Projektseminar Echtzeitsysteme

Technische Dokumentation - Team PMS

Eingereicht von
Kyoung Soon Choe, Bahri Enis Demirtel, Julian Pascal Poths, David Witon
am 13. April 2016

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: Prof. Dr. rer. nat. Schürr
Betreuer: M.Sc. Géza Kulcsar

Inhaltsverzeichnis

1 Einleitung

Im Rahmen des Projektseminars Echtzeitsysteme im WS2015/16 wurde eine Software für Modellfahrzeuge entwickelt, die verschiedene Fahrszenarien ermöglicht. Seit Beginn der Lehrveranstaltung wurde in diesem Semester erstmals das Betriebssystem ROS[1] eingesetzt.

Eine umfassende Überarbeitung der zur Verfügung stehenden Hardware gegenüber den vergangenen Semestern ermöglicht neben Ultraschall- und Hallsensoren die Nutzung von vergleichsweise rechenintensiver Bilderkennung via Webcam. Außerdem ist es möglich, die gegebene Hardware z.B. via USB-Schnittstelle zu erweitern. Auf diese Weise lässt sich beispielsweise die Genauigkeit der Positions- und Lagebestimmung durch Redundanz verbessern.

Dieses Dokument erläutert die Verwendung der verwendeten Hardware (insbesondere der Sensorik) und dient als Hilfestellung bei der Inbetriebnahme der implementierten ROS-Module.

Das Team PMS hat einen besonderen Fokus auf die Optimierung der Geradeausfahrt gelegt; daher werden in dieser Dokumentation neben den technischen Aspekten auch die mathematischen Grundlagen des verwendeten Reglers kurz erläutert.

[1] Robot Operating System – http://www.ros.org

2 ROS-Module

In diesem Kapitel werden Aufbau und Funktion der implementierten ROS-Module sowie deren Zusammenwirken beschrieben.

Im Rahmen des Projektseminars wurden die nachfolgend beschriebenen Module weitestgehend unabhängig voneinander entwickelt. Dieser Ansatz ermöglicht es in zukünftigen Veranstaltungen Teile der Implementierung einfach aus einem Gesamtsystem herauszulösen und weiterzuentwickeln.

Zentrales Modul für die Kommunikation mit dem Fahrzeug (Sensoren und Aktoren) ist der *car_controller*. Ausschließlich dieser Node liest sämtliche Werte der ursprünglich am Fahrzeug vorhandenen Sensoren aus und stellt diese über ein entsprechendes Topic zur Verfügung. Andere Module greifen auf dieses Topic zu, um die Sensorwerte zu verarbeiten. Umgekehrt werden alle Steuerbefehle für das Fahrzeug (d.h. Fahren und Lenken) über diesen Node abgewickelt, nachdem sie auf das *car_command*-Topic gepublisht wurden.

Daraus folgt auch, dass für die Nutzung aller Module vorausgesetzt wird, dass neben dem *roscore* zuvor auch der *car_controller* gestartet wurde.

```
# roscore
# rosrun car_controller car_controller_node
```

2.1 Geradeausfahrt entlang einer Wand

Ziel des Moduls ist es, eine Wandfolgeregelung in das System einzubinden, sodass das Auto einen vorgegebenen Abstand zur Wand permanent einhält. Damit der Wandabstand immer konstant bleibt, bedarf es einer geeigneten Regelung. Um einen geeigneten Regler zu finden, wurde mit Hilfe von Dr.-Ing. Eric Lenz ein Streckenmodell des Systems (siehe Anhang) entworfen. Durch die Anschauung des Modells kann eine geeignete Vorwahl für einen Regler getroffen werden. In den nachfolgenden Gleichungen sind das Streckenmodell sowie die Übertragungsfunktion des Systems zu sehen. Auf die Parameter wird im Rahmen dieses Dokuments nicht näher eingegangen. Bei Bedarf kann dies im Anhang nachgelesen werden.

$$\begin{bmatrix} \dot{y} \\ \dot{\varphi}_K \end{bmatrix} = \begin{bmatrix} 0 & v \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} y \\ \varphi_K \end{bmatrix} + \begin{bmatrix} v \cdot \frac{l_H}{l} \\ \frac{v}{l} \end{bmatrix} \cdot \varphi_L^* \tag{1.1}$$

$$y = [1 \quad 0] \cdot \begin{bmatrix} y \\ \varphi_K \end{bmatrix} \tag{1.2}$$

$$G(s) = \frac{v \cdot l_H}{l} \cdot \frac{\frac{v}{l_H} + s}{s^2} \tag{1.3}$$

In der Übertragungsfunktion (1.3) ist sofort ersichtlich, dass ein doppelter I-Anteil vorliegt. Aus diesem Grund sollte der Regler keinen zusätzlichen I-Anteil besitzen. Eine benötigte Phasenreserve von 180° könnte andernfalls nicht mehr eingehalten werden – das System könnte durch einen weiteren I-Anteil instabil werden.

Als Ansatz für einen Regler werden ein P-Regler und ein PD-Regler verwendet. Zunächst versucht man, den P-Regler mit Hilfe einer Wurzelortskurve auszulegen. Anschließend wird der P-Wert als Anfangswert für den PD-Regler verwendet und mit Hilfe von MATLAB simuliert. Die simulierten Werte werden anschließend direkt am Fahrzeug erprobt.

Zwei Eigenschaften müssen bei der Simulation bedacht werden.

- Erstens liegt eine linearisierte Strecke vor. Damit ist das Modell nur für kleine Änderungen hinreichend genau.
- Das reale Modell ist in der Wirklichkeit diskret.

Mit den Parametern $v = 0{,}3\frac{m}{s}$, $l = 0{,}3m$ und einem Verzweigungspunkt von -4 ergibt sich mit Hilfe des Wurzelortskurven-Verfahrens $P' = 8$ sowie $P = 53{,}3$.

Abbildung 2 zeigt die konstruierte Wurzelortskurve (WOK). Der P-Parameter wurde mit Hilfe des Verzweigungspunktes bestimmt, da in ihm kein imaginärer Anteil vorliegt und das System somit nicht schwingt. Die Simulation zeigt mit diesem Parameter ein einigermaßen stabiles Verhalten, jedoch weist das System in der Realität leichtes schwingen auf. Auf die Konstruktion einer WOK wird hier nicht weiter eingegangen, bei Bedarf können die Regeln z .B. im vorlesungsbegleitenden Skript zu *SDRT 1*[2] nachgelesen werden.

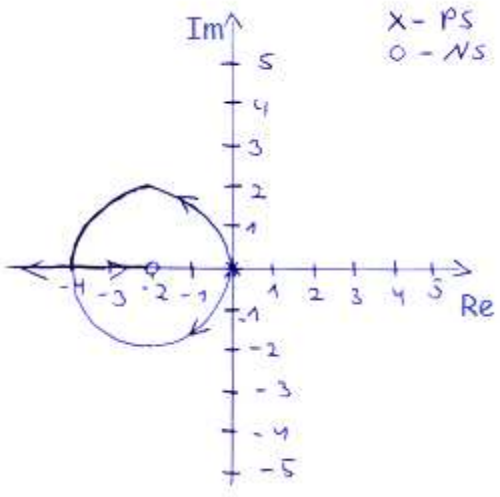

Abbildung 1 – Wurzelortskurve

Um das Schwingen zu beseitigen, wird mit der Hilfe von MATLAB und einem iterativen Vorgehen mit anschließenden Tests ein geeignetes Wertepaar für den PD-Regler bestimmt. Dabei wird mit dem Wert $P = 53{,}3$ begonnen. Die schlussendlich ermittelten Werte betragen für $P = 15$ und $D = 3$.

[2] Systemdynamik und Regelungstechnik 1, Prof. Dr.-Ing. Konigorski

Nachfolgend sind der MATLAB Regelkreis (Abbildung 2), sowie ein Beispiel-Plot (Abbildung 3) zu sehen. Die großen Abweichungen der Parameter lassen eine nicht exakte Simulation des Regelkreises vermuten.

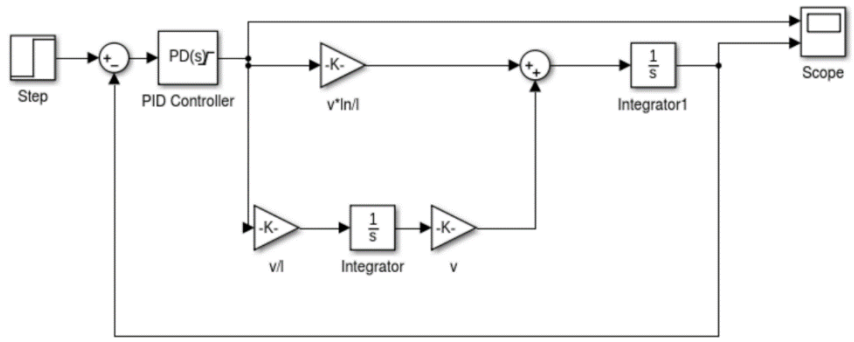

Abbildung 2 – MATLAB Regelkreis

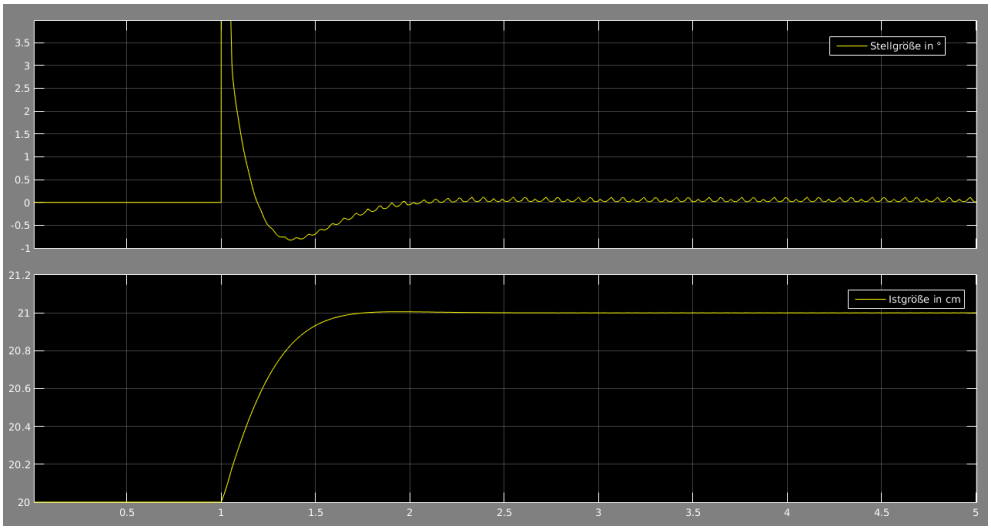

Abbildung 3 – Beispielplot der Simulation

Eine nicht unerhebliche Abweichung am realen Fahrzeug ist stellen die Totzeit und eine Ungenauigkeit bei der Positionierung des Motors dar. Teilweise braucht der Motor bei einem Lenkeinschlag fast eine Sekunde um seine Endlenkposition zu erreichen.

Nachdem der Motor den Lenkvorgang beendet hat, entspricht die eingelenkte Position nicht immer der korrekten Sollposition (mangelnde Wiederholgenauigkeit). Zudem unterliegt das Fahrzeug Versorgungsspannungsschwankungen, sodass nicht immer die vorgegebene Geschwindigkeit konstant gehalten werden kann.

Eine weitere Schwierigkeit ist die Ungenauigkeit des Ultraschallsensors, da dieser bei Schrägstellung zur Seitenwand keine genauen Werte liefern kann. Generell ist die nicht-parallele Orientierung zur Wand ein Problem, da es dem Fahrzeug nicht möglich ist zu erkennen, wann eine parallele Orientierung gegeben ist (nur dann ist die Entfernungsmessung zuverlässig). Da

die ermittelten Parameter das Fahrzeug hinreichend stabilisierten, wurde auf eine genauere Modellierung verzichtet. Tests ergaben zudem, dass sich die ausgewählten Parameter für alle Geschwindigkeiten eignen.

Verwendung des Moduls

Zum Einbinden des Moduls muss ein neuer Rosnode erstellt werden und die gepublishten Werte des Ultraschallsensors müssen abgefangen werden. Zudem wird ein Publisher benötigt welche den ermittelten Lenkwinkel (dieser ist die Eingangsgröße des Systems) an den Steuerungsnode schickt. Damit sollte der Code auf anderen Plattformen funktionieren, jedoch kann es vorkommen, dass die Reglerparameter nicht dieselben Ergebnisse liefern. Dies könnte an einem Abweichenden mechanischen Modell liegen.

2.2 Kurvenfahrt

Da Fahrzeuge nicht nur geradeaus, sondern insbesondere in Kreuzungsbereichen auch Kurven fahren, erschien es uns wichtig, die Kurvenfahrfähigkeit in unserem Projekt zu implementieren. Die bereits im Auto verbauten Sensoren lassen unseres Erachtens keine ausreichend zuverlässige Kurvenfahrt zu. Aus diesem Grund erweiterten wir das Fahrzeug durch ein Gyroskop in Kombination mit einem Arduino. Gyroskop und Arduino kommunizieren über die I^2C-Schnittstelle miteinander. Nachdem durch das Gyroskop uns der Winkel zwischen X- und Y-Achse bestimmt wurde, überprüft das *curve*-Node, ob die Kurvenfahrt abgeschlossen ist.

Erkennung einer Kurve

Steuert das Auto während einer Geradeausfahrt auf einen Kreuzungsbereich zu, erkennen die mittig in der Seite des Fahrzeugs verbauten Ultraschallsensoren im Kreuzungsbereich das plötzliche Ende der bislang verfolgten Wand (vgl. Abschnitt 2.1). Das plötzliche Ende der Seitenwand signalisiert, dass das Fahrzeug abbiegen soll.

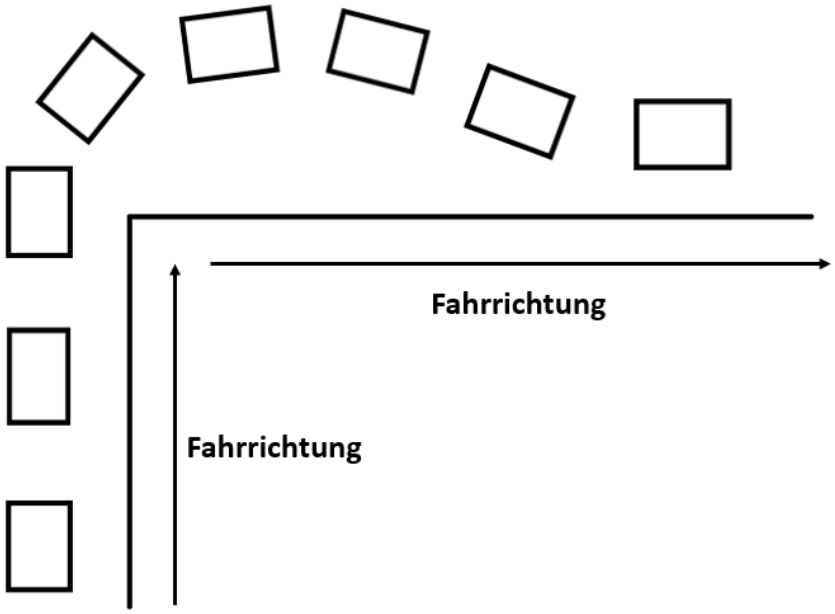

Abbildung 4 - Schematische Darstellung der Kurvenfahrt

Im kurzen Abstand von 5 Millisekunden werden die Sensorwerte vom Wagen erfasst und ermöglichen eine sofortige Reaktion in Form einer Kurvenfahrt, nachdem der Winkel zwischen X- und Y-Achse vom Sensor abgeglichen und gespeichert wurde. Nach einem 140 Grad-Winkel korrigiert das Auto selbständig seine Spur in Geradeausposition und bewegt sich so lange geradlinig weiter, bis die Sensoren einen weiteren Kreuzungsbereich erkennen.

Ausblick

Für die Implementierung der Kurvenfahrten waren zusätzliche Komponenten (hier: Gyroskop) hilfreich. Verbesserungsbedarf besteht bei der Lenkmechanik, sodass ein geringerer Kurvenradius erreicht werden kann. Um eine enge 90°-Kurve fahren zu können, ist es derzeit notwendig, „mehr“ als 90 Grad abzubiegen und anschließend entgegen zu korrigieren (siehe Abbildung 4). Erst dann kann das Fahrzeug die Geradeausfahrt an der Wand in gleichem Sollabstand fortsetzen.

2.3 Manuelles Fahren per Fernsteuerung

Für einige Fahr- und Testmanöver ist die manuelle Steuerung des Fahrzeugs unverzichtbar. Daher wurde eine simple Möglichkeit implementiert, das Fahrzeug per Tastatur (über SSH) zu steuern. Die Implementierung basiert auf *teleop_twist_keyboard*, welches bereits aus dem ROS-Tutorial bekannt ist.

Verwendung des Moduls

```
# rosrun rc_car rc_car_node
# rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

Die gedrückten Tasten, die von *teleop_twist_keyboard* gepublisht werden, werden durch das *rc_car*-Modul empfangen und verarbeitet. Dabei werden die entsprechenden Steuerbefehle direkt an den *car_controller* gesendet; die Steuerung funktioniert also analog zu *TurtleSim* aus dem ROS-Tutorial[3].

Ausblick

Für eine komfortablere Steuerung wäre die Verwendung eines Joysticks/Game-Controllers oder die App-Unterstützte Steuerung via Smartphone denkbar.

2.4 Tordurchfahrt durch Erkennung von ArUco-Markern

Im Modul *pass_gate* wird das Fahrzeug automatisch durch vorbereitete Tore gefahren. An den Toren sind ArUco-Marker angebracht, welche mit Hilfe der Kamera erkannt werden, sodass das Fahrzeug eine Fahrbewegung zum und durch das Tor ausführen kann.

Abbildung 5 – Aufbau des Szenarios "Tordurchfahrt"

Verwendung des Moduls

Nachfolgend wird vorausgesetzt, dass der *roscore* und das *car_controller*-Node bereits gestartet wurden (vgl. Beginn dieses Kapitels auf Seite 2).

[3] http://wiki.ros.org/ROS/Tutorials#Beginner_Level

Um das Modul zu starten, startet man zunächst den *usb_cam_example*-Node:

```
# roslaunch usb_cam_example uvcCameraLaunch.launch
# rosrun usb_cam_example usb_cam_example_node
```

Der Node für die Kamera und die ArUco-Marker-Erkennung wird dadurch gestartet, anschließend kann der Node *pass_gate* gestartet werden.

```
# rosrun pass_gate pass_gate_node
```

Kamerakalibrierung

Bevor die Kamera zur Erkennung der ArUco-Marker zufriedenstellend verwendet werden kann, muss die Kamera zuvor kalibriert werden. Dies dient dazu, die Kameraparameter zu bestimmen, sodass die Bildverzerrung durch physikalische Eigenschaften der Kameralinse erkannt und kompensiert werden kann.

Mit Hilfe des sog. *Chessboards* ist es möglich, die Kamera zu kalibrieren. Dazu folgt man der Anleitung im ROS-Wiki[4].

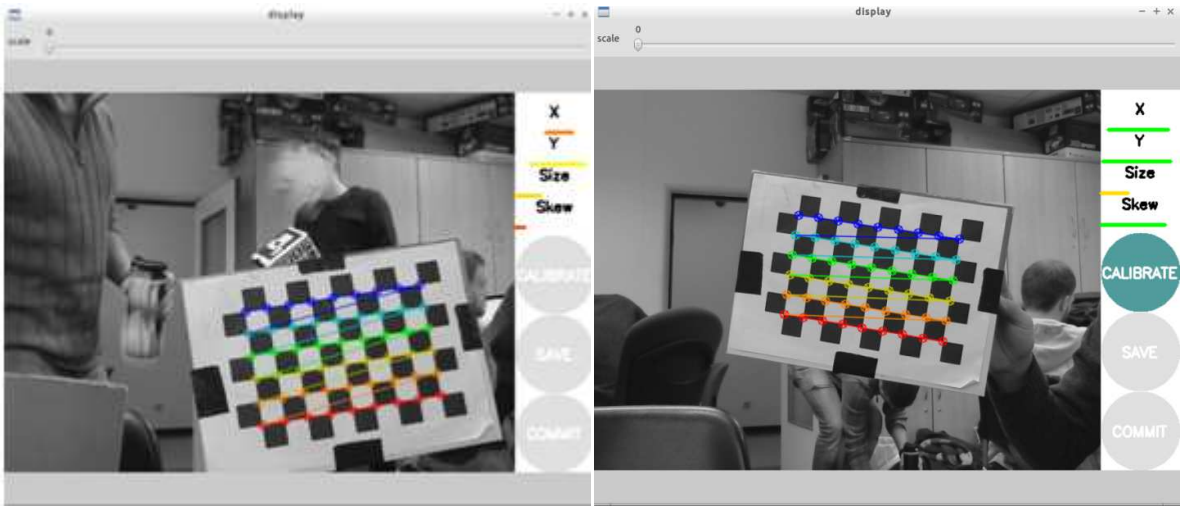

Abbildung 6 – Kalibierung der Kamera (aus Kameraperspektive)

Nach erfolgreicher Kalibrierung gewinnt man die Kameraparameter, die in Abbildung 7 dargestellt sind.

[4] Kamerakalibierung: http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration

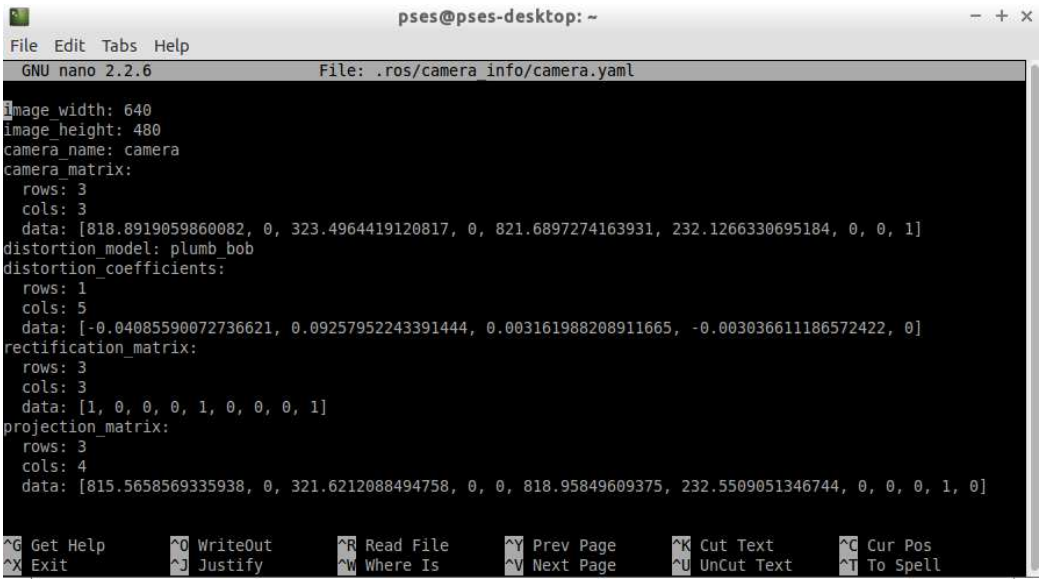

Abbildung 7 - Ermittelte Kameraparameter

Beim Starten von des *usb_cam_example*-Nodes werden diese Kameraparameter benötigt, um die vorhandene Verzerrung korrigieren zu können. Die Kameraparameter werden zunächst als *camera.yaml* gespeichert. Von *usb_cam_example* werden die Parameter jedoch als XML-Datei erwartet, daher ist eine Umwandlung von .yaml auf .xml erforderlich. Am einfachsten erledigt man dies mit Hilfe eines Texteditors manuell.

Beim Ausführen von *usb_cam_example* wird die XML-Datei eingelesen:

```
cParameters.readFromXMLFile(
      "/home/pses/catkin_ws/src/usb_cam_example/camera.xml”);
```

Erkennen der ArUco-Marker

Mit dem gegebenen Node *usb_cam_example* werden die ArUco-Marker erkannt.

Wie man in der Abbildung sehen kann, werden immer sämtliche ArUco-Marker im Kamerabild gleichzeitig erkannt.

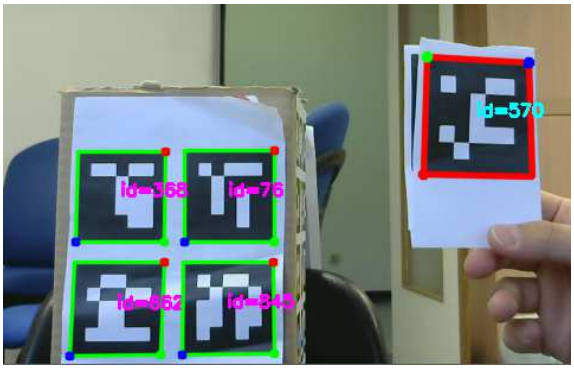

Abbildung 8 – ArUco-Marker aus Kameraperspektive

Um die Durchfahrt mehrerer hintereinander positionierter Tore (siehe Abbildung 5) sinnvoll implementieren zu können, ist es erforderlich, den Abstand der erkannten Marker einschätzen zu können. Dabei ist es ausreichend zu erkennen, welcher der Marker dem Fahrzeug am nächsten ist.

Die Koordinaten aller Eckpunkte der Marker liegen als Pixelwert (relativ zum Bildrand) vor. Mit Hilfe des Satz des Pythagoras wird die Seitenlänge aller Marken berechnet und anschließend miteinander verglichen.

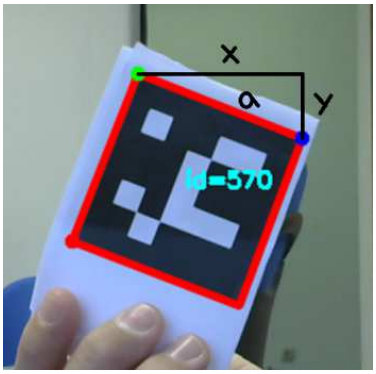

Abbildung 9 - Geometrische Eigenschaften zur Bestimmung der Kantenlänge

Nun muss das Fahrzeug zum ArUco-Marker hingesteuert werden. Um möglichst gerade durch das Tor fahren zu können, wird die orthogonale Linie vom Marker mit Hilfe der Funktion *draw3dAxis()* in *cvdrawingutils.h* zunächst berechnet und zusammen mit Koordinaten des Markers als *gate_parameter* an den *pass_gate*-Topic veröffentlicht.

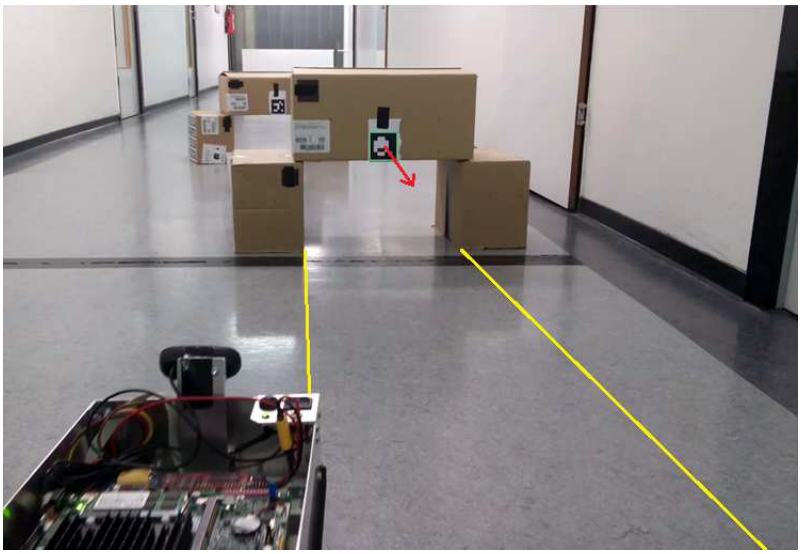

Abbildung 10 – Virtuelle Fahrbahn durch Erkennung der Marker-Perspektive

Der *pass_gate*-Node steuert das Auto zunächst auf die Fahrbahn, um das Tor möglichst gerade durchfahren zu können. Dabei werden die Koordinaten der Spitze der Orthogonalen und Mittelpunkt des Markers verglichen (d.h. die Perspektive erkannt) und abhängig von der Differenz wird das Fahrzeug nach links oder nach rechts gesteuert.

Das Auto bleibt stehen, falls kein Marker zu erkennen ist.

Ausblick

Die am Fahrzeug angebrachte Kamera liefert aktuell die Auflösung von 640x480. Die Seitengröße der in unserer Lösung verwendeten Marker beträgt 6.4cm. Der Code ist stark abhängig von diesen beiden Größen und muss dementsprechend angepasst (oder dynamisch implementiert) werden, falls sich die Kameraauflösung oder die Größe der Marker ändern.

Eine Kamera mit höherer Auflösung könnte die Erkennung weiter entfernter Marker (ab 2m) deutlich verbessern – insbesondere während der Fahrt kommt es zu erheblicher Bildunschärfe, welche die Marker-Erkennung einschränkt.

2.5 Einparken parallel zur Fahrtrichtung

Mit Hilfe des Moduls *parking* erfolgt das automatische Längseinparken.

Verwendung des Moduls

Hierfür werden die Nodes *car_controller* und *carduino* für die Messungen von Ultraschallsensor, Hallsensor und Winkel, *teleop_twist_keyboard* und *rc_car* für die manuelle Steuerung des Fahrzeugs, sowie *parking* für automatisches Einparken benötigt. Das Zusammenspiel dieser Nodes/Topics ist in Abbildung 11 dargestellt.

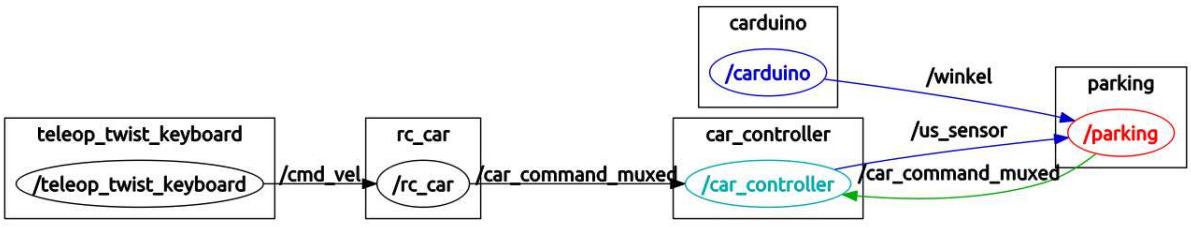

Abbildung 11 - ROS-Struktur der Nodes

Als Umgebungsszenario ist vorgesehen, zwischen zwei parkenden Fahrzeugen einzuparken. Während das Fahrzeug neben dem ersten parkenden Fahrzeug steht, wird der *parking*-Node gestartet.

```
# rosrun parking parking_node
```

Anschließend wird das Fahrzeug manuell (per Fernsteuerung, vgl. Abschnitt 2.3 „Manuelles Fahren per Fernsteuerung“ auf Seite 6) vom ersten parkenden Auto bis zum zweiten Auto

gefahren. Dabei wird die Parklücke mit Hilfe von Hallsensor (Wegstrecke/Länge der Parklücke) und Ultraschallsensor (seitlicher Abstand identifiziert parkende Fahrzeuge bzw. Freiraum) vermessen. Falls die Parklücke zu klein ist, werden alle Messgrößen zurückgesetzt, und nächste mögliche Parklücke wird gesucht. Der gesamte Vorgang ist in den nachfolgenden Abbildungen schematisch dargestellt. (Grün: parkende Fahrzeuge; Rot: einparkendes Fahrzeug; Blau: Wand)

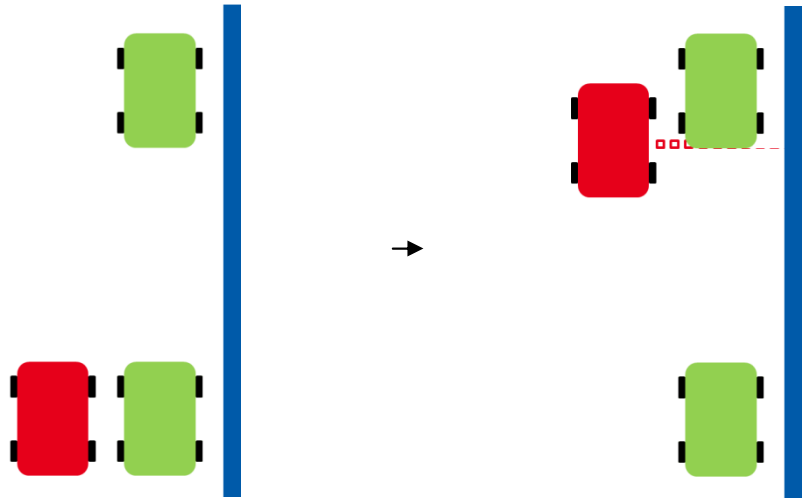

Abbildung 12 - Vermessen der Parklücke

Ist die Parklücke ausreichend groß (derzeit 80cm), wird der aktuelle Winkel über den Gyro-Sensor eingelesen und gespeichert. Das Fahrzeug lenkt voll ein und fährt rückwärts, bis die Drehung um die Fahrzeughochachse 30° beträgt (relativ zum zuvor gespeicherten Wert). Anschließend wird gegengelenkt und weiter rückwärts gefahren bis der Winkelunterschied wieder 0° beträgt.

Der Einparkvorgang ist damit erfolgreich beendet.

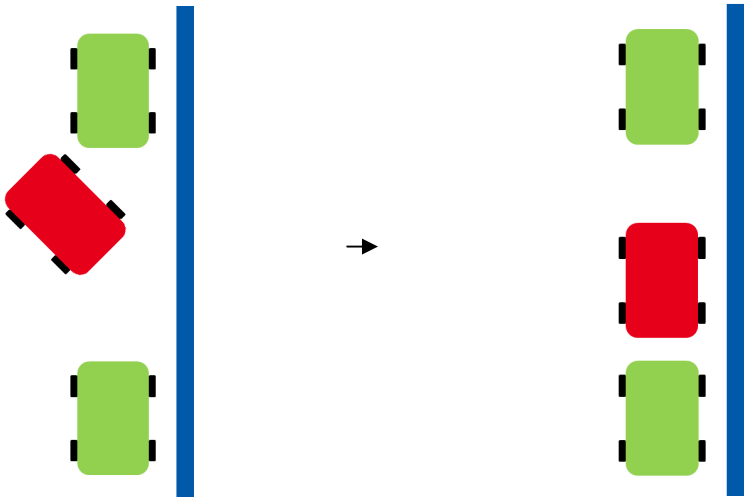

Abbildung 13 - Automatisches Einparken

Ausblick

Das Fahrzeug verfügt derzeit nur einen Ultraschallsensor pro Seite (jeweils einen vorne, links, rechts). Um ein besseres Verhalten und Ergebnis zu erzielen, können weitere Ultraschallsensoren hilfreich sein - an der rechten Seite insgesamt zwei (rechts vorne und rechts hinten) und einen weiteren an der Rückseite. Dadurch wäre es möglich, den genauen Abstand zwischen den parkenden Fahrzeugen und zur Wand zu ermitteln, um genaueres Einparken zu ermöglichen.

2.6 Anbindung eines Arduino-Entwicklerboards

In diesem Modul wird Mithilfe eines Arduino-Boards eine zusätzliche Hardware ins System eingebracht. Das Arduino wird mit einem MPU6050 (Gyro) verbunden, um die Winkelgeschwindigkeiten sowie Beschleunigungen aller drei Raumachsen zu messen. Mit Hilfe dieser Werte lässt sich das System besser beschreiben, jedoch wäre es noch besser die Geschwindigkeiten, den Weg und den aktuellen Lenkwinkel des Fahrzeuges zu kennen. Aufgrund begrenzter Hardwareressourcen können mit dem Modul nur die Beschleunigung und der Drehwinkel um die Z-Achse bestimmt werden. Der Drehwinkel wird dabei nicht gemessen, sondern mit Hilfe einer Integration bestimmt. Die gemessenen Werte werden anschließend mit der UART Schnittstelle des Arduino übertragen, dort gepublisht und anschließend von anderen Modulen abgefangen und verarbeitet.

Eine detaillierte Beschreibung, wie ein Arduino in das System eingebunden werden kann, wurde im PSES-Wiki[5] veröffentlicht. Deshalb wird in diesem Abschnitt eher auf die Hauptaufgabe des Arduino Moduls eingegangen, welche das auslesen des MPU6050 ist.

[5] http://pswiki.es.e-technik.tu-darmstadt.de/foswiki/bin/view/Main/WebHome (nur aus dem ES-Netz erreichbar)

Für einen Messvorgang werden fünf Werte gemessen, gemittelt und der Offset addiert, welcher vorher mit 3000 Messungen bestimmt wurde. Danach werden die Werte in die realen physikalischen Größen umgewandelt wie der folgende Code-Ausschnitt zeigt:

```
acX = 0;
acY = 0;
acZ = 0;
gyZ = 0;

while(a != 5){
   readData(); // liest die Sensorwerte ein und addiert diese in den oben stehenden Variablen auf
   a++;
}

acX /= 5;
acY /= 5;
acZ /= 5;
gyZ /= 5;

//Rechne Bytewerte in Beschleunigungswerde um
acX = acX / 16384 * 9.81;
acY = acY / 16384 * 9.81;
acZ = acZ / 16384 * 9.81;
```

Ziel der Integration ist es, den Drehwinkel um die Z-Achse zu bestimmen. Mit Hilfe dieses Winkels ist es dann möglich, die Szenarien *Kurvenfahrt* (Abschnitt 2.2) und *Einparken* (2.5) einfacher zu implementieren. Außerdem wäre es möglich, die genaue zurückgelegte Strecke in Koordinatenrichtungen mit Hilfe des Hall-Sensors aufzuzeichnen. Die Integration wird mit Hilfe des Timers, welcher sich auf dem Arduino befindet, realisiert. Der Timer kann entweder mit *millis()* oder *nanos()* ausgelesen werden. Da im Diskreten eine Integration der Aufsummierung der Flächeninhalte von Rechtecken entspricht, muss die Differenzzeit zwischen zwei Messwerten mit Hilfe des Timers abgeglichen werden. In dem Anwendungsfall wurde der Zeitabstand auf 10 Mikrosekunden festgelegt. Es ist auch möglich, einen kleineren Zeitabstand zu wählen, jedoch würde dies das System mehr Rechenzeit kosten.

Am Anfang der Loop-Schleife des Arduino wird der aktuelle Timerstand abgespeichert. Gegen Ende der Schleife wird der abgespeicherte Wert mit dem aktuellen Wert des Timers verglichen werden. Falls der aktuelle Wert die 10 Mikrosekunden noch nicht erreicht hat, wird gewartet, bis diese Zeit verstrichen ist. Zu beachten ist dabei, dass hier ein Fehler entstehen kann, falls die Berechnung viel schneller als 10 Mikrosekunden ist. Die eigentliche Integration erfolgt durch

```
winkel_g_z = gyZ * (t/1000000)/131*250 + winkel_g_z;
```

Dabei wird der Timerwert in Sekunden umgerechnet und mit der aktuell gemessenen Winkelgeschwindigkeit multipliziert. Schließlich wird noch der alte Wert des Winkels dazu addiert, um den absoluten Drehwinkel zu erhalten.

Integrationen sind normalerweise nicht zu empfehlen, da es zu einem wegdriften (Integrationsfehler) des Sensors kommen kann. Da aber kein Magnetometer zur Verfügung stand, wurde dieser Weg als Alternative gewählt. Ein Grund für den Integrationsfehler ist, dass der Sensor Rauschen besitzt. So kann es passieren, dass der Sensor eine Änderung der Winkelge-

schwindigkeiten anzeigt, obwohl sich das Fahrzeug nicht bewegt (abgesehen von der Z-Achse). Zudem trägt die Ungenauigkeit der diskreten Integration zum Fehler bei, da die Aufsummierung von Rechtecken nur eine Annäherung an ein kontinuierliches Integral ist. Um erstgenanntem entgegenzuwirken, versucht man den Drehwinkel über die gemessenen Beschleunigungen anzunähern:

```
winkel_a_z = atan((acZ / sqrt(acY*acY + acX*acX)))*r2d;
```

Somit stehen zwei unabhängige Messwerte zu Verfügung, was das Rauschen etwas unterdrückt. Abschließend werden beide Winkel verrechnet, wobei die Annährung des Drehwinkels weniger gewichtet wird:

```
winkel_ z = gain * winkel_g_z + (1-gain) * winkel_a_z;
```

Mit Hilfe dieser Methode ist es möglich, den Winkel im Stillstand für ca. 5 Minuten zu stabilisieren. Es ist auch möglich, mehrere Kurven zu fahren oder Einparkvorgänge zu absolvieren. Jedoch sollte der Messvorgang nach einigen Manövern während eines Stillstandes zurückgesetzt werden (z. B. nach der Erkennung einer Kurve). Ein Zurücksetzten des Integrals funktioniert aber schneller, da es nicht mehr nötig ist, die Offsetwerte erneut zu bestimmen. Eine bessere Alternative wäre, direkt ein Magnetometer oder zumindest zwei Gyroskope gleichzeitig zu verwenden.

Ausblick

Das Einbringen weiterer Sensoren würde einen größeren Lösungsraum eröffnen. Mit der Hilfe aller Zustände des Systems wäre es möglich, kartographische Ansätze einfach zu verfolgen. Für den Anfang würden weitere Ultraschallsensoren an den Seiten, zwei Magnetometer, zwei Gyroskope, ein Encoder für die Bestimmung des tatsächlichen Lenkwinkels und ein genauerer Encoder (als der Hall-Sensor) für die zurückgelegte Strecke denkbar.

1 Ackermannmodell

Zur systematischen Reglerauslegung zur Seitenführung (Abstand zur Wand) bzw. zur Diskussion des prinzipiellen Verhaltens des Fahrzeuges ist ein mathematisches Modell notwendig. Ein einfaches Modell soll hier vorgestellt werden.

Zur Beschreibung der Bewegung eines Fahrzeuges existieren verschiedene Modelle, die unterschiedliche Annahmen bzw. Vereinfachungen treffen. Eines der einfachsten Modelle ist das Ackermannmodell, welches ein rein kinematisches Modell darstellt. D. h. die Fahrzeuggeschwindigkeit (hier als Geschwindigkeit an der Hinterachse definiert) ist eine Eingangsgröße des Modells. (Ein Modell, welches die Kinetik/Dynamik berücksichtigen würde, würde als Eingangsgrößen Kräfte bzw. Momente besitzen, und die Geschwindigkeit würde daraus folgen.) Entsprechend kommen keine Parameter wie Massen oder Trägheiten, sondern nur geometrische Größen in dem Modell vor.

Das Ackermannmodell ist desweitern ein Einspurmodell, d. h. die Räder einer Achse werden zu einem gemeinsamen Rad zusammengefasst. Die wesentliche Annahme des Ackermannmodells ist, dass die Räder nicht über die Seite „schieben". D. h. zu jedem Zeitpunkt zeigt der Geschwindigkeitsvektor an den Rädern in die Richtung des jeweiligen Rades. Es wird also davon ausgegangen, dass die Räder jederzeit genügend Seitenkraft aufbringen können, eine Bewegung senkrecht zur Abrollrichtung zu vermeiden.

Ein anderes Modell wäre beispielsweise das „Einspurmodell", welches einen gewissen Schräglauf der Reifen zulässt (bzw. erfordert).

Das Ackermannmodell ist dann zulässig, wenn der Schräglauf oder das „Schieben" über die Räder vernachlässigt werden kann. Dies ist bei kleinen Geschwindigkeiten, steifen Rädern (niedriger Querschnitt), einer guten Haftung zwischen Rad und Boden und nicht zu großen Lenkwinkeln der Fall und sollte hier angenommen werden können.

Hier wird das Ackermannmodell eines Fahrzeugs mit ungelenkter Hinterachse betrachtet. Die Größen des Ackermannmodells sind in Tabelle 1.1 zusammengefasst und in Abbildung 1.1 dargestellt. Dabei sind l, l_H und die Geschwindigkeit v an der Hinterachse Parameter des Modells, der Lenkwinkel φ_L ist die Eingangsgröße und der Kurswinkel φ_K sowie die Position x in Fahrbahnrichtung (wobei die Fahrbahn hier als gerade angenommen wird) und die Querablage y sind Zustände. (Die Geschwindigkeit könnte auch als Eingangsgröße gezählt werden.) Dabei sind x und y bezogen auf einen Referenzpunkt angegeben, der über den Parameter l_H (Abstand zur Hinterachse) festgelegt ist. Der Referenzpunkt ist hier derjenige Punkt, für den der Abstand zur Wand gemessen wird.

Ziel ist es zunächst, einen Zusammenhang zwischen der Eingangsgröße Lenkwinkel φ_L und der Querablage y aufzustellen.

Tabelle 1.1: Größen des Ackermannmodells

φ_L	Lenkwinkel
φ_K	Kurswinkel
v	Geschwindigkeit an ungelenkter Hinterachse
x	Position Referenzpunkt in Fahrbahnrichtung
y	Abstand Referenzpunkt zur rechten Fahrbahnbegrenzung
l_H	Abstand Referenzpunkt zur Hinterachse
l	Achsstand

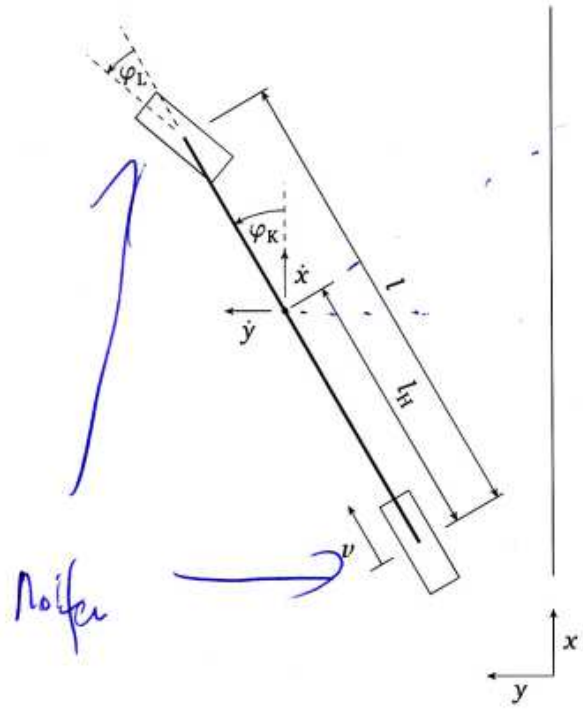

Abbildung 1.1: Größen des Ackermannmodells

1.1 Modellgleichungen

Für die Geschwindigkeiten $\dot{x}$ und $\dot{y}$ des Referenzpunktes in Fahrbahnrichtung bzw. senkrecht dazu gilt

$$\dot{x} = v \cdot \cos\varphi_K - l_H \dot{\varphi}_K \cdot \sin\varphi_K$$
$$\dot{y} = v \cdot \sin\varphi_K + l_H \dot{\varphi}_K \cdot \cos\varphi_K .$$

Die wesentlichen Größen zur Bestimmung der Geschwindigkeiten sind der Kurswinkel φ_K und dessen Ableitung. Um die Abhängigkeit des Kurswinkels vom Lenkwinkel und anderen Größen zu beschreiben, wird Abbildung 1.2 betrachtet.

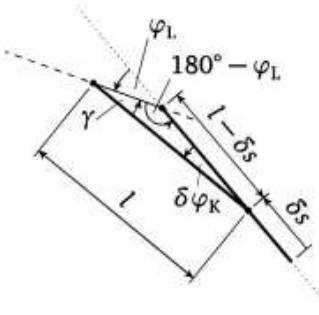

Abbildung 1.2: Zur Bestimmung der Änderung des Kurswinkels

Dort ist der Fall dargestellt, dass sich die Hinterachse um die kleine Strecke δs fortbewegt. Diese Bewegung muss aufgrund der Grundannahme des Ackermannmodells in die Richtung der Hinterachse

erfolgen, also entlang einer Gerade mit dem Winkel φ_K, die als gepunktete Linie eingezeichnet ist. Die Vorderachse muss sich entsprechend auf einer Gerade bewegen, die mit dem Lenkwinkel gegenüber der „Kursgeraden" geneigt ist. Diese Gerade ist in Abbildung 1.2 gestrichelt eingezeichnet. Die Länge l des Fahrzeuges ändert sich natürlich nicht, womit sich die in der Abbildung eingezeichneten Längen ergeben. Es ergibt sich damit eine Änderung des Kurswinkels von $\delta\varphi_K$.

Für den in Abbildung 1.2 mit γ bezeichneten Winkel gilt

$$\gamma = 180° - \delta\varphi_K - (180° - \varphi_L) = \varphi_L - \delta\varphi_K .$$

Wendet man den Sinussatz auf das durch die Punkte markierte Dreieck an, so erhält man zunächst

$$\frac{\sin(180° - \varphi_L)}{l} = \frac{\sin\gamma}{l - \delta s},$$

bzw. mit

$$\sin(180° - \varphi_L) = \sin\varphi_L$$

auch

$$\frac{\sin\varphi_L}{l} = \frac{\sin(\varphi_L - \delta\varphi_K)}{l - \delta s} . \tag{1.1}$$

Mit dem Additionstheorem

$$\sin(\alpha - \beta) = \sin\alpha \cdot \cos\beta - \cos\alpha \cdot \sin\beta$$

ergibt sich weiter

$$\frac{\sin\varphi_L}{l} = \frac{\sin\varphi_L \cdot \cos\delta\varphi_K - \cos\varphi_L \cdot \sin\delta\varphi_K}{l - \delta s} .$$

Für kleine Änderungen $\delta\varphi_K$ des Kurswinkels gilt $\cos\delta\varphi_K \approx 1$ und $\sin\delta\varphi_K \approx \delta\varphi_K$. Da später ohnehin noch der Grenzübergang $\delta\varphi_K \to 0$ ausgeführt wird, ist dies nicht nur eine Näherung, sondern führt zu einem exakten Ergebnis. Es ergibt sich zunächst

$$\frac{\sin\varphi_L}{l} = \frac{\sin\varphi_L - \cos\varphi_L \cdot \delta\varphi_K}{l - \delta s}$$

was über

$$\frac{l - \delta s}{l} = \frac{\sin\varphi_L - \cos\varphi_L \cdot \delta\varphi_K}{\sin\varphi_L}$$

$$1 - \frac{\delta s}{l} = 1 - \frac{1}{\tan\varphi_L} \cdot \delta\varphi_K$$

zu

$$\delta\varphi_K = \frac{\tan\varphi_L}{l} \cdot \delta s$$

umgeformt werden kann.[1]

[1] Aus Gl. (1.1) ergibt sich als exakter Ausdruck für die Änderung des Lenkwinkels, d. h. ohne die Annahme kleiner Änderungen, die Gleichung $\delta\varphi_K = \varphi_L - \sin^{-1}\left(\left(1 - \frac{\delta s}{l}\right) \cdot \sin\varphi_L\right)$.

Dividiert man diese Gleichung noch durch die Dauer δt, indem die Strecke δs zurückgelegt wurde, erhält man

$$\frac{\delta \varphi_K}{\delta t} = \frac{\tan \varphi_L}{l} \cdot \frac{\delta s}{\delta t},$$

und letztlich durch $\delta t \to 0$ den gesuchten Zusammenhang

$$\dot{\varphi}_K = \frac{v}{l} \cdot \tan \varphi_L$$

zwischen dem Lenkwinkel und der Änderung des Kurswinkels.

Diese Differentialgleichung bildet zusammen mit der Differentialgleichung für die Seitenablage y das Differentialgleichungssystem

$$\begin{aligned} \dot{\varphi}_K &= \frac{v}{l} \cdot \tan \varphi_L \\ \dot{y} &= v \cdot \sin \varphi_K + v \cdot \frac{l_H}{l} \cdot \cos \varphi_K \cdot \tan \varphi_L, \end{aligned}$$

welches die Kinematik beschreibt. (Die Position x in Fahrbahnrichtung wird hier nicht betrachtet.) Diese Gleichungen sind für alle Kurswinkel und alle Lenkwinkel ungleich 90° exakt.[2]

Linearisierung

Für kleine Kurswinkel gilt $\sin \varphi_K \approx \varphi_K$ und $\cos \varphi_K \approx 1$ womit man

$$\begin{aligned} \dot{\varphi}_K &= \frac{v}{l} \cdot \tan \varphi_L \\ \dot{y} &= v \cdot \varphi_K + v \cdot \frac{l_H}{l} \cdot \tan \varphi_L \end{aligned}$$

erhält.

Nimmt man zusätzlich kleine Lenkwinkel an, so kann $\tan \varphi_L \approx \varphi_L$ gesetzt werden, womit sich

$$\begin{aligned} \dot{\varphi}_K &= \frac{v}{l} \cdot \varphi_L \\ \dot{y} &= v \cdot \varphi_K + v \cdot \frac{l_H}{l} \cdot \varphi_L \end{aligned}$$

ergibt.

Allgemeiner kann die Nichtlinearität der Eingangsgröße über ein „Vorfilter" $\varphi_L^* = \tan \varphi_L$ berücksichtigt werden. D. h. das System lautet bezüglich der neuen Eingangsgröße φ_L^*

$$\begin{aligned} \dot{\varphi}_K &= \frac{v}{l} \cdot \varphi_L^* \\ \dot{y} &= v \cdot \varphi_K + v \cdot \frac{l_H}{l} \cdot \varphi_L^*. \end{aligned}$$

Nun liese sich ein linearer Regler auslegen, der eine die Eingangsgröße φ_L^* fordert. Diese kann dann über $\varphi_L = \tan^{-1} \varphi_L^*$ in die tatsächliche Eingangsgröße umgerechnet werden.

[2] Mathematisch gesehen. Praktisch ist für zu große Lenkwinkel die Ackermannannahme (kein Schieben über die Räder) nicht einzuhalten.

Systemdarstellung

Mit der Eingangsgröße φ_L^* und der Ausgangsgröße y ergibt sich die Zustandsraumdarstellung

$$\begin{bmatrix} \dot{y} \\ \dot{\varphi}_K \end{bmatrix} = \begin{bmatrix} 0 & v \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} y \\ \varphi_K \end{bmatrix} + \begin{bmatrix} v \cdot \frac{l_H}{l} \\ \frac{v}{l} \end{bmatrix} \cdot \varphi_L^*$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} y \\ \varphi_K \end{bmatrix}$$

bzw. die Übertragungsfunktion

$$G(s) = \frac{v \cdot l_H}{l} \cdot \frac{\frac{v}{l_H} + s}{s^2}.$$

Eine Blockschaltbilddarstellung ist in Abbildung 1.3 gezeigt.

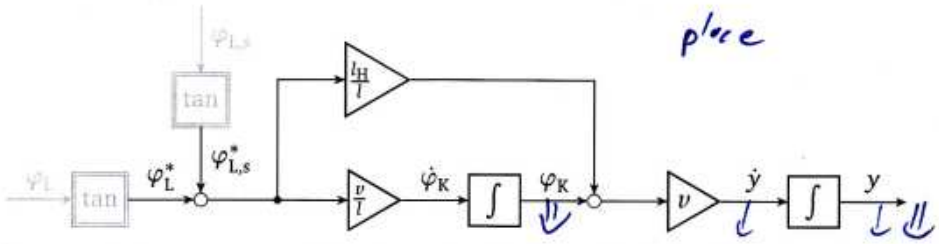

Abbildung 1.3: Ackermannmodell in Blockschaltbilddarstellung (für kleine Kurswinkel φ_K)

1.2 Diskussion

Das System besitzt zwei Pole in Null (Integratoren) und eine Nullstelle. Für negative Geschwindigkeiten ($v < 0$) ist die Nullstelle positiv und das System damit nicht minimalphasig. Dies weist aus regelungstechnischer Sicht darauf hin, dass das rückwärtsfahrende Fahrzeug deutlicher schwieriger zu regeln ist. (Anschaulich schlägt der Referenzpunkt zunächst immer gegen die eigentlich gesteuerte Richtung aus. Dies ist ein für nicht minimalphasige Systeme charakteristisches Verhalten.)

Die beiden Integratoren und der damit einhergehende Phasenabfall sind kritisch für die Regelung. So muss der Regler einen D-Anteil besitzen. Ein reiner P-Regler würde praktisch zu einem sehr schwach gedämpften Systemverhalten führen.

Die wesentliche Störung ist ein Fehler in der Lenkung (Spiel, nicht exakt arbeitende Servos, Fehler in der Nulllage). Dieser ist in Abbildung 1.3 als Störgröße $\varphi_{L,s}$ bzw. $\varphi_{L,s}^*$ dargestellt. Diese Störung wirkt direkt am Streckeneingang, so dass für eine stationär genaue Regelung trotz der beiden I-Anteile der Strecke ein I-Anteil im Regler notwendig ist. (Das vereinfachte Nyquistkriterium, d. h. das Kriterium über Amplituden- bzw. Phasenreserve ist für ein System mit drei Integratoren nicht mehr anwendbar. D. h. eine stabilisierende Regelung ist prinzipiell auch hier möglich, wobei diese praktisch relativ langsam werden wird.)

1.2.1 Ansätze/Anregungen

Das hier vorgestellte, einfache Modell für das Fahrzeug kann dazu genutzt werden, simulativ mit den Reglerparametern „zu spielen" und somit schneller zu einem Gefühl für die prinzipiellen Auswirkungen zu kommen. Auch können Effekte wie eine Aktordynamik und die Abtastung einfacher untersucht werden.

Anstelle der Reglerauslegung eines PD- oder PID-Reglers über „Probieren“ oder anhand der Wurzelortskurve oder ähnlichen Verfahren könnte der Regler über eine Polvorgabe oder einen Optimalregleransatz ausgelegt werden. Letztlich ist ein PD-Regler für das in Abbildung 1.3 dargestellte System dasselbe wie ein Zustandsregler, ein PID-Regler ein um einen I-Anteil erweiteter Zustandsregler.

Es tritt in der praktischen Umsetzung des D-Anteils auch das Problem auf, dass numerisch die Ableitung des Signals y bestimmt werden muss, welches nur in relativ großen Zeitschritten und verrauscht vorliegt. Sollte die Ausgangsgröße gefiltert werden, führt dies zu einem weiteren Phasenabfall und sollte daher bei dem Reglerentwurf berücksichtigt werden!

Hier könnte man auch einen Ansatz über einen Beobachterentwurf wählen. Letztlich stellt auch ein Beobachter hier eine Schätzung der Ableitung dar, aber der Entwurf ist etwas systematischer.

Bisher unbeachtet ist die Aktordynamik und die Abtastung. Je nachdem, wie lange der Servo benötigt, den gewünschten Lenkwinkel einzustellen, kann dieser eine für den Regelkreis relevante Dynamik oder Totzeit besitzen. Auch eine zu langsame Abtastung führt zu einem Phasenabfall und damit zu einem schlechter zu regelnden System. Zeigt sich die Abtastung als problematisch, so könnte der Reglerentwurf im Zeitdiskreten erfolgen.

Nutzen von Inertialsensorik

Es steht Inertialsensorik zur Verfügung, mit der Beschleunigungen und Drehraten gemessen werden können. Integriert man diese Werte auf, so erhält man den Kurswinkel und die Geschwindigkeit v in Fahrtrichtung sowie (unter Berücksichtigung des Kurswinkels) die Änderungsgeschwindigkeit $\dot{y}$ der seitlichen Ablage y. Praktisch kann diese Integration nur kurzzeitig durchgeführt werden, bis Offsetfehler dazu führen, dass die Werte zu weit „weglaufen“. Allerdings könnte die Intertialsensorik über ein Kalmanfilter gestützt werden, welches die Sensorsignale des Radsensors (Messung der Viertelumdrehungen) und des Ultraschalls nutzt, die integrierten Werte zu korrigieren und Offsetfehler zu schätzen.

Der Vorteil läge darin, bessere Schätzwerte für die benötigten Zustandsgrößen zu erhalten, die dazu auch noch in einem schnelleren Zeitraster vorliegen, womit die Regelung mit einer schnelleren Abtastzeit erfolgen kann.