der Lenkwinkel Soll-Wert auf dem CAN-Bus übertragen. Da die Berechnungsphase des Automaten 3-mal länger als T_{LS} ist, werden in der Zwischenzeit 3 Scanzyklen verpasst. Außerdem jittert das Senden des Soll-Lenkwinkels. (vgl. Abb. 60)

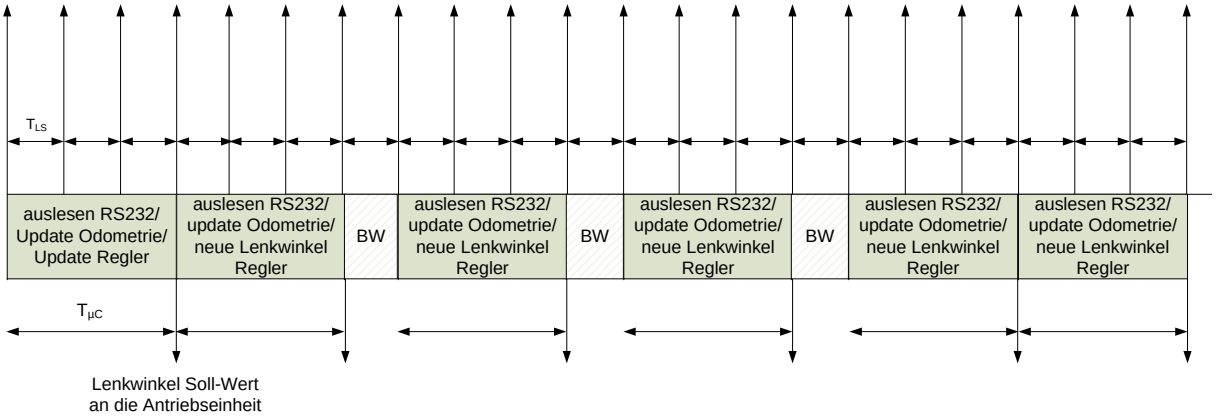

Abb. 60: Ausführungszeit des *Einparken* und *Laserscanner* Automaten, inkl. Aktualisierung der Odometrie, Berechnung der Lenkwinkel Soll-Vorgabe getriggert durch den Empfang der Lasermesswerte.

Die in der Antriebseinheit implementierte Regelung beinhaltet ein zeitgesteuertes Taskkonzept [26], das die Ist-Lenkwinkel in Abständen von T_{CAN} auf dem CAN-Bus verschickt, die Soll-Werte empfangen und alle 20 ms die Antriebs- und Lenkungsmotoren regelt. (vgl. Abb. 61)

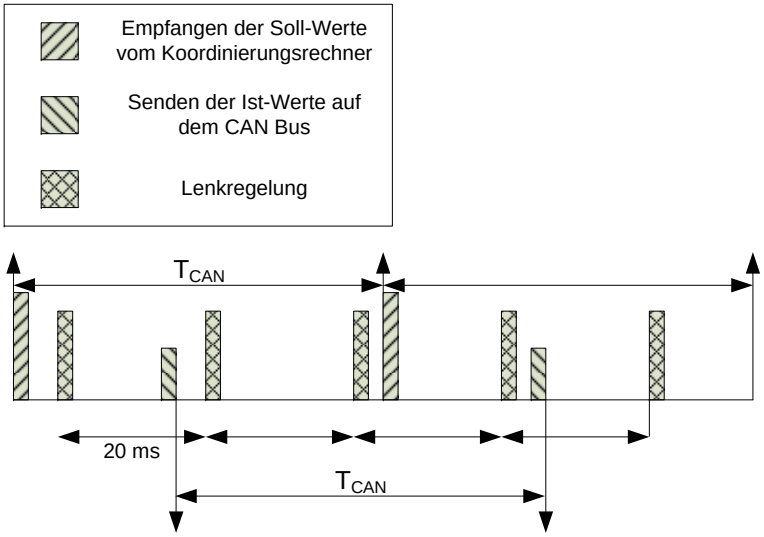

Abb. 61: Ausführungszyklen der Antriebseinheit: Regelungs-, Sende- und Empfangstask auf dem μ-Controller

Beide Zyklen besitzen unterschiedliche Perioden, damit besteht keine zeitliche Synchronisation zwischen der Hardware. In einem Worst-Case-Szenario wird ein Soll-Lenkwinkel nicht von der Antriebseinheit empfangen, wie aus Abb. 62 zu erkennen ist. Da der CAN-Treiber nicht implementiert wurde, fehlen bei der Abb. 62 die Laufzeiten des *Comm_Servers* und Kommunikationskontrollers. Nach der Implementierung des CAN-Treibers müsste dieser in den *Comm_Server* eingebunden und die Ausführungszeit für den *Comm_Server* und den Kommunikationskontroller bis zum Start

des Versendens der CAN-Nachricht gemessen werden. Dazu müsste die Übertragungszeit der CAN-Nachricht mit 111 Bits bei einer Übertragungsgeschwindigkeit von 500 kbit/s hinzugezählt werden. Sie beträgt 222 μs. T_{uC} müsste in Abb. 62 dann modifiziert werden.

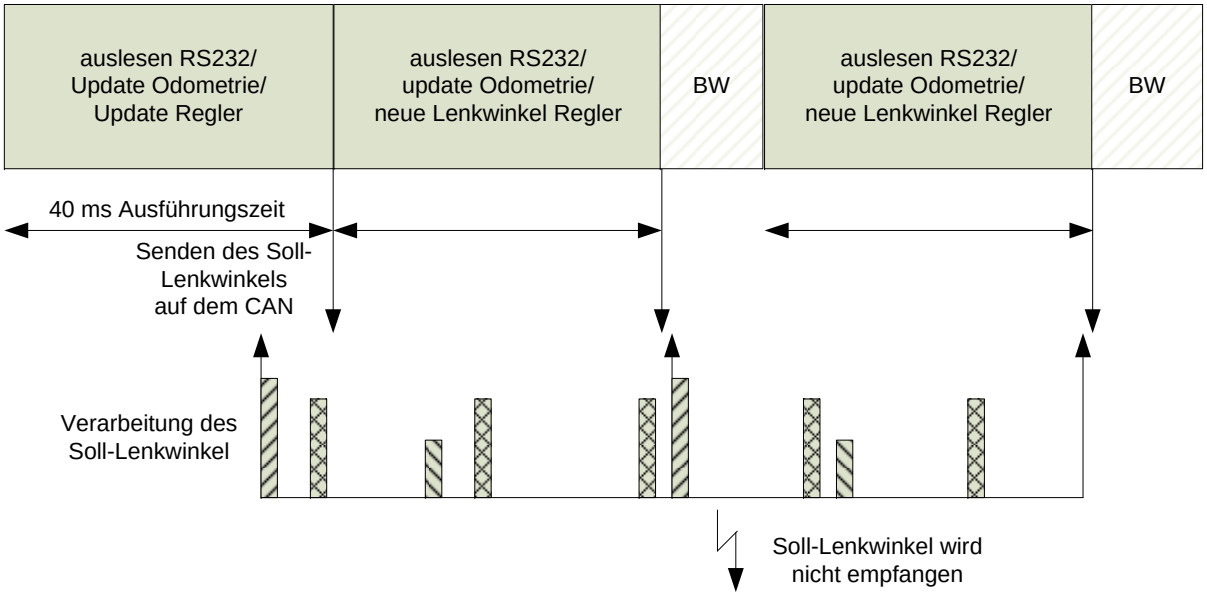

Abb. 62: Übersicht zur Kopplung des Timings des Einparkassistent-μ-Controller und der Antriebseinheit-μ-Controller

Zwar ist das Auftreten des Jitters deterministisch und eine Synchronisierung findet nachträglich, nach dem verpassten Empfang des Soll-Lenkwinkels, wieder statt. (vgl. Abb. 60) Jedoch wird die Regel für ein Echtzeitsystem, das eine Operation innerhalb einer Zeitgrenze verfügbar sein muss, verletzt und der Einparkassistent ist per Definition kein hartes Echtzeitsystem mehr. (vgl. Abb. 62)

8.3. Verbesserungs- und Erweiterungsvorschläge für den Einparkassistenten auf dem Microcontroller

Da die benötigten Schnittstellen Ethernet, CAN, RS232 auf dem ARM7 vorhanden sind, ergeben sich 2 Möglichkeiten.

- ARM7 ersetzt den Kommunikationsrechner: Einparkassistent wird auf dem μ-Controller ausgeführt. Bei dem Austausch würde der Stream der WebCAM wegfallen, weil keine USB Schnittstelle auf dem Board vorhanden ist, und der Server für den Betrieb das Linux Betriebssystem benötigt. Die Filterung der WLAN-Steuerungsbefehle wird auf dem μ-Controller in der Einparkphase durchgeführt, so dass die Befehle nicht auf dem CAN übertragen und der Bus dadurch entlastet wird. (vgl. 63)
- ARM7 ergänzt den Kommunikationsrechner: Die WebCAM streamt weiter das Video an den Leitstand. Der Kommunikationsrechner wandelt die WLAN-Nachrichten in CAN um und übernimmt die Filterung der Steuerungsbefehle in der Einparkphase. Die Soll-Lenkwinkel und die Geschwindigkeitsvorgaben werden vom Einparkassistenten auf dem μ-Controller über CAN gesendet und empfangen. Über eine CAN-Nachricht teilt der Einparkassistent den Kommunikationsrechner mit, dass er die Steuerungsbefehle filtern soll. (vgl. 64)

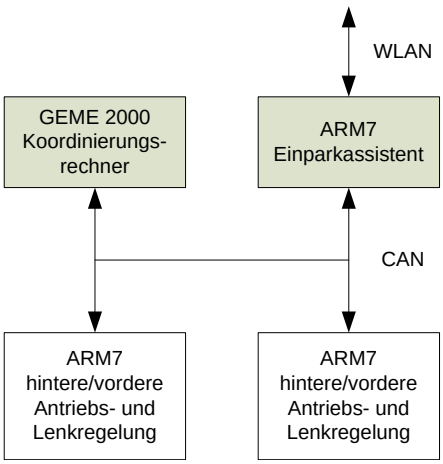

Abb. 63: Ersatz des Kommunikationsrechners durch den μ-Controller

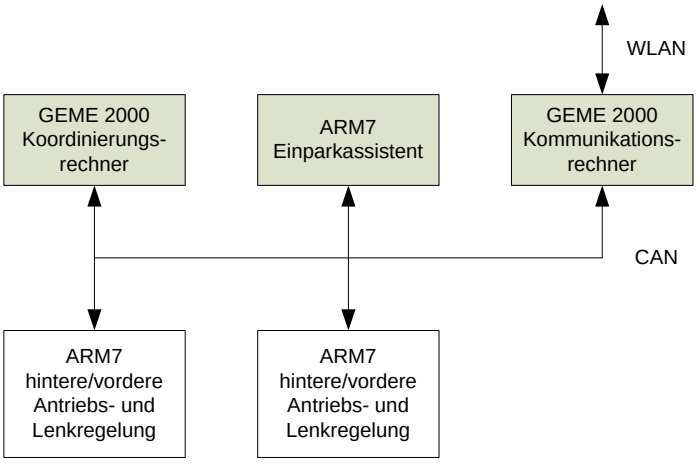

Abb. 64: Ergänzung der CAN-Funktionaliäten des Einparkassistenten durch den Kommunikationsserver

Auf dem ARM7 fehlen die Treiber für die CAN-Schnittstelle und die Einbindung in den $Comm\_Server$, da der Fokus in dieser Masterarbeit auf die Erstellung des Konzepts für die Portierung des Einparkassistenten gelegt wird.

Erst nachdem die Erweiterungen implementiert werden, sind folgende konzeptionelle Änderungen am Softwaredesign möglich:

- Im Softwaredesign werden die Werte vom CAN und Laserscanner gepollt. Mit der Verwendung von Interrupts für beide Schnittstellen würde das Busy Waiting entfallen. In den ISRs werden AIRA-Nachrichten an die jeweiligen Laserscanner und $Comm\_Server$ Automaten versendet, die eine Ausführung der Automatenkette anregt.
- Zusammen mit der Implementierung der ISRs wird die Verwendung des Schedulers [24] der die Antriebs- und Lenkungsmotoren regelt für den Einparkassistenten vorgeschlagen. Mit der Integration des Schedulers wird eine Time-Triggered-Architektur eingeführt. Die Aktualisierung der Odometrie und die Ausgabe der Soll-Lenkwinkel geschieht alle T_{CAN} ms, dabei werden die zuletzt empfangenen Messwertdaten vom Laserscanner verwendet. Durch die unterschiedlichen Perioden des Systems sind die Verzögerungen zwischen Lasermesswerten und Aktualisierung nicht konstant, da aber die Perioden vom Laserscanner bekannt sind,

können die Verzögerungszeiten berechnet werden. Alle 200 ms tritt der Fall auf, dass die Aktualisierung der Odometrie und das Auslesen der Lasermesswerte gleichzeitig starten. Der Automat *Einparken* müsste dann den Laserscanner verdrängen, siehe Abb. 65.

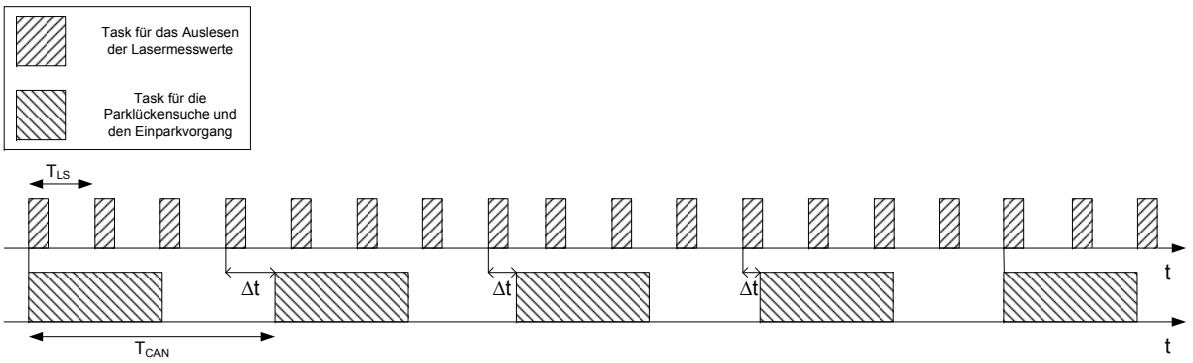

Abb. 65: Task Perioden für Laserscanzyklus, Parklückensuche/Einparkvorgang mit unterschiedlichen Abständen Δt

Das Fehlen des Coprozessors für die Fließkommazahlen-Arithmetik des ARM7 macht sich in der Ausführungszeit des Laserscanners bemerkbar. Es ergeben sich folgende Vorschläge, um die Ausführungszeit zu verringern:

- Der Tausch des ARM Modells. Der momentan verwendete Chip ist für Low-Cost-Anwendungen und μ-Controller optimiert. In den ARM7 Derivaten gibt es ein spezielles Modell für Echtzeitanwendungen. Der ARM7R ist speziell für Echtzeitsysteme geeignet und beinhaltet einen Coprozessor für Fließkommazahlen-Arithmetik.
- Der Austausch der CPU mit einer in denen eine Floating Point Unit (fpu) integriert ist, wie z.B. der TC1130 oder den RC32C. Der TC1130 besteht aus einer 32 Bit CPU, er enthält eine MMU, vier CAN-Knoten, eine Ethernet-Schnittstelle und wird mit einem Oszillator mit bis zu 150 MHz betrieben [13]. Das Linux Betriebssystem ist auf diesem μ-Controller lauffähig oder es könnte alternativ das echzeitfähige Real-Time-Linux installiert werden. In der RC32C Modellserie können die 16 Bit CPUs, mit Oszillatoren von bis zu 64 MHz betrieben werden [22]. Neben der CAN-Schnittstelle sind in einigen Derivaten FlexRay Kommunikationskontroller mit auf dem Board.
- Durch den Wechsel des Datentyps von Fließkommazahlen zu Integer wird die Fließkommazahlen-Arithmetik vermieden. Die Sinus- und Kosinusberechnungen werden dabei durch das Anlegen von Lookup-Tabellen ersetzt.

9. Zusammenfassung

Der in dieser Masterarbeit entwickelte Einparkassistent misst, mit einem vertikalen 180° Laserscanner bei einer Frequenz von 75 Hz, die rechte Seite des SCVs ab und führt diese Werte einem Entscheidungsalgorithmus zu. Dieser Entscheidungsalgorithmus wertet aus, ob es sich bei den Abständen um die seitliche Parklückenbegrenzung oder ein Hindernis in der Parklücke handelt. Anhand der Abstände die einem nichtlinearen kinematischen Einspurfahrzeugmodell zugeführt werden, berechnet ein PD-Regler Lenkwinkel Soll-Vorgaben, die das SCV während einer Rückwärtsfahrt in die Parklücke führt.

Mit der Odometrie des nichtlinearen kinematischen Einspurfahrzeugmodells wird die Position während des Einparkvorgangs mit und ohne lasergestützte Abstandserfassung ermittelt. Bei der Abstandsregelung werden die Eingangsgrößen der Geschwindigkeit, der Lenkwinkel und der Abstand zur seitlichen Parklückenbegrenzung der Odometrie zugeführt. Im Zusammenhang mit einem PD-Regler werden für den Einparkvorgang die Lenkwinkel Soll-Vorgaben an die Antriebseinheit zur Regelung der Lenkungs- und Antriebseinheit weitergeleitet. In Dymola wurde die Fahrt des SCVs in die Parklücke mit einer Odometrie zur Positionsbestimmung und der virtuellen Deichsel simuliert. Damit die Positionen während des Einparkvorgangs online berechnet werden können, wurden die Gleichungen der Odometrie in den diskreten Bereich übertragen und mit dem Runge-Kutta-Verfahren integriert. Die Positionsdaten der Odometrie-Implementierung wurden mit der Simulation verglichen, dabei war die Simulation, die Referenz für die Genauigkeit. Es wurden Abweichungen von 2,5 cm bei Größenordnungen von 1,2 bis 2 m festgestellt.

Der Einparkassistent wurde nach der Harel-Automaten-Notation modelliert. Für die Implementierung der Automaten wurde die AIRA-Laufzeitumgebung verwendet. Die Laufzeitumgebung verringert den Implementierungsaufwand, indem sie vorgefertigte Datenstrukturen und einen Reaktionsalgorithmus anbietet. Dabei übernimmt der Automat *Laserscanner* die Parametrisierung des Laserscanners und liest über die RS232/422 einen Scanzyklus aus und verschickt diese an die Automaten *Parkluecke* und *Einparken*. Die Parklückenauswertung wird von dem Automat *Parkluecke* durchgeführt, während der Automat *Einparken* die Odometrie und die Lenkwinkel Soll-Vorgaben berechnet. Der *Steuer_Parkassistent* Automat sendet dem Automaten *Einparken* die Ist-Lenkwinkel vom CAN Bus und überträgt die vom *Einparken* vorgegebenen Soll-Werte auf dem CAN.

Die Messungen der Ausführungszeiten der Automaten des Einparkassistenten haben gezeigt, dass der 650 MHz Prozessor des Kommunikationsrechners nicht ausgelastet wurde, und das eine um den Faktor 20 langsamere CPU eingesetzt werden kann. Als Konsequenz wurde der Einparkassistent auf einem ARM7TDMI μ-Controller mit 32 MHz portiert. Mit dem Wechsel auf dem μ-Controller wurde die nebenläufige Ausführung der Automaten durch ein sequentielles Verarbeitungskonzept für den Einparkassistenten entworfen.

Die Ergebnisse einer erneuten Timinganalyse zeigen, dass bei dem ARM7 Derivat die Ausführungszeiten bei Fließkomma- im Vergleich zur Fix-Point-Arithmetik um den Faktor 10 steigen. Da damit die Ausführungsdauer des Automaten *Einparken* für die Odometrie und den PD-Regler von 40 ms 3-mal länger ist als der Scanzyklus des Laserscanner T_{LS}, wurden Alternativen zur Reduzierung der Ausführungszeiten von Fließkomma-Arithmetik vorgestellt. Neben dem Austausch der CPU mit Fließkomma-Coprozessor wird die Einführung von Lookup-Tabellen vorgeschlagen. Nach einer Optimierung der Ausführungszeit könnte der Einparkassistent auf dem μ-Controller den vorher verwendeten Kommunikationsrechner ersetzten oder ergänzen. Alternativ könnte zusätzlich ein zeitgesteuertes, verteiltes Taskkonzept zur Verwaltung der Automaten eingeführt werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literatur

[1] Dynasim AB. *Dymola, Dynamic Modeling Laboratory, Users Manual*. Research Park Ideon, 2004.

[2] Christian Ameling. Steigerung der aktiven sicherheit von kraftfahrzeugen durch ein kollisionsvermeidungssystem. dissertation. Dissertation, Helmut-Schmidt Universität, 2002.

[3] Nelson Castillo Andres Calderon. Why arm's eabi matters. http://www.linuxdevices.com/articles/AT5920399313.html, 2008. [Online; zugegriffen am 23. Juni 2008].

[4] J. Deutscher B. Müller. Zweistufige trajektorienplanung für das automatische einparken. *Universität Erlangen*, 2006.

[5] Jacqueline Proulx Farrel Bradford Nichils, Dick Buttlar. *Pthreads Programming*. O'Reilly, 2002.

[6] David R. Butenhof. *Programming with POSIX Threads*. Addison-Wesley, 2007.

[7] Stefan Cordes. Automatischer bremsassistent auf basis einer laserscanner-abstandserfassung für ein fahrerloses transportsystem. Masterarbeit, Hochschule für Angewandte Wissenschaften Hamburg, 2006.

[8] Marco Cesati Daniel P. Bovet. *Understanding the Linux Kernel*. O'Reilly, 2000.

[9] Statistisches Bundesamt Deutschland. Unfälle und verunglückte im straßenverkehr. *Verkehrsunfälle*, 2007.

[10] PhyTec GmbH. phycore-at91m55800a. *Hardware Manual*, 2003.

[11] Nicolas Halbwachs. *Synchronous Programming of Reactive Systems*. Kluwer International Series in Engineering and Computer Science, 1993.

[12] Christian Laugier Igor E. Paromtchik. Motion generation and control for parking an autonomous vehicle. *International Conference on Robotics and Automation*, 1997.

[13] Infineon. Tc1130 family. http://www.infineon.com/cms/en/product/channel.html?channel=ff80808112ab681d0112ab6b7190082f, 2008. [Online; zugegriffen am 23. Juni 2008].

[14] Ning Liu John Alberts, Ingmar Gründel. Projektarbeit, master informatik, einparkassistent ws 2006/2007, 2006.

[15] Prof. Dipl. Inf. H. Kaltenhäuser. Interne dokumentation des aira-compilers, 2005.

[16] Lawrence Livermore National Laboratory. Posix threads programming. https://computing.llnl.gov/tutorials/pthreads/#Overview, 2008. [Online; zugegriffen am 29. Februar 2008].

[17] Advanced RISC Machines Ltd. The arm instruction set architecture. http://www.arm.com/products/CPUs/architecture.html, 2008. [Online; zugegriffen am 23. Juni 2008].

[18] Andreas Pröhl Oliver Neumann. Aira-runtime eine laufzeitumgebung für automaten in c für qnx, 2005.

[19] William H. Press. *Numerical Recipes in FORTRAN*. Cambridge University Press, 1994.

[20] Andreas Pröhl. Entscheidungsstrategien und steuerungskonzepte eines kollisionsvermeidungssystems. Masterarbeit, Hochschule für Angewandte Wissenschaften Hamburg, 2006.

[21] Maria Gensel Ralf Anske. Automatisches einparken. *Freie Universität Berlin*, 2005.

[22] Renesas. R32c/100 series. http://www.renesas.com/fmwk.jsp?cnt=r32c100_series_landing.jsp&fp=/products/mpumcu/m16c_family/r32c100_series/, 2008. [Online; zugegriffen am 23. Juni 2008].

[23] Denis Schetler. Automatischer ausweichassistent mit einer laserscanner - basierten abstandsregelung für ein fahrerloses transportsystem. Masterarbeit, Hochschule für Angewandte Wissenschaften Hamburg, 2007.

[24] Jörn Sellenthin. Ein zeitgesteuertes, verteiltes sw-konzept implementiert auf flexray-komponenten für ein fahrerloses transportsystem. Masterarbeit, Hochschule für Angewandte Wissenschaften Hamburg, 2006.

[25] Sick. Telegramme zur konfiguration und bedienung der lasermesssysteme lms2xx, 2005.

[26] Olaf Tetzlaff Stefan Cordes, Jörn Sellentin. Projektarbeit, entwicklung, implementierung und funktionstests der embedded hw-/sw-plattformen für den faust fahrbetrieb mit remotesteuerung. *Verkehrsunfälle*, 2007.

[27] Wikipedia. Prius wiki. http://www.priuswiki.de/wiki/Einparkautomatik, 2005. [Online; zugegriffen am 3. Januar 2008].

[28] www.motor talk.de. Park assist von volkswagen ab sofort im touran bestellbar. http://www.motor-talk.de/news/-park-assist-von-volkswagen-ab-sofort-im-touran-bestellbar-t1355956.html, 2007. [Online; zugegriffen am 3. Januar 2008].

[29] Oliver Wyman. Automobile sicherheitstechnik. http://www.oliverwyman.com/de/pdf_files/PM_Fahrzeugsicherheit_digital.pdf, 2004. [Online; zugegriffen am 3. Januar 2008].

[30] David Yuen. http://www.ele.auckland.ac.nz/ cyue001/tech/lmsintro.html, 2004.

A. Anhang

A.1. Pinbelegung des PhyCORE Extensionboards

Signal	phyCORE Module	Expansion Bus	Patch Field
PB0	30D	30D	10B
PB1	31C	31C	10F
PB2	31D	31D	11A
PB3 / IRQ4	32D	32D	11C
PB4 / IRQ5	33D	33D	11B
PB5 / IRQ6	50B	50B	44B
PB6 / AD0TRIG	45C	45C	15E
PB7 / AD1TRIG	44C	44C	15C
PB8	37B	37B	40A
PB9	38A	38A	40E
PB10	38B	38B	40B
PB11	39A	39A	40D
PB12	40A	40A	40F
PB13	40B	40B	41A
PB14	41A	41A	41E
PB15	41B	41B	41B
PB16	42B	42B	41F
PB17	43A	43A	42A
PB18 / BMS	43B	43B	42C
PB19 / TCLK0	44A	44A	42E
PB20 / TIOA0	45A	45A	42B
PB21 /TIOB0	45B	45B	42F
PB22 / TCLK1	46A	46A	43A
PB23 / TIOA1	46B	46B	43C
PB24 / TIOB1	47B	47B	43E
PB25 / TCLK2	48A	48A	43B
PB26 / TIOA2	48B	48B	43F
PB27 / TIOB2	49A	49A	44A

Abb. 66: Port B Pinbelegung des AT91S55800A und des Extensionboards [10]

A.2. Zustandsübergangstabellen

Zustand	Signal	Ereignis	Übergang in Zustand
IDLE	S_PL_Parkwunsch	Empfang der Nachricht zum Start der Parklückensuche	INIT_SCANNER
INIT_SCANNER	S_PL_Scanner_Fehler	Initialisierung des Laserscanners ist fehlgeschlagen	IDLE
INIT_SCANNER	S_PL_Abbruch	Initialisierung des Laserscanners wird abgebrochen	IDLE
INIT_SCANNER	S_PL_Laserscanner _nicht_verfuegbar	Ein Versuch den Laserscanner zu initialisieren ist fehlgeschlagen	-/nach dem 3 mal IDLE
INIT_SCANNER	S_PL_Laserscanner _bereit	Der Laserscanner ist initialisiert	SUCHEN
SUCHEN	S_PL_LaengeReichtAus	Die ermittelte Parklücke ist von der Länge und Tiefe ausreichend	ANBIETEN
SUCHEN	S_PL_Abbruch	Die Parklückensuche wird abgebrochen	STANDBY
SUCHEN	N_PL_Start_Ack	LMS bestätigt den Start der Parklücken- suche und überträgt den Pointer auf den Ringpuffer	-
SUCHEN	N_PL_Verfahrweg	Speicherung des zurück gelegten Weges	-
SUCHEN	S_PL_Data_Ready	LMS meldet, dass neue Messwerte im Ring puffer vorhanden sind	-
SUCHEN	S_PL_Timeout	Beim Laserscanner ist ein Timeout aufgetreten. Beim 3. mal Erzeugung S_PL_ScannerFehler Nachricht	-
SUCHEN	S_PL_ScannerFehler	Beim Laserscanner ist ein Fehler aufgetreten	IDLE

ANBIETEN	S_PL_Parkluecke_Verwerfen	Parklücke wird ab-gelehnt	SUCHEN
ANBIETEN	S_PL_ParklueckenDatenSenden	Parklücketiefe wird angefordert	STANDBY
ANBIETEN	N_PL_Verfahrweg	Speicherung des zurückgelegten Weges	-
ANBIETEN	N_PL_ParklueckeAnbieten	Übernahme der Parklücken tiefe aus Zustand SUCHEN	-
ANBIETEN	S_PL_VerfahrwegSenden	Anforderung des Verfahrwegs bis zur Startposition des Einparkvorgangs	-
STANDBY	S_PL_Parkwunsch	Empfang der Nachricht zum Starten der Parklückensuche	SUCHEN

Tabelle 1: Zustandübergangstabelle für das Automat Parkluecke

Zustand	Signal	Ereignis	Übergang in Zustand
IDLE	N_LMS_Init	Empfang der Nachricht zur Parametrisierung des Laserscanners	INIT
INIT	S_LMS_Start	Start der Messwertübertragung	SCAN
INIT	S_LMS_Stop	Abbruch der Initialisierung	IDLE
INIT	S_LMS_Fehler	Bei der Initialisierung des Laserscanners ist ein Fehler aufgetreten	IDLE
SCAN	S_LMS_Stop	Abbruch der Messwerterfassung	IDLE
SCAN	S_LMS_Pause	Unterbrechung der Messwertübertragung	-
SCAN	S_LMS_Start	Fortsetzung der Messwertübertragung	-
SCAN	Timer_Tick	Überprüfe die RS422 Schnittstelle auf neue Messwerte vom Laserscanner	-

Tabelle 2: Zustandübergangstabelle für das Automat Laserscanner

EINPARKEN_STANDBY	S_EP_Unterbrechen	Der Einparkvorgang wird unterbrochen	-
EINPARKEN_STANDBY	N_EP_speichere _Geschwindigkeit	Die Geschwindigkeit vor der Unterbrechung des Einparkvorgang wird gespeichert	-
EINPARKEN_STANDBY	S_EP_Weiter _Parken	Der Einparkvorgang wird fortgesetzt	EINPARKEN

Zustand	Signal	Ereignis	Übergang in Zustand
IDLE	N_EP_Verfahrweg	Empfang und Weiterleitung des Verfahrwegs bis zur Startposition des Einparkvorgangs	X_POS_ER.
X_POS_ER.	N_EP_Verfahrweg	Empfang des Verfahrwegs bis zur Startposition	-
X_POS_ER.	S_EP_Unterbrechen	Unterbrechung des Einparkvorgangs	X_POS_ER. _STANDBY
X_POS_ER.	S_EP_Abbrechen	Abbruch des Einparkvorgangs	IDLE
X_POS_ER.	S_EP_Xposerreicht	Startposition des Einparkvorgangs ist erreicht	EINPARKEN
X_POS_ER.	Timer_Tick	Berechne den verbleibenden Verfahrweg bis die Startposition erreicht ist	-
X_POS_ER. _STANDBY	S_EP_Unterbrechen	Die Fahrt in die Startposition wird unterbrochen	-
X_POS_ER. _STANDBY	N_EP_speichere _Geschwindigkeit	Die Geschwindigkeit vor der Unterbrechung wird gespeichert	-
X_POS_ER. _STANDBY	S_EP_Weiter _Parken	Die Fahrt in die Startposition wird fortgesetzt	-
EINPARKEN	S_EP_Abbrechen	Der Einparkvorgang wird abgebrochen	IDLE
EINPARKEN	S_EP_Endposerreicht	Der Einparkvorgang ist abgeschlossen	IDLE
EINPARKEN	S_EP_Unterbrechen	Der Einparkvorgang wird untergebrochen	EINPARKEN _STANDBY
EINPARKEN	N_EP_Abstandzurwand	Die Tiefe der Parklücke wird empfangen	-
EINPARKEN	Timer_Tick	Der nächste berechnete Lenkwinkel wird versendet	-

Tabelle 3: Zustandübergangstabelle für das Einparken Automat

Zustand	Signal	Ereignis	Übergang in Zustand
IDLE	S_STP_Parkwunsch _anmelden	Empfang der Nachricht zum Start der Parklückensuche	PL_SUCHEN
PL_SUCHEN	S_STP_ScannerFehler	Beim Laserscanner ist ein Fehler aufgetreten	IDLE
PL_SUCHEN	S_STP_Parkwunsch _abgebrochen	Der Einparkvorgang ist vom Fahrer abgebrochen worden.	IDLE
PL_SUCHEN	N_STP_ParkenlueckenDaten	Empfang der Tiefe der Parklücke	WARTE_AUF_GAS
PL_SUCHEN	S_STP_Parkenluecke _annehmen	Die Parklücke wird vom Fahrer angenommen	-
PL_SUCHEN	S_STP_Parkenluecke _ablehnen	Die Parklücke wird vom Fahrer abgelehnt	-
PL_SUCHEN	S_STP_Parkenluecke _gefunden	Es wurde eine Parklücke für den Einparkvorgang gefunden	-
PL_SUCHEN	S_STP_Parkwunsch _abbrechen	Der Einparkvorgang wird vom Fahrer abgebrochen	-
PL_SUCHEN	N_STP_Verfahrweg	Der Verfahrweg bis zum Start des Einparkvorgangs wird angefordert	-
WARTE _AUF_GAS	S_STP_ScannerFehler	Beim Laserscanner ist ein Fehler aufgetreten	IDLE
WARTE _AUF_GAS	S_STP_Parkwunsch _abbrechen	Der Einparkvorgang wird vom Fahrer abgebrochen	-
WARTE _AUF_GAS	S_STP_set_V_und_W _von_LST[V>5]	Das Betätigen des Fußpedals hat den Schwellwert von 5 mm/s überschritten	EINPARKEN _TICK
WARTE _AUF_GAS	Timer_Tick	Sende den Lenkwinkel und die Geschwindigkeit an die Antriebseinheit	-

Tabelle 4: Zustandübergangstabelle für das Steuer_Parkassistent Automat

EINPARKEN_TICK	N_STP_set_V	Setze Geschwindigkeit des SCVs	-
EINPARKEN_TICK	N_STP_set_W	Setze Lenkwinkel des SCVs	-
EINPARKEN_TICK	S_STP_Parkwunsch-abbrechen	Der Einparkvorgang wird vom Fahrer abgebrochen	-
EINPARKEN_TICK	S_STP_set_V_ und_W _von_LST [V=<5]	Das Gaspedal ist nicht betätigt und der Einparkvorgang soll unterbrochen werden	-
EINPARKEN_TICK	S_STP_set_V_ und_W _von_LST [V>5 AND unterbrochen==true]	Das Gaspedal ist betätigt und der Einparkvorgang soll fortgeführt werden	-
EINPARKEN_TICK	Timer_Tick	Die Parklücke wird vom Fahrer abgelehnt	-
EINPARKEN_TICK	S_STP_Eingeparkt	Der Parkvorgang ist abgeschlossen	WARTE_AUF _GAS_NULL
WARTE _AUF_GAS _NULL	S_STP_ScannerFehler	Beim Laserscanner ist ein Fehler aufgetreten	IDLE
WARTE _AUF_GAS _NULL	S_STP_Parkwunsch _abbrechen	Der Einparkvorgang wird vom Fahrer abgebrochen	IDLE
WARTE _AUF_GAS _NULL	S_STP_set_V_und_W _von_LST [V > 5]	Das Betätigen des Fußpedals hat den Schwellwert von 5 mm/s unterschritten	IDLE

A.3. Abweichungen der Odometrie und der virtuellen Deichsel, mit ChScite und Dymola

Time	X_dl	Y_dl	Alpha_L	Theta_L	X_m	X_p	Y_m	Y_p
0	2440	971	0,981315	0,028687	-760	0	0	0
0,05	2426	971	0,95654	0,0558	-745	14	-20	1
0,1	2411	970	0,932138	0,082446	-729	29	-39	2
0,15	2396	968	0,908107	0,108618	-712	44	-58	3
0,2	2381	966	0,884444	0,134307	-694	59	-75	5
0,25	2365	964	0,861144	0,159507	-675	75	-92	7
0,3	2349	961	0,838206	0,184211	-656	92	-107	10
0,35	2332	958	0,815626	0,208415	-636	108	-122	13
0,4	2315	954	0,793399	0,232113	-615	125	-136	17
0,45	2298	950	0,771523	0,255303	-593	142	-149	22
0,5	2280	945	0,749992	0,277981	-571	160	-160	26
0,55	2263	940	0,728803	0,300144	-549	177	-171	32
0,6	2245	934	0,70795	0,321792	-526	195	-181	37
0,65	2227	928	0,687429	0,342924	-503	213	-190	44
0,7	2208	921	0,667234	0,363537	-479	232	-198	50
0,75	2190	914	0,64736	0,383633	-455	250	-205	58
0,8	2171	906	0,6278	0,403212	-431	269	-211	65
0,85	2153	898	0,608547	0,422274	-406	287	-216	73
0,9	2134	889	0,589595	0,44082	-382	306	-220	82
0,95	2115	880	0,570938	0,458853	-357	325	-224	91
1	2096	871	0,552566	0,476373	-332	344	-226	101
1,05	2077	861	0,534471	0,493382	-307	363	-227	111
1,1	2058	850	0,516646	0,509883	-282	382	-228	121
1,15	2039	839	0,499081	0,525878	-257	401	-228	132
1,2	2020	828	0,481767	0,54137	-232	420	-227	143
1,25	2001	817	0,464694	0,55636	-207	439	-225	155
1,3	1982	805	0,447851	0,570852	-182	458	-222	167
1,35	1963	792	0,431227	0,584848	-157	477	-219	179
1,4	1944	779	0,414812	0,59835	-133	496	-215	192
1,45	1926	766	0,398592	0,61136	-108	514	-210	205
1,5	1907	753	0,382555	0,623882	-84	533	-204	218
1,55	1888	739	0,366688	0,635917	-60	552	-198	232
1,6	1869	725	0,350978	0,647468	-36	571	-191	246
1,65	1851	711	0,335408	0,658535	-12	589	-183	260
1,7	1832	696	0,319965	0,669121	12	608	-175	275
1,75	1813	682	0,304631	0,679226	35	627	-166	290
1,8	1795	666	0,28939	0,688852	58	645	-156	305
1,85	1776	651	0,274225	0,697998	81	664	-146	320
1,9	1758	636	0,259115	0,706665	104	682	-136	336
1,95	1740	620	0,244042	0,714852	126	700	-125	352
2	1721	604	0,228983	0,722559	148	719	-113	368

2,05	1703	588	0,213918	0,729783	170	737	-101	384
2,1	1685	571	0,198823	0,736522	192	755	-89	400
2,15	1667	555	0,183672	0,742774	213	773	-76	417
2,2	1649	538	0,168439	0,748534	234	791	-62	433
2,25	1631	521	0,153097	0,753799	255	809	-48	450
2,3	1613	504	0,137617	0,758563	276	827	-34	467
2,35	1595	487	0,121965	0,762819	296	845	-19	484
2,4	1577	470	0,106111	0,766562	316	863	-4	501
2,45	1559	452	0,090018	0,769782	335	881	12	519
2,5	1541	435	0,07365	0,77247	354	899	28	536
2,55	1523	418	0,056968	0,774616	373	917	44	554
2,6	1505	400	0,039932	0,776209	392	935	60	571
2,65	1487	383	0,022498	0,777235	411	953	77	589
2,7	1470	365	0,004622	0,777681	429	970	95	606
2,75	1452	348	-0,013741	0,777531	446	988	112	624
2,8	1434	330	-0,032636	0,776768	464	1006	130	641
2,85	1416	313	-0,052112	0,775374	481	1024	149	659
2,9	1398	295	-0,072214	0,77333	498	1042	167	676
2,95	1381	278	-0,092986	0,770616	514	1059	186	694
3	1363	260	-0,114469	0,767209	530	1077	205	711
3,05	1345	243	-0,136701	0,763088	546	1095	225	728
3,1	1327	226	-0,159712	0,75823	561	1113	244	745
3,15	1309	209	-0,183521	0,752611	576	1131	264	762
3,2	1291	193	-0,208137	0,746209	591	1149	285	779
3,25	1273	176	-0,233555	0,739001	605	1167	305	795
3,3	1255	160	-0,259749	0,730967	619	1185	326	811
3,35	1237	144	-0,286674	0,722088	632	1203	347	827
3,4	1219	128	-0,314259	0,712348	646	1221	368	843
3,45	1201	113	-0,342408	0,701735	658	1239	390	858
3,5	1183	98	-0,370999	0,690243	671	1257	411	873
3,55	1165	83	-0,399882	0,677868	683	1275	433	888
3,6	1147	69	-0,428883	0,664616	694	1293	455	902
3,65	1129	55	-0,45781	0,650496	706	1311	478	916
3,7	1112	42	-0,486454	0,635525	717	1328	500	930
3,75	1094	29	-0,514599	0,619728	727	1346	523	942
3,8	1076	16	-0,542032	0,603135	738	1364	546	955
3,85	1059	5	-0,568545	0,58578	748	1381	568	967
3,9	1041	-7	-0,593949	0,567705	758	1399	591	978
3,95	1024	-18	-0,618077	0,548955	768	1416	614	989
4	1006	-28	-0,640787	0,529576	778	1434	637	999
4,05	989	-38	-0,661967	0,509619	787	1451	660	1009
4,1	972	-47	-0,681535	0,489133	797	1468	683	1019
4,15	955	-56	-0,699439	0,468169	807	1485	706	1027
4,2	938	-65	-0,715652	0,446775	817	1502	729	1036

4,25	921	-73	-0,730172	0,425001	827	1519	752	1044
4,3	904	-80	-0,743016	0,402891	837	1536	775	1051
4,35	887	-87	-0,754218	0,380491	847	1553	798	1058
4,4	870	-93	-0,763824	0,357842	858	1570	820	1065
4,45	853	-100	-0,771891	0,334983	869	1587	843	1071
4,5	836	-105	-0,778482	0,311952	880	1604	865	1077
4,55	819	-111	-0,783663	0,288783	892	1621	887	1082
4,6	802	-115	-0,787504	0,26551	904	1638	909	1087
4,65	785	-120	-0,790074	0,242163	917	1655	931	1091
4,7	768	-124	-0,791441	0,218771	930	1672	952	1095
4,75	751	-127	-0,791673	0,19536	943	1689	973	1099
4,8	734	-131	-0,790834	0,171957	957	1706	994	1102
4,85	716	-133	-0,788985	0,148586	972	1724	1014	1105
4,9	699	-136	-0,786184	0,125268	987	1741	1034	1107
4,95	681	-138	-0,782484	0,102027	1003	1759	1053	1109
5	663	-139	-0,777937	0,078881	1019	1777	1073	1111
5,05	646	-141	-0,772587	0,055852	1035	1794	1091	1112
5,1	628	-141	-0,766478	0,032958	1052	1812	1109	1113
5,15	610	-142	-0,759646	0,010218	1070	1830	1127	1113
5,2	591	-142	-0,752124	-0,012351	1088	1849	1144	1113

Tabelle 5: Fahrzeugparameter Dymola virtuelle Deichsel, Startabstand=1400

Time	X_dl	Y_dl	Alpha_L	Theta_L	X_m	X_p	Y_m	Y_p
0,00	2440	971	0,981315	0,028687	-760	0	0	0
0,05	2427	971	0,956202	0,056030	-745	13	-20	1
0,10	2412	970	0,931490	0,082905	-729	28	-39	2
0,15	2398	968	0,907163	0,109304	-712	42	-58	3
0,20	2383	967	0,883218	0,135217	-695	57	-75	5
0,25	2367	964	0,859651	0,160637	-676	73	-92	7
0,30	2352	962	0,836458	0,185559	-657	88	-108	10
0,35	2335	958	0,813635	0,209976	-636	105	-123	13
0,40	2319	955	0,791178	0,233883	-615	121	-136	17
0,45	2302	950	0,769082	0,257277	-594	138	-149	21
0,50	2285	946	0,747343	0,280155	-572	155	-161	25
0,55	2267	941	0,725955	0,302513	-549	173	-172	31
0,60	2249	935	0,704914	0,324350	-526	191	-182	36
0,65	2232	929	0,684213	0,345665	-503	208	-191	42
0,70	2213	922	0,663847	0,366457	-479	227	-199	49
0,75	2195	915	0,643810	0,386725	-455	245	-206	56
0,80	2177	907	0,624095	0,406470	-430	263	-213	64
0,85	2158	899	0,604695	0,425692	-406	282	-218	72
0,90	2140	891	0,585603	0,444393	-381	300	-222	81
0,95	2121	882	0,566811	0,462574	-356	319	-225	90
1,00	2102	872	0,548312	0,480237	-331	338	-228	99

1,05	2083	862	0,530096	0,497383	-306	357	-229	109
1,10	2064	852	0,512154	0,514015	-280	376	-230	119
1,15	2045	841	0,494479	0,530136	-255	395	-229	130
1,20	2026	830	0,477058	0,545747	-230	414	-228	141
1,25	2007	818	0,459883	0,560851	-205	433	-226	153
1,30	1988	806	0,442942	0,575451	-180	452	-224	165
1,35	1969	794	0,426224	0,589550	-155	471	-220	177
1,40	1950	781	0,409718	0,603149	-130	490	-216	190
1,45	1931	768	0,393410	0,616253	-105	509	-211	203
1,50	1913	755	0,377288	0,628863	-81	527	-205	216
1,55	1894	741	0,361338	0,640981	-56	546	-199	230
1,60	1875	727	0,345545	0,652610	-32	565	-192	244
1,65	1856	713	0,329895	0,663752	-8	584	-184	258
1,70	1838	698	0,314372	0,674408	15	602	-176	273
1,75	1819	683	0,298960	0,684580	39	621	-167	288
1,80	1801	668	0,283640	0,694268	62	639	-158	303
1,85	1782	653	0,268394	0,703474	85	658	-147	319
1,90	1764	637	0,253204	0,712197	108	676	-137	334
1,95	1745	621	0,238048	0,720438	131	695	-126	350
2,00	1727	605	0,222905	0,728194	153	713	-114	366
2,05	1709	589	0,207752	0,735466	175	731	-102	382
2,10	1691	572	0,192565	0,742251	196	749	-89	399
2,15	1673	556	0,177319	0,748546	218	767	-76	415
2,20	1654	539	0,161986	0,754349	239	786	-62	432
2,25	1636	522	0,146538	0,759654	260	804	-48	449
2,30	1619	505	0,130944	0,764457	280	821	-34	466
2,35	1601	488	0,115172	0,768752	300	839	-19	483
2,40	1583	471	0,099188	0,772532	320	857	-4	501
2,45	1565	453	0,082955	0,775790	340	875	12	518
2,50	1547	436	0,066436	0,778515	359	893	28	535
2,55	1529	418	0,049590	0,780699	378	911	44	553
2,60	1512	401	0,032375	0,782330	397	928	61	571
2,65	1494	383	0,014747	0,783394	415	946	78	588
2,70	1476	365	-0,003340	0,783880	433	964	95	606
2,75	1459	348	-0,021935	0,783770	451	981	113	623
2,80	1441	330	-0,041085	0,783048	468	999	131	641
2,85	1423	313	-0,060840	0,781697	485	1017	149	659
2,90	1405	295	-0,081249	0,779697	502	1035	168	676
2,95	1388	278	-0,102359	0,777027	518	1052	187	694
3,00	1370	260	-0,124214	0,773666	534	1070	206	711

3,05	1352	243	-0,146854	0,769591	549	1088	226	729
3,10	1334	226	-0,170313	0,764777	564	1106	246	746
3,15	1317	208	-0,194614	0,759202	579	1123	266	763
3,20	1299	192	-0,219769	0,752840	594	1141	286	780
3,25	1281	175	-0,245774	0,745669	608	1159	306	796
3,30	1263	158	-0,272605	0,737666	621	1177	327	813
3,35	1245	142	-0,300215	0,728809	634	1195	348	829
3,40	1228	126	-0,328533	0,719081	647	1212	370	845
3,45	1210	111	-0,357457	0,708468	660	1230	391	861
3,50	1192	95	-0,386857	0,696958	671	1248	413	876
3,55	1174	81	-0,416570	0,684547	683	1266	435	891
3,60	1156	66	-0,446409	0,671237	694	1284	457	905
3,65	1139	52	-0,476160	0,657036	705	1301	479	919
3,70	1121	38	-0,505592	0,641958	715	1319	502	933
3,75	1103	25	-0,534466	0,626026	725	1337	524	946
3,80	1086	13	-0,562543	0,609270	735	1354	547	959
3,85	1068	1	-0,589592	0,591726	745	1372	570	971
3,90	1051	-11	-0,615403	0,573436	754	1389	593	982
3,95	1034	-22	-0,639792	0,554446	763	1406	616	993
4,00	1017	-33	-0,662605	0,534807	772	1423	639	1004
4,05	999	-43	-0,683726	0,514571	781	1441	662	1014
4,10	983	-52	-0,703075	0,493792	790	1457	685	1023
4,15	966	-61	-0,720603	0,472523	799	1474	708	1032
4,20	949	-70	-0,736295	0,450818	807	1491	731	1041
4,25	932	-78	-0,750162	0,428728	817	1508	754	1049
4,30	916	-85	-0,762238	0,406301	826	1524	776	1056
4,35	899	-92	-0,772574	0,383586	835	1541	799	1063
4,40	882	-99	-0,781233	0,360626	845	1558	822	1070
4,45	866	-105	-0,788289	0,337463	855	1574	844	1076
4,50	849	-110	-0,793819	0,314136	866	1591	866	1082
4,55	833	-116	-0,797905	0,290681	877	1607	889	1087
4,60	816	-121	-0,800628	0,267132	888	1624	910	1092
4,65	800	-125	-0,802068	0,243520	900	1640	932	1096
4,70	783	-129	-0,802303	0,219876	912	1657	953	1100
4,75	766	-132	-0,801407	0,196226	925	1674	974	1104
4,80	749	-136	-0,799450	0,172596	938	1691	995	1107
4,85	732	-138	-0,796498	0,149012	952	1708	1015	1110
4,90	715	-141	-0,792612	0,125495	966	1725	1035	1112
4,95	698	-143	-0,787849	0,102067	981	1742	1055	1114
5,00	681	-144	-0,782260	0,078750	996	1759	1074	1116
5,05	664	-146	-0,775893	0,055563	1012	1776	1093	1117
5,10	646	-146	-0,768788	0,032526	1029	1794	1111	1118
5,15	628	-147	-0,760985	0,009655	1046	1812	1129	1118

Tabelle 6: Fahrzeugparameter ChScite virtuelle Deichsel, Startabstand=1400

A.4. Abweichungen des PD-Reglers und der virtuellen Deichsel, mit ChScite

Time	X_dl	Y_dl	Alpha_L	Theta_L	X_m	X_p	Y_m	Y_p
0	1680	943	1,15159	0	0	760	0	0
0,05	1670	942	1,12399	0,0298575	11	770	-23	0
0,1	1658	942	1,09679	0,0593255	23	782	-44	1
0,15	1647	941	1,06999	0,0883883	36	793	-66	2
0,2	1635	940	1,04359	0,117031	51	805	-86	3
0,25	1622	938	1,01759	0,14524	66	818	-106	4
0,3	1609	936	0,991978	0,173003	83	831	-124	7
0,35	1595	933	0,966765	0,200308	100	845	-142	9
0,4	1581	930	0,941948	0,227147	119	859	-159	12
0,45	1566	927	0,917524	0,253509	138	874	-175	16
0,5	1551	923	0,893491	0,279386	158	889	-190	20
0,55	1536	918	0,869847	0,304772	179	904	-204	24
0,6	1521	913	0,84659	0,329661	200	919	-216	30
0,65	1505	907	0,823716	0,354047	222	935	-228	35
0,7	1489	901	0,801222	0,377926	245	951	-239	41
0,75	1472	894	0,779103	0,401295	268	968	-249	48
0,8	1456	887	0,757356	0,42415	291	984	-258	55
0,85	1439	880	0,735974	0,44649	315	1001	-265	63
0,9	1423	871	0,714953	0,468313	339	1018	-272	71
0,95	1406	862	0,694285	0,489618	364	1034	-277	80
1	1389	853	0,673966	0,510405	388	1051	-282	89
1,05	1372	843	0,653986	0,530674	413	1068	-286	99
1,1	1354	833	0,634339	0,550427	438	1086	-288	109
1,15	1337	822	0,615015	0,569662	463	1103	-290	120
1,2	1320	811	0,596006	0,588383	488	1120	-290	131
1,25	1303	799	0,577301	0,606591	513	1137	-290	143
1,3	1286	787	0,558889	0,624287	538	1154	-289	155
1,35	1268	775	0,54076	0,641474	563	1172	-287	168
1,4	1251	762	0,522901	0,658153	588	1189	-284	181
1,45	1234	748	0,505299	0,674327	612	1206	-280	194
1,5	1217	734	0,48794	0,689998	637	1223	-275	208
1,55	1200	720	0,470809	0,705168	661	1240	-270	223
1,6	1183	705	0,453889	0,719839	685	1257	-264	237
1,65	1166	690	0,437163	0,734013	710	1274	-257	252
1,7	1149	675	0,420612	0,747691	733	1291	-249	267
1,75	1133	659	0,404215	0,760874	757	1307	-241	283
1,8	1116	643	0,387952	0,773564	780	1324	-232	299
1,85	1100	627	0,371797	0,78576	803	1340	-222	316
1,9	1083	610	0,355725	0,797462	826	1357	-212	332
1,95	1067	593	0,339707	0,80867	848	1373	-201	349
2	1051	576	0,323712	0,819381	871	1389	-189	366
2,05	1035	559	0,307704	0,829593	892	1405	-177	384
2,1	1019	541	0,291647	0,839303	914	1421	-164	401
2,15	1003	523	0,275497	0,848506	935	1437	-151	419
2,2	987	505	0,259207	0,857196	956	1453	-137	437

2,25	971	487	0,242723	0,865364	976	1469	-123	456
2,3	955	468	0,225986	0,873003	996	1485	-108	474
2,35	940	449	0,208928	0,8801	1016	1500	-93	493
2,4	924	430	0,191471	0,886642	1036	1516	-77	512
2,45	909	411	0,173527	0,892612	1055	1531	-61	531
2,5	893	392	0,154996	0,897992	1073	1547	-44	550
2,55	878	373	0,135759	0,902759	1091	1562	-27	570
2,6	863	353	0,115684	0,906886	1109	1577	-9	589
2,65	847	334	0,094613	0,91034	1127	1593	9	609
2,7	832	314	0,0723652	0,913087	1143	1608	27	629
2,75	817	294	0,0487291	0,915081	1160	1623	46	648
2,8	802	274	0,0234589	0,916272	1176	1638	65	668
2,85	786	254	-0,00373096	0,916602	1191	1654	85	688
2,9	771	235	-0,03317	0,916001	1206	1669	105	708
2,95	756	215	-0,065235	0,91439	1220	1684	126	728
3	741	195	-0,100349	0,911678	1234	1699	147	747
3,05	725	175	-0,138974	0,90776	1247	1715	168	767
3,1	710	156	-0,181593	0,90252	1259	1730	190	786
3,15	695	137	-0,228668	0,895829	1270	1745	212	806
3,2	680	118	-0,280578	0,887552	1280	1760	235	824
3,25	665	100	-0,337517	0,877553	1290	1775	258	843
3,3	650	82	-0,399365	0,865707	1298	1790	282	861
3,35	635	65	-0,465547	0,851915	1305	1805	306	878
3,4	620	48	-0,53494	0,836123	1310	1820	330	894
3,45	606	33	-0,605876	0,818337	1315	1834	355	910
3,5	592	18	-0,676316	0,79863	1318	1848	380	924
3,55	579	5	-0,744146	0,777142	1319	1861	405	938
3,6	566	-8	-0,807519	0,754062	1320	1874	430	950
3,65	554	-19	-0,865095	0,729605	1320	1886	455	961
3,7	542	-29	-0,916136	0,703991	1319	1898	480	972
3,75	531	-39	-0,960442	0,677424	1317	1909	505	981
3,8	520	-47	-0,998219	0,650086	1315	1920	530	990
3,85	509	-55	-1,02992	0,622133	1313	1931	555	997
3,9	499	-62	-1,05614	0,593693	1311	1941	579	1005
3,95	489	-69	-1,07747	0,56487	1309	1951	604	1011
4	479	-75	-1,09453	0,535751	1308	1961	629	1017
4,05	469	-80	-1,10786	0,506404	1306	1971	654	1023
4,1	459	-86	-1,11796	0,476888	1306	1981	679	1028
4,15	449	-90	-1,12523	0,447248	1305	1991	704	1033
4,2	440	-95	-1,13006	0,417523	1306	2000	729	1037
4,25	430	-99	-1,13276	0,387746	1306	2010	754	1042
4,3	420	-103	-1,13359	0,357946	1308	2020	779	1046
4,35	410	-107	-1,13278	0,328146	1310	2030	804	1049
4,4	400	-110	-1,13054	0,298368	1314	2040	829	1052
4,45	390	-113	-1,12703	0,268631	1317	2050	854	1055
4,5	379	-116	-1,12239	0,238951	1322	2061	878	1058
4,55	369	-118	-1,11677	0,209345	1328	2071	902	1060
4,6	358	-120	-1,11026	0,179827	1334	2082	927	1063

4,65	347	-122	-1,10298	0,15041	1342	2093	951	1064
4,7	336	-123	-1,095	0,121107	1350	2104	974	1066
4,75	324	-125	-1,08642	0,0919281	1359	2116	997	1067
4,8	312	-126	-1,07731	0,0628852	1369	2128	1020	1068
4,85	301	-126	-1,06774	0,0339879	1380	2139	1043	1069
4,9	288	-126	-1,05778	0,00524564	1392	2152	1065	1069
4,95	276	-126	-1,04749	-0,0233331	1404	2164	1087	1069
5	263	-126	-1,03695	-0,0517405	1418	2177	1108	1068
5,05	251	-125	-1,02621	-0,0799696	1432	2189	1128	1067
5,1	238	-124	-1,01534	-0,108015	1447	2202	1148	1066
5,15	224	-122	-1,00441	-0,13587	1463	2216	1168	1065
5,2	211	-120	-0,993489	-0,163533	1479	2229	1186	1063

Tabelle 7: ChScite Deichsel, Startabstand=1400

Time	X_dl	Y_dl	Alpha_L	Theta_L	X_m	X_p	Y_m	Y_p
0	1680	943	0,930883	0,027669	0	760	0	0
0,05	1665	942	0,905384	0,054811	16	775	-19	1
0,1	1650	941	0,880613	0,081422	33	790	-38	2
0,15	1635	939	0,856552	0,107502	50	805	-55	3
0,2	1619	938	0,833178	0,133052	69	821	-72	5
0,25	1602	935	0,810472	0,158072	88	838	-88	7
0,3	1585	932	0,788416	0,182565	109	855	-103	10
0,35	1568	929	0,766991	0,206535	129	872	-117	14
0,4	1551	925	0,746177	0,229985	151	889	-130	17
0,45	1533	921	0,72596	0,252921	173	907	-142	22
0,5	1515	916	0,70632	0,275347	195	925	-153	26
0,55	1497	911	0,687241	0,29727	218	943	-163	32
0,6	1479	905	0,668709	0,318696	242	961	-172	37
0,65	1460	899	0,650706	0,339632	266	980	-181	44
0,7	1441	892	0,633218	0,360086	290	999	-188	50
0,75	1423	885	0,61623	0,380065	314	1017	-195	58
0,8	1404	877	0,599727	0,399577	338	1036	-200	65
0,85	1385	869	0,583697	0,41863	363	1055	-205	73
0,9	1366	860	0,568125	0,437232	388	1074	-209	82
0,95	1347	851	0,552998	0,455392	413	1093	-212	91
1	1327	842	0,538303	0,473118	438	1113	-214	100
1,05	1308	832	0,524029	0,490419	463	1132	-215	110
1,1	1289	822	0,510163	0,507304	489	1151	-216	121
1,15	1270	811	0,496693	0,523781	514	1170	-216	131
1,2	1251	800	0,483609	0,539859	539	1189	-215	142
1,25	1232	789	0,470898	0,555547	564	1208	-213	154
1,3	1213	777	0,458551	0,570853	589	1227	-211	166

1,35	1194	765	0,446558	0,585786	614	1246	-208	178
1,4	1175	752	0,434907	0,600355	639	1265	-205	190
1,45	1157	739	0,423589	0,614567	664	1283	-200	203
1,5	1138	726	0,412594	0,628432	689	1302	-195	217
1,55	1119	712	0,401915	0,641957	713	1321	-190	230
1,6	1101	698	0,39154	0,655151	738	1339	-184	244
1,65	1082	684	0,381462	0,668022	762	1358	-177	258
1,7	1064	670	0,371672	0,680577	786	1376	-170	273
1,75	1046	655	0,362163	0,692826	810	1394	-162	287
1,8	1028	640	0,352925	0,704774	834	1412	-154	302
1,85	1010	625	0,343951	0,716431	857	1430	-145	318
1,9	992	609	0,335234	0,727802	881	1448	-136	333
1,95	974	594	0,326766	0,738896	904	1466	-126	349
2	957	577	0,31854	0,74972	927	1483	-116	365
2,05	939	561	0,310549	0,760281	950	1501	-105	381
2,1	922	545	0,302787	0,770585	973	1518	-94	398
2,15	905	528	0,295247	0,78064	995	1535	-83	414
2,2	888	511	0,287922	0,790451	1017	1552	-71	431
2,25	871	494	0,280807	0,800026	1039	1569	-59	448
2,3	854	477	0,273895	0,80937	1061	1586	-47	466
2,35	837	459	0,267181	0,81849	1083	1603	-34	483
2,4	821	442	0,260659	0,827392	1104	1619	-20	501
2,45	804	424	0,254323	0,836081	1125	1636	-7	519
2,5	788	406	0,248169	0,844563	1146	1652	7	537
2,55	772	388	0,24219	0,852845	1167	1668	21	555
2,6	756	369	0,236383	0,860931	1188	1684	35	573
2,65	740	351	0,230741	0,868826	1208	1700	50	592
2,7	724	332	0,225261	0,876537	1228	1716	65	610
2,75	709	313	0,219938	0,884067	1248	1731	80	629
2,8	693	294	0,214766	0,891423	1268	1747	96	648
2,85	678	275	0,209743	0,898608	1288	1762	112	667
2,9	662	256	0,204863	0,905627	1307	1778	128	686
2,95	647	237	0,200123	0,912486	1326	1793	144	706
3	632	217	0,195518	0,919187	1345	1808	160	725
3,05	617	198	0,191045	0,925737	1364	1823	177	745
3,1	602	178	0,1867	0,932138	1383	1838	194	764
3,15	588	158	0,182479	0,938396	1401	1852	211	784
3,2	573	138	0,178378	0,944514	1419	1867	228	804
3,25	559	118	0,174395	0,950495	1437	1881	245	824
3,3	544	98	0,170526	0,956345	1455	1896	263	844
3,35	530	78	0,166768	0,962066	1473	1910	281	864
3,4	516	58	0,163117	0,967662	1491	1924	299	885
3,45	502	37	0,15957	0,973136	1508	1938	317	905
3,5	488	17	0,156125	0,978493	1525	1952	335	926

3,55	474	-4	-1,761487	0,983735	1542	1966	353	946
3,6	461	-24	-1,712238	0,950008	1519	1979	363	966
3,65	463	-20	-1,664398	0,916054	1496	1977	374	962
3,7	465	-17	-1,617926	0,881953	1475	1975	387	960
3,75	467	-15	-1,572782	0,847783	1454	1973	401	958
3,8	468	-15	-1,52893	0,813613	1435	1972	417	957
3,85	468	-15	-1,486331	0,779508	1417	1972	435	957
3,9	467	-15	-1,44495	0,745525	1401	1973	454	958
3,95	465	-17	-1,404752	0,711719	1386	1975	473	959
4	463	-19	-1,365704	0,678138	1373	1977	494	961
4,05	460	-21	-1,327772	0,644825	1361	1980	516	964
4,1	456	-24	-1,290925	0,61182	1351	1984	539	967
4,15	451	-28	-1,255131	0,579158	1342	1989	562	970
4,2	445	-31	-1,220361	0,546871	1335	1995	586	974
4,25	439	-35	-1,186585	0,514986	1330	2001	610	978
4,3	432	-39	-1,153775	0,483527	1326	2008	634	982
4,35	423	-43	-1,121902	0,452516	1324	2017	659	986
4,4	414	-48	-1,090942	0,421971	1324	2026	684	990
4,45	405	-52	-1,060866	0,391906	1325	2035	708	994
4,5	394	-56	-1,03165	0,362336	1327	2046	733	999
4,55	383	-60	-1,00327	0,333271	1331	2057	757	1003
4,6	371	-64	-0,975701	0,304719	1337	2069	781	1007
4,65	358	-68	-0,94892	0,276688	1344	2082	805	1011
4,7	345	-72	-0,922906	0,249182	1352	2095	828	1014
4,75	331	-75	-0,897635	0,222205	1361	2109	850	1018
4,8	317	-78	-0,873086	0,195758	1372	2123	873	1021
4,85	302	-81	-0,84924	0,169842	1384	2138	894	1024
4,9	286	-84	-0,826075	0,144456	1396	2154	915	1026
4,95	270	-86	-0,803572	0,1196	1410	2170	936	1028
5	253	-88	-0,781713	0,095269	1425	2187	955	1030
5,05	236	-89	-0,760479	0,071461	1441	2204	974	1032
5,1	219	-90	-0,739853	0,048171	1457	2221	992	1033
5,15	201	-91	-0,719816	0,025393	1475	2239	1010	1033
5,2	182	-91	-0,700351	0,003124	1493	2258	1026	1034

Tabelle 8: ChScite PD-Regler, Startabstand=1400

A.5. Zustände der Pthreads

Ein Pthread kann eine von vier Zuständen annehmen:

- bereit, der Thread ist lauffähig und wartet auf die Zuteilung der CPU durch den Scheduler. Sobald ein Thread mit der pthread_create() Funktion gestartet wird, befindet es sich zunächst im bereit Zustand.
- blockiert, der Thread ist nicht lauffähig und wartet auf eine Ressource, in Form eines Mutex, einer Bedingungsvariable oder I/O.

- läuft, der Thread wird momentan vom Prozessor ausgeführt. Auf einem Multiprozessorsystem, können mehrere Threads sich in diesem Zustand befinden. Dieser Zustand wird verlassen, wenn der Scheduler dem Thread die CPU Zeit entzieht, der Thread auf eine Ressource wartet oder der Thread beendet wird.
- beendet, der Thread kehrt aus der Thread Funktion zurück und ruft pthread_exit auf, oder wurde abgebrochen.

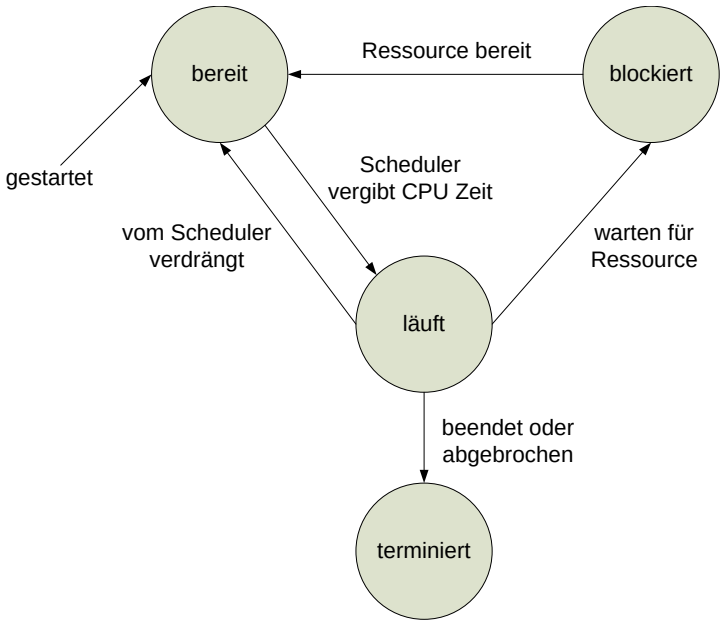

Abb. 67: Die vier Zustände, bereit, laufend, blockiert und beendet, die ein Pthread im Ausführungskontext einnehmen kann.

A.6. Das Starten eines Pthreads

In einem Programm werden die Threads über einem Identifier in der Datenstruktur pthread_t erkannt, unter Linux handelt es sich um einen primitiven Datentyp in Form eines Integers. Um einen Thread zu starten, muss zunächst die Datenstruktur als Variable im Programm deklariert werden, um sie später der pthread_create() Methode zu übergeben.

```
int pthread_create(pthread_t *thread,
const pthread_attr_t *attr,
void*(*start)(void *), void *arg);
```

Der Prototyp für die Methode zum Starten eines Threads enthält 4 Parameter.

- 1. Parameter, zur eindeutigen Identifizierung des Threads.
- 2. Parameter, zur Einstellung des Schedulers.
- 3. Parameter, die Thread Funktion.
- 4. Parameter, einen Parameter für die Thread Funktion.

Mit dem 2. Parameter die über eine Datenstruktur vom Typ pthread_attr_t übergeben werden, lässt sich mehr parametrisieren als den Scheduler, wie die Größe und Adresse des Stacks, usw. Jedoch ist für den Einparkassistenten nur die Einstellung des Schedulers relevant.

A.7. Messungen der Ausführungszeit 650 MHz

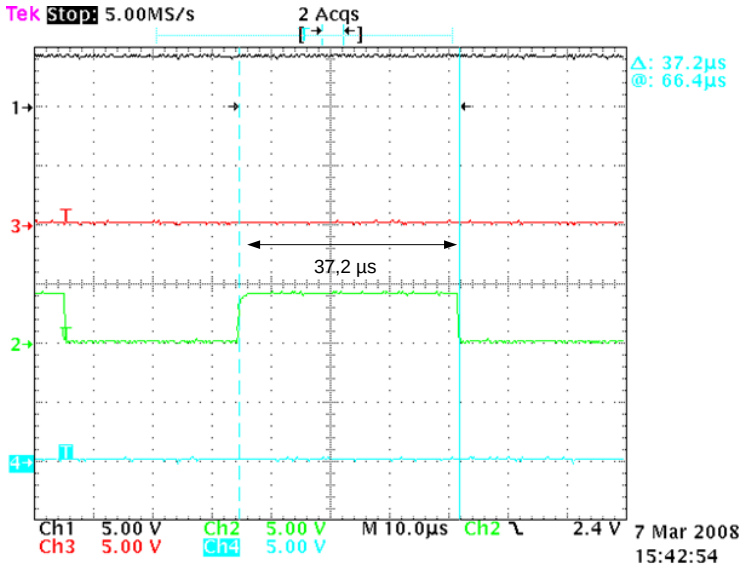

Abb. 68: Byte auslesen, read()-Feinere Auflösung der Abb. 52, Auflösung 10μs/Raster und 5V/Kasten. Ch1=Reaktionsmethode Laserscanner Ch2=read()-Funktion Ch4=Automat Parkluecke

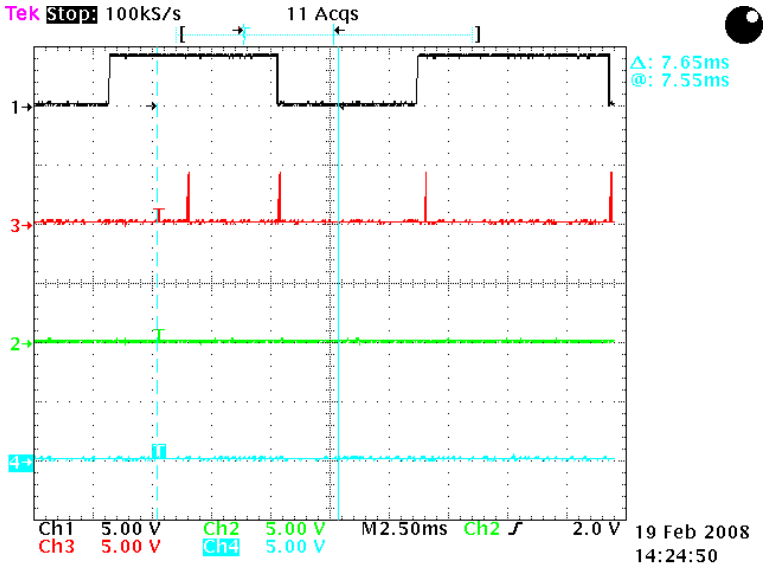

Abb. 69: Kontextwechsel Laserscanner, Steuer_Parkassistent, Auflösung 2,5ms/Raster und 5V/Kasten. Ch1=Reaktionsmethode Laserscanner Ch2=read()-Funktion Ch3=Automat Steuer_Parkassistent

A.8. Messungen der Ausführungszeit 32 MHz

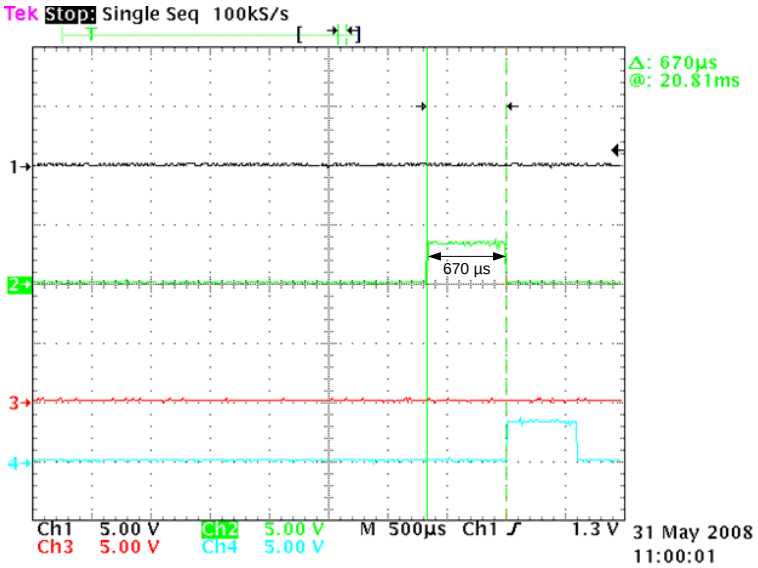

Abb. 70: Auflösung 500μs/Raster und 5V/Kasten. Ch1=Automat Laserscanner Ch2= Automat Steuer_Parkassistent Ch3= Automat EinparkenCh4=Automat Parkluecke

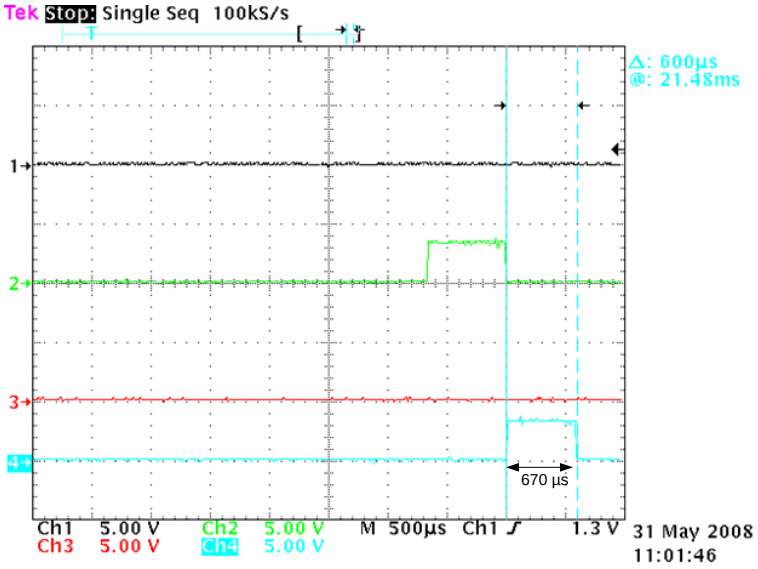

Abb. 71: Auflösung 500μs/Raster und 5V/Kasten. Ch1=Automat Laserscanner Ch2= Automat Steuer_Parkassistent Ch3= Automat EinparkenCh4=Automat Parkluecke

B. Code

B.1. Parallel Port GEME2000

```
#include <Parport.h>

    int fd,ioperm_ret;
    unsigned char datareg = 0x00;

void initParport(void)
{
    ioperm_ret = ioperm(PARPORT, 3, 1);
}

void write_par(unsigned char param)
{
  initParport();
  outb(param, PARPORT);
  closeParport();
}

unsigned char read_par(void)
{

  //Pins 2..9 wieder als Ausgang konfigurieren
    initParport();
        datareg=inb(PARPORT);
        printf("read inb %x\n",datareg);
        closeParport();

  return datareg;
}

void closeParport(void)
{
    ioperm_ret=ioperm(PARPORT, 3, 0);
}

void setD7(void)
{
  unsigned char mask;

  initParport();
  mask = inb(PARPORT);
  mask = mask | D7;
  //printf("Setting d7 0x%x\n",mask);
  outb(mask, PARPORT);
```

```
  closeParport();
}

void setD6(void)
{
  unsigned char mask;

  initParport();
  mask = inb(PARPORT);
  mask = mask | D6;
  //printf("Setting d6 0x%x\n",mask);
  outb(mask, PARPORT);
  closeParport();
}

void setD4(void)
{
  unsigned char mask;

  initParport();
  mask = inb(PARPORT);
  mask = mask | D4;
  //printf("Setting d4 0x%x\n",mask);
  outb(mask, PARPORT);
  closeParport();
}

void setD3(void)
{
  unsigned char mask;

  initParport();
  mask = inb(PARPORT);
  mask = mask | D3;
  //printf("Setting d3 0x%x\n",mask);
  outb(mask, PARPORT);
  closeParport();
}

void clrD7(void)
{
  unsigned char mask;
  initParport();
  mask = inb(PARPORT);
  mask = mask & ~(D7);
  //printf("Clearing d7 0x%x\n",mask);
  outb(mask, PARPORT);
  closeParport();
```

```
}

void clrD6(void)
{
  unsigned char mask;

  initParport();
  mask = inb(PARPORT);
  mask = mask & ~(D6);
  //printf("Clearing d6 0x%x\n",mask);
  outb(mask, PARPORT);
  closeParport();
}

void clrD4(void)
{
  unsigned char mask;

  initParport();
  mask = inb(PARPORT);
  mask = mask & ~(D4);
  //printf("Clearing d4 0x%x\n",mask);
  outb(mask, PARPORT);
  closeParport();
}

void clrD3(void)
{
  unsigned char mask;

  initParport();
  mask = inb(PARPORT);
  mask = mask & ~(D3);
  //printf("Clearing d3 0x%x\n",mask);
  outb(mask, PARPORT);
  closeParport();
}
```

Listing 5: Parallel Port Treiber

B.2. AIRA-Laufzeitumgebung

```
/*-------------------------------------------------------------*/
/* automaten_impl.c                                            */
/* Implementierung der Funktionen fuer                         */
/* das Laufzeitsystem fuer reaktive Systeme*/
/*-------------------------------------------------------------*/
#include <globals.h>
```

```
#include <aira-runtime.h>
#include <list.h>

// Prueft, ob die Aktivitaet einen Ihrer Endzustaende erreicht hat


T_zustand* aktivitaet_akt_zustand(T_aktivitaet *arg_aktivitaet){
  //printf("Akt state in %i?\n",arg_aktivitaet->akt_zustand);
  T_zustand* zustand = (T_zustand*) list_get
(arg_aktivitaet->zustaende,arg_aktivitaet->akt_zustand);
 // printf("Akt state out?\n");
  return zustand;
}

T_logical ist_akt_beendet (T_aktivitaet *arg_aktivitaet ){
  return ( aktivitaet_akt_zustand(arg_aktivitaet)->
zustandsart == Endzustand ? True : False );
}

/*-------------------------------------------------*/
/* Implementierung der Routinen
/*-------------------------------------------------*/

int aktivitaet_init(T_aktivitaet **arg_aktivitaet){
  (*arg_aktivitaet) = (T_aktivitaet*)malloc(sizeof(T_aktivitaet));
  list_init(&(*arg_aktivitaet)->zustaende);
  (*arg_aktivitaet)->akt_zustand = 0;
  return 0;
}

int aktivitaet_free(T_aktivitaet *arg_aktivitaet){
  T_List_iterator * it;
  T_zustand* zustand;

  if(arg_aktivitaet != NULL){
    it = (T_List_iterator*) malloc(sizeof (T_List_iterator));
    zustand = (T_zustand*) list_get_first(arg_aktivitaet->zustaende,it);
    while(zustand != NULL){
      zustand_free(zustand);
      zustand = (T_zustand*) list_get_next(it);
    }
    free(it);
    free(arg_aktivitaet->zustaende);
    free(arg_aktivitaet);
    return 0;
  }
  else
    return 1;
```

```
}

void aktivitaet_eintragen(T_zustand* arg_zustand, T_aktivitaet
arg_aktivitaet, int arg_pos){
  list_add(arg_zustand->aktivitaeten,arg_aktivitaet,arg_pos);
}


int zustand_init(T_zustand **arg_zustand){
  (*arg_zustand) = (T_zustand*)malloc(sizeof(T_zustand));
  list_init(&(*arg_zustand)->aktivitaeten);
  (*arg_zustand)->entry_aktion = NULL;
  (*arg_zustand)->exit_aktion = NULL;
  (*arg_zustand)->ext_reaktionen = NULL;
  (*arg_zustand)->int_reaktionen = NULL;
  (*arg_zustand)->lambda_reaktionen = NULL;
  (*arg_zustand)->max_timer = NULL;
  (*arg_zustand)->min_timer = NULL;
  (*arg_zustand)->tick_timer = NULL;
  (*arg_zustand)->zustandsart = Normalzustand;
  return 0;
}

int zustand_free(T_zustand *arg_zustand){
  T_List_iterator * it;
  T_aktivitaet *akt;

  if(arg_zustand != NULL){
    it = (T_List_iterator*) malloc(sizeof (T_List_iterator));
    akt = (T_aktivitaet*) list_get_first(arg_zustand->aktivitaeten,it);
    while(akt != NULL){
      aktivitaet_free(akt);
      akt = (T_aktivitaet*) list_get_next(it);
    }
    free(it);
    if(arg_zustand->max_timer != NULL)
      timer_free(arg_zustand->max_timer);
    if(arg_zustand->min_timer != NULL)
      timer_free(arg_zustand->min_timer);
    if(arg_zustand->tick_timer != NULL)
      timer_free(arg_zustand->tick_timer);

    free(arg_zustand->aktivitaeten);
    free(arg_zustand);
    return 0;
  }
  else
    return 1;
```

```
}

// Eintragen eines weiteren Zustandes in eine Aktivitdt
void zustand_eintragen ( T_aktivitaet *arg_aktivitaet,
T_zustand    *arg_zustand ){
  /* Vorbedingungen */
  if (arg_aktivitaet == NULL) exit (1);
  if (arg_zustand == NULL) exit (1);
  /* Routinenanweisungen */
  list_enqueue(arg_aktivitaet->zustaende,arg_zustand);
}

/*
 * Aktionen beim Verlassen eines Zustandes
 */

void zustand_verlassen ( T_aktivitaet *arg_aktivitaet){
  T_List * liste;
  T_List_iterator * it;
  T_aktivitaet *akt;
  /* Vorbedingungen */
  if (arg_aktivitaet == NULL) exit (1);

  /* Routinenanweisungen */
  // eventuell vorhandene Subaktivitdten des aktuellen
  // Zustands beenden
  liste = aktivitaet_akt_zustand(arg_aktivitaet)->aktivitaeten;
  it = (T_List_iterator*) malloc(sizeof (T_List_iterator));
       akt = (T_aktivitaet*) list_get_first(liste,it);
  while(akt != NULL){
              aktivitaet_beenden(akt);
              akt = (T_aktivitaet*) list_get_next(it);
  }
  // eventuell vorhandene Exit-Aktionen ausf|hren
  if (aktivitaet_akt_zustand(arg_aktivitaet)->exit_aktion != NULL)
      aktivitaet_akt_zustand(arg_aktivitaet)->exit_aktion ();

  free(it);
}


// Aktionen beim Betreten eines Zustandes
void local_zustand_annehmen(T_aktivitaet *arg_aktivitaet,
T_zustand_nr arg_zust_nr, int runExit ){
       T_List * liste;
  T_List_iterator * it;
  T_aktivitaet *akt;
```

```
    /* Vorbedingungen */
  if (arg_aktivitaet == NULL) exit (2);
  if ((arg_zust_nr < 0) || (arg_zust_nr >
arg_aktivitaet->zustaende->size-1))
  {
        printf("Fehler: arg_zust_nr ausserhalb der
erlaubten Grenzen %d\n",arg_zust_nr);
        exit (2);
  }

  /* Aktuellen Zustand verlassen -> Exit ausf|hren */
  if(runExit){
    if(aktivitaet_akt_zustand(arg_aktivitaet)->max_timer != NULL){
      timer_stop(aktivitaet_akt_zustand(arg_aktivitaet)->max_timer);
    }
    if(aktivitaet_akt_zustand(arg_aktivitaet)->tick_timer != NULL){
      timer_stop(aktivitaet_akt_zustand(arg_aktivitaet)->tick_timer);
    }
    zustand_verlassen(arg_aktivitaet);
  }

  /* Routinenanweisungen */
  arg_aktivitaet->akt_zustand = arg_zust_nr;

  /* eventuell vorhandene Entry-Aktionen ausf|hren */
  if (aktivitaet_akt_zustand(arg_aktivitaet)->entry_aktion != NULL){
    if(aktivitaet_akt_zustand(arg_aktivitaet)->max_timer != NULL){
      timer_start(aktivitaet_akt_zustand(arg_aktivitaet)->max_timer);
    }
    if(aktivitaet_akt_zustand(arg_aktivitaet)->min_timer != NULL){
      timer_start(aktivitaet_akt_zustand(arg_aktivitaet)->min_timer);
    }
    if(aktivitaet_akt_zustand(arg_aktivitaet)->tick_timer != NULL){
timer_reset(aktivitaet_akt_zustand(arg_aktivitaet)->tick_timer,
aktivitaet_akt_zustand(arg_aktivitaet)->tick_timer->start_count);
      timer_start(aktivitaet_akt_zustand(arg_aktivitaet)->tick_timer);
    }
    aktivitaet_akt_zustand(arg_aktivitaet)->entry_aktion ();
  }

  /* eventuell vorhandene Subaktivitdt starten */
  liste = aktivitaet_akt_zustand(arg_aktivitaet)->aktivitaeten;
  it = (T_List_iterator*) malloc(sizeof (T_List_iterator));
       akt = (T_aktivitaet*) list_get_first(liste,it);
  while(akt != NULL){
              aktivitaet_starten(akt);
              akt = (T_aktivitaet*) list_get_next(it);
  }
```

```
  free(it);
}

void zustand_annehmen ( T_aktivitaet *arg_aktivitaet,
T_zustand_nr arg_zust_nr){
        T_zustand lokal_aktueller_zustand =
*aktivitaet_akt_zustand(arg_aktivitaet);
        if(lokal_aktueller_zustand.min_timer == NULL){
                local_zustand_annehmen(arg_aktivitaet,arg_zust_nr,TRUE);
        }
        else if(lokal_aktueller_zustand.min_timer->timer_expired){
                local_zustand_annehmen(arg_aktivitaet,arg_zust_nr,TRUE);
        }
}

/*
 *  Abwicklung der Reaktion einer Aktivitdt auf eine Nachricht
 */
T_akt_reaktion aktivitaet_reagieren (T_aktivitaet* arg_aktivitaet,
REF_T_Nachricht arg_nachricht){
  int reaktion_erfolgt = 0;
  int subakt_beendet = 0;
  T_akt_reaktion lokal_subakt_reaktion = AKT_Hat_Nicht_Reagiert;
  T_zustand lokal_aktueller_zustand;
  T_List * liste;
  T_List_iterator * it;
  T_aktivitaet *akt;

  // Vorbedingungen
  if (arg_aktivitaet == NULL) exit (1);


  // Reaktion, wenn die Aktivitaet beendet ist
  if (ist_akt_beendet (arg_aktivitaet) == True)
    return (AKT_Ist_Beendet);


  /*
    * Reaktion, wenn eine eingebettete Sub-Aktivitaet vorhanden ist,
    * in der unteren Ebene abhandeln
    */
  lokal_aktueller_zustand = *aktivitaet_akt_zustand(arg_aktivitaet);

  liste = lokal_aktueller_zustand.aktivitaeten;
  it = (T_List_iterator*) malloc(sizeof (T_List_iterator));
        akt = (T_aktivitaet*) list_get_first(liste,it);
  while(akt != NULL){
                lokal_subakt_reaktion =
```

```
aktivitaet_reagieren(akt,arg_nachricht);
                if(lokal_subakt_reaktion == AKT_Hat_Reagiert){
                        reaktion_erfolgt = 1;
                }
                if(lokal_subakt_reaktion == AKT_Hat_Geendet){
                        subakt_beendet = 1;
                }
                akt = (T_aktivitaet*) list_get_next(it);
  }
  free(it);

  if(subakt_beendet){
    lokal_subakt_reaktion = AKT_Hat_Geendet;
  }
  else if(reaktion_erfolgt){
    lokal_subakt_reaktion = AKT_Hat_Reagiert;
  }

  // Reaktionsmoeglichkeiten fuer die aktuelle Ebene pruefen

  switch (lokal_subakt_reaktion) {

      case AKT_Hat_Nicht_Reagiert:
      case AKT_Ist_Beendet :

        /* Da keine Subaktivitaet vorhanden ist bzw.
eine vorhandene nicht reagiert hat,
            in der aktuellen Ebene nach einer
Reaktionsmoeglichkeit suchen */

        /* Versuch eine zustandserhaltenden Reaktionen
reagieren zu lassen */

        if (lokal_aktueller_zustand.int_reaktionen != 0) {
          if ((*lokal_aktueller_zustand.int_reaktionen)
(arg_aktivitaet, arg_nachricht) == ZST_Hat_Reagiert )
              return (AKT_Hat_Reagiert);
        }
        /* Versuch eine zustandsaendernden Reaktionen
reagieren zu lassen */

        if (lokal_aktueller_zustand.ext_reaktionen != 0) {
          if(lokal_aktueller_zustand.min_timer == NULL){
            if ((*lokal_aktueller_zustand.ext_reaktionen)
(arg_aktivitaet, arg_nachricht) == ZST_Hat_Reagiert){
              if(ist_akt_beendet(arg_aktivitaet)){
                  return (AKT_Hat_Geendet);
              }else{
```

```
                return (AKT_Hat_Reagiert);
              }
            } else {
              return (AKT_Hat_Nicht_Reagiert);
            }
          }else if (lokal_aktueller_zustand.min_timer->timer_expired){
            if ((*lokal_aktueller_zustand.ext_reaktionen)
(arg_aktivitaet, arg_nachricht) == ZST_Hat_Reagiert){
              if(ist_akt_beendet(arg_aktivitaet)){
                return (AKT_Hat_Geendet);
              }else{
                return (AKT_Hat_Reagiert);
              }
            } else {
              return (AKT_Hat_Nicht_Reagiert);
            }
          }
          else{
            return (AKT_Hat_Nicht_Reagiert);
          }
        } else {
            return (AKT_Hat_Nicht_Reagiert);
        }
        break;



      case AKT_Hat_Reagiert :

        /* Da die vorhandene Subaktivitdt reagiert hat, aber nicht
        geendet hat, erfolgt in der aktuellen Ebene keine Reaktion.
            Es wird nur weitergemeldet, dass reagiert wurde */

        return (AKT_Hat_Reagiert);

        break;



      case AKT_Hat_Geendet :

        /* Da eine vorhandene Subaktivitdt reagiert hat und in einen
Endzustand gelaufen ist, wird in der aktuellen Ebene nach
einer Lambda-Reaktion gesucht  */

        if (lokal_aktueller_zustand.lambda_reaktionen != 0){
if ((*lokal_aktueller_zustand.lambda_reaktionen)
(arg_aktivitaet, arg_nachricht) == ZST_Hat_Reagiert){
```

```
            if (ist_akt_beendet(arg_aktivitaet))
                return (AKT_Hat_Geendet);
            else
                return (AKT_Hat_Reagiert);
          } else
            return (AKT_Hat_Nicht_Reagiert);
        } else
            return (AKT_Hat_Reagiert);

        break;

  } /* end of switch */
        return (AKT_Hat_Nicht_Reagiert);
}

void aktivitaet_starten (T_aktivitaet *arg_aktivitaet){
        /* Hier muss noch der Aktuelle startzustand anhand der
Wdchter gefunden werden.
         * Wdchter sind methoden der Subzustdnde
         */
        local_zustand_annehmen(arg_aktivitaet,0,FALSE);
}

void aktivitaet_beenden (T_aktivitaet *arg_aktivitaet){
        zustand_verlassen(arg_aktivitaet);
}
```

Listing 6: AIRA-Laufzeitumgebung Reaktionsalgorithmus

B.3. AIRA-Automat Laserscanner

```
/**
 * @file Z_LMS_SCAN.c
 * @brief Z_LMS_SCAN-Zustand. Fuehrt die Messwerterfassung durch.
 * @author Yegor Yefremov (yefrem_y@informatik.haw-hamburg.de)
 */
#include <A_LMS.h>
#include <Z_LMS_SCAN.h>
#include <MsgQueue.h>
#include <int_vars.h>
#include <ringbuffer.h>
#include <LMSlib.h>
#include <aira_dbg_lms.h>

static signed int LMS_surpressing;
/**
 * Kopiert die Messwerte aus dem Bytestrom in den Ringbuffer
 * @param len die Laenge
 * @param buf Bytestom mit Messwerten
 */
void copy_values_to_rb(int len, uchar *buf)
{
  int i,j;
  short val[len/2];
  for(i=0,j=0; i<len; i=i+2,j++) {
    val[j]=(short)( (buf[i+1] & 0x1f) <<8  |buf[i]);
    //printf("%d ", val[j]);

  }
```

```
  ringbuffer_put(tRB, len/2, val);
  //printf("Length %d\n", len/2);

}

/**
 * Liest die Messwerte aus dem Ringbuffer und gibt die aus. Fuer debug Zwecke
 * @param len die Laenge
 */
void read_values_from_rb(int len)
{
  int msg_len = len/2;
  short vals[msg_len];
  ringbuffer_get(tRB, msg_len, vals);
  printf("Vals: %d, %d, %d\n", vals[0], vals[1], vals[2]);
}

/**
 * entry-Funktion
 */
void Z_LMS_SCAN_Entry()
{
        AIRA_PRINTF(AIRA_DBG_LOW, "Z_LMS_SCAN entry!\n");
        /* Das Verschicken der Scanwerte vom Laser wird beim Eintritt grundsätzlich freigegeben */
        LMS_surpressing=0;
}

/**
 * exit-Funktion
 */
void Z_LMS_SCAN_Exit()
```

```
{
        AIRA_PRINTF(AIRA_DBG_LOW, "Z_LMS_SCAN exit!\n");
}

/**
 * Externe Reaktion.
 * @param arg_aktivitaet Aktivitaet
 * @param arg Nachricht
 * @return reaktion des Zustandes
 */
T_zst_reaktion Z_LMS_SCAN_ext_reaktion (T_aktivitaet* arg_aktivitaet,REF_T_Nachricht arg){
        T_zst_reaktion reaktion = ZST_Hat_Reagiert;
        T_Nachricht* arg_nachricht = (T_Nachricht*)arg;

        switch(arg_nachricht->n_art){

                /**
                 * #S_LMS_Stop: der Laserscanner wird resetet. LMS wechselt sich nach Z_LMS_IDLE-Zustand.
                 **/
                case S_LMS_Stop:
                        AIRA_PRINTF(AIRA_DBG_LOW, "S_LMS_Stop received\n");
                        //stopLMS(fd);
                        //resetLMS(fd);
                        LMS_surpressing=1;
                        zustand_annehmen(arg_aktivitaet,ID_Z_LMS_IDLE);
                        break;

                case S_LMS_Parken:
                        AIRA_PRINTF(AIRA_DBG_LOW, "S_LMS_Parken received\n");
                        zustand_annehmen(arg_aktivitaet,ID_Z_LMS_SCAN_PARKEN);
                        break;
```

```
                default:
                        reaktion = ZST_Hat_Nicht_Reagiert;
                        break;
        }
        return reaktion;
}

/**
 * Interne Reaktion.
 * @param arg_aktivitaet Aktivitaet
 * @param arg Nachricht
 * @return reaktion des Zustandes
 */
T_zst_reaktion Z_LMS_SCAN_int_reaktion (T_aktivitaet* arg_aktivitaet,REF_T_Nachricht arg){
        int datalen;
        uchar buf[MAXNDATA];
    T_zst_reaktion reaktion = ZST_Hat_Reagiert;
    T_Nachricht* arg_nachricht = (T_Nachricht*)arg;

        //setD7();
    switch(arg_nachricht->n_art){
                T_Nachricht *send_msg_T;
                T_Nachricht *send_msg_Timeout;

                /**
                 * #A_Tick: die Daten von dem LMS200 werden ausgelesen und S_PL_Data_Ready
                 * wird an Parkluecke-Automat geschickt.
                 */
                case A_Tick:
                        datalen = readLMSSubrangeData(fd, buf);
```

```
        if (datalen != 0)
        {
                //showdata(datalen, buf+11);
                copy_values_to_rb(datalen, buf+11);
                //read_values_from_rb(datalen);
                chkstatus(buf[datalen+11]);
                if(LMS_surpressing==0)
                {
                        send_msg_T = (T_Nachricht*) malloc(sizeof(T_Nachricht));
                        send_msg_T->n_art = S_PL_Data_Ready;
                        msg_enqueue(msgRecvQueueParkluecke,send_msg_T);
                }
        }
        else
        {
            send_msg_Timeout = (T_Nachricht*) malloc(sizeof(T_Nachricht));
            send_msg_Timeout->n_art = S_PL_Timeout;
            msg_enqueue(msgRecvQueueParkluecke,send_msg_Timeout);
        }
        break;
                                                        /**
 * #S_LMS_Pause: LMS wechselt sich nach Z_LMS_SCAN_IDLE.
 */
case S_LMS_Pause:
        AIRA_PRINTF(AIRA_DBG_LOW, "S_LMS_Pause␣received\n");

        /* Das Verschicken der Scanwerte vom Laser wird unterdrückt777 */
        LMS_surpressing=1;
        break;

case S_LMS_Start:
```

```
                        AIRA_PRINTF(AIRA_DBG_LOW, "S_LMS_Start␣received\n");

                        arg_nachricht = (T_Nachricht*) malloc(sizeof(T_Nachricht));
                        arg_nachricht->n_art = S_PL_Laserscanner_bereit;
                        msg_enqueue(msgRecvQueueParkluecke,arg_nachricht);

                        LMS_surpressing=0;
                        break;

                default:
                                                reaktion = ZST_Hat_Nicht_Reagiert;
                break;
    }
    //clrD7();
    return reaktion;
}
```

Listing 7: Zustand Z_LMS_SCAN

B.4. Odometrie und PD-Regler in ChScite

```
#include <chplot.h>

#define T_END     4.4
#define A         1200
#define VF1       1

#define HALBEFAHRZEUGBREITE 457.5
#define HINTERRADVORDERKANTE 980
#define FAHRZEUGBREITE       915
#define FAHRZEUGLAENGE      1220
#define TOLERANZ             100
#define PARKLUECKENLAENGE   ((FAHRZEUGLAENGE+TOLERANZ)*2)
#define PARKLUECKENTIEFE    (FAHRZEUGBREITE+TOLERANZ)
#define FASTGERADE          0
#define RADABSTAND 760
#define VVORDERRAD 500
#define VRUECK -500

#define VF2 4
#define T_DELTA 0.05

  /* Einspurfahrzeugmodell

  Y_dl = H_y - y_p;
  X_dl = H_x - x_p;

  theta_L = arctan(8*(Y_dl / X_dl)) - theta_L;
  V_p = V_m * cos(alpha_L);

  // Winkel theta
  der(theta_L) = V_m / L_r * sin(alpha_L);

  lenkbare Vorderrad
  der(x_m) = V_m * cos(theta_L - alpha_L);
  der(y_m) = V_m * sin(theta_L - alpha_L);

  lenkbare Hinterrad
  der(x_p) = V_p * cos(theta_L);
  der(y_p) = V_p * sin(theta_L);

  */

double v_m;
double l_r;
```

```
double beta_c;
double hyp_c;

int x_plot = 0;
int toggle = 0;

v_m = 500;
l_r = 760;
beta_c= atan(HALBEFAHRZEUGBREITE/HINTERRADVORDERKANTE);
hyp_c =
sqrt((HALBEFAHRZEUGBREITE*HALBEFAHRZEUGBREITE)
+(HINTERRADVORDERKANTE*HINTERRADVORDERKANTE));

/*++++++++++++ Interne Daten für Regler ++++++++++++++++++*/
double last_regl_diff = 0;          // 1/z Glied
double last_z_a0_d    = 0;          // 1/z Glied

void reset_regler(){
        last_regl_diff = 0;          // 1/z Glied
        last_z_a0_d    = 0;          // 1/z Glied
}

double pd_regler(double fuehrung_ay,double x,double y){
        //double fuehrung_ay;
        double soll_lenkwinkel = 0.0;
        double s_ax;
        double s_ay;
        double regl_diff;                          // Regeldifferenz
        double sum1, sum2;

        //~ // Reglerparameter nach Matlab Berechnung
        //~ // sample period ist nach timer: 0,048 sec
        //~ // nach i
        double z_b0   = 3.402;
        double z_b1   = -3.4;
        // double z_a0_d = 0.39246;
        // double z_a1   = -2.5;
    double z_a0_d = 0.448;
        double z_a1   = -2.2;

    toggle=0;
    x_plot=1;

        //~ // nach h
        //~ double z_b0   = 3.402;
        //~ double z_b1   = -3.4;
    //~ double z_a0_d = 0.24704;
        //~ double z_a1   = -4;
```

```
        // aktuelle Messwerte
        s_ax = x;
        s_ay = y;

        regl_diff = fuehrung_ay - (VF1*HALBEFAHRZEUGBREITE) - s_ay;
        //~ regl_diff = fuehrung_ay - (HALBEFAHRZEUGBREITE) - s_ay;

        // Limiter zur Vermeidung von Lenkwinkeluebersteuerung
// und Vorzeichenumkehr
        if(regl_diff < -0.628)
        {
                regl_diff = 0.628;
        }
        else if(regl_diff > 0.628)
        {
                regl_diff = - 0.628;
        }
        else
        {
                regl_diff = - regl_diff;
        }

        // digitaler PD-Regler
        sum1 = last_regl_diff * z_b1 - last_z_a0_d * z_a1;
        sum2 = regl_diff * z_b0 + sum1;
        soll_lenkwinkel = sum2 * z_a0_d;

        // 1/z Glieder
        last_regl_diff = regl_diff;
        last_z_a0_d    = soll_lenkwinkel;

        return (double)((soll_lenkwinkel) * (-1));
}

// Trapez Integration
double ti_xm(double x_m,double theta_l, double dtheta_l, double alpha_l,
double dalpha_l, double h){
    double dxm;

    dxm =   x_m
            +(0.5 * h * v_m * cos(theta_l - alpha_l))
            +(0.5 * h * v_m * cos(dtheta_l- dalpha_l));

    return dxm;
}

double ti_ym(double y_m,double theta_l, double dtheta_l,
```

```
double alpha_l, double dalpha_l, double h){
    double dym;

    dym =   y_m
            +(0.5 * h * v_m * sin(theta_l - alpha_l))
            +(0.5 * h * v_m * sin(dtheta_l- dalpha_l));

    return dym;
}

double ti_xp(double x_p, double theta_l, double dtheta_l,
double alpha_l, double v_p, double h){
    double dxp,dv_p;

    dv_p = v_m  * cos(alpha_l);

    dxp =   x_p
            +(0.5 * h * v_p * cos(theta_l))
            +(0.5 * h * dv_p * cos(dtheta_l));

    return dxp;
}

double ti_yp(double y_p,double theta_l ,double dtheta_l ,
double alpha_l, double v_p, double h){
    double dyp,dv_p;

    dv_p = v_m  * cos(alpha_l);

    dyp =   y_p
            +((0.5 * h * v_p * sin(theta_l))
            +(0.5 * h * dv_p * sin(dtheta_l)));

    return dyp;
}

double ti_theta(double theta_l, double alpha_l, double v_p,
double y_dl, double x_dl, double h){
    double dtheta,dalpha;

    //dalpha = h*(atan(8*(y_dl/x_dl)) - theta_l);

    dtheta = theta_l
            +(0.5*h*v_m * sin(alpha_l)
            +(0.5*h*v_m * sin(alpha_l))
             )/l_r;

    return dtheta;
```

```
}

double rk_mx(double x_m, double theta_l, double alpha_l){
    double dmx1,dmx2,dtheta;

    dtheta = T_DELTA*(((double)((double)v_m / (double)RADABSTAND))
* sin(alpha_l));

    dmx1 = T_DELTA*(((double)v_m) * cos(theta_l - alpha_l ));
    dmx2 = T_DELTA*(((double)v_m) * (cos(theta_l- alpha_l )
+(dtheta/((double)2))));

    return (x_m + dmx2);
}

double rk_my(double y_m, double theta_l, double alpha_l){
    double dmy1,dmy2,dtheta;

    dtheta = T_DELTA*(((double)((double)v_m / (double)RADABSTAND))
* sin(alpha_l));

    dmy1 = T_DELTA*(((double)v_m) * sin(theta_l - alpha_l ));
    dmy2 = T_DELTA*(((double)v_m) * (sin(theta_l - alpha_l )
+(dtheta/((double)2))));

    return (y_m + dmy2);
}

double rk_px(double x_p, double theta_l, double alpha_l, double v_p){
    double dpx1,dpx2,dtheta;

    dtheta = T_DELTA*(((double)((double)v_m / (double)RADABSTAND))
* sin(alpha_l));

    dpx1 = T_DELTA*(v_p * cos(theta_l));
    dpx2 = T_DELTA*(v_p * (cos(theta_l)+(dtheta/((double)2))));

    return (x_p + dpx2);
}

double rk_py(double y_p, double theta_l, double alpha_l, double v_p){
    double dpy1,dpy2,dtheta;

    dtheta = T_DELTA*(((double)((double)v_m / (double)RADABSTAND))
* sin(alpha_l));

    dpy1 = T_DELTA*(v_p * sin(theta_l));
    dpy2 = T_DELTA*(v_p * (sin(theta_l)+(dtheta/((double)2))));
```

```
    return (y_p + dpy2);
}

double rk_theta(double theta_l, double alpha_l, double v_p,
double x_p, double y_p)
{
    double dpsi1,dpsi2,dalpha;

    dalpha = T_DELTA*(pd_regler(A,x_p,y_p));

    dpsi1 = T_DELTA*(v_m / ((double)RADABSTAND) * sin(alpha_l));
    dpsi2 = T_DELTA*(v_m / ((double)RADABSTAND) *
(sin(alpha_l)+(dalpha/((double)2))));

    return (theta_l + dpsi2);
}

void all_funct(double h)
{
       int i;

       class CPlot plot, lissajous;
       int steps = T_END/T_DELTA;
       array double     x_m[steps], y_m[steps], x_p[steps] ,
y_p[steps],y_b[steps], theta_l[steps], alpha_l[steps],
x_dl[steps], y_dl[steps], v_p[steps], t[steps], h_y[steps],
y_g[steps];
       array int winkel[steps],achs[steps];
       double x_h,y_h,y_t,x_t;

        // Koordinatensytem
        x_m[0] = (double)0;
        y_m[0] = (double)0;
        x_p[0] = (double)RADABSTAND;
        y_p[0] = (double)0;
        y_h        = (double)(A);
        x_h    = (double)2*FAHRZEUGLAENGE;


       /*********************Startwerte*************************/
       // Deichselschwerpunkt
       y_dl[0]=(double)(y_h - ((double)
(HALBEFAHRZEUGBREITE)) - y_p[0]);
       x_dl[0]=(double)(x_h - x_p[0]);

       // Winkel
       //alpha_l[0] = atan((VF2)*y_dl[0]/x_dl[0]);
```

```
        y_t = y_p[0];
        x_t = x_p[0];
        alpha_l[0] = pd_regler(y_h,x_t,y_t);

        // Geschwindigkeit
        v_m = (double)VVORDERRAD;
        v_p[0]= v_m * cos(alpha_l[0]);

        theta_l[0] = rk_theta(0, alpha_l[0], v_p[0], x_p[0], y_p[0]);

        y_b[0] = y_m[0] + (sin(theta_l[0]+beta_c)*hyp_c);

        /*************************************************/
        printf("t; x_dl; y_dl; alpha_l; theta_l;
x_m; x_p; y_m; y_p; v_m\n");
        // nur fÃ¼r die Schleife
        t[0]=0;
        for(i=0; i<steps-1; i++)
        {
        // Deichselschwerpunkt
        y_dl[i]=(double)(y_h - ((double)
(HALBEFAHRZEUGBREITE)) - y_p[i]);
        x_dl[i]=(double)(x_h - x_p[i]);

        // Winkel geradeaus fahren
        if(((theta_l[i]<FASTGERADE) || (v_m<0)) && (i>30))
        {
            alpha_l[i] = 0;
            v_m = (double)VRUECK;
        }
        else
        {
            //alpha_l[i] = atan(((double)VF2)*y_dl[i]/x_dl[i])
- theta_l[i];
            y_t=y_p[i];
            x_t=x_p[i];
            alpha_l[i] = pd_regler(y_h,x_t,y_t);
            printf("%f; %f; %f; %f; %f; %f; %f; %f; %f\n",
t[i], x_dl[i], y_dl[i],alpha_l[i],theta_l[i],x_m[i],x_p[i],
y_m[i],y_p[i],v_m);
        }
        /*****************/
        /* Runge Kutta 2      */
        /*****************/
        theta_l[i+1] = rk_theta(theta_l[i],alpha_l[i],
v_p[i],x_p[i],y_p[i]);
        //theta_l[i+1] = ti_theta(theta_l[i], alpha_l[i], v_p[i],
 y_dl[i], x_dl[i], T_DELTA);
```

```
 // Geschwindigkeit
 v_p[i+1]= v_m * cos(alpha_l[i]);


 // Koordinatensytem
 x_m[i+1] = rk_mx(x_m[i],theta_l[i],alpha_l[i]);
 y_m[i+1] = rk_my(y_m[i],theta_l[i],alpha_l[i]);
 x_p[i+1] = rk_px(x_p[i],theta_l[i],alpha_l[i],v_p[i]);
 y_p[i+1] = rk_py(y_p[i],theta_l[i],alpha_l[i],v_p[i]);

 t[i+1]=t[i]+T_DELTA;


 // zur hilfe
 y_g[i] = A;
 y_b[i] = y_m[i] + (sin(theta_l[i]+beta_c)*hyp_c);
 winkel[i] = alpha_l[i] * 180 / 3.141;
 achs[i] = theta_l[i]* 180 / 3.141;

 }
 y_g[i+1] = A;

if(toggle==1)
{
    plot.data2D(t,y_g);
    plot.legend("grenze",0);

    plot.data2D(t,y_m);
    plot.legend("ym",1);

    plot.data2D(t,y_p);
    plot.legend("yp",2);

    if(x_plot==1)
    {
        plot.data2D(t,x_p);
        plot.legend("xp",3);

        plot.data2D(t,x_m);
        plot.legend("xm",4);
    }
}
else
{
//~ plot.data2D(t,y_g);
//~ plot.legend("grenze",0);
//~ plot.data2D(t,theta_l);
//~ plot.legend("theta_l",0);
```

```
        //~ plot.data2D(t,alpha_l);
        //~ plot.legend("alpha_l",1);

        //~ plot.data2D(t,winkel);
        //~ plot.legend("winkel",0);

        //~ plot.data2D(t,achs);
        //~ plot.legend("achs",1);

        //~ plot.data2D(t,y_m);
        //~ plot.legend("y_m",0);

        //~ plot.data2D(t,y_p);
        //~ plot.legend("y_p",1);
        //~ }

        plot.data2D(t,x_m);
        plot.legend("x_m",0);

        plot.data2D(t,x_p);
        plot.legend("x_p",1);
        }

         plot.plotting();
}



int main() {

   all_funct(T_DELTA);

   return 0;
}
```

Listing 8: Implementierung der Odometrie und des PD Reglers mit ChScite

B.5. Odometrie und virtuelle Deichsel mit Dymola

```
model ParkAssistent

    //#################################################################

    constant Real Hfb_h =  457.5;
//Halbe Fahrzeugbreite

    constant Real Ra_s =  230;
```

```
//Abstand Mitte des Vorderrades zum Vorderrand

    constant Real L_r = 760;
 //Radstand in m

    constant Real H_y = Grenze;
//Start y-Koordinate des detektierten Hindernises

    constant Real H_x = 2440;
//Start x-Koordinate des detektierten Hindernises
    constant Real V_m = 500;

    constant Real vf2 = 4;
 // Verstärkungsfaktor für Sigma

    constant Real vf1 = 1;
// Verstärkungsfaktor für dY_p

    constant Real Grenze=1200;
 // Parklücketiefe

    Real V_p(start=0);
//Geschwindigkeit des starren Hinterrades in m/s

    Real Y_m(start= 0);
//y-Koordinate des lenkbaren Vorderrades

    Real X_m(start= 0);
//x-Koordinate des lenkbaren Vorderrades

    Real X_p(start= L_r);
//x-Koordinate des starren Hinterrades

    Real Y_p(start= 0);
//y-Koordinate des starren Hinterrades

    Real Alpha_L(start=0);
//Lenkwinkel in RAD

    Real Sigma_L(start=0);
 //arctan(ydl/xdl) in RAD

    Real Theta_L(start=0);
//Achswinkel zum Deichselzugpunkt in RAD

    Real Y_dl(start=0);
 //y-Koordinate des Zugpunktes der Deichsel
```

```
    Real X_dl(start=0);
//x-Koordinate des Zugpunktes der Deichsel

    Real Lenkwinkel(start=0);
        //Lenkwinkel in Grad

    Real Achswinkel(start=0);
//Achswinkel in Grad

    //Real Y_de (start= Grenze + R - Hfb_h);
//y-Koordinate der vorderen linken Ecke des Fahrzeugs

    //Real X_de (start= Ra_s);
//x-Koordinate der vorderen linken Ecke des Fahrzeugs

    VisualShape Chassi(r0={-0.94, 0, 0}, Shape="box",
                    Length=1.2, Width=0.9, Height=1,
                    LengthDirection={1,0,0}, WidthDirection={0,1,0},
                    Material={0.9, 0.2, 0.2, 0.5});

    /*VisualShape R1(r0={0, 0, 0}, Shape="box",
                    Length=0.2, Width=1, Height=1,
                    LengthDirection={1,0,0}, WidthDirection={0,1,0},
                    Material={0.9, 0.2, 0.2, 0.5});
    VisualShape R2(r0={0, 0, 0}, Shape="box",
                    Length=1, Width=0.2, Height=1,
                    LengthDirection={1,0,0}, WidthDirection={0,1,0},
                    Material={0.9, 0.2, 0.2, 0.5});*/

    VisualShape Vorderrad(
                    r0={0, 0, 0}, Shape="box",
                    Length=0.3, Width=0.1, Height=0.3,
                    LengthDirection={1,0,0}, WidthDirection={0,1,0},
                    Material={0.2, 0.2, 0.9, 0.5});

    /*VisualShape Hinterrad(
                    r0={0, 0, 0}, Shape="box",
                    Length=0.1, Width=0.05, Height=0.3,
                    LengthDirection={1,0,0}, WidthDirection={0,1,0},
                    Material={0.9, 0.2, 0.2, 0.5});*/

    VisualShape Wand(
                    r0={0, 0, 0}, Shape="box",
                    Length=20, Width=0.05, Height=0.3,
                    LengthDirection={1,0,0}, WidthDirection={0,1,0},
                    Material={0.9, 0.2, 0.2, 0.5});

    VisualShape Hindnernis(
```

```
                    r0={-2, Grenze, 0}, Shape="box",
                    Length=2, Width=0.9, Height=1,
                    LengthDirection={1,0,0}, WidthDirection={0,1,0},
                    Material={0.5, 0.5, 0.5, 0.5});

    VisualShape DeichselSchwerpunkt(
                    r0={0, Grenze - Hfb_h, 0}, Shape="box",
                    Length=0.1, Width=0.1, Height=0.1,
                    LengthDirection={1,0,0}, WidthDirection={0,1,0},
                    Material={0.9, 0.5, 0.7, 0.5});
    //###########################################################

equation
  //###########################################################

  //Einspurfahrzeugmodell

  V_p = V_m * cos(Alpha_L);

  der(Theta_L) = V_m / L_r * sin(Alpha_L);

  // lenkbare Vorderrad
  der(X_m) = V_m * cos(Theta_L - Alpha_L);
  der(Y_m) = V_m * sin(Theta_L - Alpha_L);

  // lenkbare Hinterrad
  der(X_p) = V_p * cos(Theta_L);
  der(Y_p) = V_p * sin(Theta_L);

  //#########################################################

  //Konzept der "virtuellen Deichsel"

    Y_dl = H_y - (vf1*Hfb_h) - Y_p;

    X_dl = H_x - X_p;

  Sigma_L = arctan(vf2* Y_dl / X_dl);

  Alpha_L = Sigma_L - Theta_L;

  //#########################################################

  //Umrechnung der RAD Werte in Grad

  Lenkwinkel = Alpha_L * 180 / Modelica.Constants.PI;

  Achswinkel = Theta_L * 180 / Modelica.Constants.PI;
```

```
  // rot starr
  Chassi.S = {{cos(Theta_L),-sin(Theta_L),0},
{sin(Theta_L),cos(Theta_L),0},{0,0,1}};
  Chassi.r = {X_p, Y_p, -0.45};

  // blau lenkbar
  Vorderrad.S = {{cos(Theta_L - Alpha_L),-sin(Theta_L - Alpha_L),0},
{sin(Theta_L - Alpha_L),cos(Theta_L - Alpha_L),0},{0,0,1}};
  Vorderrad.r = {X_m,Y_m, 0};

  // rot starr
  //Hinterrad.S = {{cos(Theta_L),-sin(Theta_L),0},
{sin(Theta_L),cos(Theta_L),0},{0,0,1}};
  //Hinterrad.r = {X_p,Y_p,0};

  Wand.S = identity(3);
  Wand.r = {-10, H_y + Grenze, 0};

  // grau starr
  Hindnernis.S = identity(3);
  Hindnernis.r = {H_x-3.2, H_y - Hfb_h,0};

  DeichselSchwerpunkt.S = identity(3);
  DeichselSchwerpunkt.r = {X_dl,Y_dl,0};

end ParkAssistent;
```

Listing 9: Implementierung der Odometrie und der virtuellen Deichsel mit Dymola

*Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 17. Juli 2008

Ort, Datum

Unterschrift