Bachelorarbeit

Entwurf und Realisierung autonomer Fahrfunktionen in Modellfahrzeugen

Von:	Fabian Freihube
Eingereicht:	Oktober 2016
Hochschule:	HTWK Leipzig
Fakultät:	Informatik, Mathematik und Naturwissenschaften
Studiengang:	Medieninformatik (B.Sc.)
1. Gutachter:	Prof. Dr. Schwarz
2. Gutachter:	Prof. Dr. Bastian

Zusammenfassung

Die vorliegende Arbeit beschreibt alle notwendigen Schritte zur Implementierung einer Ansteuerung und Trajektorienplanung für ein autonomes Modellfahrzeug mit dem Ziel, eine Basis für die höhere Fahrzeuglogik bereitzustellen. Dabei wird auf die Integration von Methoden und Technologien aus den Bereichen Sensorfusion, Regelungstechnik und Aktuatorensteuerung in das Robot Operating System (ROS) eingegangen. Eine Eignung der entwickelten Trajektorienplanung und Geschwindigkeitsregelung für den normalen Straßenverkehr wurde in zwei Versuchen bestätigt.

Eidesstatliche Erklärung

Ich versichere, dass die Bachelorarbeit mit dem Titel “Entwurf und Realisierung autonomer Fahrfunktionen in Modellfahrzeugen“ nicht anderweitig als Prüfungsleistung verwendet wurde und diese Bachelorarbeit noch nicht veröffentlicht worden ist. Die hier vorgelegte Bachelorarbeit habe ich selbstständig und ohne fremde Hilfe abgefasst. Ich habe keine anderen Quellen und Hilfsmittel als die angegebenen benutzt. Diesen Werken wörtlich oder sinngemäß entnommene Stellen habe ich als solche gekennzeichnet.

Leipzig, 16. Oktober 2016 Unterschrift

Inhaltsverzeichnis

1 Einleitung

1.1 Projektbeschreibung

Das Team HTWK Smart Driving ist eine Gruppe von Studenten der Hochschule für Technik, Wirtschaft und Kultur (HTWK) Leipzg, die sich mit der Entwicklung autonomer Fahrzeuge auf Basis von Modellfahrzeugen beschäftigt. Nach den erfolgreichen Teilnahmen des Teams bei dem Audi Autonomous Driving Cup 2015 und 2016, wird nun ein eigenes Modellfahrzeug aufgebaut. Dieses Fahrzug soll für die Entwicklung von Algorithmen sowie einer Softwarearchitektur für autonome Fahrzeuge genutzt werden. Dabei stehen insbesondere die Teilbereiche der Sensorfusion, Bildverarbeitung, Trajektorienplanung und Ansteuerungslogik im Fokus des Teams.

Bei dem Fahrzeug handelt es sich um ein Modellfahrzeug aus dem Hobbybereich im Maßstab 1:10, das durch Sensorik und Recheneinheiten ergänzt wurde, um den Fahraufgaben eines urbanen Geländes gerecht zu werden.

Diese Bachelorarbeit entstand im Rahmen des Aufbau des Fahrzeuges und beschäftigt sich mit der grundlegenden Ansteuerung des Modellautos. Die Aufgabenstellung ist detailliert im folgenden Abschnitt 1.2 erläutert.

1.2 Aufgabenstellung

Ein autonomes Fahrzeug soll in der Lage sein, verschiedene Fahrmanöver auszuführen, wie z.B. das Folgen einer kurvigen Straße oder das zentimetergenaue

Einparken. Dazu muss eine zuvor geplante Trajektorie möglichst genau, im Sinne von Geschwindigkeit und Position, umgesetzt werden können.

Aus dieser Zielsetzung ergeben sich mehrere Teilaufgaben. So ist für eine Regelung der Geschwindigkeit eine präzise Erfassung der aktuellen Geschwindigkeit unerlässlich. Zu diesem Zweck sollen die vorhandenen Sensordaten kontinuierlich fusioniert werden, um Messungenauigkeiten der einzelnen Sensoren zu minimieren. Als umgesetzt kann diese Aufgabe angesehene werden, wenn die Zustandsschätzung sowohl in Kurvenabschnitten als auch auf Geraden ausreichend genaue Geschwindigkeitsdaten liefert, um damit eine Geschwindigkeitsregelung durchzuführen und eine Trajektorie, wie sie auch im echten Straßenverkehr vorkommt, abgefahren werden kann.

Daraus resultiert eine zweite Teilaufgabe. Es ist eine Schnittstelle zu implementieren, die eine Kommunikation der Fahrzeuglogik mit der Ansteuerung des Fahrzeuges ermöglicht. Es soll möglich sein, dem Ansteuerungsmodul sowohl Befehle zur Änderung der Geschwindigkeit als auch der Orientierung im Raum zu übermitteln.

Mit der Umsetzung dieser Fahraufgaben beschäftigt sich der dritte und vierte Teilaspekt dieser Arbeit. Es soll eine Geschwindigkeitsregelung entworfen und implementiert werden, die das Fahren einer konstanten Geschwindigkeit, unabhängig von verschiedenen Störgrößen, ermöglichen soll. Weiterhin soll auch eine Änderung der Geschwindigkeit flüssig und rasch erfolgen. Zusammenfassend kann gesagt werden, dass die Regelung der Geschwindigkeit alle Anforderungen des normalen Straßenverkehrs erfüllen muss.

Weiterhin soll das Fahrzeug in der Lage sein, eine Trajektorie zu einem Zielpunkt selbstständig zu berechnen und anschließend abzufahren. Ein wichtiges Qualitätsmerkmal ist dabei die Präzision der Positionsänderung. So soll das Fahrzeug beispielsweise bei bekannter Ausgangspostion dazu in der Lage sein, bei einer Kreuzungsgröße von $1m \times 1m$, selbstständig abzubiegen ohne die Fahrbahnmarkierungen zu überfahren. Das Fahrzeug hat eine angemessene Geschwindigkeit selbst zu wählen.

1.3 Überblick

Die Gliederung dieser Arbeit orientiert sich an den einzelnen Teilaufgaben dieser Arbeit. In jedem Kapitel ist die Abfolge Sensordatenerfassung und -fusion, Trajektorienplanung und Geschwindigkeitsregelung wiederzufinden.

In Kapitel 2 wird auf die notwendigen mathematischen und technischen Grundlagen eingegangen um die zuvor in Abschnitt 1.2 definierten Aufgaben zu erfüllen. Im Kapitel 3, dem Entwurf, wird eine entsprechende Architektur entwickelt. Weiterhin erfolgt hier die Schnittstellendefinition der Fahraufgabe und die Auswahl eines passenden Reglers für das System. Daran schließt sich das Kapitel 4, die Umsetzung, an. Hier wird vor allem auf die Realisierung des Entwurfes eingegangen. Weiterhin findet man hier Informationen über die Einbindung der verwendeten Sensoren und Aktuatoren in das Robot Operating System (ROS). Im letzten Kapitel vor dem Fazit wird die Validierung der Ergebnisse sowie die verwendeten Methoden besprochen. Schlussendlich wird in Kapitel 6 ein Fazit gezogen, die erbrachten Ergebnisse kritisch bewertet und ein Ausblick auf eine mögliche Weiterführung der Arbeit, im Hinblick auf Verbesserungspotential, gegeben.

2 Grundlagen

2.1 Mathematische Grundlagen

2.1.1 Der Zahlenbereich der Quaternionen

Die Quaternionen sind eine Erweiterung des reellen Zahlenbereichs. Sie wurden im Jahr 1843 von Sir William Rowan Hamilton erfunden und werden daher auch häufig als Hamilton-Zahlen bezeichnet. Daraus leitet sich auch das Symbol des Zahlenbereiches ab - $\mathbb{H}$. [2, S. 30]

Ähnlich den komplexen Zahlen entstehen die Quaternionen aus den reellen Zahlen indem man 3 weitere Zahlen hinzufügt. Somit sind die Quaternionen ein vierdimensionaler Vektorraum. Sie bestehen aus einem einelementigen Realteil und einem dreielementigen Imaginärteil. Angelehnt an die imaginäre Einheit der komplexen Zahlen, wurden den Komponenten des Imaginärteils die Namen i, j, k gegeben. Ein Quaternion, dessen Realteil Null ist, nennt man reines Quaternion.

$$q := w + x \cdot i + y \cdot j + z \cdot k \tag{2.1}$$

In 2.1 sieht man die Definition eines Quaternion q. Ein Quaternion kann als Vektor mit den Komponenten w, x, y und z vollständig beschrieben werden. Die Grundrechenarten Addition und Subtraktion werden, der Vektorraumeigenschaft folgend, elementweise durchgeführt. Vom Vektorraum wird auch die skalare Multiplikation

übernommen. Eine Multiplikation zweier Basiselemente ergibt, wie in 2.2 und 2.3, immer das dritte Element. Diese Rechenregel wird als Hamilton-Regel bezeichnet.

$$i \cdot j = k \qquad j \cdot k = i \qquad k \cdot i = j \tag{2.2}$$

$$j \cdot i = -k \qquad k \cdot j = -i \qquad i \cdot k = -j \tag{2.3}$$

Besonders in der Informatik haben Quaternionen eine große Bedeutung. Bei der Beschreibung von Drehungen im dreidimensionalen euklidischen Raum liefern die Quaternionen effiziente Verfahren, so dass sie häufig an Stelle von Drehmatrizen zum Einsatz kommen. Insbesondere bei der Kombination vieler Drehungen kann mit Quaternionen eine signifikante Menge an Rechenoperationen eingespart werden.

Mit Quarternionen kann zudem das Problem des Gimbal Locks, was häufig bei der Verwendung von inertialen Navigationssystemen auftritt, gelöst werden. Bei der Rotation eines Objektes in einem globalen Koordinatensystem mit Hilfe von Eulerwinkeln werden nacheinander drei Rotationen durchgeführt. Eine Drehung um 90° bewirkt das Zusammenfallen zweier Rotationsachsen. Wenn man z.B. 90° um die Y-Achse rotiert, fallen die X- und die Z-Achse zusammen. Das heißt eine Rotation um die X-Achse ergibt nun das gleiche Ergebnis, wie eine Rotation um die Z-Achse. Somit geht ein Freiheitsgrad verloren.

2.1.2 Der Kalman-Filter

Der Kalman-Filter ist der vermutlich am weitesten verbreitete Filter zur Zustandsschätzung linearer und nichtlinearer Systeme. Er wird immer dann eingesetzt, wenn aufgrund von bekannten Messunsicherheiten verschiedener Sensoren der Zustand eines Systems geschätzt werden muss. Die besten Leistungen werden dabei erzielt,

wenn das System mit Informationen verschiedener Sensoren mit unterschiedlichen Fehlercharakteristiken gespeist wird.

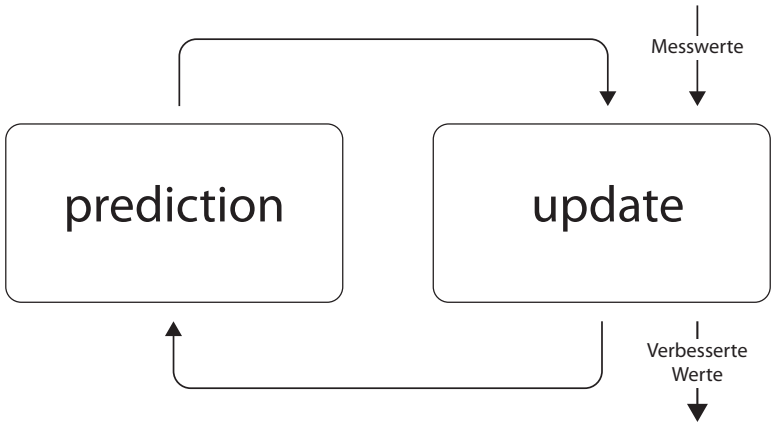


Abbildung 2.1: Funktionsweise eines Kalman-Filters

Wie in Abbildung 2.1 zu sehen ist, handelt es sich bei dem Filter um einen Kreisprozess, dessen zwei Abschnitte man grob in *prediction* und *update* einteilen kann. Die zwei wichtigen Größen sind dabei der Systemzustand und die Kovarianz. Beide werden abwechselnd den neuen, verrauschten Messwerten angepasst.

Für den *prediction* Schritt sieht das wie folgt aus.

$$\mathbf{x}_{k+1|k} = \mathbf{A} \cdot \mathbf{x}_k \tag{2.4}$$

$$\mathbf{P}_{k+1|k} = \mathbf{P}_k \cdot \mathbf{A} \cdot \mathbf{A}^T + \mathbf{Q} \tag{2.5}$$

Zunächst wird in 2.4 der Systemzustandsvektor $\mathbf{x}_k$ mit Hilfe der Dynamikmatrix $\mathbf{A}$ für den Schritt $k+1$ berechnet. Es wird also der Systemzustand des nächsten Schrittes vorhergesagt (engl. *prediction*). Die Schreibweise $\mathbf{x}_{k+1|k}$ bedeutet, dass $\mathbf{x}_{k+1}$ zum Zeitpunkt k berechnet wird.

Die Bestimmung der Matrix $\mathbf{A}$ ist der Kern bei der Entwicklung eines Kalman-Filters. Sie modelliert die Dynamik zwischen zwei Zuständen und ist somit von der physikalischen Beschreibung des Problems abhängig. Ein Beispiel dafür ist die gleichförmige Bewegung. Multipliziert mit dem Systemzustand $\mathbf{x}_k$, muss die Dynamikmatrix den Systemzustand $\mathbf{x}_{k+1}$ liefern.

Als nächstes wird in 2.5 die Kovarianz $\mathbf{P}$ aktualisiert. Dabei ist $\mathbf{Q}$ die Prozessrauschkovarianzmatrix, die die Störung der Dynamik abbildet. Für das Beispiel der gleichförmigen Bewegung wäre $\mathbf{Q}$ die Varianz der Bewegung, die z.B. durch eine unebene Straße oder Windeinfluss hervorgerufen wird. Die Kovarianz $\mathbf{P}$ ist somit die Unsicherheit des Systemzustands $\mathbf{x}_k$. $\mathbf{Q}$ ist hingegen die Unsicherheit des Übergangs $\mathbf{x}_k \rightarrow \mathbf{x}_{k+1}$.

Der *update* Schritt folgt direkt im Anschluss an die *prediction* und betrachtet, ob die berechnete Vorhersage zutrifft und ob diese angepasst werden muss.

$$\mathbf{y}_{k+1} = \mathbf{z}_{k+1} - (\mathbf{H} \cdot \mathbf{x}_{k+1|k}) \tag{2.6}$$

$$\mathbf{K}_{k+1} = \frac{\mathbf{P}_{k+1|k} \cdot \mathbf{H}^T}{\mathbf{H} \cdot \mathbf{H}^T \cdot \mathbf{P}_{k+1|k} + \mathbf{R}} \tag{2.7}$$

Dazu werden in 2.6 und 2.7 zwei Hilfsgrößen eingeführt. Zum einen die Inovation des Systemzustandsvektor $\mathbf{y}_k$ und das Kalman-Gain $\mathbf{K}_k$. Letzteres wird geringer, wenn die Messwerte dem vorhergesagten Systemzustand entsprechen. Um die Innovation zu berechnen, benötigt man neben den aktuellen Messwerten $\mathbf{z}_k$ noch eine Messmatrix $\mathbf{H}$. Diese gibt an, in welchem Verhältnis die neuen Messwerte zum Zustandsvektor stehen und wie sie auf diesen abgebildet werden können. Der Summand $\mathbf{R}$ in der Berechnung des Kalman-Kalman Gains ist die Messrauschkovarianzmatrix. Diese ist zusammen mit der Kovarianz $\mathbf{P}$ und der Prozessrauschkovarianz $\mathbf{Q}$ die dritte Unsicherheit im System und hängt von der Genauigkeit der eingesetzten Sensoren ab.

$$\mathbf{x}_{k+1} = \mathbf{x}_{k+1|k} + (\mathbf{K}_k \cdot \mathbf{y}_k) \tag{2.8}$$

$$\mathbf{P}_{k+1} = (\mathbf{I}_n - (\mathbf{K}_k \cdot \mathbf{H})) \cdot \mathbf{P}_{k+1|k} \tag{2.9}$$

In 2.8 und 2.9 werden dann der Systemzustand und die Kovarianz angepasst. Anschließend beginnt der Prozess von vorn. In einer softwaretechnischen Umsetzung würde man nach dem *update*-Schritt, den verbesserten Systemzustand abgreifen, bevor von diesem Wert aus, erneut der *prediction*-Schritt eingeleitet wird. [4]

2.2 Technische Grundlagen

2.2.1 Der Regelkreis

Ein Regelkreis ist ein in sich geschlossener Wirkungsablauf, der das Ziel verfolgt, eine bestimmte physikalische Größe auf einen gewünschten Wert zu bringen und dort zu halten. Dabei nennt man die Größe, die geregelt werden soll, *Regelgröße* und den Wert, auf den geregelt werden soll, *Sollwert*. Der Regelkreis führt fortlaufend die Aufgaben Messen, Vergleichen und Stellen aus. Durch die Rückführung der Ausgangsgröße entsteht eine kreisförmige Struktur. Daraus ergibt sich die Bezeichnung Regelkreis. [8, S. 5 ff.]

In Abbildung 2.2 sieht man einen Standardregelkreis mit negativer Rückkopplung. Ein Regelkreis mit negativer Rückkopplung oder auch Gegenkopplung genannt, kommt immer dann zum Einsatz, wenn ein Gleichgewicht hergestellt werden soll. Die Regelgröße y wird dabei mit dem Sollwert w verglichen. Die Abweichung $e = w - y$ wird Regelabweichung genannt und dem Regler wieder zugeführt. Der Regler ist in der Lage, daraus die nötige Stellgröße u zu berechnen, mit der die Regelstrecke, also der Aktuator, der die zu regelnde physikalische Größe enthält, angesteuert

werden muss, um die Regelabweichung zu minimieren. Der Faktor z ist eine Größe, die nicht gesteuert werden kann und den Prozess nicht zielorientiert beeinflusst. Daher wird diese Größe als Störgröße bezeichnet. [3, S. 2 ff.]

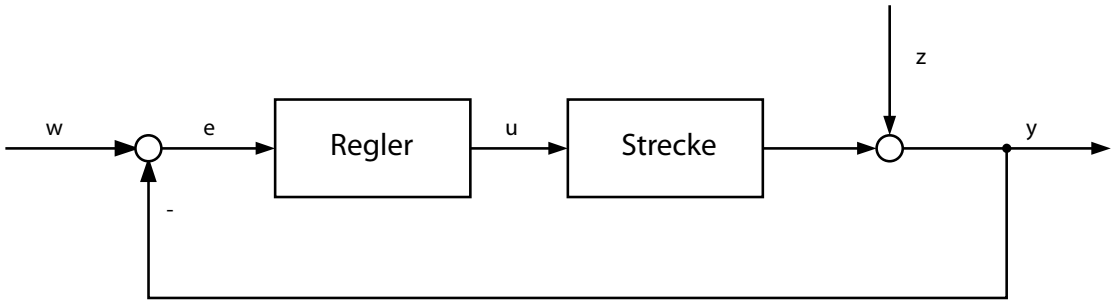


Abbildung 2.2: Standardregelkreises mit negativer Rückkopplung

In einem realen System, wie z.B. der Ansteuerung eine Fahrzeuges, ist der Sollwert die Geschwindigkeit, die gefahren werden soll, die Regelgröße die Geschwindigkeit, die aktuell gefahren wird und die Störgröße eine Vielzahl von Einflüssen, wie Bodenunebenheiten, Wind oder Reibung im System.

2.2.2 Reglertypen

Ein Regler ist ein Teil des technischen Regelkreises und vergleicht fortlaufend den Sollwert mit der Regelgröße. Er bestimmt mit Hilfe der Regelabweichung die Stellgröße, die nach einer gewissen Einstellzeit die Regelabweichung, bezüglich des Sollwertes, minimieren soll. Wie aus der Regelabweichung die Stellgröße berechnet wird, hängt von dem verwendeten Reglertyp ab.

Ein Regler wird durch das Zeitverhalten der Regelstrecke charakterisiert, also mit welchem dynamischen Verhalten sich die Regelgröße auf die Stellgröße einstellt. Dabei unterscheidet man das proportionale (P), das integrale (I) und das differentiale (D) Zeitverhalten. [3, S. 93, 106 ff.]

Im folgenden Abschnitt wird näher auf die gängigsten Reglertypen eingegangen und deren Eigenschaften und Anwendungsmöglichkeiten diskutiert. Anhand dieser

Informationen soll in Kapitel 3.3 der passende Regler für das betrachtete System ausgewählt werden.

P-Regler

Ein P-Regler ist ein Regler mit ausschließlich proportionalem Anteil. Somit existiert eine lineare Abhängigkeit zwischen der Regelgröße und dem Sollwert. Eingangs- und Ausgangsgröße sind proportional. Der Regler gibt nur ein Signal ab, wenn eine Regelabweichung existiert.

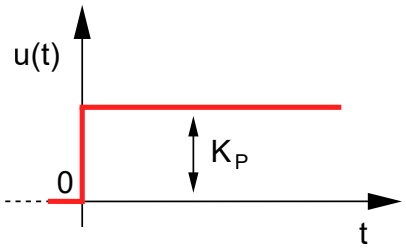


Abbildung 2.3: Sprungantwort eines P-Reglers

Die Gleichung des Reglers sieht man in Formel 2.10. Die Kenngröße ist der Faktor K_P, der als Reglerverstärker bzw. als Reglerproportionalwert bezeichnet wird. Den Einfluss von K_P kann man in der Darstellung der Sprungantwort des Reglers in Abbildung 2.3 sehen. Der Regler multipliziert die Reglerabweichung mit dem Reglerverstärker. Der proportionalwirkende Regler multipliziert die Regelabweichung mit seinem Verstärkungsfaktor K_P und gibt das Ergebnis unverzögert weiter.

$$u(t) = K_P \cdot e(t) \tag{2.10}$$

Aufgrund der Proportionalität ist dieser Reglertyp anfällig für bleibende Regelabweichungen. Der Regler verstellt die Stellgröße immer nur um den Wert der Reglerabweichung, der speziell bei Annäherung an das Ziel sehr klein werden kann. In vielen technischen Prozessen führt das dazu, dass der Sollwert niemals erreicht wird. Dies kann durch ein I-Glied verhindert werden (siehe PI-Regler). [8, S. 144 ff.]

I-Regler

Ein I-Regler ist ein integralwirkender Regler. Er summiert die Reglerabweichung über die Zeit auf und multipliziert die Summe (in Formel 2.11 also das Integral) mit dem Faktor $\frac{1}{T_N}$. T_N steht für die Nachstell- bzw. die Integrierzeit des Reglers und ist dessen Kenngröße. In Abbildung 2.4 sieht man, dass sie den Gradienten des Anstiegs bestimmt. Eine alternative Kenngröße ist $K_I = \frac{1}{T_N}$ und wird als Integrierbeiwert bezeichnet.

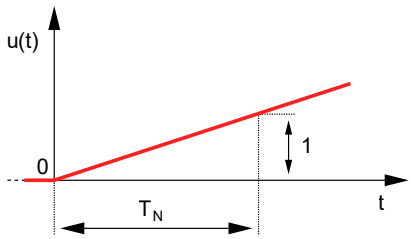


Abbildung 2.4: Sprungantwort eines I-Reglers

$$u(t) = \frac{1}{T_N} \int_0^t e(\tau)\,\mathrm{d}\tau \quad (2.11)$$

Im Vergleich zu der sprunghaften Stellgrößenänderung des P-Reglers reagiert der I-Regler, entsprechend der Reglergleichung 2.11, nur langsam auf eine Änderung der Eingangsgröße. Jedoch verstellt er die Stellgröße so lange, bis die Regeldifferenz zu null geworden ist. Er hinterlässt keine bleibende Regelabweichung. [8, S. 146 f.]

PI-Regler

Ein PI-Regler besteht aus einem P-Glied und einem I-Glied. Er verknüpft somit die Eigenschaften eines P-Reglers und eines I-Reglers. Beim Auftreten einer Regeldifferenz regelt das P-Glied schnell die Stellgröße, während das I-Glied eine fortlaufende Änderung der Stellgröße bewirkt. Der durch das I-Glied erworbene Vorteil der Vermeidung einer konstanten Regelabweichung hat auch den Nachteil, dass der PI-Regler durch das I-Glied zu den langsamen Reglern gezählt wird.

Wie in Abbildung 2.5 zu sehen ist, ordnet das P-Glied der Stellgröße sofort eine sprunghafte Änderung zu. Das I-Glied sorgt im weiteren für einen rampenförmigen Verlauf der Stellgrößenänderung. Er benötigt die Nachstellzeit T_N bis er die Gleiche Änderung der Stellgröße vorgenommen hat, wie sie durch das P-Glied sofort erfolgt ist.

In Formel 2.12 sieht man, dass der Regler zwei Kenngrößen besitzt. Zum einen die aus dem I-Regler bekannte Nachstell- bzw. Integrierzeit T_N und zum anderen den Proportionalwert K_P des P-Reglers. [8, S. 147 f.]

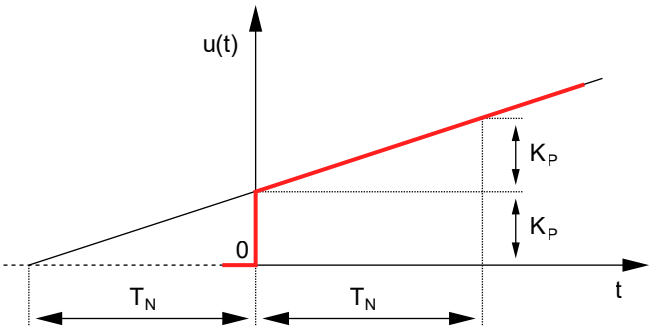


Abbildung 2.5: Sprungantwort eines PI-Reglers

$$u(t) = K_P \left[e(t) + \frac{1}{T_N} \int_0^t e(\tau)\,\mathrm{d}\tau \right] \tag{2.12}$$

PD-Regler

Ein PD-Regler besteht aus einem P-Glied und einem D-Glied. Das D-Glied ist ein Übertragungsglied, das als Differenzierer wirkt. Somit berechnet ein PD-Regler die Stellgrößenänderung mit Hilfe der Regeldifferenz und deren erster Ableitung der Änderungsgeschwindigkeit.

Diese Kombination aus P- und D-Glied sorgt dafür, dass schon bei einer langsamen Änderung der Regeldifferenz eine Reaktion des Reglers ausgelöst wird.

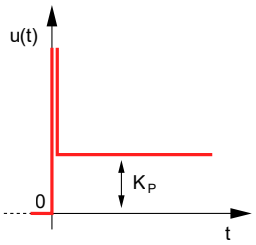


ABBILDUNG 2.6: Sprungantwort eines idealen PD-Reglers

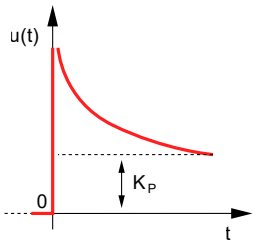


ABBILDUNG 2.7: Sprungantwort eines realen PD-Reglers

$$u(t) = K_P \left[e(t) + T_V \cdot \dot{e}(t) \right] \tag{2.13}$$

In Abbildung 2.6 sieht man die Sprungantwort eines idealen PD-Reglers. Durch die Differenzierung der Reglerabweichung ist der Ausschlag der Steuergröße am Beginn der Sprungfunktion unendlich groß.

Ein Regler mit solchem Verhalten ist in der Realität nicht möglich. Es würde in jedem Fall eine Verzögerung vorliegen, so dass das Sprungantwortverhalten, wie in Abbildung 2.7 dargestellt, aussehen würde.

Die Kenngröße eines PD-Reglers ist neben dem Proportionalwert K_P die Vorhaltzeit T_V. Die Vorhaltzeit stellt ein, wie stark der D-Anteil des Reglers der Regelgrößenänderung entgegenwirkt. Da die Regeldifferenz beim Annähern an den Sollwert kleiner wird, ist auch der Einfluss des D-Übertragungsglieds negativ und dämpft somit die Steuergröße ab. Dies kann ein potentielles Überschwingen des Reglers vermindern. [8, S. 149 f.]

PID-Regler

Ein PID-Regler setzt sich aus den Anteilen des P-Übertragungsglieds, des I-Übertragungsglieds und des D-Übertragungsglieds zusammen. Er verbindet deren positive Eigenschaften. Der PID Regler ist der anpassungsfähigste der Standard-Regler

und kann, wie in Formel 2.14 zu sehen, anhand der drei Parameter K_P, T_N und T_V eingestellt werden. Durch eine geeignete Wahl der Werte dieser Größen lässt sich das Zeitverhalten des Reglers an das der Regelstrecke anpassen, um das gewünschte Verhalten zu erreichen.

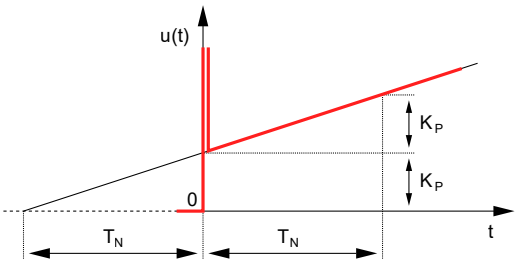


ABBILDUNG 2.8: Sprungantwort eines idealen PID-Reglers

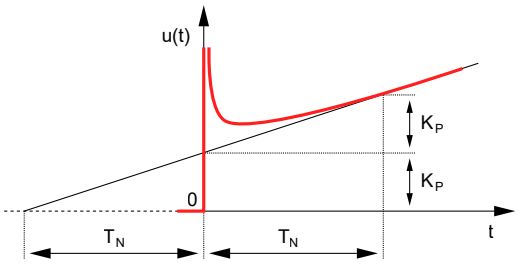


ABBILDUNG 2.9: Sprungantwort eines realen PID-Reglers

$$u(t) = K_P \left[e(t) + \frac{1}{T_N} \int_0^t e(\tau)\,\mathrm{d}\tau + T_V \cdot \dot{e}(t) \right] \tag{2.14}$$

In den Abbildungen 2.8 und 2.9 ist das Zeitverhalten des PID-Reglers zu sehen. Zunächst wirkt der D-Anteil, der jedoch entsprechend der Definition der Sprungfunktion für $t > 0$ wieder verschwindet. Anschließend überlagern sich die Anteile des P- und des I-Übertragungsglieds. Dies führt dann schließlich zur typischen Anstiegsfunktion. [8, S. 152 ff.]

2.2.3 Das Robot Operating System

Das Robot Operating System (ROS) ist eine Sammlung von verschiedenen Software-Frameworks zur Entwicklung und dem Betrieb von Robotersystemen. Es wurde 2007 an der Stanford Universität im Stanford Artificial Intelligence Laboratory

entwickelt. Heute wird es von der Open Source Robotics Foundation (OSRF) verwaltet und weiterentwickelt. Die aktuelle Release Version von ROS ist Kinetic Kame (siehe Abbdilung 2.10). [6]

Das ROS besitzt betriebssystemähnliche Funktionalitäten und eine Clusterarchitektur. Hauptaugenmerk liegt dabei auf Hardwareabstraktion, Gerätetreiber, Softwarebibliotheken, Visualisierungen, Nachrichtenvermittlung und Paketverwaltungen. [1]

Abbildung 2.10: Logo von ROS Kinetic Kame

Veröffentlicht wurde ROS unter der BSD-Lizenz und ist eng mit der Open-Source-Szene verknüpft. Es wurde während der Entwicklung auf einen guten kollaborativen Arbeitsfluss geachtet. Von ROS gibt es Client-Libaries für C++ und Python.

Es stehen dem Entwickler verschiedene Kontrollstrukturen zur Verfügung. Die Node ist dabei die wichtigste und stellt einen kleinen abgeschlossenen Prozess dar. Zur Steuerung eines ganzen Roboters arbeiten daher viele Nodes in einem Flussgraphen zusammen. Sie können miteinander kommunizieren indem sie Daten Streams mittels Topics aufbauen oder Services nutzen.

Bei einer Kommunikation über Topics, können Nodes als Publisher, Subscriber oder beidem fungieren. Möchte eine Node Daten senden, dann tut sie dies als Publisher auf ein Topic. Jede andere Node kann nun als Subscriber fungieren und diese Daten empfangen. Als Datenstruktur kommen Messages zum Einsatz. Sie definieren den Aufbau einer Nachricht. Eine Message kann aus primitiven Datentypen und weiteren Messages bestehen. Die Kommunikation über Topics wird vor allem für kontinuierliche Datenströme verwendet.

Services hingegen werden häufig für das Aufrufen eigenständiger Prozeduren verwendet, sogenannten Remote Procedure Calls (RPCs). Eine Node kann einen Service zur Verfügung stellen, indem sie diesen unter einem festen Namen bekannt

macht. Eine andere Node kann eine Anfrage an diesen Service stellen, indem sie eine Request-Message sendet. Die Service-Node antwortet darauf mit einer Reply-Message. Diese Message-Toupel aus Request und Reply ist zuvor festgelegt worden und charakterisiert den Service.

Das zentrale Element in jeder ROS-Architektur ist der Master. Er sorgt für die Registrierung der einzelnen Nodes und dient der Beantwortung von Anfragen zur Namensauflösung. Nach der Verbindung der Nodes über den Master kommunizieren diese peer-to-peer. Weiterhin dient der Master als Parameter- und Zeitserver für die Nodes.

3 Entwurf

3.1 Komponentenübersicht

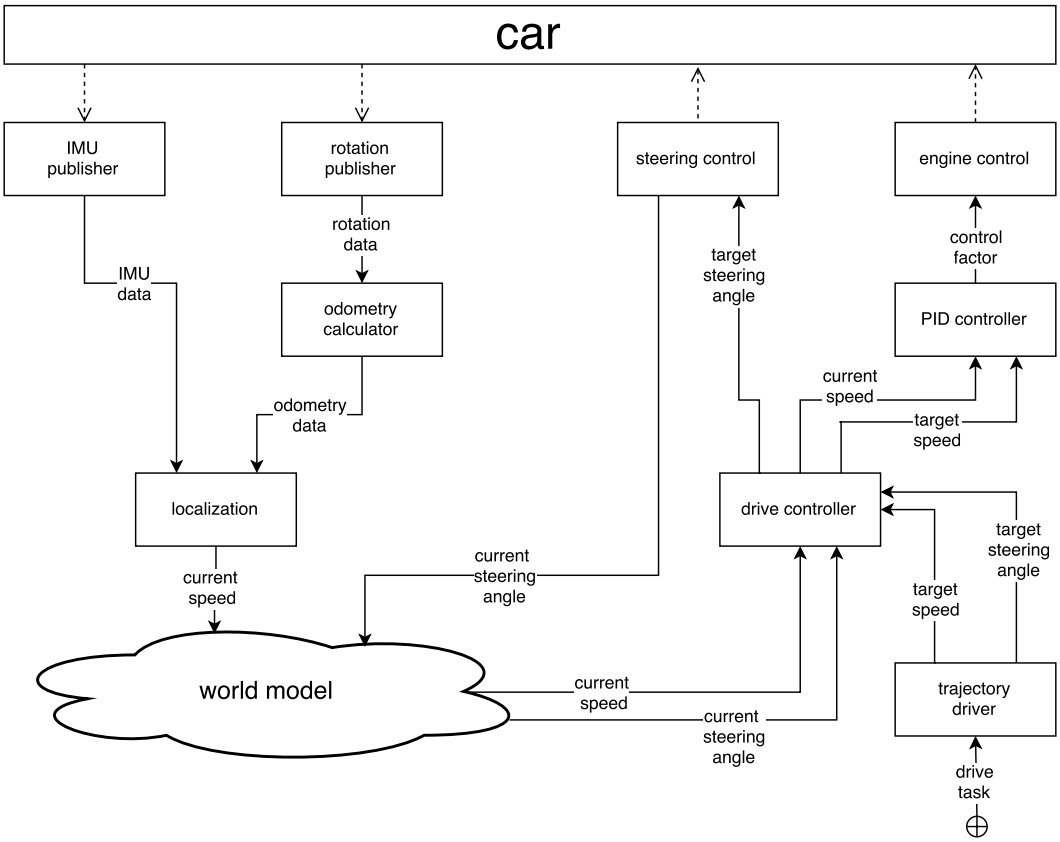


Abbildung 3.1: Übersicht über die Komponenten der Fahrzeugsteuerung

In Abbildung 3.1 sieht man eine Übersicht über die einzelnen Komponenten, die für die Ansteuerung des Fahrzeuges zusammenarbeiten. Die Kommunikation mit den

Sensoren und den Aktuatoren erfolgt mittels seriellen Verbindungen und Arduino Micro Boards. Diese Kommunikation ist in Abbildung 3.1 durch die gestrichelten Linien zwischen dem Fahrzeug und den jeweiligen Nodes dargestellt.

Informationen über die aktuelle Bewegung und Position des Fahrzeugs werden durch die inertial measurement unit (IMU, deut. Inertiale Messeinheit) und den Drehzahlmesser generiert. Diese Daten werden von den Arduinos aufgenommen und mit Hilfe von seriellen Verbindungen an das ROS weitergegeben.

Nach der Vorverarbeitung der Rotationsdaten des Drehzahlmessers werden die Daten der IMU und der berechneten Odometrie mit Hilfe eines Kalmanfilters fusioniert. Eine detaillierte Beschreibung dieser Vorgänge findet man im Abschnitt 4.2 dieser Arbeit.

Die Informationen über die Pose des Fahrzeugs und seiner Geschwindigkeit werden den anderen Komponenten zur Verfügung gestellt. Da es zum Zeitpunkt der Arbeit noch kein Weltmodell für das Fahrzeug gibt, werden die Daten in einem globalen Koordinatensystem weitergegeben.

Der drive controller nutzt diese Informationen, um die Steuerbefehle für die Aktuatoren zu berechnen. Er kann von jedem anderen Teil der Fahrzeuglogik dazu genutzt werden, um Steuerbefehle an die Aktuatoren zu senden.

Der trajectory driver ist ein zusätzliches Modul, das mit Hilfe des drive controllers Trajektorien berechnen und fahren kann. Näheres findet man dazu im folgenden Abschnitt 3.2.

Der drive controller gibt direkt die Steuerbefehle für die Lenkung an den jeweiligen Arduino, der dann den Servo ansteuert. Weiterhin sorgt der drive controller für die Steuerung der Geschwindigkeit. Das geschieht mit Hilfe eines Reglers. Dieser erhält die aktuelle Geschwindigkeit sowie die Zielgeschwindigkeit und berechnet daraus die Stellgröße für die Ansteuerung des Motors.

3.2 Trajektorienplanung

Das Planen und Fahren von Trajektorien ist eine der Teilaufgaben welche am Anfang im Abschnitt 1.2 definiert worden sind. Im Folgenden ist zunächst die Schnittstelle erklärt, über die eine Fahraufgabe an die Trajektorienberechnung übermittelt werden kann. Anschließend werden verschiedene Komplexitätsgrade beschrieben, die eine solche Fahraufgabe haben kann. Danach wird die Berechnung einer Trajektorie des Komplexitätsgrades I erläutert.

3.2.1 Schnittstellendefinition

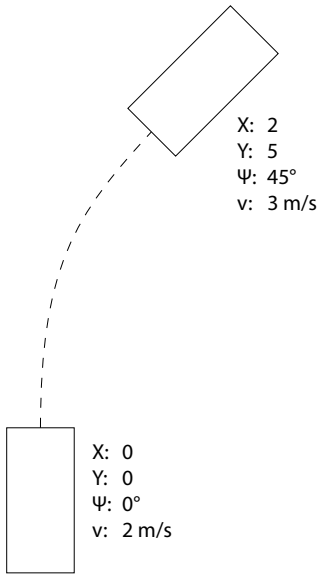


Abbildung 3.2: Visualisierung der Schnittstelle Fahraufgabe

Der Begriff Fahraufgabe steht hier für die Schnittstelle zwischen der Fahrzeuglogik (Bildverarbeitung etc.) und der Ansteuerung des Fahrzeuges. Es wird ein Interface definiert, das mit Hilfe von vier Werten die eindeutige Änderung der Pose und Geschwindigkeit des Fahrzeuges beschreibt. Zu diesen Werten zählen die X-

und Y-Werte des Zielpunktes in kartesischen Koordinaten, die Orientierung des Fahrzeuges als Yaw-Winkel Ψ, und die Geschwindigkeit v in diesem Punkt.

In Abbildung 3.2 sieht man die Fahraufgabe eines Fahrzeuges, das eine langgezogene 45-Grad-Kurve ausführt. Das Fahrzeug verschiebt sich dabei um die Werte $X = 2, Y = 5$ und beschleunigt um $1m/s$ von $2m/s$ auf $3m/s$.

3.2.2 Komplexität der Fahraufgabe

Wie im echten Straßenverkehr unterscheidet sich die Komplexität der Ausführung einer Positionsänderung eines Fahrzeuges zum Teil enorm. Aus diesem Grund werden im folgenden die Komplexitätsgrade I, II und III einer Fahraufgabe definiert und erläutert.

Komplexitätsgrad I

Der Komplexitätsgrad I ist der einfachste der drei Komplexitätsgrade. Er zeichnet sich durch einfache Manöver aus. Die Trajektorie setzt sich ausschließlich aus Geraden und Kreisbahnen am Anfangs- und Endpunkt zusammen. Dabei ist die Änderung des Orientierungswinkels auf $\Delta\Psi < 180°$ beschränkt. Eine weitere notwendige Bedingung ist, dass die Radien der Kreisbahnen für den direkten Weg zum Punkt größer als der minimale Wendekreis des Fahrzeuges sind. Das heißt, das Fahrzeug kann immer auf dem direkten Weg den Punkt erreichen und muss z.B. nicht in die entgegengesetzte Richtung ausholen. Weiterhin muss bei der Planung der Trajektorie nicht auf mögliche Hindernisse geachtet werden. Ein Wechsel der Fahrtrichtung während der Fahraufgabe ist nicht möglich.

Mögliche Manöver für diesen Komplexitätsgrad sind das Abbiegen an einer Kreuzung, das Fahren einer Kurve oder das Wechseln der Spur, um ein langsamer fahrendes Fahrzeug zu überholen. In Abbildung 3.3 sind drei mögliche Fahraufgaben zu sehen. Ausgangspunkt ist jeweils das hervorgehobene Fahrzeug im unteren Bildteil.

Komplexitätsgrad II

Der Komplexitätsgrad II ist aus Sicht der Trajektorienplanung deutlich komplexer als Grad I. Die Trajektorie setzt sich ebenfalls aus Geraden und Kreisbahnen zusammen. Kreisbahnen können in jedem Abschnitt des Pfades vorkommen und sind nicht auf Anfangs- und Endpunkt beschränkt. Die Änderung des Orientierungswinkels Ψ unterliegt keinen Beschränkungen. Hindernisse werden auch hier vernachlässigt und fließen nicht in die Trajektorienplanung ein. In Abbildung 3.4 sieht man ein mögliches Manöver des Komplexitätsgrads II. Da der Wendekreis des Fahrzeuges (gepunktete Linie) zu groß ist, um direkt an die Zielposition zu fahren und da das Fahrzeug bei einer Rechtskurve falsch herum stehen würde, muss es nach links fahren und sich dem Ziel von hinten näheren. Ein Wechsel der Fahrtrichtung während der Fahraufgabe ist in diesem Komplexitätsgrad möglich.

Ein mögliches Manöver des Grads II wäre das Wenden des Fahrzeuges oder das Parken ohne Hindernisse. Eine Fahraufgabe des Grads II kann immer in mehrere Aufgaben des Grads I geteilt und nacheinander ausgeführt werden.

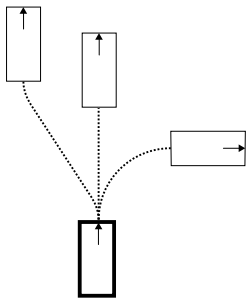


Abbildung 3.3: Mögliche Fahraufgaben Komplexitätsgrad I

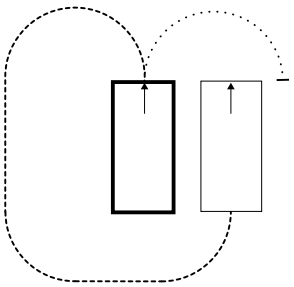


Abbildung 3.4: Mögliche Fahraufgaben Komplexitätsgrad II

Komplexitätsgrad III

Der Komplexitätsgrad III ist ein Spezialfall. Er unterscheidet sich zu dem Grad II nur dadurch, dass bei der Planung des Fahrwegs Hindernisse berücksichtigt werden und diese umfahren werden müssen. Es handelt sich in dieser Kategorisierung

um einen Spezialfall, da es je nach Softwarearchitektur des Fahrzeugs nicht nötig und auch nicht gewünscht ist, dass sich die Trajektorienplanung damit beschäftigt Hindernisse zu erkennen und zu umfahren. Oftmals geschieht eine ausgeprägtere Pfadplanung in anderen Modulen des autonomen Robotersystems. Dabei würde vor der Übergabe der Fahraufgabe an den drive controller bereits darauf geachtet werden, dass der Fahrweg frei von Behinderungen ist. Ist dies der Fall, so ist eine sichere Navigation durch eine Hindernislandschaft mit Fahraufgaben des Grads I bzw. II problemlos möglich.

Mögliche Manöver für den Komplexitätsgrad III ist das Einparken zwischen anderen Fahrzeugen oder das selbstständige Umfahren von Hindernissen.

3.2.3 Berechnung der Trajektorie

Um eine Fahraufgabe umzusetzen, muss das Fahrzeug selbstständig einen Weg zwischen Startpunkt und Zielpunkt berechnen, der von dem Fahrzeug auch gefahren werden kann.

Die zu fahrende Trajektorie setzt sich aus Kreisbahnen mit gleichem Radius und Geraden zusammen. In Abbildung 3.5 sieht man eine beispielhafte Fahraufgabe. Das Fahrzeug soll von Punkt A im Koordinatenursprung zu Punkt $B = (-7|5)$ fahren. Die Orientierung soll sich dabei nicht ändern. Solch eine Trajektorie wäre z.B. bei einem Spurwechsel notwendig.

In der Abbildung sieht man die möglichen Kreisbahnen die mit einem bestimmten Radius an Start- und Endpunkt gefahren werden können. Eine notwendige Bedingung ist, dass der Radius nicht kleiner als der Radius des kleinstmöglichen Wendekreises des Fahrzeuges ist. Zur Berechnung der Trajektorie müssen zunächst von den vier Kreisen, M_1 bis M_4, die zwei bestimmt werden, die für die Trajektorie notwendig sind. Dabei werden verschiedene Parameter betrachtet, wie das Delta aus x_{Start} und x_{Ende} sowie die Entfernung der Kreismittelpunkte M_1, M_2, M_3 und M_4 zueinander. Für die in Abbildung 3.5 dargestellte Fahraufgabe werden die Kreise um M_1 und M_4 benötigt.

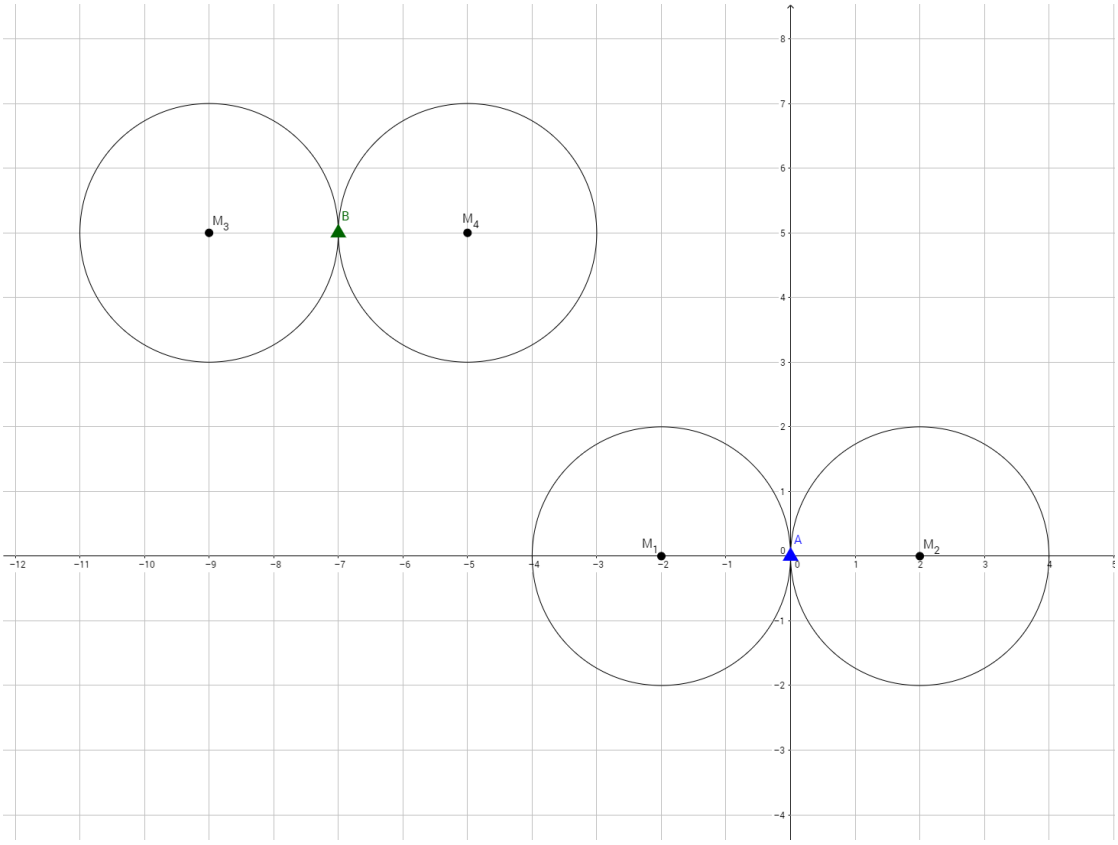


Abbildung 3.5: Wendekreise des Fahrzeuges

Zwischen zwei sich nicht schneidenden Kreisen kann man immer vier Tangenten finden. Dabei unterscheidet man zwischen Außen- und Innentangenten.

In Abbildung 3.6 ist das Beispiel der Spurwechseltrajektorie fortgesetzt. Die Außentangenten sind hier t_1 und t_2. Die Innentangenten sind t_3 und t_4. Die Berechnung der Außentangenten ist im Falle gleich großer Radien der Kreise trivial, da es sich um Parallelen zur Strecke $\overline{M_1 M_4}$ handelt. Die Berechnung der Innentangenten erfolgt über den Schnittpunkt dieser im Punkt D. Dabei handelt es sich gleichzeitig auch um den Mittelpunkt zwischen M_1 und M_4. Vom Ausgangspunkt A auf dem Kreis um M_1 kommen nur die Tangenten t_2 und t_4 als Pfad in Frage. Betrachtet man den Kreis um M_4, sieht man, nur wenn man auf Tangente t_4 fährt, kann der Zielpunkt B mit der korrekten Orientierung erreicht werden. Daher wird diese Tangente als Fahrweg ausgewählt. Die Punkte C und E sind die Wende-

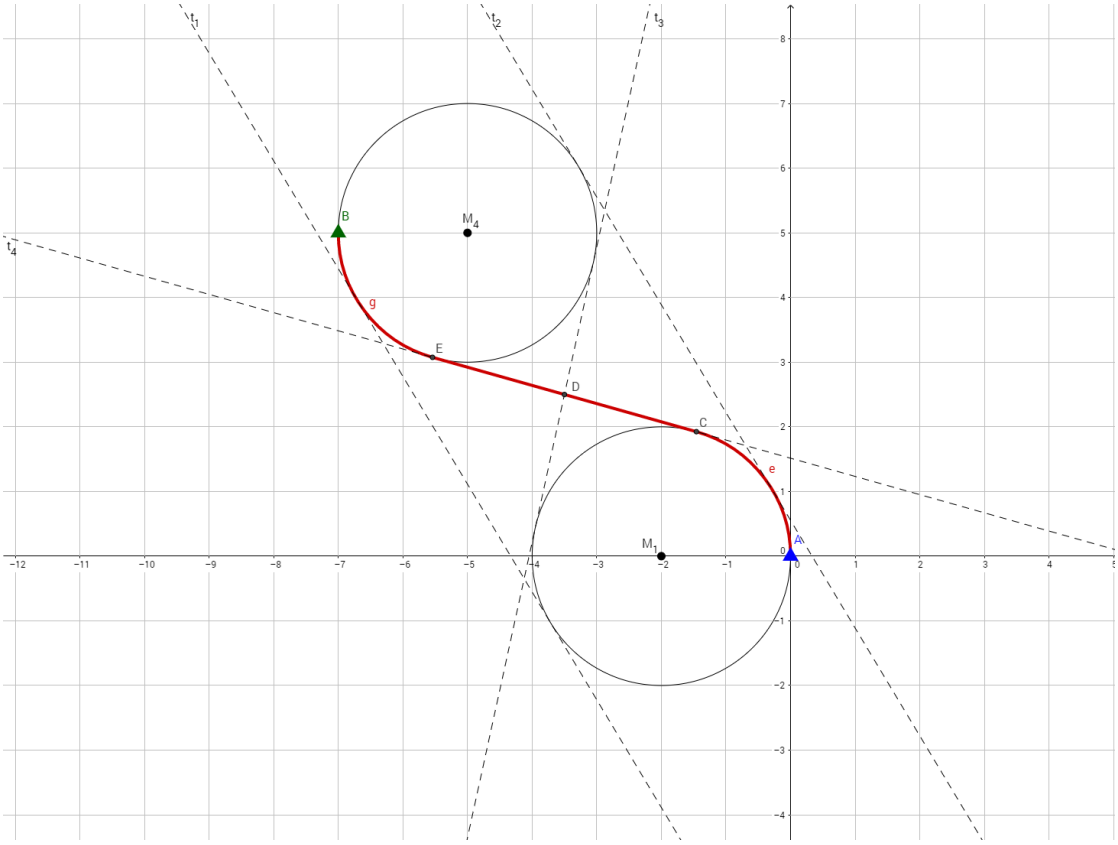


Abbildung 3.6: Berechnung der Trajektorie anhand von Kreistangenten

punkte der Trajektorie und damit wichtige Kennwerte. An diesen Punkten ändert sich der Lenkwinkel. Die Strecke $\overline{CE}$ liegt genau zwischen den Wendepunkten und wird mit einem Lenkwinkel von 0° gefahren.

Wie in Abbildung 3.7 zu sehen ist, kann man mit dem Kurvenradius, dem Mittelpunktwinkel des Startkreises, dem Mittelpunktwinkel des Endkreises und der Strecke dazwischen die Trajektorie vollständig beschreiben. Die Umsetzung dieser Trajektorienberechnung und die Integration in ROS findet man in Abschnitt 4.3.

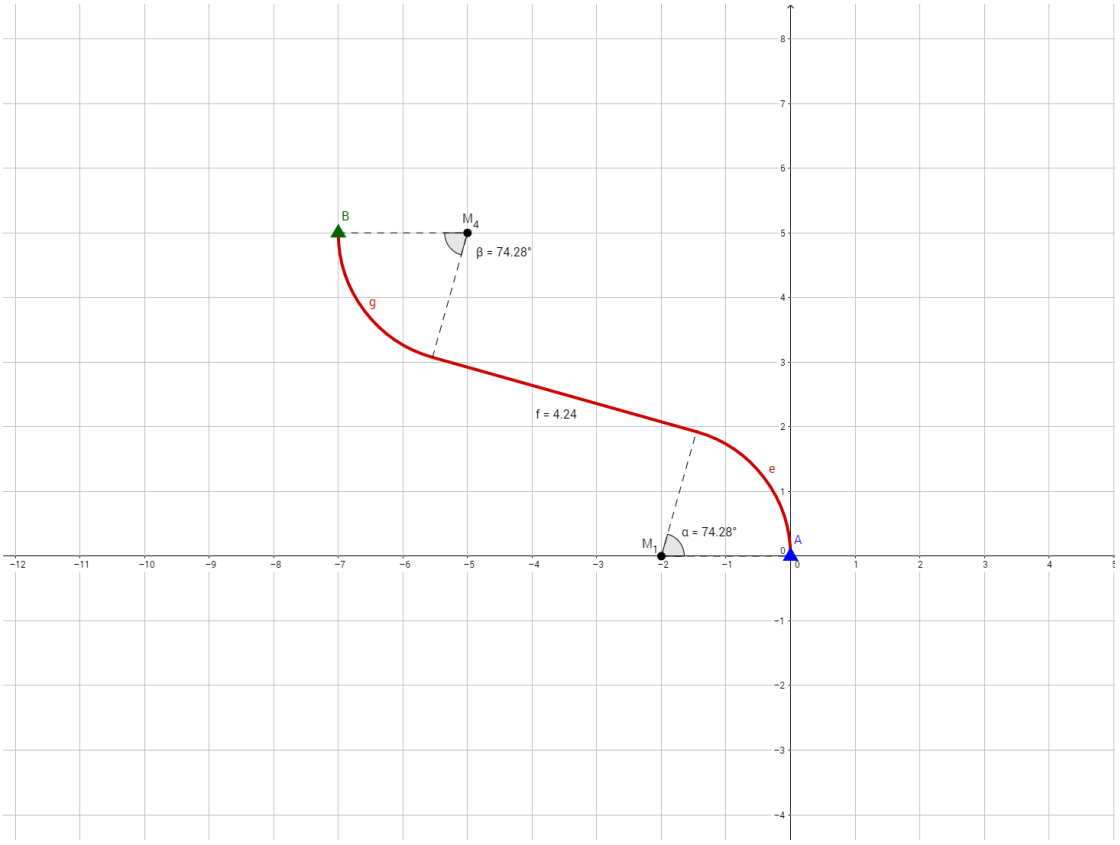


Abbildung 3.7: Die berechnete Trajektorie zwischen zwei Punkten in der Ebene

3.3 Reglerauswahl

Wie in Abschnitt 2.2.2 aufgezeigt, ist der PID-Regler der flexibelste der verglichenen Reglertypen. Er vereint die Vorteile des proportionalen Reglers mit denen des integralen und differentialen. Außerdem kann man durch nullen des jeweiligen Parameters sowohl das I als auch das D Glied flexibel entfernen, ohne den Programmcode zu ändern.

In Abschnitt 4.4 ist die Integration des gewählten Reglerkonzepts in das ROS zu finden. Hier wird auch der Prozess der Parametereinstellung beschrieben.

4 Umsetzung

4.1 Architektur im ROS

Das ROS setzt auf eine untypische Softwarearchitektur, die auf dem Zusammenspiel mehrerer Nodes basiert. Jede Node ist dabei ein für sich selbst agierendes, ausführbares Programm. Nähere Informationen findet man dazu in dem Grundlagenkapitel 2.2.3 über ROS.

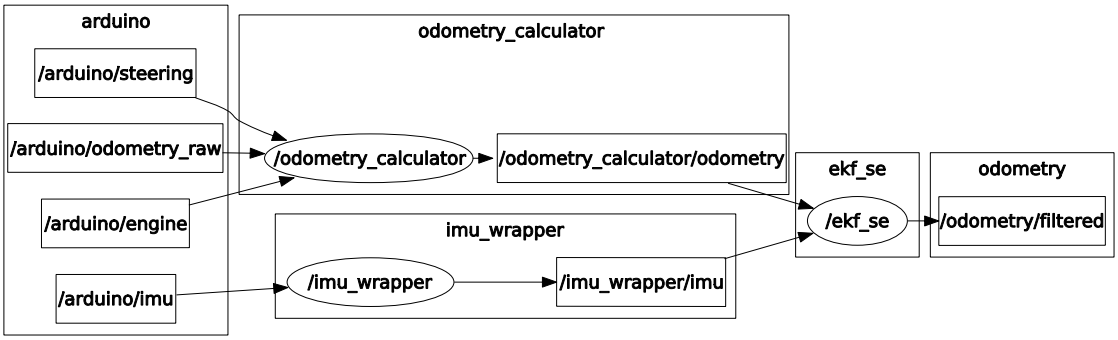


Abbildung 4.1: ROS-Graph von den Sensoren bis zum Kalmanfilter

In den Abbildungen 4.1 und 4.2 sieht man den Datenfluss im ROS. Die Grafiken wurden automatisch mit Hilfe von `rqt_graph` generiert. Dies ist ein in ROS integriertes Programm zur Darstellung von ROS-Architekturen. In Abbildung 4.1 sieht man den Graph von den Sensoren bis zur Sensorfusion im Kalmanfilter. Die Abbildung 4.2 zeigt die anschließende Verarbeitung der Informationen und die Ansteuerung der Aktuatoren. Im Anhang ist eine vollständige Abbildung der Architektur im ROS zu finden.

Jedes Rechteck mit Pfeil zeigt hier eine Kommunikation über Topics an. Der Pfeil zeigt vom Sender zum Empfänger. In den Rechtecken stehen die Namen der Topics, auf die sich die Nodes subscriben können. Die Ovale stellen die Nodes dar.

Nach dem Empfangen der Daten der Sensoren werden diese in den jeweiligen Nodes für Odometrie und IMU vorverarbeitet. Anschließend werden sie fusioniert. Die Sensorfusion geschieht in der Node `ekf_se`. Dabei steht `ekf` für extended kalman filter (deut. erweiterter Kalmanfilter). Genauere Informationen zur Vorverarbeitung und zur Fusion findet man in Abschnitt 4.2 dieses Kapitels.

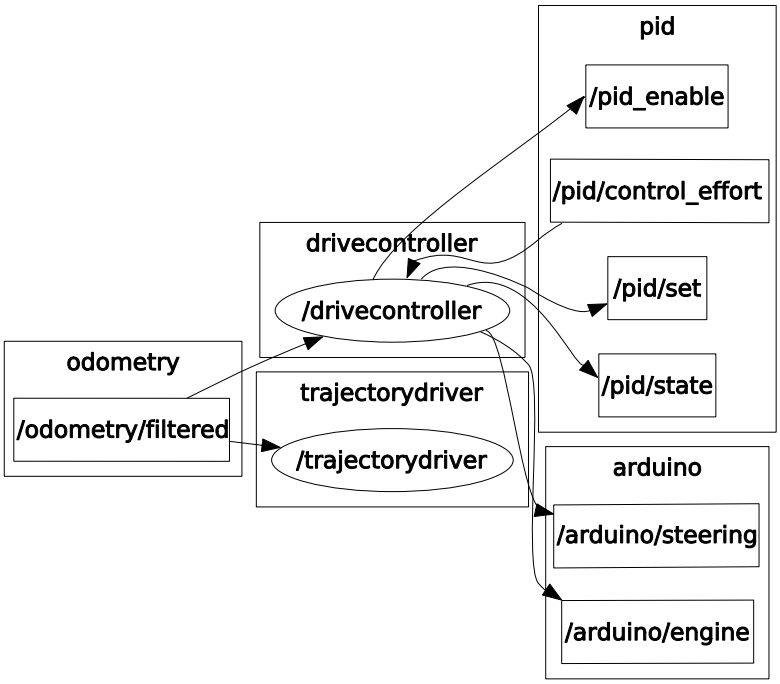


Abbildung 4.2: ROS-Graph vom Kalmanfilter bis zu den Aktuatoren

Das Topic `/odometry/filtered` ist die Schnittstelle zwischen Sensoraufbereitung und Ansteuerung. Darin befinden sich sowohl fusionierte Daten über die absolute Position und Orientierung des Fahrzeuges, als auch dessen Geschwindigkeit.

Die Node `drivecontroller` ist das zentrale Element der Ansteuerung. Es baut Services auf, die von anderen Nodes genutzt werden können, um eine Geschwin-

digkeit einzustellen oder einen bestimmten Winkel oder Kreisradius zu fahren. Die `trajectorydriver` Node nutzt ebenfalls einen Service, um Fahraufgaben von anderen Teilen der Fahrzeuglogik oder von der Kommandozeile des Fahrzeuges zu empfangen und die nötigen Steuerbefehle zu berechnen. Die Ausführung erfolgt über die Services `setSpeed` und `setSteering` des `drivecontroller`. Die Kommunikation mittels Services wird von `rqt_graph` nicht dargestellt.

Während der `drivecontroller` die Befehle zur Lenkung des Fahrzeuges direkt an den jeweiligen Arduino weitergibt, werden die Befehle zur Geschwindigkeitsregelung zunächst an den PID-Regler übermittelt. Dieser baut verschiedene Topics auf. Mit Hilfe von `/pid_enable` kann der Regler ab- oder angeschaltet werden. Der Sollwert und die Regelgröße werden über `/pid/set` und `/pid/state` gesetzt. Hinter `/pid/control_effort` verbirgt sich die Steuergröße. Diese bestimmt letztlich den Wert, der an den Arduino zur Motorsteuerung, übergeben wird. Weitere Informationen zu den Aktuatoren gibt es in diesem Kapitel in Abschnitt 4.5.

4.2 Sensoraufbereitung

4.2.1 Inertiale Messeinheit

Bei der IMU handelt es sich um eine MPU-6050 des Herstellers InvenSense, die fest mit dem Chassis des Fahrzeuges verbaut worden ist. Das Gerät kann sechs degrees of freedom (DoF, deut. Freiheitsgrade) aufzeichnen und verarbeiten. Dazu besitzt das Modul einen Gyroskop-Sensor, der die Orientierung des Fahrzeuges um die X, Y, Z Achse aufzeichnen kann. Im Kontext eines Lagesensors spricht man hier von dem Roll-Nick-Gier-Winkel (engl. roll-pitch-yaw-angle). Zusätzlich zeichnet die MPU-6050 die Beschleunigung des Fahrzeuges in X, Y und Z Richtung auf. Neben diesen Sensoren besitzt das Modul einen zusätzlichen Prozessor, der von dem Hersteller InvenSense als Digital Motion Processor (DMP) beworben wird. Dieser ist dazu in der Lage, verschiedene bewegungsverarbeitende Algorithmen bereits auf dem Gerät durchzuführen.

Im Rahmen dieser Arbeit wird der DMP dazu verwendet, um die Rotationswinkel der Orientierung in den Zahlenbereich der Quaternionen zu überführen. Dies hat mehrere Vorteile, die im Abschnitt 2.1.1 dieser Arbeit besprochen werden.

Zum Abgreifen der Daten wird ein Arduino Micro Board verwendet. Dieses kommuniziert über den Inter-Integrated Circuit (I^2C) Standard mit der IMU. Zur Kommunikation kommt die C++ Implementierung der I2Cdevlib-MPU6050 von Jeff Rowberg [7] zum Einsatz.

Der Arduino wird wiederum über das ROS-Paket rosserial mit dem zentralen Rechner des Fahrzeuges verbunden. Rosserial stellt Funktionen zur Verfügung um die ROS-Kommunikation mit Hilfe serieller Schnittstellen auf andere Geräte, wie Arduinos, zu erweitern.

Quellcode 4.1: Die Message `imu_stamped` des ROS-Pakets `htwk_smart_driving`

```
1 std_msgs/Header header
2 geometry_msgs/Quaternion orientation
3 geometry_msgs/Vector3 linear_acceleration
```

Im Codeauszug 4.1 sieht man die verwendete Message. Sie beinhaltet, neben den ROS-Standard Messages für Quaternionen und dreidimensionalen Vektoren, auch einen Header. Dieser besteht aus einer Sequenznummer, einem Zeitstempel und einer Frame ID. Die Sequenznummer und der Zeitstempel werden von dem benutzten Arduino Micro bei jedem Versenden der Message aktualisiert. Die Frame ID ist in diesem Fall "$\backslash imu$". Sie gibt an, in welchem Kontext die aufgezeichneten Werte gesehen werden müssen, und ob sie bereits einem, der in den ROS Enhancement Proposals(REP, deut. ROS Verbesserungsvorschläge) 105 definierten Koordinatensystemen, zugeordnet werden können.

Die Daten der IMU werden in einer Wrapper-Node in das ROS-Standardformat `sensor_msgs/Imu` gebracht. Dies ist für die verwendeten Pakete der Sensorfusion notwendig. Aufgrund der hohen Speicheranforderung konnte dieses Format nicht direkt auf dem stark speicherlimitierten Arduino Micro eingesetzt werden.

4.2.2 Drehzahlmesser

Bei dem verwendeten Drehzahlmesser handelt es sich um einen Halleffekt Absolutwertgeber Serie MAB25 des Herstellers Megatron. Er deckt einen Winkelbereich von 360° mit einer Auflösung von 12 Bit ab. Der Sensor liefert mit einer maximalen Frequenz von $10000Hz$, Angaben über den Drehwinkel der Welle. In unserem System wurde der Drehzahlmesser mit dem Hauptzahnrad des Motors verbunden. Ein Arduino nimmt die Angaben zum Drehwinkel auf und sendet sie mittels rosserial an den zentralen Rechner des Fahrzeuges.

Auf dem Rechner erfolgt die Verarbeitung des Drehwinkels. Anhand von Messungen konnte festgestellt werden, dass eine Umdrehung der Welle rund $7,51cm$ gefahrener Strecke entspricht. Daraus kann, zusammen mit der Zeit zwischen den einzelnen Messungen, die aktuelle Geschwindigkeit in Fahrtrichtung berechnet werden.

Im Gegensatz zu den sonst üblichen Raddrehzahlmessern, liefert ein Drehzahlmesser am Hauptzahnrad Informationen über die Rotation aller Räder gleichzeitig. Durch das verbaute Differenzialgetriebe kann keine Aussage über die Rotation einzelner Räder getroffen werden. Erst durch die Rückführung des aktuellen Lenkwinkels und die aktuelle Motoransteuerung ist es möglich, Daten über die Positionsänderung des Fahrzeuges zu generieren.

4.2.3 Fusion und Berechnung des Systemzustands

Die Sensorfusion geschieht mit Hilfe eines Kalman-Filters. Die Arbeitsweise dieses Filters wird in Abschnitt 2.1.2 beschrieben. Als Implementierung kommt das ROS-Package robot-localization zum Einsatz. [5]

In die Berechnung des Systemzustands fließen die Daten der Orientierung und Beschleunigung der IMU sowie die Geschwindigkeitsdaten des Drehzahlmesser. Daraus werden in Echtzeit fusionierte Informationen über Position, Geschwindigkeit, Orientierung und Orientierungsänderung generiert. Die Sensorfusion arbeitet

mit einer Taktrate von 60Hz. Dies ist ein Kompromiss aus Rechenaufwand und Genauigkeit.

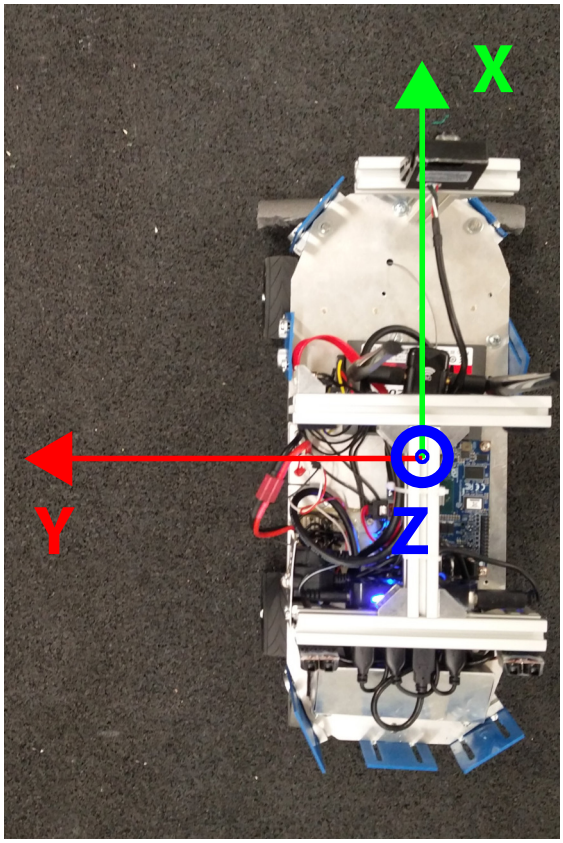


Abbildung 4.3: Das verwendete Koordinatensystem mit dem Fahrzeug im Ursprung

In Abbildung 4.3 sieht man eine Visualisierung des Koordinatensystems des Fahrzeuges zum Zeitpunkt null, also dem Starten des Fahrzeuges und damit der Sensorfusion. Das Fahrzeug hat keine Informationen über andere Bezugssysteme wie z.B. Geokoordinaten der Erde. Daher startet das Fahrzeug immer im Koordinatenursprung dieses Weltkoordinatensystems. Dabei hat es die Orientierung $\Psi = 0°$. Es handelt sich bei Ψ um die Rotation um die Z-Achse, die auch als Gierwinkel (engl. Yaw) bezeichnet wird. Alle anderen Rotationsachsen werden nicht betrachtet, da sie bei einem Fahrzeug auf der Straße vernachlässigt werden können. Genau so verhält es sich auch mit der Beschleunigung. Diese wird in Z-Richtung konstant auf null gesetzt, da sie keinen Mehrwert für die Berechnung der Position des Fahrzeuges in der X-Y-Ebene bietet.

4.3 Fahren der Trajektorie

Die Berechnung der Trajektorie, wie sie in 3.2.3 beschrieben ist, liefert drei Werte zurück, die die Trajektorie von Start- bis Endpunkt vollständig beschreibt. Bei diesen Werten handelt es sich um die Orientierungsänderung am Startpunkt, am Endpunkt und die Länge des geraden Streckenabschnitts in der Mitte.

Quellcode 4.2: Die Methode zum Fahren von Geraden mittels der `trajectorydriver`-Node

```
1 bool straight(double distance, HTWKPoint startPoint) {
2     setSteering(0, false);
3     HTWKPoint currentPosition(currentOdoMsg.pose.pose.position.x,
4                               currentOdoMsg.pose.pose.position.y);
5
6     return startPoint.dist(currentPosition) < distance;
7 }
```

In den Codeauszügen 4.2 und 4.3 sieht man die Methoden zum Fahren von Kurven und Geraden. Sie sind vom Typ Bool und werden solange aufgerufen, bis sie jeweils `false` zurückliefern. In der Methode `straight` geschieht das in Zeile 6 sobald das Fahrzeug die entsprechende Distanz zurückgelegt hat.

In der Methode `turn` wird zunächst zwischen einer Kurve nach links und einer Kurve nach rechts unterschieden. Anschließend wird die Kurve solange mit einem festen Radius weitergefahren, bis der Ziel-Gierwinkel gleich dem Aktuellen-Gierwinkel ist. Dabei gelten die Winkel als gleich, wenn sie sich um weniger als 5° unterscheiden. Dies ist notwendig, da der Gierwinkel nur Werte zwischen −180° bis +180° annehmen kann. Eine einfache Abfrage, ob der aktuelle Winkel kleiner dem Zielwinkel ist, würde nicht ausreichen, da der Gierwinkel, wenn er z.B. 180° in positiver Drehrichtung überschreitet, wieder bei −180° beginnt.

Quellcode 4.3: Die Methode zum Fahren von Kurven mittels der trajectorydriver-Node

```
1 bool turn(float radius, double curveYaw, double targetYaw) {
2     double currentYaw = HTWKMathUtils::quatToYaw(currentOdoMsg)
3
4     ROS_INFO("currentYaw: %f", currentYaw);
5
6     if (curveYaw > 0) //left
7     {
8         if (HTWKUtils::Equals(currentYaw, targetYaw, 5)
9           && currentYaw > targetYaw) {
10            return false;
11        } else {
12            setSteering(radius, true);
13            return true;
14        }
15    } else if (curveYaw < 0) { //right
16        if (HTWKUtils::Equals(currentYaw, targetYaw, 5)
17          && currentYaw < targetYaw) {
18            return false;
19        } else {
20            setSteering(-radius, true);
21            return true;
22        }
23    } else {
24        return false;
25    }
26 }
```

Im Codeauszug 4.4 ist die Schleife zu sehen, die die `turn` Methode aufruft. Der Aufruf der `straight` Methode funktioniert analog dazu. Innerhalb der Schleife in Zeile 2 wird `ros::spinOnce()` aufgerufen. Dies ist wichtig, da ansonsten die Node blockieren würde, bis der Trajektorienabschnitt abgeschlossen ist. Die Variable `currentYaw` könnte so nicht aktualisiert werden.

Quellcode 4.4: Die Schleife zum Fahren einer Kurve in der `trajectorydriver`-Node

```
1         while (turn(radius, trajectory.startYaw, yawTarget)) {
2             ros::spinOnce();
3             cycleTime.sleep();
4         }
```

Diese Aufrufe geschehen mit einer Frequenz von $60Hz$. Dieser Wert ist in der Variable `cycleTime` zuvor festgelegt worden.

4.4 Geschwindigkeitsregelung

Für eine Geschwindigkeitsregelung muss die aktuelle Geschwindigkeit zurückgeführt werden. Da diese, wie in Abschnitt 4.2.3 beschrieben, jedoch in ihre X- und Y-Komponente aufgespalten ist, muss die korrekte Geschwindigkeit erst berechnet werden. Der Betrag des Geschwindigkeit V aus 4.1 ergibt sich dabei zu der gesuchten Geschwindigkeit.

$$V := \begin{pmatrix} v_x \\ v_y \end{pmatrix} \tag{4.1}$$

Wie in Abschnitt 3.3 beschrieben, wird die Geschwindigkeitsregelung durch einen PID-Regler realisiert. Die Implementation eines solchen Reglers folgt den Formeln, die in dem Grundlagenkapitel 2.2.2 zu den verschiedenen Reglertypen nachzulesen sind. Da bereits eine C++ Implementation eines PID-Reglers als ROS-Package verfügbar ist, wird diese verwendet. [9]

Hauptaufgabe bei der Integration eines PID-Reglers ist es passende Parameter für den Proportional-, den Integral-, und den Differenzialteil zu bestimmen. Eine ver-

breitete Methode diese Parameter zu bestimmen, ist das Verfahren nach Methode von Ziegler und Nichols. Dabei handelt es sich um ein heuristisches Verfahren. Das heißt, das System muss zum Zeitpunkt der Reglerauslegung bereits existieren und nutzbar sein. Das Verfahren liefert Werte für einen Regler, der ein gutes Störverhalten erreicht. Ein Nachteil des Verfahrens ist, dass der so konzipierte Regler zu Schwingungsverhalten neigt. [10, S. 759-768]

Das Verfahren gibt es in zwei Varianten. Zum einen gibt es einen empirischen Ansatz und zum anderen einen mathematischen Ansatz. Der mathematische Ansatz setzt zur Berechnung der Reglerparameter auf die Betrachtung der Wendetangente der Sprungantwort. Aus den daraus ablesbaren Werten für Verstärkung, Nachstellzeit und Vorhaltzeit, können die Parameter des jeweiligen Anteils des PID-Reglers berechnet werden.

Die so ermittelten Parameter führten bei dem untersuchten Modellfahrzeug jedoch zu einem starken Schwingungsverhalten im Geschwindigkeitsbereich zwischen $0m/s - 1m/s$.

Durch den Einsatz der zweiten Variante der Methode von Ziegler und Nichols wurde eine Paramterkombination gefunden, die die Anforderungen an die Geschwindigkeitsregelung, die in 1.2 definiert wurden, erfüllt. Die Einhaltung von Qualitätskriterien wird genauer in Abschnitt 5.1 betrachtet.

Bei der empirischen Methode wird zunächst nur der proportionale Anteil des Reglers soweit erhöht, bis ein deutliches Schwingungsverhalten auftritt. Anschließend werden der Integral- und Differenzialteil soweit erhöht, dass dem Schwingen des Proportionalteils entgegengewirkt wird. [3, S. 220 ff.]

Die daraus gewonnen Parameter können der Tabelle 4.1 entnommen werden.

K_P	T_N	T_V
4,1	6,0	0,8

Tabelle 4.1: Parameter des PID-Reglers

4.5 Aktuatoren

Die Aktuatoren des Fahrzeuges, also der Lenk- und der Motorservo, werden ebenfalls von einem Arduino Micro mittels eines pulsweitenmodulierten Signales angesteuert. Dabei ist eine Pulsweite von 1500 Mikrosekunden die Neutralstellung beider Servos. Sie können in einem Bereich von 1000 bis 2000 Mikrosekunden angesteuert werden.

Bei dem Motorservo bedeutet eine Ansteuerung von 2000 Mikrosekunden eine maximale Beschleunigung nach vorne und eine Ansteuerung von 1000 Mikrosekunden eine maximale Beschleunigung nach hinten. Der Lenkservo lenkt bei Werten > 1500 Mikrosekunden nach rechts und bei Werten < 1500 Mikrosekunden nach links.

Um Befehle an die Aktuatoren zu übermitteln, wurden im ROS die Klassen `SteeringControl` und `EngineControl` implementiert. Diese kapseln die Methoden zum Senden der Befehle an den Arduino. Die Übertragung geschieht mit Hilfe von Topics.

Darüber hinaus enthält die Klasse `SteeringControl` Methoden zum Umrechnen von Lenkwinkel oder Kurvenradius in die benötigte Pulsweite.

5 Validierung

5.1 Fahren einer Geschwindigkeit

Die Implementation eines PID-Reglers hatte das Ziel, präzise eine vorgegebene Geschwindigkeit fahren zu können. Um das korrekte Arbeiten des Reglers zu überprüfen, wurde ein Versuch durchgeführt, bei dem das Fahrzeug eine vorgegebene Strecke mit abgemessener Länge fährt. Aus der Zeit, die das Fahrzeug benötigt, um die Strecke zu fahren und der Länge der Strecke, kann berechnet werden, wie schnell das Fahrzeug in Wirklichkeit fährt. Ein Kurzprotokoll des Versuches folgt dieser Arbeit im Anhang. Diesem ist die genaue Methodik und der Versuchsaufbau zu entnehmen.

Bei dem Versuch wurden zwei verschiedene Geschwindigkeiten getestet. Für jede der Geschwindigkeiten wurden drei Fahrten durchgeführt.

In Tabelle 5.1 kann man die Ergebnisse des Versuches sehen. Dabei ist v_s die Geschwindigkeit auf die geregelt werden soll, v_m ist die gemessene Geschwindigkeit der einzelnen Fahrten und v_r ist der Durchschnitt der Geschwindigkeiten der drei Durchläufe.

v_s in m/s	v_m in m/s			v_r in m/s
	Fahrt 1	Fahrt 2	Fahrt 3	
0,5	0,493	0,492	0,498	0,494
1,3	1,418	1,395	1,420	1,411

Tabelle 5.1: Ergebnisse des Versuchs "Fahren einer Geschwindigkeit"

Es fällt auf, dass bei einer Zielgeschwindigkeit von $0,5m/s$ sehr präzise Werte mit einer Abweichung von $< 0,01m/s$ gefahren werden. Bei einem Sollwert von $1,3m/s$ ist die Abweichung mit $> 0,1m/s$ deutlich größer. Eine Erklärung dafür ist, dass Regler jeden Typs zum Überschwingen neigen. Durch begrenzte Platzverhältnisse hatte das Fahrzeug nur wenige Meter, um seine Zielgeschwindigkeit zu erreichen, bevor der in den Abschnitt der Zeitmessung eingefahren ist. Während bei der niedrigeren Geschwindigkeit die Distanz ausreichte, um nach dem Überschwingen auf den Sollwert zu regeln, war dies bei der höheren Geschwindigkeit möglicherweise nicht der Fall. Eine genauere Fehlerbetrachtung ist ebenfalls im Protokoll im Anhang zu finden.

Die Geschwindigkeitsregelung erfüllt die Qualitätsanforderungen, die in Abschnitt 1.2 gestellt worden sind. Es wurde eine Sensorfusion implementiert, die ausreichend genaue Werte liefert um eine Geschwindigkeitsregelung durchzuführen. Es ist möglich, eine vorgegebene Geschwindigkeit zu fahren.

Weiterhin soll die Geschwindigkeitsreglung den Ansprüchen des normalen Straßenverkehrs genügen. Der beschriebene Versuch gibt bereits Hinweise darauf, dass auch dieses Kriterium erfüllt worden ist. Ein abschließendes Urteil kann darüber jedoch erst nach Auswertung des zweiten durchgeführten Versuches, "Fahren einer Trajektorie", gefällt werden.

5.2 Fahren einer Trajektorie

In einem zweiten Versuch ging es darum, dass das Fahrzeug eine vorgegebene Trajektorie möglichst präzise abfahren soll. Dazu wurden dem Fahrzeug fünf Punkte übergeben, die nacheinander gefahren werden sollten. Der Versuch wurde in 3 Fahrten durchgeführt. Auch zu diesem Versuch findet man ein Kurzprotokoll mit dem Versuchsaufbau im Anhang.

Die Strecke des Versuches bestand aus zwei Kreuzungsfahrten und kurzen Geraden.

Abbildung 5.1: Die Positionen des Fahrzeuges während einer Kurvenfahrt

In Abbildung 5.1 sieht man die verschiedenen Positionen des Fahrzeuges während einer Kurvenfahrt. Bei der Kurve handelt es sich um eine 90° Linkskurve und um die zweite der beiden Kreuzungsfahrten. Man kann deutlich sehen, dass die Kreuzung ohne Berührung der Fahrbahnmarkierung gefahren wird, obwohl die Lokalisierung des Fahrzeuges durch die bereits zurückgelegte Strecke eine gewisse Varianz aufweist.

Insgesamt hat das Fahrzeug während seiner Fahrt eine Strecke von rund 6 Metern zurückgelegt. Am Ende der Strecke hatte das Fahrzeug eine Abweichung von $0,6$ bis $0,8$ Meter gemessen von der Zielposition. Außerdem war eine Abweichung der Orientierung von durchschnittlich 20° zu messen. Die Abweichung der Position des Fahrzeuges passiert fast vollständig auf dem letzten Streckenabschnitt. Zum einen

ist hier die Genauigkeit der Selbstlokalisierung am geringsten, zum anderen lässt sich das Fahrzeug nach Erreichen des letzten Wegpunktes ausrollen. Eine genauere Fehlerbetrachtung ist im Protokoll im Anhang zu finden.

Mit diesen Ergebnissen ist zu erkennen, dass die in Abschnitt 1.2 gestellten Aufgaben erfüllt worden sind. Es wurde eine Schnittstelle implementiert, um Fahraufgaben an die Ansteuerung zu übermitteln. Diese wurde während des Versuches “Fahren einer Trajektorie“ erfolgreich Verwendet. Weiterhin hat das Fahrzeug gezeigt, dass es dazu in der Lage ist, eine Trajektorie selbständig zu berechnen und abzufahren. Der Algorithmus hat selbstständig darauf geachtet, dass die Geschwindigkeit in den Kurven nicht zu groß ist, um Einflüsse durch die Trägheit des Fahrzeuges zu vermeiden.

Die Geschwindigkeitsregelung mittels PID-Regler kam auch in diesem Versuch zum Einsatz. Es wurde gezeigt, dass die Geschwindigkeitsregelung auch in Kurvenfahrten zuverlässig funktioniert.

6 Fazit und Ausblick

Die Erkenntnisse die in dieser Arbeit gewonnen wurden, schließen eine Lücke in der bisherigen Arbeit des Teams HTWK Smart Driving. Das erlangte Wissen um ROS, Sensorfusion, Reglerparametrierung, Trajektorienberechnung und Ansteuerung dient als Grundlage für die zukünftige Arbeit des Teams. Insbesondere fließen diese Erkenntnisse in ein mögliches nächstes Fahrzeug des Teams.

Mit dem in dieser Arbeit betrachteten Fahrzeug konnten Erfolge erzielt werden. Es ist bereits jetzt dazu in der Lage, einem vorher vorgebenden Straßenverlauf eine gewisse Zeit zu folgen. Vorgänge, wie das Abbiegen an einer Kreuzung oder das Fahren einer Kurve sind nun möglich. Es wurde eine Grundlage geschaffen, auf die weitere Teile der Fahrzeuglogik aufbauen können. Dabei können Elemente, wie die Bildverarbeitung, nun auf ein gekapseltes und erprobtes System zur Ansteuerung des Fahrzeuges zurückgreifen. Weiterhin wurden erste Schritte in Richtung eines Weltmodells unternommen, in dem das Fahrzug nun in der Lage ist, sich mit Hilfe der vorhandenen Sensoren selbst zu lokalisieren.

Mögliche zukünftige Verbesserungen der Hard- als auch der Software kann man in jedem der bearbeiteten Teilbereiche finden. Insbesondere im Bereich der Odometrie konnte festgestellt werden, dass ein einzelner Drehzahlmesser am Hauptzahnrad des Getriebes nicht optimal ist, um Odometriedaten zu generieren. Auch wenn das Auslesen der Sensordaten mittels der Arduinos zuverlässig funktioniert, so sind die beschränkten Hardwareressourcen dieser Platinen ein Faktor, der die Genauigkeit der Ansteuerung sowie die Architektur des Systems negativ beeinflusst hat.

Insbesondere der Versuch “Fahren einer Trajektorie“ hat gezeigt, dass neben der Regelung der Geschwindigkeit durch einen Regler auch ein manuelles Bremsen

notwendig ist. Auch wenn das Fahrzeug hardwaretechnisch über keine Bremsanlage verfügt, so ist ein softwaretechnisches Bremsen mittels des Elektromotors denkbar und erstrebenswert.

Letztlich ist es gelungen, ein Modellfahrzeug aus dem Hobbybereich, das durch verschiedene Sensorik und Rechenhardware ergänzt worden ist, um autonome Fahrfunktionen zu erweitern.

Quellcodeverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar

Aktuator technisches Bauteil, das elektrische Signale in Bewegungen oder andere physikalische Größen umsetzt, Gegenstück zum Sensor; siehe Seite 36

Inertiale Messeinheit kombinierte Sensoreinheit aus Beschleunigungssensoren und Drehratensensoren; siehe Seite 28f

Inter-Integrated Circuit serieller Datenbus zur Kommunikation verschiedener Bauteile wie Sensoren oder Aktuatoren

Kalman-Filter Filter zur Zustandsschätzung linearer und nichtlinearer Systeme; siehe Seite 5ff

Odometrie Methode zur Bestimmung der Pose eines Fahrzeuges oder Roboters anhand seiner Antriebselemente; siehe Seite 30f

Pose Verbindung aus Position und Orientierung

Quaternionen Zahlenebreich bestehend aus einelementigen Realteil und dreielementigen Imaginärteil; siehe Seite 4f

Robot Operating System Software-Framework zur Entwicklung und dem Betrieb von Robotersystemen; siehe Seite 14f

Sensor technisches Bauteil, das physikalische Größen messen und in ein weiterverarbeitbares elektrisches Signal umwandeln kann, Gegenstück zum Aktuator; siehe Seite 28ff

Trajektorie Pfad oder Bahn der Bewegung eines Punktes, hier Fahrweg des Fahrzeuges; siehe Seite 22ff

Abkürzungsverzeichnis

HTWK	Hochschule für Technik, Wirtschaft und Kultur
IMU	inertial measurement unit
ROS	Robot Operating System
OSRF	Open Source Robotics Foundation
DoF	degrees of freedom
DMP	Digital Motion Processor
REP	ROS Enhancement Proposals
RPC	Remote Procedure Call
I²C	Inter-Integrated Circuit

Literatur

[1] *About ROS*. abgerufen am 28.08.2016. URL: http://www.ros.org/about-ros/.

[2] Albrecht Beutelspacher. *Lineare Algebra*. Vieweg Friedr. + Sohn Ver, 2006.

[3] H. Jaschek, L. Merz und H. Voos. *Grundkurs der Regelungstechnik: Einführung in die praktischen und theoretischen Methoden*. Oldenbourg, 2010.

[4] Simon J Julier und Jeffrey K Uhlmann. „New extension of the Kalman filter to nonlinear systems“. In: *AeroSense'97*. International Society for Optics und Photonics. 1997.

[5] T. Moore und D. Stouch. „A Generalized Extended Kalman Filter Implementation for the Robot Operating System“. In: *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, 2014.

[6] *Open Source Robotics Foundation | Willow Garage*. abgerufen am 29.08.2016. Apr. 2012. URL: http://www.willowgarage.com/blog/2012/04/16/open-source-robotics-foundation.

[7] Jeff Rowberg. *I2Cdevlib-MPU6050*. https://github.com/jrowberg/i2cdevlib.git. 2011–2016.

[8] Hildebrand Walter. *Grundkurs Regelungstechnik: Grundlagen für Bachelorstudiengänge aller technischen Fachrichtungen und Wirtschaftsingenieure (German Edition)*. Springer Vieweg, 2013.

[9] Andy Zelenak. *PID*. https://bitbucket.org/AndyZe/pid.git. 2015–2016.

[10] J. G. Ziegler und N. B. Nichols. „Optimum Settings for Automatic Controllers“. In: *Transactions of ASME* (1942).

Bildquellen

2.3: Wdwd, https://commons.wikimedia.org/wiki/File:Idealer_P_Sprungantwort.svg (16.10.2016)

2.4: Wdwd, https://commons.wikimedia.org/wiki/File:Idealer_I_Sprungantwort.svg (16.10.2016)

2.5: Wdwd, https://commons.wikimedia.org/wiki/File:Idealer_PI_Sprungantwort.svg (16.10.2016)

2.6: Wdwd, https://commons.wikimedia.org/wiki/File:Idealer_PD_Sprungantwort.svg (16.10.2016)

2.8: Wdwd, https://commons.wikimedia.org/wiki/File:Idealer_PID_Sprungantwort.svg (16.10.2016)

2.10: Wdwd, http://wiki.ros.org/kinetic?action=AttachFile&do=get&target=kinetic.png (16.10.2016)

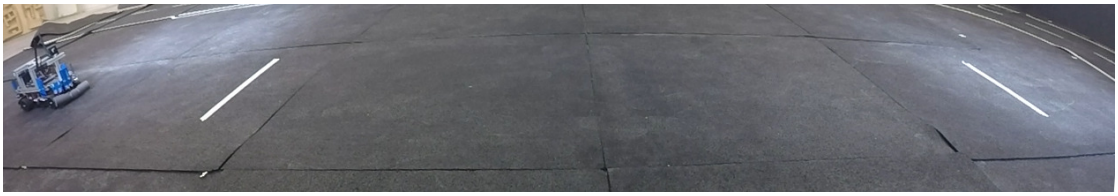
Anhang

Kurzprotokoll Versuch “Fahren einer Geschwindigkeit“

Thema des Versuchs

Der Versuch wird durchgeführt, um zu überprüfen, ob der PID-Regler des Fahrzeuges korrekt arbeitet. Es soll festgestellt werden, ob das Fahrzeug in der Lage ist, eine vorgegebene Geschwindigkeit zu fahren.

Aufbau



Versuchsaufbau “Fahren einer Geschwindigkeit“

Es wurden zwei weiße Linien im Abstand von genau $3m$ mit Klebeband auf den Boden aufgeklebt. Die Startposition des Fahrzeuges ist ca. $1,5m$ vor der ersten Linie. Eine Fischaugenobjektivkamera mit einer Framerate von 60 Bildern pro Sekunde wurde so angebracht, dass sie die komplette Durchfahrt des Fahrzeuges durch den Messabschnitt aufnehmen kann.

Durchführung

Das Fahrzeug besitzt einen Service mit dem man die Geschwindigkeit einstellen kann. Dieser wird mit Hilfe des Kommandozeilenprogrammes `rosservice` aufgerufen und eine Geschwindigkeit eingestellt. Die Lenkung wird in Neutralstellung belassen.

Es werden zwei Durchgänge mit jeweils drei Fahrten ausgeführt. Dabei wird im ersten Durchgang der Sollwert der Geschwindigkeitsregelung auf $0,5m/s$ eingestellt. Bei dem zweiten Durchgang soll eine Geschwindigkeit von $1,3m/s$ gefahren werden. Jede Fahrt wird von der Fischaugenobjektivkamera aufgezeichnet.

Anschließend wird am PC das Videomaterial ausgewertet. Dabei wird die Anzahl der Frames gezählt, von dem Moment des Überfahrens der Startlinie bis zum Überfahren der Ziellinie. Anschließend kann anhand der Framerate der Kamera und der gezählten Frameanzahl die Zeit berechnet werden, die das Fahrzeug benötigt, um die Strecke zurückzulegen. Aus Weg und Zeit kann die gefahrene Geschwindigkeit berechnet werden.

Beobachtungen

		Fahrt 1	
v_s in m/s	Frames	t in s	v_m in m/s
0,5	365	6,0834	0,493
1,3	127	2,1167	1,418
		Fahrt 2	
0,5	366	6,1	0,492
1,3	129	2,15	1,395
		Fahrt 3	
0,5	362	6,034	0,498
1,3	128	2,113	1,420s

Messungen Versuch “Fahren einer Geschwindigkeit“

Die Messdaten und die daraus errechneten Werte kann man obiger Tabelle entnehmen. Dabei ist v_s der Sollwert der Geschwindigkeitsregelung, t die Zeit für die Durchfahrt und v_m die daraus errechnete Geschwindigkeit. Dieser Arbeit ist ein Datenträger beigelegt, der die Videoaufnahmen des Versuches enthält.

Auswertung

Es ist zu erkennen, dass bei den Fahrten mit der niedrigeren Geschwindigkeit von $0,5m/s$ die tatsächliche Geschwindigkeit der angestrebten Geschwindigkeit sehr nah kommt. Hier wurden durchschnittlich Abweichungen von weniger als $0,01m/s$ erreicht. Bei einem Sollwert von $1,3m/s$ ist die Abweichung durchschnittlich größer als $0,1m/s$. Beide Messungen können als Erfolg gewertet werden. Das Fahrzeug ist in der Lage, verschiedene Geschwindigkeiten mit hinreichender Genauigkeit zu fahren. Im Umkehrschluss bedeutet dies, die aktuelle Geschwindigkeit des Fahrzeuges wird von der Sensorfusion zuverlässig ermittelt.

Fehlerbetrachtung

Zahlreiche Störfaktoren sind bei diesem Versuch zu beachten. Die eingesetzten Sensoren sind nicht fehlerfrei und weisen Messungenauigkeiten auf. Weiterhin konnten Start- und Endframe im Video nur geschätzt werden. Durch den Einsatz der Fischaugenobjektivkamera kommt es zu Verzerrungen, so dass nicht zweifelsfrei bestimmt werden kann, wann das Fahrzeug die jeweiligen Linien überquert hat. Außerdem kann es bei dem Abstecken der Testgerade zu Messungenauigkeiten gekommen sein.

Die größeren Abweichungen der Fahrten mit der höheren Geschwindigkeit, können an einem Überschwingen des Reglers liegen. Da aufgrund von beschränkter Platzverhältnisse nur wenige Meter Anlauf zur Verfügung standen, ist es möglich, dass das Fahrzeug seine endgültige Geschwindigkeit noch nicht erreicht hatte.

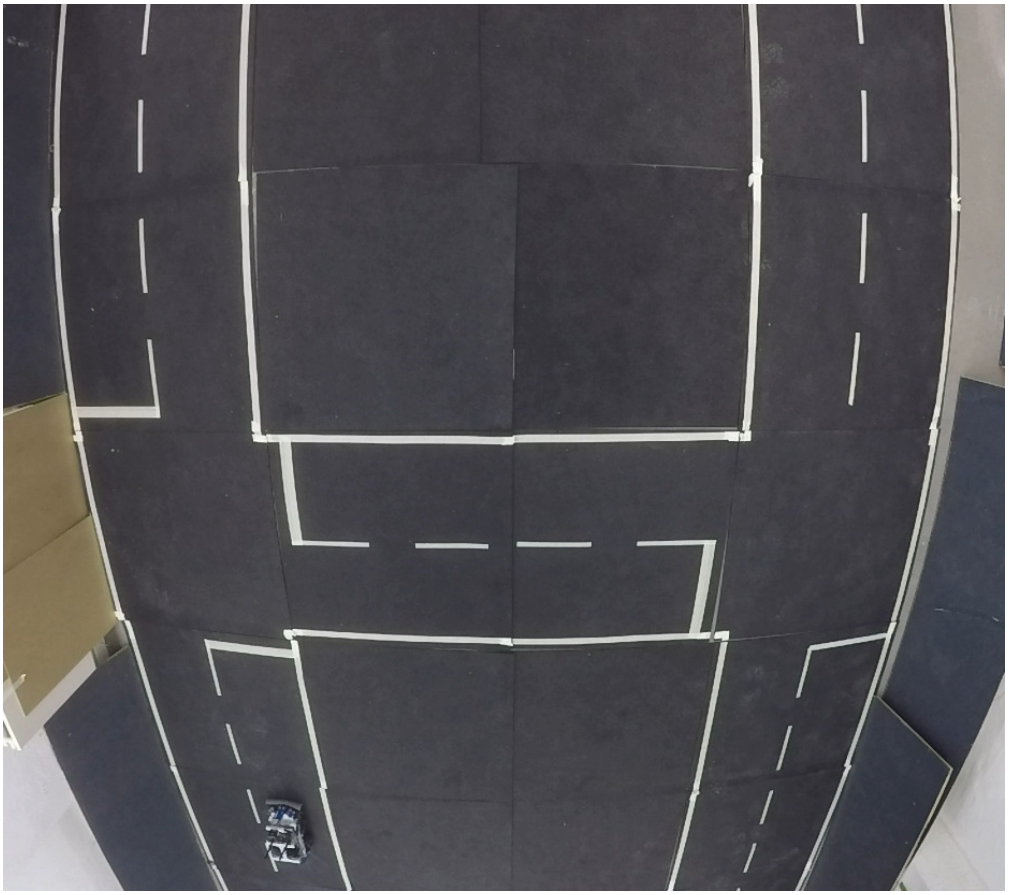
Kurzprotokoll Versuch “Fahren einer Trajektorie“

Thema des Versuchs

In diesem Versuch soll überprüft werden, ob die implementierte Trajektorienberechnung und die Ansteuerung dazu in der Lage sind, das Fahrzeug durch einen stadtähnlichen Parcour zu navigieren.

Aufbau

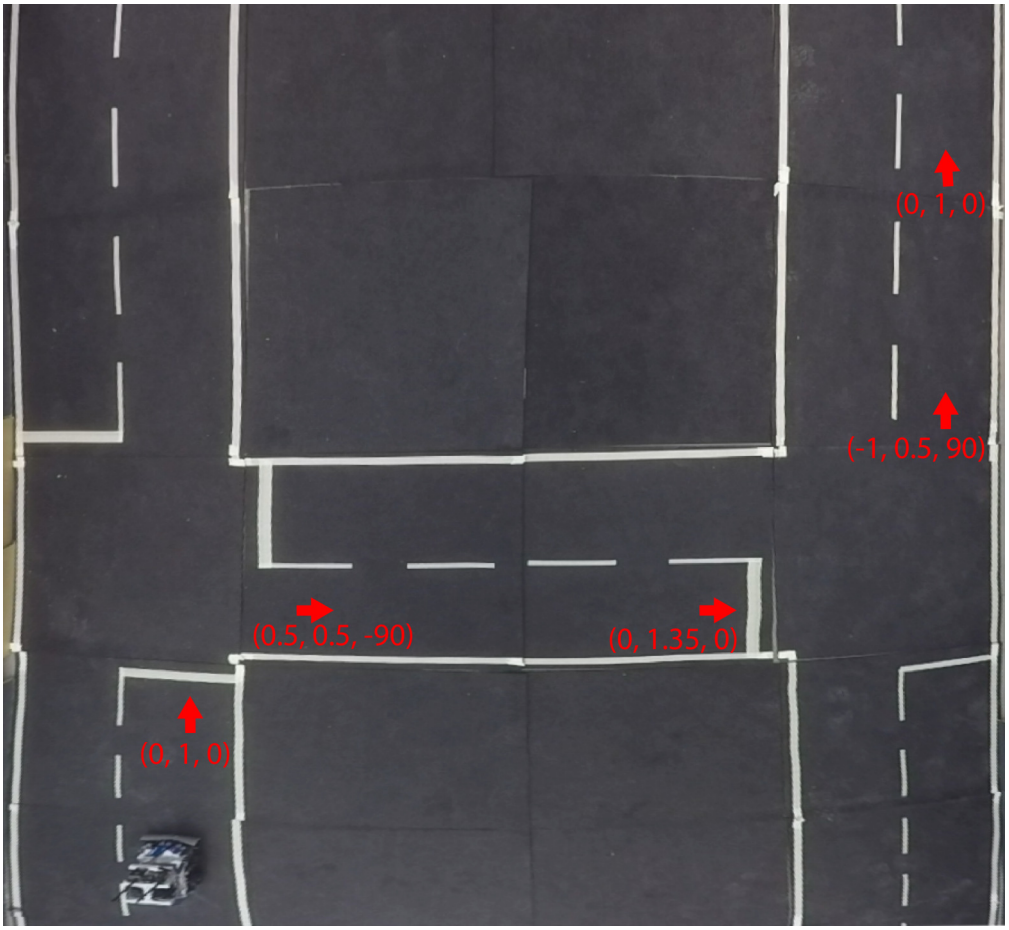
Es wurde auf Gummimatten eine Strecke aufgeklebt, die dem Straßenverlauf einer Stadt ähnelt. Das Fahrzeug wird ca. einen Meter vor einer der beiden Kreuzungen plaziert. Eine Fischaugenobjektivkamera mit einer Framerate von 60 Bildern pro Sekunde wurde an der Decke des Raumes angebracht. Von dort kann die Kamera die gesamte Strecke überblicken.



Versuchsaufbau “Fahren einer Geschwindigkeit“

Durchführung

Das Fahrzeug ist in der Lage, Fahraufgaben über einen Service entgegenzunehmen. Für den Versuch werden 5 dieser Serviceaufrufe hintereinander durchgeführt, so dass das Fahrzeug eine komplexe Fahraufgabe zu bewältigen hat. Das Fahrzeug wird zunächst auf die Strecke gestellt und anschließend wird via SSH auf dem Fahrzeug das Skript gestartet, das die Serviceaufrufe ausführt. Der untenstehenden Grafik können die Punkte entnommen werden. Die Koordinaten sind jeweils relativ zur Position des Fahrzeuges zum Zeitpunkt des Serviceaufrufs. Dabei ist der erste Wert die x-Koordinate, der zweite Wert die y-Koordinate und der dritte Wert die Rotation um die z-Achse.



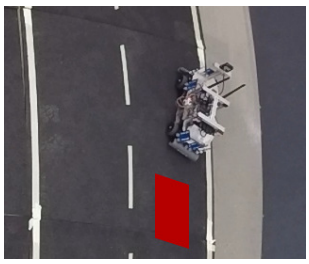
Die zu fahrende Trajektorie

Es werden drei Fahrten der gleichen Trajektorie durchgeführt. Jede Fahrt wird von der Kamera an der Decke aufgezeichnet. Anschließend wird das Videomaterial am PC ausgewertet.

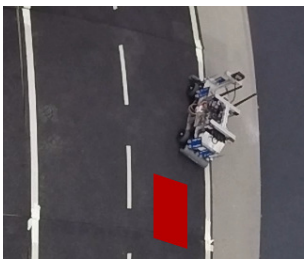
Beobachtungen

In den drei Abbildungen in diesem Abschnitt sieht man die jeweiligen Endpositionen des Fahrzeuges nach den drei Fahrten. In den Abbildungen wurde Rot markiert, an welcher Position das Fahrzeug bei einer perfekten Fahrt zum stehen

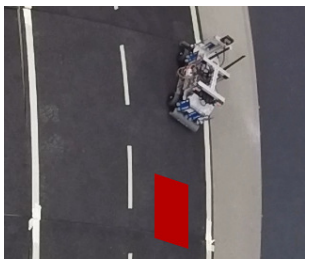
kommen müsste. Zusätzlich sieht man in der Tabelle die Abweichungen von Zielposition und Orientierung der einzelnen Fahrten. Dieser Arbeit ist ein Datenträger beigelegt, der die Videoaufnahmen des Versuches enthält.



Enposition Fahrt 1



Enposition Fahrt 2



Enposition Fahrt 3

Fahrt	x in m	y in m	Ψ in °
1	0,7	0,15	-15
2	0,6	0,3	-25
3	0,8	0,2	-15

Messungen Versuch “Fahren einer Trajektorie“

Auswertung

Die Abweichungen sind gering. Selbst am Ende der Fahrt überschreitet die Abweichung der realen Position von der Soll-Position nicht 15%. Es wurden zwei Abbiegevorgänge durchgeführt, ohne die Fahrbahnmarkierungen zu verletzen. Das Fahrzeug hat gezeigt, dass es auch eine komplexe Trajektorie zuverlässig fahren kann. Weiterhin hat der Versuch gezeigt, dass alle Teilaspekte des Fahrzeuges zuverlässig zusammenarbeiten. Sowohl Selbstlokalisierung, Geschwindigkeitsregelung als auch Trajektorienberechnung arbeiten zuverlässig.

Fehlerbetrachtung

Hauptfehlerursache bei diesem Versuch sind die Messungenauigkeiten der Sensoren. Weiterhin lässt sich auf Grund des Fahralgorithmuses das Fahrzeug bei Erreichen des letzten Wegpunktes ausrollen, ohne zu bremsen.

Vollständige Architekturübersicht im ROS