Projektseminar Echtzeitsysteme

Team MoveIT
Projektseminar eingereicht von
Florian Fischer, Carsten Heinz, Sven Hellwig, David Nass, Matthias Sterzik
am 18. April 2017

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachgebiet Echtzeitsysteme

Elektrotechnik und Informationstechnik (FB18)

Zweitmitglied Informatik (FB20)

Prof. Dr. rer. nat. A. Schürr
Merckstraße 25
64283 Darmstadt

www.es.tu-darmstadt.de

Gutachter: Géza Kulcsár
Betreuer: Géza Kulcsár

Erklärung zum Projektseminar

Hiermit versicheren wir, das vorliegende Projektseminar selbstständig und ohne Hilfe Dritter angefertigt zu haben. Gedanken und Zitate, die wir aus fremden Quellen direkt oder indirekt übernommen haben, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen und wurde bisher nicht veröffentlicht.

Wir erklären uns damit einverstanden, dass die Arbeit auch durch das Fachgebiet Echtzeitsysteme der Öffentlichkeit zugänglich gemacht werden kann.

Darmstadt, den 18. April 2017

(Florian Fischer, Carsten Heinz, Sven Hellwig, David Nass, Matthias Sterzik)

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Im Rahmen des Projektseminars Echtzeitsysteme im Wintersemester 2016/2017 wurden in einem Team von fünf Studenten Aufgaben aus dem Bereich Autonomes Fahren mithilfe eines vom Fachbereich zur Verfügung gestelltes Modellautos bearbeitet.

Ziel dabei war nicht nur das Erlangen von Programmiererfahrung sowie die Auseinandersetzung mit dem verwendeten Software Framework Robot Operation System (ROS), sondern auch das Sammeln von Erfahrung bei Arbeiten im Team und der damit verbundenen Aufgabenaufteilung und geeigneten Zeitplanung. Die Umsetzung und Ergebnisse der bearbeiteten Aufgaben von der Gruppe MoveIT sind in dieser Ausarbeitung dokumentiert.

1.1 Aufgabenstellung

Zum Start des Projektseminars wurden fünf Aufgabenstellungen präsentiert, aus denen jede Gruppe jeweils drei Aufgaben zur Bearbeitung auswählen konnte. Die vorgestellten Aufgaben umfassten folgende Disziplinen, die mit dem zur Verfügung gestellten Modellautos realisiert werden sollten:

1. Rundstrecke fahren ohne Hindernisse
2. Rundstrecke fahren mit Hindernissen
3. Parklücke finden und parallel einparken
4. Fahrbahnmarkierung erkennen und Spur halten
5. Autonomes Erkunden eines unbekannten Terrains

Die erste Aufgabe war obligatorisch und musste von jeder Gruppe bearbeitet werden. Darüber hinaus wurden vom Team MoveIT die Rundstrecke mit Hindernissen sowie das autonome Erkunden eines unbekannten Terrains als Disziplinen zur Bearbeitung ausgewählt.

1.2 Gliederung der Ausarbeitung

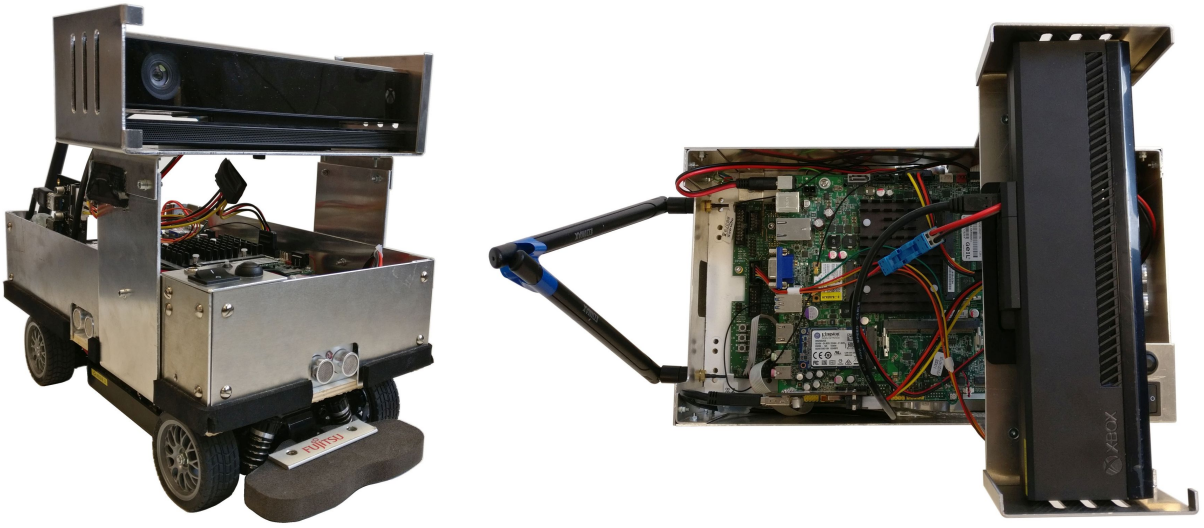
Im weiteren Verlauf dieser Arbeit werden in Kapitel 2 kurz die verwendeten Hard- und Software Plattformen vorgestellt. In Kapitel 3 wird die zur Navigation benötigte Lokalisierung erläutert sowie die Implementierung mit der verwendeten Hardware und Software erklärt. Das zur Regelung erforderliche Fahrzeugmodell sowie die Regelung des Motors und der Lenkung des Modellautos wird in Kapitel 4 verdeutlicht. Anschließend wird die Herangehensweise und die Implementierung der ausgewählten und bearbeiteten Aufgaben — Rundstrecke fahren ohne Hindernisse (Kapitel 5), Rundstrecke fahren mit Hindernissen (Kapitel 6) und das autonome Erkunden eines unbekannten Terrains (Kapitel 7) — beschrieben sowie die jeweiligen Ergebnisse präsentiert und diskutiert. Zum Schluss gibt es im Kapitel **??** noch eine Zusammenfassung und ein Ausblick für die folgenden Projektseminare.

2 Verwendete Hard- und Software

Für die Bearbeitung der Aufgaben und der jeweiligen Disziplinen wurde ein Modellauto zur Verfügung gestellt. Im folgenden Abschnitt werden kurz die wichtigsten genutzten Hardwarebestandteile sowie die verwendete Software beschrieben.

2.1 Hardware

In Abbildung 2.1 ist das Modellauto dargestellt, das im Rahmen des Projektseminars dieses Jahr eingesetzt wurde. Auf dem Auto befindet sich ein Mainboard, das mit dem Betriebssystem Lubuntu läuft und als Computer verwendet wird. Über die UART-Schnittstelle ist das Mainboard per USB-Anschluss mit einem Mikrocontroller-Board verbunden, das zur Steuerung der Aktorik sowie zur Erfassung und Verarbeitung der Sensorik eingesetzt wird.



(a) Frontansicht des Modellautos **(b)** Draufsicht des Modellautos

Abbildung 2.1: Ansichten des zur Verfügung gestellten Modellautos

Als Sensoren kommen ein Hall-Sensor, ein Beschleunigungs- und Drehratensensor und ein Magnetometer zum Einsatz, die im Detail in Kapitel 3.1 beschrieben werden. Oberhalb des Autos ist eine Kinect von Microsoft angebracht, die hauptsächlich zur Lokalisierung (siehe Kapitel 3) und zur Identifizierung von Hindernissen auf der Fahrbahn (siehe Kapitel 6) verwendet wird.

2.2 Software

Auf dem PC kommt das Robotic Operation System (ROS) zum Einsatz. ROS ist ein Software-Framework für Roboter, das unter Linux läuft und Dienste (u.a. die Hardwareabstraktion) sowie häufig benötigte Funktionen zur Verfügung stellt. Zusätzlich stellt ROS auch die Kommunikation zwischen verschiedenen Prozessen bereit, die als Nodes bezeichnet werden. Diese Prozesse können zudem auf mehrere Systeme im Netzwerk verteilt werden. ROS enthält eine Open-Source-Bibliothek, die viele Implementierungen

von allgemeinen Funktionen und Algorithmen in der Robotik beinhaltet. Die in diesem Projekt eingesetzten Pakete werden in den folgenden Kapiteln beschrieben.

3 Lokalisierung

Die Lokalisierung, d.h. die Ermittlung von Position und Orientierung, spielt eine wichtige Rolle für die Navigation autonomer mobiler Roboter. Die Herausforderung hierbei besteht darin, aus verrauschten, fehlerbehafteten Messdaten oft auch redundanter Sensoren eine möglichst gute Schätzung der unbekannten Position zu berechnen. Zu diesem Zweck werden i.A. Methoden der Daten- und Sensorfusion eingesetzt [DWH08].

Sowohl die Pfadfolgeregelung beim Rundkurs ohne Hindernisse, als auch die Trajektorienplanung beim Rundkurs mit Hindernissen benötigen Informationen zur aktuellen Fahrzeugposition. Auch beim autonomen Erkunden wird die Pose bei der Generierung der Karte mit einbezogen. Für diesen Zweck wurde bereits eine einfache Odometrie zur Verfügung gestellt, die die zurückgelegte Strecke über den Radsensor und den Gierwinkel per Integration des Gyroskop-Messwerts schätzt. Aufgrund von Messfehlern, insbesondere des Gyrosensors, ist diese Art der Positionsbestimmung driftbehaftet und führt ohne Korrektur zu einer immer größer werdenden Abweichung des Schätzwerts von der tatsächlichen Position. Zudem werden nicht alle vorhandenen Sensordaten verwendet.

Zur Verbesserung der Lokalisierung wurde deshalb eine häufig verwendete Methode der Sensorfusion, das *Extended Kalman Filter* (EKF), implementiert und mit dem ROS-Paket AMCL kombiniert. In den folgenden Abschnitten wird dazu zunächst auf die Sensoren und AMCL eingegangen, ein kinematisches Systemmodell des Fahrzeugs hergeleitet und das EKF vorgestellt. Zuletzt werden die Ergebnisse dieser Methode mit der bisherigen Odometrie verglichen.

3.1 Sensoren

3.1.1 Koordinatensysteme

Zur mathematischen Beschreibung der Sensoren werden zwei Koordinatensysteme (KOS) festgelegt: ein ortsfestes Inertialsystem und ein fahrzeugfestes Koordinatensystem. Das Inertialsystem wird mit dem Index I bezeichnet. Das fahrzeugfeste KOS F hat seinen Ursprung in der Mitte der Hinterachse. Die x-Achse des fahrzeugfesten Systems zeigt in Längsrichtung des Autos, die y-Achse nach links und die z-Achse nach oben. Desweiteren ist $\mathbf{r} = \begin{bmatrix} r_x, & r_y \end{bmatrix}^{\mathrm{T}}$ der Ortsvektor der IMU im fahrzeugfesten System (Abb. 3.1).

3.1.2 Gyroskop und Beschleunigungssensor

Auf der IMU befinden sich ein Drei-Achsen-Gyroskop und ein Drei-Achsen-Beschleunigungssensor, die alle 5ms neue Messwerte zur Verfügung stellen. Für die Modellierung wird davon ausgegangen, dass sich das Fahrzeug nur in Längsrichtung bewegen kann und der Nick- und Rollwinkel stets null ist. Es werden deshalb nur die Messwerte des Gyroskops um die z-Achse und des Beschleunigungssensors in x-Richtung verwendet.

Der Gierwinkel wird mit θ, die Gierrate mit $\dot{\theta}$ bezeichnet. Der Messwert $\tilde{\omega}$ des Gyroskops hat einen konstanten Bias b_{gyro} und ist mit einem mittelwertfreien Rauschen n_{gyro} überlagert:

$$\tilde{\omega} = \dot{\theta} - b_{\text{gyro}} - n_{\text{gyro}}. \tag{1}$$

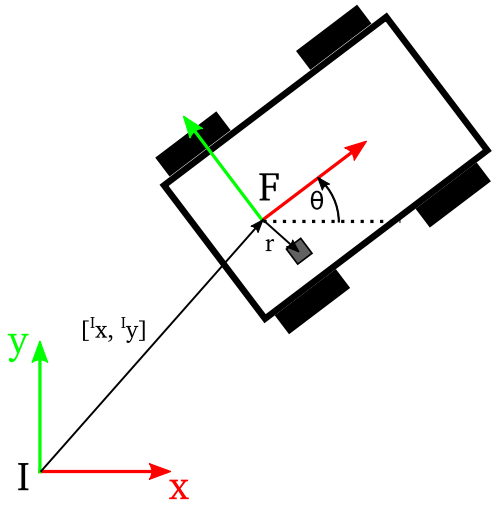


Abbildung 3.1: Koordinatensysteme und Ortsvektoren

Auch die Messwerte des Beschleunigungssensors sind bias- und rauschbehaftet. Der Bias b_a wird ebenfalls als konstant angenommen, das Rauschen analog mit n_a bezeichnet. Der Sensor misst in x-Richtung des fahrzeugfesten KOS

$$\tilde{a}_x = {}^F\ddot{x} - r_x\dot{\theta}^2 - r_y\ddot{\theta} - b_{a,x} - n_{a,x}. \tag{2}$$

Die Terme $r_x\dot{\theta}^2$ und $r_y\ddot{\theta}$ sind vernachlässigbar klein, sodass sich für den Messwert näherungsweise

$$\tilde{a}_x \approx {}^F\ddot{x} - b_{a,x} - n_{a,x} \tag{3}$$

ergibt.

Es wird desweiteren davon ausgegangen, dass das Messrauschen normalverteilt ist. Um die Varianzen zu bestimmen, werden die Messwerte am ruhenden Fahrzeug einige Minuten lang aufgezeichnet. Mit der Formel für die Stichprobenvarianz

$$\mathrm{Var}(\mathbf{x}) \approx \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2, \qquad \bar{x} = \frac{1}{N}\sum_{i=1}^{N}x_i \tag{4}$$

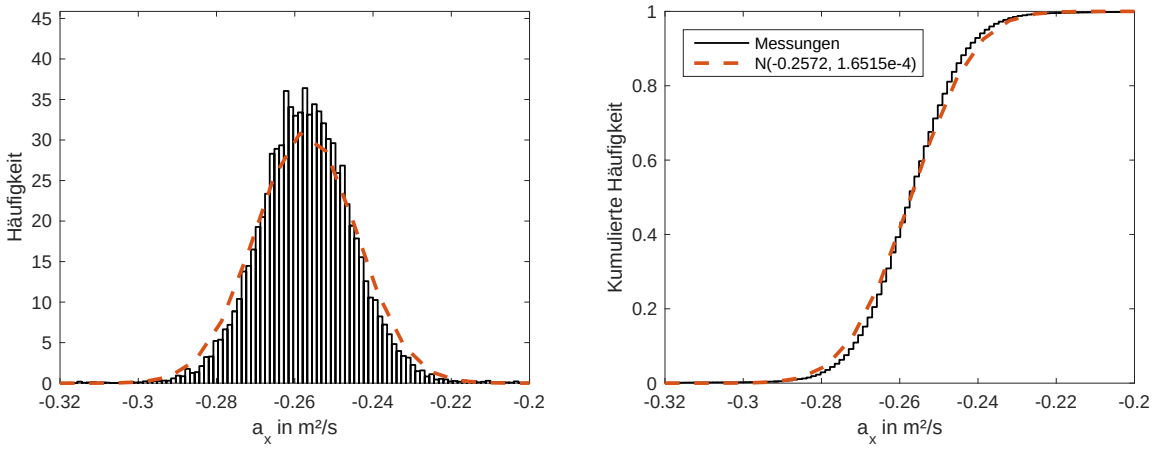
ergeben sich die in Abbildung 3.2 und 3.3 dargestellten Verteilungen.

3.1.3 Magnetometer

Zusätzlich befindet sich auf der IMU ein Drei-Achsen-Magnetometer. Dieses misst die Stärke der magnetischen Flussdichte in drei Richtungen. Aus diesem Wert kann bei Annahme einer konstanten Ausrichtung in z-Richtung ein Gierwinkel bestimmt werden.

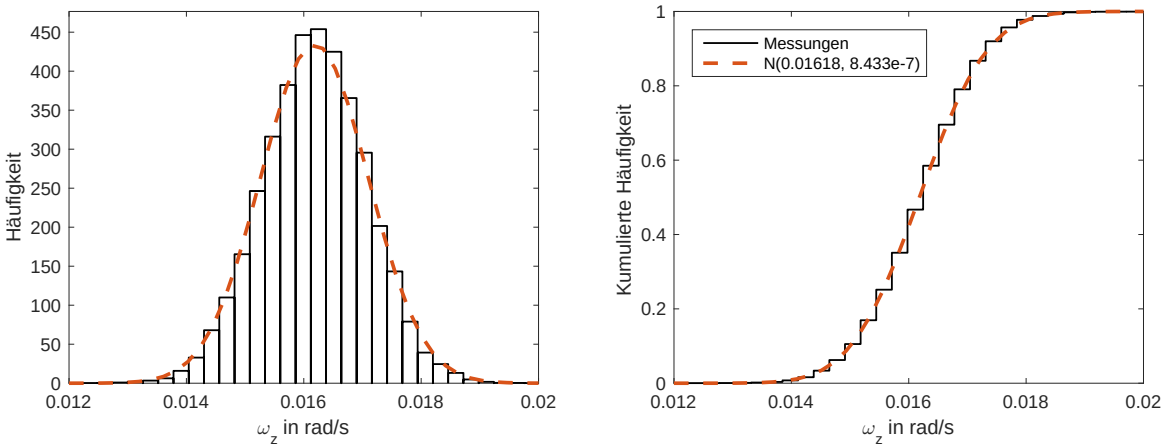
$$\theta_{\mathrm{mag}} = \mathrm{atan2}\left(B_x, B_y\right) \tag{5}$$

Diese Berechnung setzt eine Kalibrierung des Sensors voraus. Ein ideales Magnetometer misst bei Rotation um alle Achsen Werte auf der Oberfläche einer Kugel um den



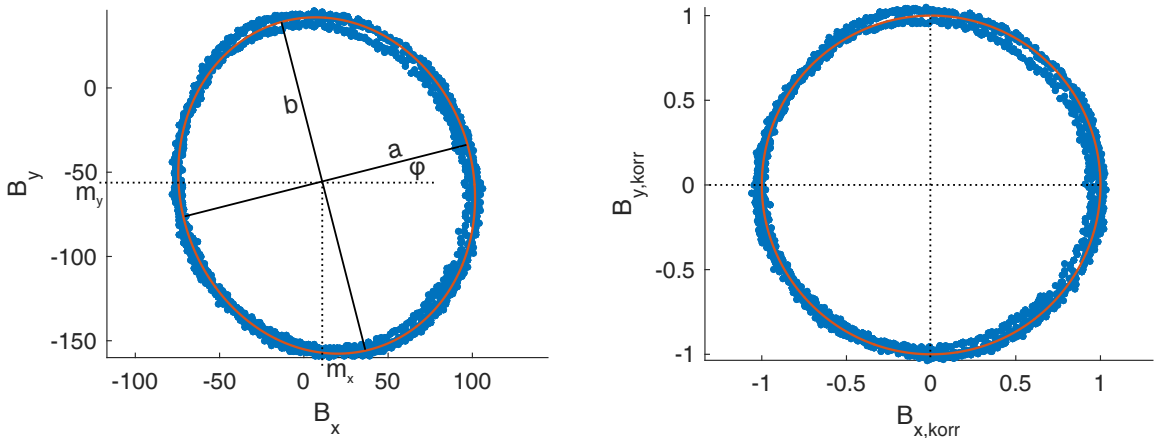
(a) Dichtefunktion **(b)** Verteilungsfunktion

Abbildung 3.2: geschätzte Verteilung der Messwerte des Beschleunigungssensors



(a) Dichtefunktion **(b)** Verteilungsfunktion

Abbildung 3.3: geschätzte Verteilung der Messwerte des Gyroskops



(a) ohne Korrektur **(b)** nach Korrektur

Abbildung 3.4: Kalibrierung des Magnetometers

Nullpunkt. Das Magnetfeld am Sensor wird allerdings z.B. durch das Gehäuse des Autos beeinflusst. Durch eine Offset-Korrektur wird zuerst der Mittelpunkt des ermittelten Ellipsoids auf den Nullpunkt verschoben [GEEPP06]. Anschließend wird eine Transformation des Ellipsoids (bzw. im zweidimensionalen Fall der Ellipse) in eine Kugel bzw. Kreis bestimmt. Im zweidimensionalen Fall erhält man die normierten, korrigierten Flussdichten

$$\mathbf{B}_{\text{korr}} = \begin{bmatrix} a^{-1} & 0 \\ 0 & b^{-1} \end{bmatrix} \begin{bmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{bmatrix} \left(\begin{bmatrix} B_x \\ B_y \end{bmatrix} - \begin{bmatrix} m_x \\ m_y \end{bmatrix} \right) \tag{6}$$

mit dem Ellipsenmittelpunkt m_x, m_y, den Radien a und b und dem Drehwinkel φ (Abb. 3.4). Ohne externe Störungen kann damit ein genauer, absoluter Gierwinkel bestimmt werden.

Zuerst wurde das Magnetometer der IMU verwendet. Dieses befindet sich über einem Motor und wird daher bei Motorbewegungen beeinflusst. Daher wurde ein externes Magnetometer an eine Außenseite montiert, wo der Motor nahezu keine Beeinflussung zeigt. Da das Magnetfeld in Gebäuden sehr schwach ist, treten auf dem Rundkurs vermehrt Störungen von magnetischen Gegenständen auf. Schlussendlich wurde aufgrund der Störungen daher das Magnetometer nicht zur Lokalisierung verwendet.

3.1.4 Drehgeber

Am linken Hinterrad sind acht Magnete befestigt. Ein Hall-Sensor, der neben dem Rad am Chassis angebracht ist, zählt, wie oft sich ein Magnet an ihm vorbeibewegt und misst die Zeitdifferenzen. Bei bekanntem Raddurchmesser ergibt sich die durchschnittliche Geschwindigkeit zwischen zwei Messungen mit

$$|\bar{v}| = \frac{\pi d}{8 \cdot \Delta t}. \tag{7}$$

Die Drehrichtung muss aus der Stellgröße des Motors bestimmt werden. Der so gewonnene Messwert ist

$$\tilde{v}'_{\text{Rad}} = {}^{F}\dot{x} - \frac{B}{2} \cdot \dot{\theta} + n_v \tag{8}$$

mit der Fahrzeugbreite B. Durch den Rauschterm n_v sollen z.B. Ungenauigkeiten bei der Anbringung der Magnete und numerische Fehler der Messung berücksichtigt werden. Der Radschlupf wird vernachlässigt.

Problematisch ist diese Art der Geschwindigkeitsmessung, wenn das Fahrzeug sehr langsam fährt oder stehen bleibt: Solange kein Magnet sich am Sensor vorbeibewegt, wird kein neuer Messwert zur Verfügung gestellt. Deshalb wird davon ausgegangen, dass die Geschwindigkeit null ist, wenn nach 0,3s keine neue Messung vorliegt.

Der Einfluss der Gierrate auf den Messwert (Gl. 8) wird mithilfe der Gierratenmessung (bzw. des aktuellen Schätzwerts für $\dot{\theta}$) kompensiert:

$$\tilde{v}_{\text{Rad}} = \tilde{v}'_{\text{Rad}} + \frac{B}{2} \cdot \hat{\dot{\theta}} \approx {}^{F}\dot{x} + n_v. \tag{9}$$

3.2 AMCL

AMCL steht für Adaptive Monte-Carlo Lokalisierung, welche ein probabilistischer Ansatz zur Lokalisierung eines Roboters auf einer Karte anhand einer Partikel-Menge ist. Für ROS steht ein gleichnamiges Paket[1] zur Verfügung, welches diesen Algorithmus implementiert und in diesem Projekt eingesetzt wurde.

AMCL benötigt als Input eine Karte und einen Laserscan. Diese Daten werden miteinander abgeglichen, um daraus die aktuelle Position schätzen zu können, während sich das Modellauto bewegt. Zu Beginn benötigt AMCL eine Initialposition, die manuell über das Dashboard gesetzt werden kann. Der Laserscan wird für die Lokalisierung mithilfe des Tiefenbilds der Kinect erzeugt, welches im folgenden Abschnitt beschrieben wird.

3.2.1 Tiefenbild der Kinect

Um die Partikel für die Lokalisierung zu erhalten, wird das Tiefenbild der Kinect verwendet. Der Wert eines Pixels im Tiefenbild ist proportional zum Abstand. Mit dem ROS-Paket depthimage_to_laserscan[2] kann aus diesem Tiefenbild eine zweidimensionale Partikel-Menge bestimmt werden. Dabei wird eine Linie im Tiefenbild definiert und daraus die Abstände extrahiert.

Jedoch enthält das Tiefenbild Störungen und Rauschen. Daher wurde ein Filter auf Basis der Bildverarbeitungsbibliothek OpenCV erstellt. Dazu wird zuerst das Bildformat von ROS in das von OpenCV umgewandelt. Im ersten Schritt werden Störungen entfernt. Kann die Kinect keinen Abstand bestimmen, setzt sie die zugehörigen Pixel im Tiefenbild auf 0. Diese würden als Objekte direkt vor dem Auto wahrgenommen werden und werden deshalb durch Pixel mit maximalem Wert (d.h. maximaler Distanz) ersetzt.

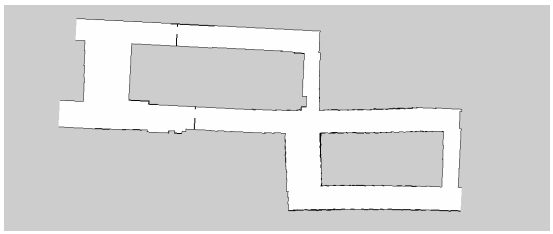
[1] http://wiki.ros.org/amcl

[2] http://wiki.ros.org/depthimage_to_laserscan

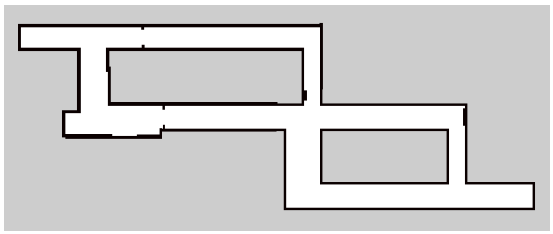
Anschließend wird das Rauschen behandelt. Das Rauschen des Tiefenbilds ähnelt einem *Salt and Pepper* Rauschen, d.h. es gibt einige Ausreißer im Bild. Dies kann mit einem Median-Filter kompensiert werden.

3.2.2 Verbesserte Karte

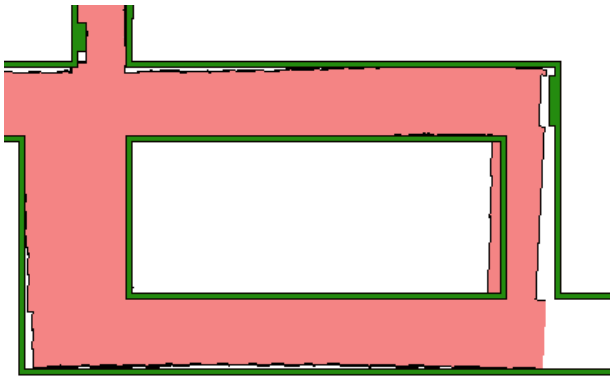
Für eine korrekte und möglichst exakte Lokalisierung mit AMCL ist es wichtig, eine genaue Karte zu haben. Die zur Verfügung gestellte Karte in Abbildung 3.5a ist ein etwas verzerrt und bezüglich der Abstände und Entfernungen zu ungenau.



(a) Alte Karte



(b) Optimierte Karte



(c) Vergleich des Rundkurses

Abbildung 3.5: Verbesserte Karten

Daher wurde unter Berücksichtigung der korrekten Abstände und Längen der Wände, die Karte der Umgebung neu erstellt (Abbildung 3.5b). In Abbildung 3.5c ist die neue Karte im Vergleich zur gegebenen Karte dargestellt, wobei die rote Fläche die alte Karte und die grüne Umrandung die neue Karte darstellt.

3.3 Extended Kalman Filter

3.3.1 Kinematisches Systemmodell

Für das EKF muss zunächst ein passendes Systemmodell aufgestellt werden[3]. Dazu werden die Beschleunigung und die Gierrate als Eingangsgrößen betrachtet. Dies bietet sich an, weil die Messdaten des Beschleunigungssensors und des Gyroskops regelmäßig und vergleichsweise oft vorliegen. Der Zustandsvektor $\mathbf{x} = [{}^{I}x, {}^{I}y, {}^{F}\dot{x}, \theta]^{\mathrm{T}}$ besteht aus der x- und y-Position im Inertialsystem, der Geschwindigkeit in x-Richtung des fahrzeugfesten

[3] Die Herleitung des Systemmodells und des EKF geschieht in Anlehnung an das Vorgehen im von Dr. Lenz zur Verfügung gestellten Handout.

KOS und dem Gierwinkel (vgl. Abb. 3.1). Man erhält die folgenden Differentialgleichungen:

$$\dot{\mathbf{x}} = \begin{bmatrix} {}^{I}\dot{x} \\ {}^{I}\dot{y} \\ {}^{F}\ddot{x} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} {}^{F}\dot{x} \cdot \cos\theta \\ {}^{F}\dot{x} \cdot \sin\theta \\ b_{a,x} \\ b_{\text{gyro}} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_x \\ \omega \end{bmatrix}. \tag{10}$$

Die tatsächliche Beschleunigung und Gierrate ist nicht bekannt. Deshalb werden die Gleichungen (1) und (3) nach a_x bzw. ω aufgelöst und in Gleichung (10) eingesetzt. Um später die unbekannten Sensorbiase schätzen zu können, wird der Zustandsvektor um $b_{a,x}$ und b_{gyro} erweitert. Offensichtlich gilt $\dot{b} = 0$. Somit ergibt sich das zeitkontinuierliche nichtlineare System

$$\dot{\mathbf{x}} = \begin{bmatrix} {}^{I}\dot{x} \\ {}^{I}\dot{y} \\ {}^{F}\ddot{x} \\ \dot{\theta} \\ \dot{b}_{a,x} \\ \dot{b}_{\text{gyro}} \end{bmatrix} = \begin{bmatrix} {}^{F}\dot{x} \cdot \cos\theta \\ {}^{F}\dot{x} \cdot \sin\theta \\ b_{a,x} \\ b_{\text{gyro}} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \tilde{a}_x \\ \tilde{\omega} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ n_{a,x} \\ n_{\text{gyro}} \\ 0 \\ 0 \end{bmatrix} = \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{n}_x). \tag{11}$$

Mit dem Euler-Cauchy-Verfahren

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \int_{t_k}^{t_{k+1}} \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{n}_x)\,\mathrm{d}t \approx \mathbf{x}_k + T \cdot \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{n}_x) \tag{12}$$

und der Abtastzeit $T = (t_{k+1} - t_k)$ erhält man schließlich das zeitdiskrete System

$$\mathbf{x}_{k+1} = \begin{bmatrix} {}^{I}x \\ {}^{I}y \\ {}^{F}\dot{x} \\ \theta \\ b_{a,x} \\ b_{\text{gyro}} \end{bmatrix}_{k+1} = \mathbf{x}_k + \begin{bmatrix} T \cdot {}^{F}\dot{x} \cdot \cos\theta \\ T \cdot {}^{F}\dot{x} \cdot \sin\theta \\ T \cdot b_{a,x,k} \\ T \cdot b_{\text{gyro},k} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ T & 0 \\ 0 & T \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \tilde{a}_{x,k} \\ \tilde{\omega}_k \end{bmatrix} + \mathbf{n}_{x,k}. \tag{13}$$

Die Gleichungen der restlichen Sensoren werden zum Ausgangsvektor

$$\tilde{\mathbf{y}}_k = \begin{bmatrix} \tilde{v}_{\text{Rad}} \\ \tilde{\theta}_{\text{mag}} \\ \tilde{x}_{\text{AMCL}} \\ \tilde{y}_{\text{AMCL}} \\ \tilde{\theta}_{\text{AMCL}} \end{bmatrix}_k = \mathbf{C}\mathbf{x}_k + \mathbf{n}_{y,k} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} {}^{I}x \\ {}^{I}y \\ {}^{F}\dot{x} \\ \theta \\ b_{a,x} \\ b_{\text{gyro}} \end{bmatrix}_k + \mathbf{n}_{y,k} \tag{14}$$

zusammengefasst.

3.3.2 Auslegung des Filters

Das Extended Kalman Filter ist eine Version des Kalmanfilters für nichtlineare Systeme. Es stellt anhand des Systemmodells im *Prädiktionsschritt* regelmäßig Vermutungen zum aktuellen Systemzustand an und korrigiert diese im *Korrekturschritt*, wenn neue Messungen vorliegen. Das EKF berücksichtigt außerdem stochastische Größen, nämlich die (Ko-)Varianzen des System- und Messrauschens.

Der Prädiktionsschritt wird durch folgende Formeln beschrieben [Sim06, S. 409]:

$$\hat{\mathbf{x}}_k^- = \mathbf{f}\left(\mathbf{x}_{k-1}^+, \mathbf{u}_{k-1}\right) \tag{15}$$

$$\mathbf{P}_k^- = \mathbf{F}_{k-1}\mathbf{P}_{k-1}^+\mathbf{F}_{k-1}^{\mathrm{T}} + \mathbf{Q}_{k-1}. \tag{16}$$

Die Notation $\hat{\mathbf{x}}_k$ bedeutet dabei „Schätzwert für $\mathbf{x}$ zum Zeitpunkt t_k“. Prädizierte Werte werden mit „−“, korrigierte mit „+“ gekennzeichnet. Je kleiner die Einträge der Kovarianzmatrix $\mathbf{P}$ sind, desto sicherer ist sich das Filter seiner Schätzung. Die Matrix

$$\mathbf{F}_{k-1} = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{x}_{k-1}, \mathbf{u}_{k-1}} \tag{17}$$

ist die Jacobimatrix der Systemfunktion und muss i.A. in jedem Prädiktionsschritt neu berechnet werden.

Der Korrekturschritt

$$\mathbf{K}_k = \mathbf{P}_k^-\mathbf{C}^{\mathrm{T}}\left(\mathbf{C}\,\mathbf{P}_k^-\mathbf{C}^{\mathrm{T}} + \mathbf{R}_k\right)^{-1} \tag{18}$$

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{K}_k\left(\mathbf{y}_k - \mathbf{C}\,\hat{\mathbf{x}}_k^-\right) \tag{19}$$

$$\mathbf{P}_k^+ = \left(\mathbf{I} - \mathbf{K}_k\mathbf{C}\right)\mathbf{P}_k^- \tag{20}$$

wird jedes mal, wenn ein neuer Messwert zur Verfügung steht, ausgeführt. Es ist zu beachten, dass hier keine Jacobimatrix berechnet werden muss, da die Ausgangsgleichung des Systems linear ist.

Die Kovarianzmatrix $\mathbf{Q}$ des Systemrauschens ist hier

$$\mathbf{Q} = \mathrm{diag}\left(0,\, 0,\, 1{,}65 \cdot 10^{-4},\, 8{,}4 \cdot 10^{-7},\, 0,\, 0\right). \tag{21}$$

Die Werte wurden mithilfe der empirisch bestimmten Varianzen aus Abschnitt 3.1.2 und durch Simulation mit aufgenommenen Messdaten in Simulink gewählt. Der Eintrag für den Hall-Sensor in der Kovarianzmatrix $\mathbf{R}$ des Messrauschens wurde auf 0,005 festgelegt. AMCL liefert eine Kovarianzmatrix zu seinen Schätzwerten, die hier verwendet wird.

Wegen der unregelmäßigen Abtastzeiten — zwischen zwei Schätzungen von AMCL vergehen oft mehrere Sekunden, die Frequenz der Hall-Sensor-Messungen ist abhängig von der Geschwindigkeit des Fahrzeugs — liegt nicht immer ein kompletter Messvektor $\mathbf{y}_k$ gleichzeitig vor. Deshalb werden Messwerte unabhängiger Sensoren stattdessen sequentiell verarbeitet: Der Korrekturschritt wird jeweils nur mit den entsprechenden Zeilen von $\mathbf{C}$ und $\mathbf{R}_k$ ausgeführt [Sim06, S. 150ff].

Zuletzt müssen noch die Startwerte $\hat{\mathbf{x}}_0^+$ und $\mathbf{P}_0^+$ bestimmt werden. Das Fahrzeug ist zu Beginn der Schätzung in Ruhe. Damit kann für die Geschwindigkeit ${}^F\dot{x}_0 = 0$ angenommen werden. Die Startposition und der Gierwinkel wird vom Benutzer über die GUI vorgegeben. Für die Sensorbiase wird ein Startwert durch Mittelung von 1000 Messwerten zu Beginn der Fahrt festgelegt.

Der Startwert $\mathbf{P}_0^+$ drückt aus, wie genau der Startzustand des Systems bekannt ist. Je größer die Einträge der Matrix sind, desto stärker korrigiert das Filter zu Beginn die Prädiktion. Als guter Wert hat sich

$$\mathbf{P}_0^+ = \mathrm{diag}\left(1 \cdot 10^{-2}, 1 \cdot 10^{-2}, 1 \cdot 10^{-10}, 1 \cdot 10^{-3}, 1 \cdot 10^{-2}, 1 \cdot 10^{-4}\right) \tag{22}$$

herausgestellt.

3.4 Ergebnisse

Zur Bewertung und zum Vergleich der vorgegebenen Odometrie und des EKF wurde der Rundkurs im HBI einmal abgefahren und die Daten aufgezeichnet. In Abbildung 3.6a sind die Positionsschätzungen der Odometrie, des EKF und des EKF in Kombination mit AMCL dargestellt. Die drei Methoden liefern ähnliche Ergebnisse. Die Odometrie zeigt geringe Abweichungen von den anderen Schätzungen. Dies ist zum einen auf den Drift zurückzuführen, der bei der Integration der Gierrate entsteht: Die Odometrie zieht zwar zur Biaskompensation einen konstanten Wert vom Gyromesswert ab, aktualisiert diesen Wert nicht zur Laufzeit. Die Biasschätzung des EKF findet dagegen auch bei ungünstigen Startwerten nach kurzer Zeit sinnvolle Biasparameter (Abb. 3.6b).

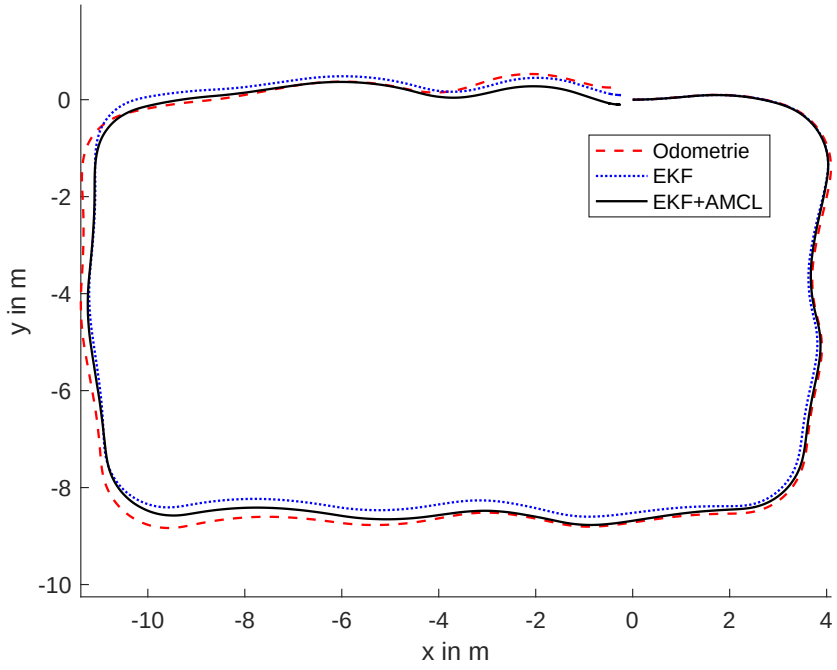
Darüber hinaus ist es schwierig, die Ergebnisse zu bewerten, weil die tatsächlich gefahrene Strecke nicht genau bekannt ist. Hierfür wäre es sinnvoll, eine Teststrecke zu markieren oder vorhandene Markierungen zu verwenden (z.B. in einer Sporthalle). Möglicherweise können bei der Positionsschätzung ähnliche Ergebnisse mit einer Kombination von AMCL und der Odometrie erzielt werden, wie mit AMCL und EKF. Es muss je nach Rahmenbedingungen und Aufgaben des Fahrzeugs entschieden werden, ob der Implementierungsaufwand gerechtfertigt ist.

Vorteil des EKF ist allerdings, dass zu bekannten Abtastzeiten Schätzwerte des Systemzustands zur Verfügung stehen. Dies vereinfacht den Reglerentwurf. Zudem kann das Filter einfach erweitert werden, sodass weitere Sensoren — z.B. zusätzliche Drehgeber an den anderen Rädern oder visuelle Odometrie — berücksichtigt werden können. Zudem können die Sensor- und Rauschmodelle angepasst werden, z.B. indem die Sensorbiase als zeitlich veränderlich angenommen werden, falls das Fahrzeug über einen längeren Zeitraum in Betrieb ist.

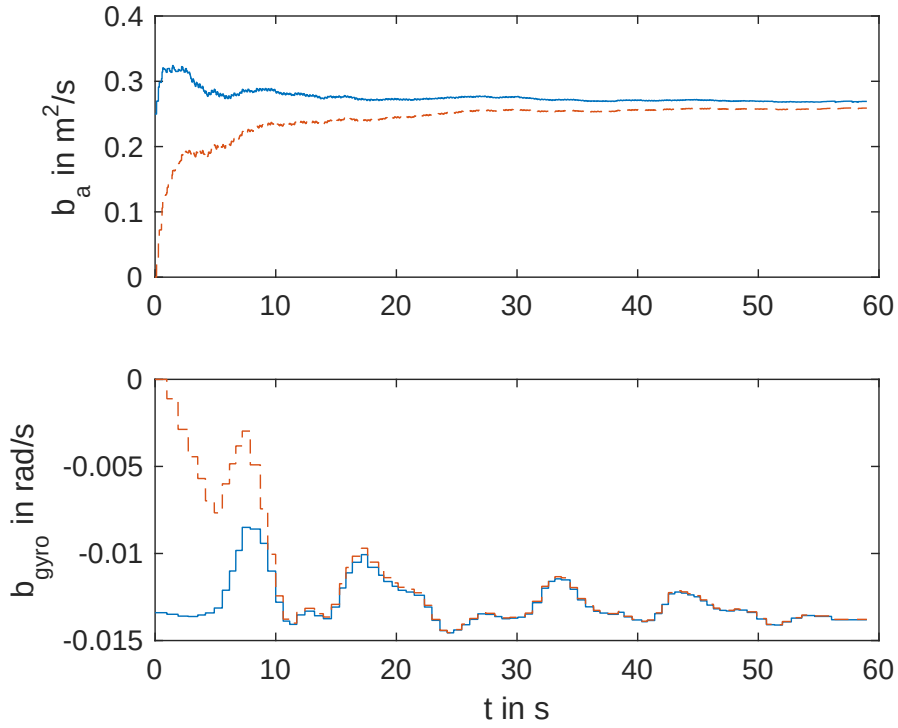
Eine mögliche Verbesserung für das hier vorgestellte Filter wäre die Verwendung von Quaternionen statt Eulerwinkeln im Systemmodell. Hierdurch würde zum einen die mühsame Behandlung des eingeschränkten Wertebereichs der Eulerwinkel ($-\pi$ bis π) entfallen. Zum anderen würden die Berechnungen numerisch effizienter [Keh14].

Des Weiteren könnte der Gierratenterm bei der Geschwindigkeitsmessung (Gl. (8)) direkt im Systemmodell berücksichtigt werden.

Es wäre außerdem denkbar, das Magnetometer einzusetzen, um Landmarken (Heizungen, Aufzugtüren etc.) zu erkennen und so die Lokalisierung zu stützen [GT12], [WSE+12].



(a) Vergleich der Lokalisierungsmethoden



(b) Bias-Schätzung mit guten (durchgezogen) und schlechten Startwerten (gestrichelt)

Abbildung 3.6: Ergebnisse der Lokalisierung

4 Modellbildung und Identifikation

4.1 Fahrzeugmodell

Damit eine Regelung des Fahrzeuges erfolgen kann, muss zunächst ein mathematisches Modell hergeleitet werden. Dazu gibt es verschiedene Ansätze unterschiedlicher Komplexität. Besonders einfach ist das Ackermannmodell. Es betrachtet als Eingangsgröße die Geschwindigkeit und vernachlässigt somit jegliche Massen und Trägheiten. Auch wird angenommen, dass die Bewegung des Fahrzeugs nur in Längsrichtung der Räder erfolgen kann. Somit kann auch die Reibung und der Schlupf zwischen Rad und Boden außer Betracht gelassen werden und die Bewegung ergibt sich ausschließlich aus geometrischen Beziehungen. Des Weiteren werden die Räder einer Achse zu einem einzelnen zusammengefasst und man erhält ein sogenanntes Einspurmodell (Abbildung 4.1) [SHB13, S.223ff].

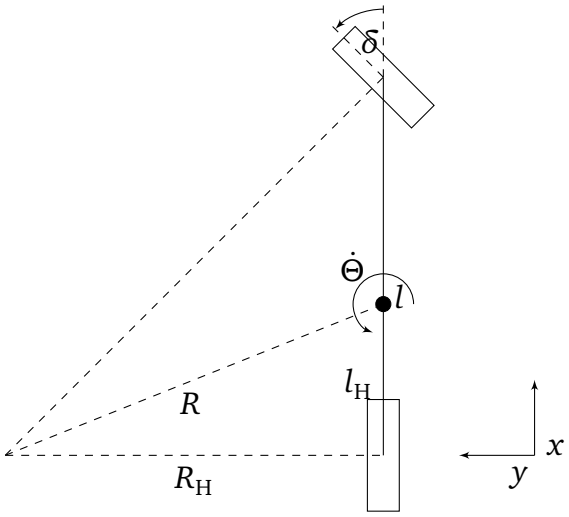


Abbildung 4.1: Einspurmodell

Im späteren Verlauf dieser Arbeit wird eine Methode vorgestellt, die das Fahrzeug einem bestimmten Pfad folgen lässt. Dazu wird angenommen, dass die Geschwindigkeit in x-Richtung (Fahrzeugrichtung) konstant ist. Da ein Pfad unabhängig von der Zeit und die Geschwindigkeit vorgegeben ist, reicht es aus die Querabweichung in y-Richtung zum Pfad auszuregeln.

Aus den oben genannten Annahmen gelingt es folgendes Modell herzuleiten:

$$\begin{bmatrix} \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & v \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y \\ \theta \end{bmatrix} + \begin{bmatrix} v\frac{l_H}{l} \\ \frac{v}{l} \end{bmatrix} \delta^* \tag{23}$$

Die Geschwindigkeit ist mit v bezeichnet. Der Lenkwinkel wird mit δ und der Gierwinkel um die z-Achse ist θ. Die Fahrzeuglänge ist l und l_H ist die Länge von Hinterachse zum Schwerpunkt. Da die Geschwindigkeit, wie bereits erwähnt als konstant angenommen wird, ist das System linear. Es ist zu beachten, dass $\delta^* = \tan(\delta)$ gilt. Somit kann

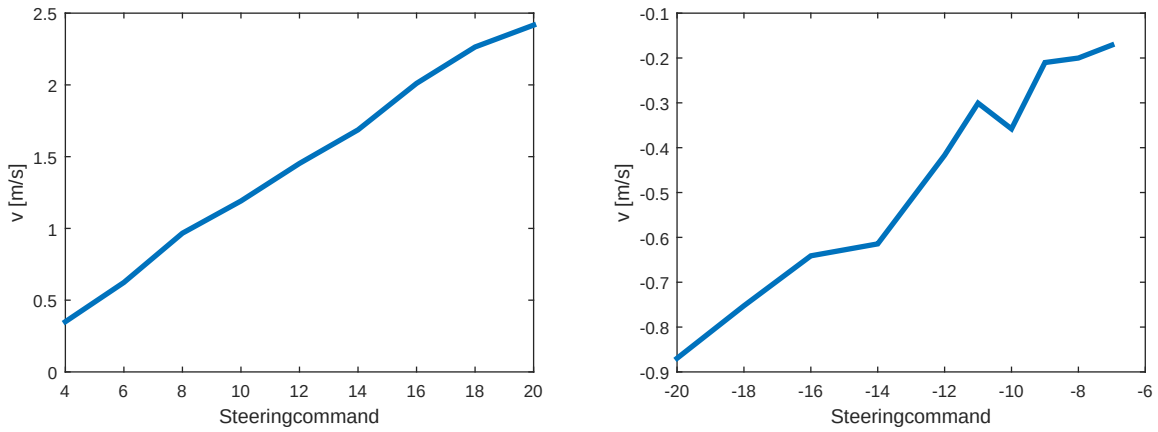
für kleine Winkel $\delta^* = \delta$ angenommen werden. Aus der Zustandsgleichung ergibt sich folgende Übertragungsfunktion:

$$G(s) = \frac{y(s)}{\delta(s)} = \frac{v l_H}{l} \frac{\frac{v}{l_H} + s}{s^2} \tag{24}$$

Für dieses System kann nun ein Regler entworfen werden. Der Reglerentwurf und das genaue Regleprinzip werden im Abschnitt 5 vorgestellt.

4.2 Unterlagerte Regler

Das Ackermannmodell geht davon aus, dass die Stellgrößen, also Geschwindigkeit v und Lenkwinkel δ, ideal auf das System wirken. Die Ansteuerung der Stellglieder erfolgt allerdings über *Motor-* bzw *Steeringcommands*. Das heißt, um eine Geschwindigkeit in $\frac{m}{s}$ oder einen Lenkwinkel in rad einzustellen, muss eine Umrechnung in das jeweilige *Command* erfolgen. Leider hat sich gezeigt, dass diese Umrechnung nichtlinear erfolgt und durch Störungen verzerrt wird. Somit fällt es schwer, eine gewünschte Größe in ausreichend guter Genauigkeit einzustellen. Um dem entgegenzuwirken soll eine Kennlinie, die das Verhältnis zwischen Sollgröße und dem jeweiligen *Command* angibt, aufgezeichnet werden. Sonstige Störungen, aber auch Ungenauigkeiten der Kennlinie werden zusätzlich mit einem Regler eliminiert.



(a) Vorwärts **(b)** Rückwärts

Abbildung 4.2: Kennlinie Geschwindigkeit

- Motor-Identifizierung
  Während einer Testfahrt wurden die Ist-Geschwindigkeit des Fahrzeugs aufgezeichnet und die *Steeringcommands* schrittweise erhöht. Aus den daraus gewonnenen Daten konnte anschließend eine Kennlinie erstellen werden, mit deren Hilfe die Umrechnung von Sollgeschwindigkeit in *Steeringcommand* erfolgen kann. Die Kennlinie ist in Abbildung 4.2 zu sehen. Für das Rückwärsfahren ist bei Steeringcommand -9 ein deutlicher Ausreißer zu sehen. Dabei handelt es sich wahrscheinlich um einen Messfehler. Für eine Reglerauslegung muss zunächst eine gewisse Kenntnis über das System vorhanden sein. Dies kann unter anderem

durch Analyse der Sprungantwort des Systems gelingen. Dazu wurde die Sollgröße sprunghaft auf einen Wert erhöht und die daraus resultierende Geschwindigkeit aufgezeichnet. Anschließend konnte die Sprungantwort mit Hilfe der *Curve Fitting Toolbox* von Matlab näherungsweise durch ein PT1-Glied identifiziert werden (Abbildung 4.4). Es folgt die Übertragungsfunktion aus Gleichung (25). Die Parameter K_{PT1} und T_{PT1} wurden durch Matlab errechnet.

$$G_{\mathrm{PT1}}(s) = \frac{K_{\mathrm{PT1}}}{T_{\mathrm{PT1}}s + 1} \tag{25}$$

Mit den gewonnenen Systeminformationen kann nun ein Reglerentwurf vollzogen werden. Damit das System trotzt Störungen eine stationäre Genauigkeit garantiert wurde ein PI-Regler gewählt.

$$G_{\mathrm{PI}}(s) = K_{\mathrm{P}} + K_{\mathrm{I}}\frac{1}{s} = K_{\mathrm{R}}\frac{T_{\mathrm{n}}s + 1}{s}, \quad \text{mit} \quad K_{\mathrm{P}} = K_{\mathrm{R}}T_{\mathrm{n}} \quad \text{und} \quad K_{\mathrm{I}} = K_{\mathrm{R}} \tag{26}$$

Auch die Reglerparameter können mit Matlab angenähert werden. Da es sich bei dem System aber nur um eine relativ ungenaue Näherung handelt, müssen die Parameter experimentell angepasst und optimiert werden. Es ergibt sich die Reglerstruktur aus Abbildung 4.3.

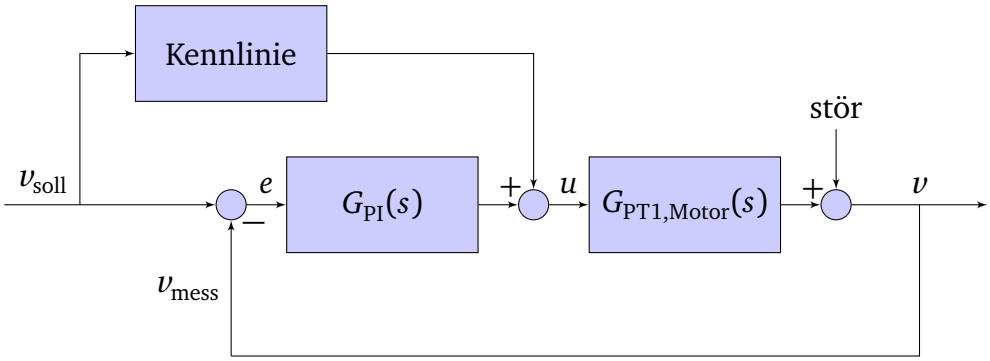


Abbildung 4.3: Reglerstruktor des Motorsystems

- Lenkungs-Identifizierung
  Die Kennlinie der Lenkung wurde ähnlich wie zuvor durch eine Testfahrt erstellt. Nacheinander wurden Kreise mit unterschiedlichen Radien gefahren und das *Steeringcommand* sowie der Gierwinkel aufgezeichnet. Der Gierwinkel wird zur Berechnung des Lenkwinkels benötigt, da dieser nicht direkt gemessen werden kann. Aus der Geometrie des Fahrzeuges kann unter der Bedingung, dass die Geschwindigkeit $v \neq 0$ ist, der Lenkwinkel bestimmt werden.

$$\delta = \tan^{-1}\left(\frac{l\dot{\theta}}{v}\right) \tag{27}$$

Diese Beziehung lässt sich direkt aus Abbildung 4.1 ableiten, mit $R_{\mathrm{H}} = \frac{v}{\dot{\theta}}$.

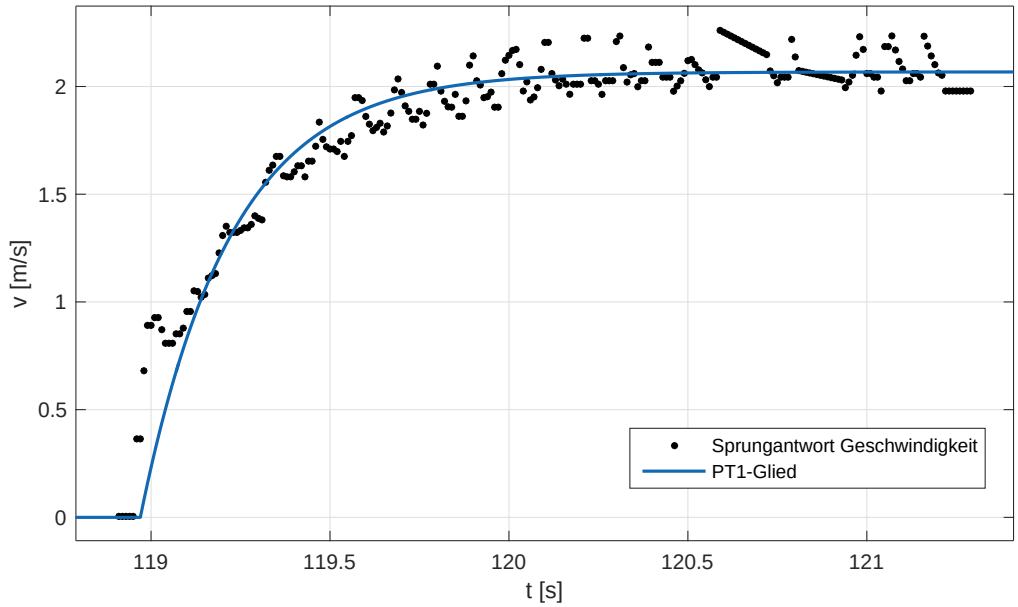


Abbildung 4.4: Curvefitting der Sprungantwort des Motors

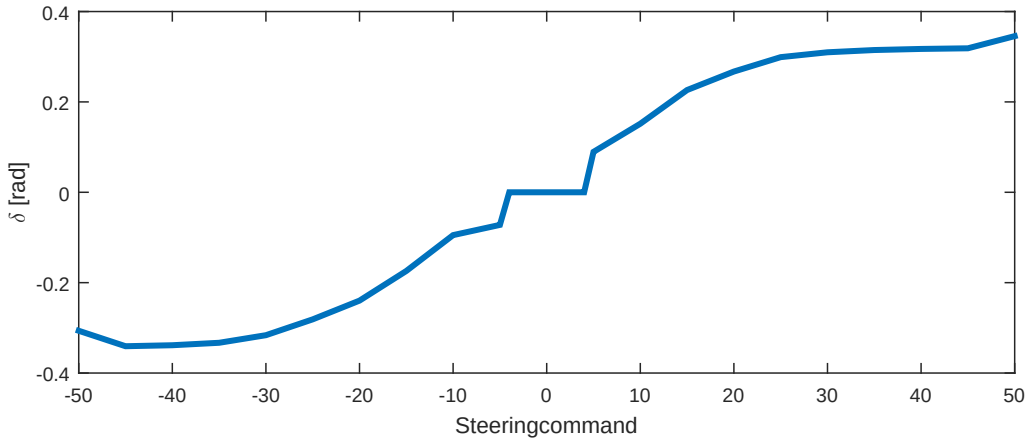


Abbildung 4.5: Kennlinie Lenkwinkel

Die daraus folgende Kennlinie ist in Abbildung 4.5 zu sehen. Anschließend wurde analog zur Motor-Indentifizierung, eine Identifikation des Lenksystems durch eine Sprungantwort durchgeführt. Es zeigte sich, dass ein PT1-Glied als Übertragungsfunktion das Systemverhalten ausreichend genau wiedergibt 4.7. Als Regler wurde ein PI-Regler ausgewählt dessen Parameter mit Matlab näherungsweise bestimmt und experimentell optimiert wurden. Die resultierende Reglerstruktur ist in Abbildung 4.6 zu sehen.

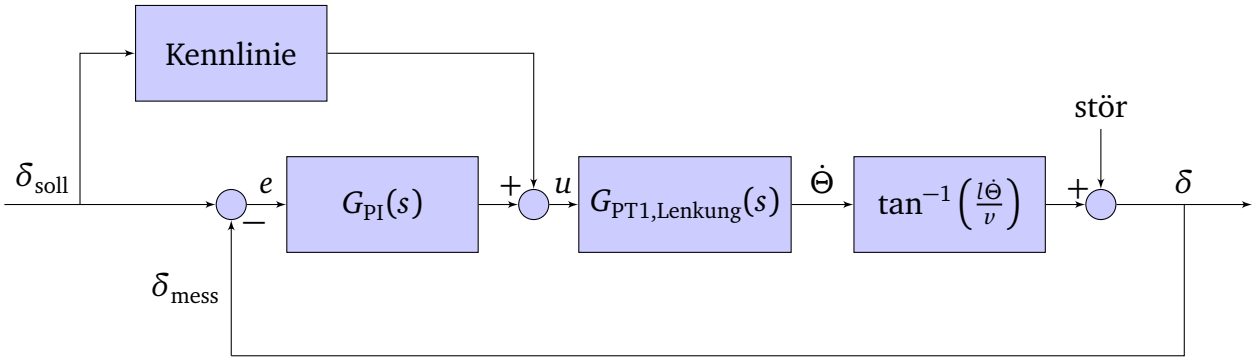


Abbildung 4.6: Reglerstruktor des Lenksystems

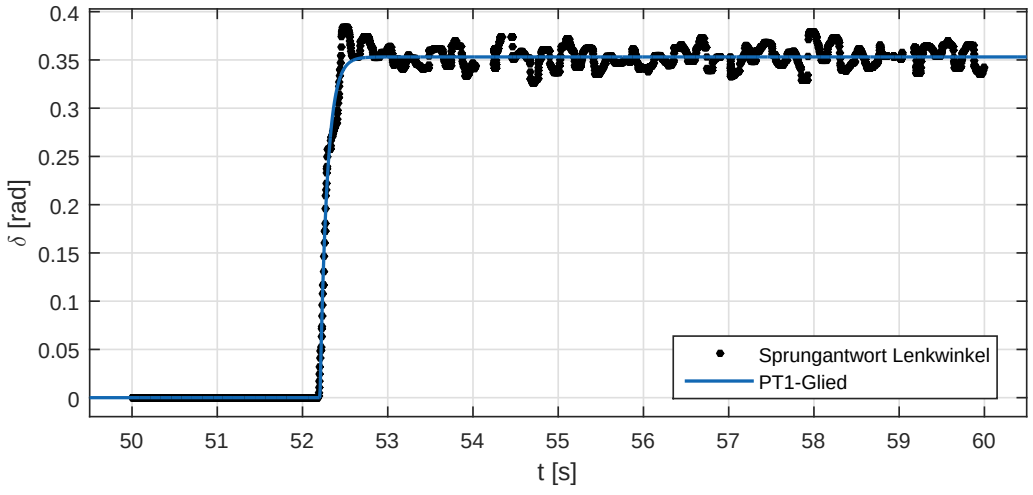


Abbildung 4.7: Curvefitting der Sprungantwort des Lenkwinkels

5 Rundkurs ohne Hindernisse

Als erste Disziplin soll das Modellauto den Rundkurs ohne Hindernisse nach Abbildung 3.5c fahren. Diese Aufgabe wurde unterteilt in die Probleme Pfadplanung und Regelung.

5.1 Pfadplanung

Für den Rundkurs ohne Hindernisse folgt das Auto einem Pfad. Von einer bekannten Startposition wird mithilfe eines globalen Planers dieser Pfad erstellt. Hierzu wird der Search-Based-Planning-Laboratory (SBPL) Lattice Planner verwendet. Dieser Planner berechnet seinen Pfad ausschließlich mittels einer vorgegebenen Karte. Das bedeutet, dass das Ergebnis stark von ihrer Güte abhängt.

Die wichtigste Konfiguration für den Planer sind die *motion primitives*, welche die möglichen Bewegungen des Roboters beschreiben und deren Kosten festgelegt. Das Modellauto folgt einem Ackermann-Modell und kann daher nicht seitwärts fahren und der Kurvenradius ist beschränkt. Für das Fahren rückwärts werden hohe Kosten festgelegt, da das Tiefenbild in diese Richtung fehlt und Kollisionen daher nicht ausgeschlossen sind.

5.2 Pfadfolgeregelung

Die Aufgabe der Pfadfolgeregelung besteht darin, den Querabstand des Fahrzeuges zum Pfad auszuregeln. Dazu muss zunächst der Abstand des Fahrzeugschwerpunkts zum Pfad berechnet werden. Dies ist natürlich nur möglich, wenn die genaue Position im Raum bekannt ist (Abschnitt 3). Der Querabstand wird im Folgenden mit d bezeichnet und berechnet sich durch:

$$d = \sqrt{(x_S - x_P)^2 + (y_S - y_P)^2} \tag{28}$$

Dabei ist $\mathbf{x}_S = [x_S \quad y_S]^T$ die absolute Position des Fahrzeugschwerpunkts und $\mathbf{x}_P = [x_P \quad y_P]^T$ der Punkt auf dem Pfad mit dem kleinsten Abstand zu $\mathbf{x}_S$. Nun ist $\mathbf{x}_P$ nicht bekannt und muss zunächst berechnet werden, indem das $\mathbf{x}_P$ gefunden wird, für welches Gleichung (28) minimiert wird. Verwendet wurde hierzu ein Least-Squares-Verfahren in Kombination mit einem Newtonverfahren. Für eine genaue Beschreibung des Vorgehens wird auf [WKA02] verwiesen.

Die Pfadfolgeregelung teilt sich auf in eine Vorsteuerung und eine parallel geschaltete Regelung auf (Abbildung 5.1). Die Vorsteuerung erwartet als Eingangsgröße den Kehrwert des Bahnradius R des Pfades. Diese Größe wird als Bahnkrümmung $\kappa = \frac{1}{R}$ bezeichnet. Betrachtet man Abbildung 4.1 so kann eine geometrische Beziehung zwischen R und dem Lenkwinkel δ hergestellt werden [Keh07]:

$$R = \frac{1}{\tan(\delta_s)} \sqrt{l^2 + l_H^2 \tan(\delta_s)^2} \tag{29}$$

Mit Einführung der Bahnkrümmung und auflösen nach δ_s ergibt sich.:

$$\delta_s = \arctan\left(\frac{l \cdot \kappa}{\sqrt{1 - l_H^2 \kappa^2}}\right) \tag{30}$$

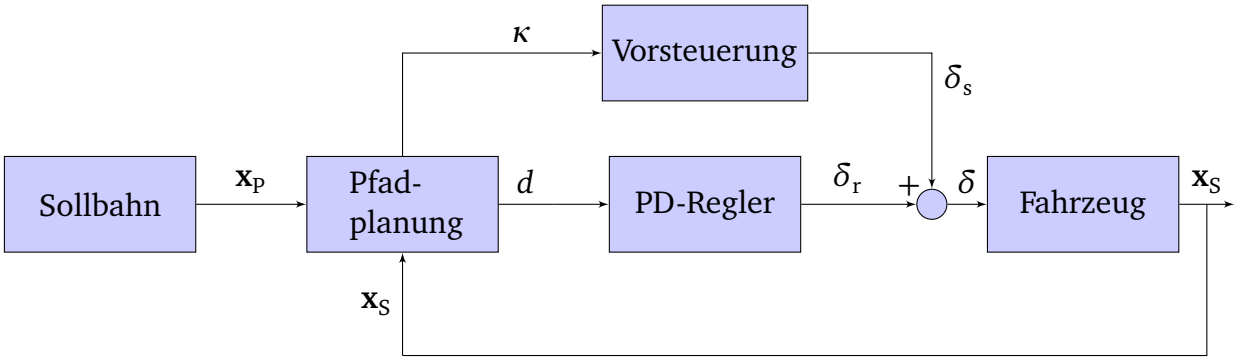


Abbildung 5.1: Reglerstruktor der Pfadfolgeregelung

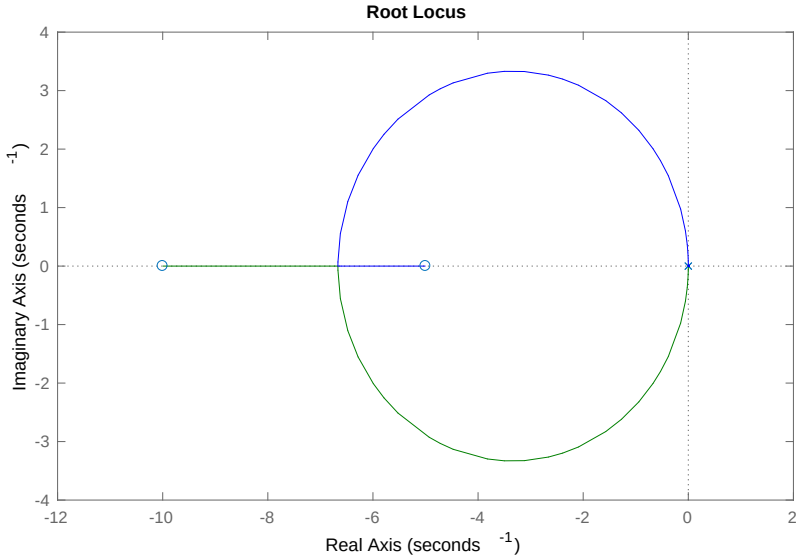


Abbildung 5.2: Qualitative Wurzelortskurve des Pfadfolgeregelungssystems

Gleichung (30) wird als Vorsteuerung der Pfadfolgeregelung verwendet. Alle weiteren Modellfehler oder Störungen werden durch einen PD-Regler ausgeglichen. Dieser setzt sich wie folgt zusammen [Keh07]:

$$\delta_r = K_P d + K_D \dot{d}. \tag{31}$$

Dass ein PD-Regler für die Regelung von Nöten ist, ist sehr gut an der Übertragungsfunktion des Ackermanmodells aus Gleichung (24) zu erkennen. Sie besitzt einen doppelten Pol in Null, sowie eine einfache Nullstelle. Ein zusätzlicher I-Anteil würde das Regelverhalten verschlechtern bzw. zur Instabilität führen. In Abbildung 5.2 ist eine qualitative Wurzelortskurve zu sehen, die zeigt, dass das System mit einem PD-Regler stabilisiert wird.

Anschließend werden aus Vorsteuerung und Regelung der resultierende Lenkwinkel zusammengefasst:

$$\delta = \delta_s + \delta_r \tag{32}$$

Die Bestimmung der Regelparameter K_P und K_D erfolgte zunächst mittels Simulation in Matlab und wurden anschließend am Fahrzeug experimentell getestet und optimiert. Wie in Gleichung (24) deutlich zu sehen ist, hängt sowohl die Verstärkung, als auch die Nullstelle von der Geschwindigkeit ab. So muss für jede Geschwindigkeit ein neuer Regler entworfen und getestet werden. Dies erwies sich oft als mühsam, da die mit einer Simulation erhaltenen Werte oft nicht das gewünschte Verhalten am realen Modell aufzeigten.

5.3 Ergebnisse

Mit unserer Kombination aus Lokalisierung und Pfadfolgeregelung konnte das Modellauto den Rundkurs bewältigen. Nach der experimentellen Ermittlung der Reglerparameter war das System stabil und Schwingungen konnten zu einem Minimum reduziert werden.

Als Verbesserung könnte beim Rundkurs die Geschwindigkeit dynamisch abhängig vom Kurvenradius gewählt werden, sodass z.B. bei Kurven abgebremst wird. Dadurch wären höhere Geschwindigkeiten auf den geraden Strecken möglich. Dies könnte eine Reduktion der Gesamtzeit bewirken.

6 Rundkurs mit Hindernissen

Als Erweiterung des Rundkurses ohne Hindernisse werden in dieser Aufgabe Hindernisse auf der Strecke platziert.

6.1 Pfadplanung

Der globale Pfad wird wie beim Rundkurs ohne Hindernisse mit dem SBPL Planer berechnet (siehe Abschnitt 5.1). Zusätzlich müssen die Hindernisse berücksichtigt werden. Dies erfolgt mit einem lokalen Planer, der dynamisch auf Hindernisse reagiert.

Der verwendete Planer ist der Timed-Elastic-Band (TEB) Local Planner. Da es sich bei diesem Planner um ein bereits fertiggestelltes ROS-Paket handelt, bestand die Schwierigkeit darin, die zahlreichen Parameter auf unsere Anforderungen anzupassen, um schlussendlich das bestmögliche Ergebnis zu erzielen. Die Informationen zu Hindernissen werden aus dem Tiefenbild der Kinect gewonnen. Aus den erkannten Hindernissen wird eine Costmap erstellt, die jedem Punkt auf der Karte einen Wert zuweist. Ein hoher Wert ist dabei mit hohen Kosten verbunden und teilt dem Local Planner mit, dass er sich nahe eines Hindernissen befindet. Ziel ist es einen möglichst kurzen Pfad mit minimalen Kosten zu berechnen.

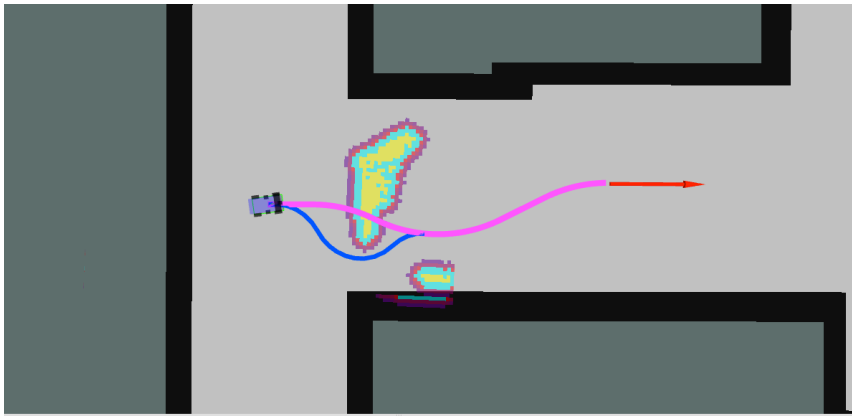


Abbildung 6.1: Lokaler Planer mit Hindernis

6.2 Ergebnisse

Bei der Bearbeitung dieser Aufgabe hat es sich positiv ausgewirkt, dass wir bereits im Rundkurs ohne Hindernisse auf die Navigation von ROS gesetzt haben. Dadurch war eine Wiederverwendung möglich und es konnte sich auf den Local Planner konzentriert werden.

Bei Hindernissen, die kleine Umwege vom geplanten Pfad erfordern, kann ein gutes Ergebnis erzielt werden. Da aber der Local Planner in gewissem Maße dem globalen Pfad folgt, werden nicht beliebig große Abweichungen berechnet. Dies kann zu Problemen bei komplexen Hindernissen führen.

Da auch bei der Lokalisierung mit AMCL das Tiefenbild verwendet wird, kommt es mit zu vielen Hindernissen zu Problemen bei der Lokalisierung. Zur Verbesserung sollte

hier verstärkt mit den Daten der Inertialnavigation gerechnet oder ein zweiter Laserscan für AMCL verwendet werden, der über die Hindernisse hinweg schaut.

7 Autonomes Erkunden

Das autonome Erkunden kann in drei grundlegende Teilgebiete gegliedert werden. An erster Stelle steht das ROS-Paket *Gmapping*[4], welches für die schrittweise erfolgende Kartografie der Umgebung, sowie der Lokalisierung des Autos im Raum zuständig ist. Aus den Daten des Gmapping kann dann das *Automap* Paket berechnen, wohin das Auto fahren soll, um seine Umgebung komplett und durch möglichst einfache Lenkmanöver erkunden zu können. Und letztendlich benötigt man einen funktionierenden Navigation Stack, der diese Zielvorgaben dann entgegennimmt und die Steuerbefehle für das Auto erzeugt.

7.1 Gmapping

Der große Unterschied hierbei zu den vorangehenden Aufgaben liegt darin, dass beim Erkunden keine statische Karte vorgegeben ist, sondern ohne Anhaltspunkte gestartet wird und sich die Umgebung Stück für Stück zusammensetzt, d.h. mit jeder Bewegung des Fahrzeuges ändert sich das Bild der Umgebung mit. Zum Bewältigen dieser komplexen Aufgabe kommt das *Gmapping* Paket zum Einsatz. Neben dem Erstellen der Karte besteht die andere Aufgabe des *Gmapping* noch darin, die aktuelle Position des Autos auf der Karte zu bestimmen. Da beide Aufgaben parallel ablaufen, ist der mapping Algorithmus auch häufig unter dem erweiterten Begriff *SLAM Gmapping* anzutreffen, wobei SLAM (Simultaneous Localization and Mapping) eben für das gleichzeitige Ablaufen der Lokalisierung und des Mapping steht. Als Input benötigt das Paket den Laserscan der Kinect und die Odometriedaten des Autos. Der Laserscan wird in unserem Fall noch durch einen Filter verbessert, was in Kapitel 3 erwähnt wird. Mit jedem Eintreffen von neuen Daten kann der Algorithmus diese mit den bisherigen Werten vergleichen und die Karte aktualisieren bzw die aktuelle Position des Fahrzeugs darin schätzen. Dies funktioniert auch mit ziemlich guter Genauigkeit, sofern sich das Auto in Gebieten mit vielen Anhaltspunkten, wie z.B. viele zur Orientierung benötigte Kanten, befindet. Sobald man allerdings auf offen gelegene Gebiete zufährt, zeigt sich der Algorithmus häufig fehleranfällig und liefert nicht immer gute Ergebnisse. Das resultiert dann in verrutschten Kanten auf der Karte, wie in Abbildung 7.1 dargestellt, und Positionssprüngen. Um diesem Verhalten entgegenzuwirken, kann man am Algorithmus einige Parametereinstellungen vornehmen, so dass schätzungsweise ungenaue Laserscans verworfen werden, welche immer dann eintreffen, wenn man gerade eine große offene Fläche entdeckt hat. Weiterhin kann man genauere Ergebnisse erzielen, indem man die Updatefrequenz (=mehr Inputdaten) für das Verarbeiten der eingehenden Laserscans erhöht. In dem Fall steigt die Genauigkeit auf Kosten der Performance, welches speziell in Kombination mit dem *Automap* Algorithmus eine Rolle spielt und in Sektion 7.4 weiter erläutert wird.

[4] http://wiki.ros.org/gmapping

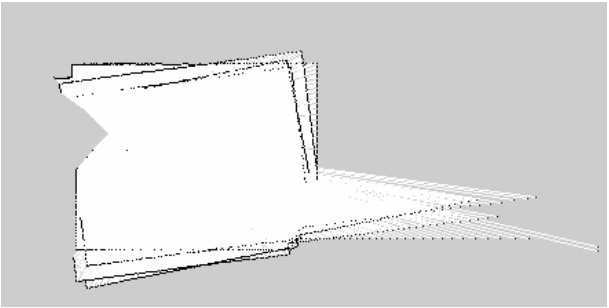


Abbildung 7.1: verrutschte Kanten beim Gmapping

7.2 Automap

Das *Automap*[5] Paket wurde von Sebastian Ehmes, vorjähriger Teilnehmer des Projektseminar Echtzeitsysteme und Betreuer für das diesjährige Projektseminar, im Rahmen seiner Bachelorarbeit entwickelt und ist verantwortlich für die eigentliche Erkundungsstrategie. Wenn man den Schnappschuss in Abbildung 7.2 betrachtet, hat man auf der einen Seite das bereits bekannte Terrain (hier in grau dargestellt) und auf der anderen Seite die unbekannten Gebiete, welche voneinander abgegrenzt sind. Im ersten Schritt findet nun der *Automap* Algorithmus alle solche Grenzen auf der bisher aufgezeichneten Karte, um dann im zweiten Schritt zu berechnen, welche dieser Grenzen am kostengünstigen angefahren werden kann. In unserem Fall bedeutet dies, dass Ziele in Fahrtrichtung bevorzugt werden, da unser Auto aufgrund der Ackermann Lenkung in seiner Mobilität eingeschränkt ist und große Wendemanöver sehr aufwendig sind. Da sich die Karte mit jeder Bewegung des Fahrzeugs ändern kann und sich wiederum auch die Grenzen verschieben, können aktuelle Anfahrtsziele vorzeitig abgebrochen werden und der *Automap* Algorithmus berechnet erneut die optimale Route. Das *Automap* Paket konnte sehr gut in unser Projekt integriert werden, da sich viele unserer Ansätze mit denen aus Sebastians Bachelorarbeit gedeckt haben, so z.B. die Benutzung von *Gmapping* und der Aufbau des Navigation Stacks, währenddessen sich die Integration anderer ROS Pakete, welche ebenfalls einen Erkundungsalgorithmus implementieren, als schwierig erwiesen hat, da diese teils andere Voraussetzungen (holonome Roboter statt Ackermann Modell) erwarteten.

7.3 Navigation Stack

Der Navigation Stack hat die Aufgabe die eigentlichen Motor- bzw Lenkbefehle an das Auto zu senden. Dafür wird zunächst die Information benötigt, wohin das Fahrzeug fahren soll. Dieses geschieht über das in Sektion 7.2 beschriebene Automapping. Außerdem wird eine Pfaderstellung vom aktuellem Standpunkt des Fahrzeugs zum Zielort benötigt, welches mithilfe eines globalen bzw lokalen Planners erfolgt. Die Funktionsweise der Planner gleicht der aus dem Kapitel 5 und Kapitel 6, weshalb hier nicht weiter darauf eingegangen wird. Während bei den Rundkursdisziplinen die Zielposition stets

[5] https://github.com/arg0n1s/automap

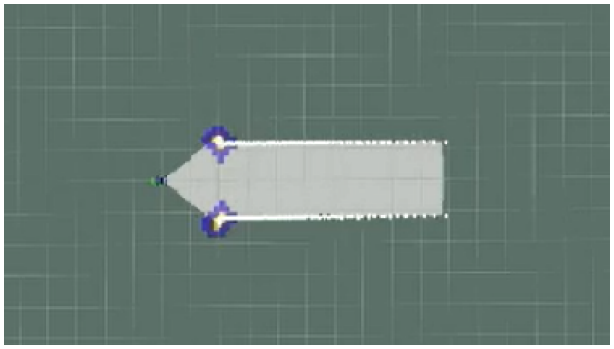


Abbildung 7.2: Gmapping Auschnitt

fest vorgegeben war, muss diese beim autonomen Erkunden ständig angepasst werden und darin liegt auch der größte Unterschied der beiden Aufgaben.

7.4 Probleme

Wie schon bereits in Sektion 7.1 angedeutet, spielt die Balance zwischen Qualität des Endproduktes (gezeichnete Karte) und Performance eine wesentliche Rolle. So kam es regelmäßig vor, dass das Modellauto während des Erkundens entweder seine Position verloren hat (Fokus: Performance>Genauigkeit), oder die Algorithmen die Menge an Daten nicht korrekt verarbeiten konnten und das Auto auf der Stelle verharrte (Fokus: Genauigkeit>Performance), insbesondere beim Erkunden von großen Flächen. Das Finden der optimalen Parameterwerte für die jeweiligen Pakete ist eine Kunst für sich und hatte den Trend Unmengen an Zeit zu verschlingen. Während das Erkunden von kleinen abgeschlossen Räumen noch funktioniert hat, konnten wir das selbe Ergebnis nicht für weitläufigere Gebiete reproduzieren.

8 Zusammenfassung und Ausblick

Die im Rahmen des Projektseminars Echtzeitsysteme gestellten Aufgaben und Disziplinen konnten vom Team MoveIT erfolgreich gelöst werden. Dabei wurden nicht nur Programmiererfahrungen gesammelt, sondern vor allem auch das Arbeiten mit ROS erlernt. Bei der Bearbeitung der Aufgaben wurde der Schwerpunkt auf die Lokalisierung sowie die Regelung des Fahrzeugs gelegt, um den Rundkurs mit und ohne Hindernissen mithilfe der Navigation von ROS absolvieren zu können.

8.1 Verbesserungsvorschläge

Prinzipiell war die zur Verfügung gestellte Hardware-Plattform sehr stabil und hat bei der Bearbeitung der Aufgaben kaum Probleme bereitet. Als Verbesserungsvorschlag wäre ein zusätzlicher Hall-Sensor am anderen Hinterreifen hilfreich, um die Genauigkeit der gemessenen Geschwindigkeit (z.B. in Kurven) zu erhöhen.

Literatur

[DWH08] Hugh Durrant-Whyte and Thomas C. Henderson. *Multisensor Data Fusion*, pages 585–610. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[GEEPP06] Demoz Gebre-Egziabher, Gabriel H. Elkaim, J. David Powell, and Bradford W. Parkinson. Calibration of strapdown magnetometers in magnetic field domain. *Journal of Aerospace Engineering*, 19(2):87–102, 2006.

[GT12] E. Le Grand and S. Thrun. 3-axis magnetic field mapping and fusion for indoor localization. In *2012 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pages 358–364, Sept 2012.

[Keh07] Steffen Kehl. *Querregelung eines Versuchsfahrzeugs entlang vorgegebener Bahnen*. Berichte aus dem Institut für Systemdynamik Universität Stuttgart. Shaker Verlag, 2007.

[Keh14] Andrew Gale Kehlenbeck. Quaternion-based control for aggressive trajectory tracking with a micro-quadrotor uav. Master's thesis, University of Maryland (College Park, Md.), 2014.

[SHB13] Dieter Schramm, Manfred Hiller, and Roberto Bardini. *Modellbildung und Simulation der Dynamik von Kraftfahrzeugen*. Springer Verlag, 2013.

[Sim06] Dan Simon. *Optimal state estimation: Kalman, H [infinity] and nonlinear approaches*. Wiley-Interscience, Hoboken, N.J., 2006.

[WKA02] Hongling Wang, Joseph Kearney, and Kendall Atkinson. Robust and efficient computation of the closest point on a spline curve. In *In Proceedings of the 5th International Conference on Curves and Surfaces*, pages 397–406, 2002.

[WSE+12] He Wang, Souvik Sen, Ahmed Elgohary, Moustafa Farid, Moustafa Youssef, and Romit Roy Choudhury. No need to war-drive: Unsupervised indoor localization. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12, pages 197–210, New York, NY, USA, 2012. ACM.