PG 629: F1/10 - Autonomous Racing: Head-To-Head Racing Capabilities

Autonomous Racing Group
LS 12 TU Dortmund

Robin Thunig, Patrick Schmelter, Marcel Ebbrecht,
Jan-Henrik Bruhn, Jan Peter Meyer,
Timo Gojowczyk, Marvin Langenkämper,
Dr. Kuan-Hsun Chen, Niklas Ueter

September 2020 (1.2)

Inhaltsverzeichnis

T Vergleich - 500 **230**

1 Einleitung

Seit jeher findet ein steter Wandel der Mobilität statt, sei es vor einem gesellschaftlichem, wirtschaftlichem oder pragmatischem Hintergrund. Eine Entwicklung, die in den letzten Jahren aus vielerlei Gründen wachsende Popularität erlangt, ist das autonome Fahren [5]. Neben vielen anderen Aspekten spielt die Sicherheit, sowohl die der Insassen als auch anderer Verkehrsteilnehmer eine übergeordnete Rolle. Um dies gewährleisten zu können, muss, neben einer korrekten Wegfindung und dem Einhalten gemeinsamer Regeln, auch das korrekte Reagieren auf nicht vorhersehbare Situationen realisiert werden.

Die Arbeit der Projektgruppe *Autonomous Racing: Head-To-Head Racing Capabilities* baut auf den Ergebnissen der Projektgruppe *Autonomous Racing* aus dem Wintersemester 2018/2019 und dem Sommersemester 2019 auf. Diese legte die Grundlage für das Fahren auf einer Rennstrecke u.A. durch die Entwicklung eines Wallfollowing-Algorithmus. Im Ziel dieser Projektgruppe war zum einen die Fahrleistung deutlich zu steigern, zum anderen Hindernisse, die sich auf der Route des Kraftfahrzeuges befinden (statisch) und sich unter Umständen bewegen (dynamisch) zu behandeln und dementsprechend die Fahrweise und Trajektorienplanung dynamisch anzupassen. Zudem wurden weitere Verbesserungen vorgenommen. Der Bezug zum Rennsport ist offensichtlich: Befinden sich mehrere Fahrzeuge auf der Strecke, müssen diese als solche erkannt werden, um eine Kollision zu vermeiden und ein Überholen zu ermöglichen. Dies erfordert eine effiziente und schnelle Berechnung unter Ressourcenbeschränkung, da diese in der Regel auf dem Fahrzeug erfolgen muss und aufgrund der zeitlichen Kritikalität nicht auf beliebig leistungsfähige externe Systeme ausgelagert werden kann. Zudem muss auf die unvorhersehbaren Änderungen von Geschwindigkeit und Richtung in möglichst kurzer Zeit reagiert werden.

Darüber hinaus wird beabsichtigt, am Ende der Projektgruppe an der F1TENTH Challenge [23] teilzunehmen, die in diesem Jahr aufgrund der Covid19-Pandemie nicht real, sondern virtuell stattfindet. Dies setzt die Einbindung des F1TENTH Simulators [1] voraus, der für die virtuellen Rennen von der F1TENTH zur Verfügung gestellt wird.
Dieser stellt dem Fahralgorithmus auch Informationen zur Verfügung, die mit dem richtigen Ansatz sehr wirkungsvoll sein können, wie beispielsweise eine vollständige Karte der Umgebung oder präzise Positions- und Geschwindigkeitsdaten. Da unser Ansatz zustandslos agiert, da in der Realität solche Informationen mit dieser präzision und einfachheit nicht zu erreichen sind, nutzen wir diese

[1] `https://github.com/f1tenth/f1tenth_simulator`

Möglichkeit nicht. Daraus ergeben sich unter Umständen Nachteile gegenüber anderen teilnehmenden Gruppen.

Die Möglichkeit an einem Wettbewerb teilnehmen zu können setzt eine auf Geschwindigkeit ausgelegte Hard- und Software voraus. Als Fahrzeug kommt ein Ford Focus im Maßstab 1:10 von Traxxas [30] zum Einsatz. Nach Abschluss der Tests im August 2020 sind wir jedoch zu der Überzeugung gelangt, dass im Rahmen des Projektes eine hinreichend leistungsfähige Plattform realisiert wurde, worauf im Laufe des Berichtes eingegangen wird. Der vorliegende Bericht soll einen Überblick über folgende Punkte geben:

- Einleitend werden ausgewählte Grundlagen (siehe Abschnitt 2) präsentiert, deren Kenntnis und Einsatz für das Erreichen der gesteckten Ziele unerlässlich ist.
- In den folgenden Abschnitten 3 und 4 werden die Ergebnisse der Vorgängergruppe in Auszügen behandelt und die durch diese Projektgruppe geschaffene und betriebene Infrastruktur beschrieben.
- Anschließend (Abschnitt 5, 6, 7 und 9) folgt die Beschreibung des Projektmanagements, der Modellbildung, eingesetzter bzw. entwickelter Software und Erweiterungen und einem verworfenen Teilprojekt.
- In Abschnitt 10 werden die Testergebnisse gezeigt, welche mit den Algorithmen der Projektgruppe erzielt wurden.
- Abschließend (Abschnitt 11 und 12) wird auf aktuelle Fragestellungen eingegangen, ein Ausblick auf mögliche Aufgabenstellungen gegeben und eine Resümee der Arbeit gezogen.

Zustandsloses Fahren

Zu Beginn des Projektes standen diverse Ansätze zur Diskussion. Sie lassen sich in mehrere Kategorien einteilen:

- Zustandsorientiertes Fahren: Hierbei werden während der Fahrt Daten gesammelt und im weiteren Verlauf für die Streckenplanung genutzt. Ein Beispiel hierfür ist SLAM[2] (Simultaneous Localization and Mapping): Im Rahmen des sog. Mappings sammelt das System im Laufe des Betriebs

[2] https://de.wikipedia.org/wiki/Simultaneous_Localization_and_Mapping

kontinuierlich Informationen über seine Umgebung und fügt diese in Form einer Karte zusammen, aus der anschließend, im Falle des Autos, eine optimale Strecke berechnet werden kann. Anschließend wird mittels der Localization die Position bestimmt und es werden entsprechende Bewegungen ausgeführt um der im Rahmen des Mappings errechneten Route zu folgen. In der Regel benötigen solche Ansätze einen Vorlauf oder bereits gespeicherte Daten um ihre volle Leistungsfähigkeit zu entfalten.

- Zustandsloses Fahren: Im Gegensatz zum vorher genannten Ansatz werden hier die Entscheidungen immer nur aufgrund der gerade erfassten Daten getroffen, eine Speicherung erfolgt in der Regel nicht oder nur für eine sehr kurze Zeit. Diese Methoden haben relativ hohe Anforderungen an die Präzision der Messdaten und die auswertenden Algorithmen, dafür benötigen sie keinen nennenswerten Vorlauf oder vorher gespeicherte Daten. Die Leistung steht von Beginn an zur Verfügung.

- Hybride Ansätze bei denen zuvor genannte Methoden verbunden oder durch maschinelles Lernen optimiert werden. Im Bereich des maschinellen Lernens bietet es sich an optimale Werte verschiedener Parameter der beiden zuvor genannten Ansätze zu erlernen.

Im Rahmen der Projektgruppe wurden die Ansätze vieler anderer Teams gesichtet. Leider legt kein Team die Details oder die Leistungsfähigkeit ihrer jeweiligen Implementierung offen, die entscheidenden Informationen werden zurückgehalten. Soweit es sich aus Videos oder veröffentlichen Teilen des jeweiligen Programmcode erkennen lies, scheint der überwiegende Teil der Teams auf zustandsorientierte Verfahren zu setzen, wobei in keinem Fall konkrete Messdaten über die tatsächliche Leistung der Verfahren vorlagen.

Die Entscheidung fiel auf die Umsetzung von zustandslosem Fahren. Diese gründet zum einen auf der Einschätzung, dass es sich hierbei um relativ einfache, klare Verfahren handelt, zum anderen aber auch auf dem Anspruch an das Ergebnis am Ende der Projektgruppe: Es galt nicht nur eine konkurrenzfähige Plattform zu schaffen sondern auch durch die Optimierung eines bisher wenig beachteten Verfahrens einen wissenschaftlichen und technologischen Beitrag zu leisten.

Grundsätzlich kann in diesem Bericht nur das Ergebnis in Form von Messwerten dargestellt und ein Vergleich mit theoretischen Maximalwerten angestellt werden. Ein Vergleich mit anderen Gruppen ist aufgrund mangelnder Daten nicht möglich. Dennoch scheint es, dass die zustandslosen Verfahren gewisse Vorteile bieten, was sich unter Anderem daraus schließen lässt, dass der Gewinner des letzten offiziellen Wettkampfs, der real stattgefunden hat, ein solches Verfahren einsetzt[3], das sich nicht dramatisch von den hier beschriebenen Ansätzen unterscheidet.

[3] https://www.nathanotterness.com/2019/04/the-disparity-extender-algorithm-and.html

1.1 Änderunghistorie

- **1.0** - Abgabe zum Projektgruppenende, 30.09.2020
- **1.1** - Ausarbeitung ROS (Anhang E) durch korrekte Version ersetzt, Änderunghistorie hinzugefügt, 08.10.2020
- **1.2** - Titel geändert, 02.12.2020

2 Grundlagen

In diesem Abschnitt werden die Grundlagen zu den in diesem Projekt verwendeten Themen erläutert. Diese wurden bereits zu Beginn des Projektes durch die Teilnehmer in Ausarbeitungen betrachtet, welche diesem Bericht als Anhang beigefügt sind. Die Darstellung erfolgt hier in gekürzter Form und behandelt nur die Punkte, die bisher eine Rolle bei der Entwicklung und beim Testen gespielt haben.

2.1 Koordinatensystem

Die in Anhang B beschriebene mögliche Realisierung, eine Karte aus den Daten des Gyro-, Beschleunigungs- und Erdmagnetfeldsensors zu erstellen, wird bislang noch nicht umgesetzt, da die eingebauten Sensoren eine zu geringe Genauigkeit haben. Jedoch werden Eigenschaften von Koordinatensystemen und Koordinatentransformationen an anderer Stelle verwendet: Im Wallfollowing-Algorithmus werden aus den LiDAR-Daten, welche in Polarkoordinaten erzeugt werden, kartesische Koordinaten berechnet. Mit diesen kann dann die weitere Verarbeitung, wie das Berechnen der auf die rechte und linke Wand der Rennstrecke approximierten Kreise, erfolgen. Auch bei der ZED-Kamera findet die Transformation von Polarkoordinaten zu kartesischen Koordinaten Einsatz. Zudem soll für ein dynamisches Überholen die Kenntnis der Geschwindigkeit und Position des zu überholenden Fahrzeuges nutzbar sein. Hier wird die Dynamik in bewegten Bezugssystemen und Koordinatentransformation relevant sein.

2.2 Fahrphysik

Die physikalischen Grundlagen werden in Anhang C im ersten Abschnitt ausgiebig behandelt und dienen als Grundlage für die folgende Betrachtung. Im Laufe des Projektes stellte sich heraus, dass die meisten Effekte keine oder eine untergeordnete Rolle spielen. Der Stabilitätsverlust beim Befahren von Kurven (Unter- bzw. Übersteuern) oder mangelnde Traktion aufgrund eines unzureichenden Ausgleichs durch das Differential traten nicht auf. Dies ist nicht zuletzt darauf zurückzuführen, dass die Algorithmen darauf ausgelegt sind einen Verlust der Haftung zwischen Reifen und Fahrbahn zu vermeiden.

Unter Bezugnahme auf die einzelnen Punkte in der Ausarbeitung lässt sich folgendes konstatieren: Die Annahmen zu den Eigenschaften des Fahrzeugs sind insofern korrekt, als dass keiner der dort genannten negativen Aspekte eine Rolle spielte. Da, wie bereits genannt, kein Stabilitätsverlust eintrat, sind alle Annahmen bezüglich des Antriebs bzw. der Bremsen und des Differentials nicht relevant. Eine Ausnahme bildet der Punkt Reifen: Hier wurde zu Beginn von einem $\mu = 0.9$ bei dem Befahren eines aspahltartigen Untergrunds ausgegangen. Da für die Tests kein derartiger Boden zur Verfügung stand (es wurde auf PVC-Boden getestet welcher scheinbar einen Reibwert von $\mu \approx 0.5$ aufweist), kann die Frage nach den tatsächlichen Eigenschaften der Reifen nicht analytisch geklärt werden, sondern muss im Zweifel durch Fahrtests ermittelt werden. Da dieser Wert einen enormen Einfluss auf die errechnete Maximalgeschwindigkeit und die mögliche Beschleunigung hat wurde dieser Parameter so im Code hinterlegt, dass er sich zur Laufzeit anpassen lässt.

2.2.1 Bremspunkt

Es soll die Distanz berechnet werden, die das Auto braucht, um zum Kurveneingang die benötigte Kurvengeschwindigkeit zu erreichen. Um die Rundenzeit zu verkürzen, muss das Auto die maximale Zeit beschleunigen. Der Bremspunkt muss so gewählt werden, dass die Geschwindigkeit über einen möglichst langen Zeitraum gehalten werden kann. Es ist allerdings von höchster Bedeutung, dass die Kurvengeschwindigkeit mit Beginn der Kurve nicht überschritten wird. Hierbei entspricht s_a der Distanz vom Auto zum Bremspunkt in der beschleunigt wird, s_b der Entfernung vom Bremspunkt zum Kurveneingang in der das Bremsen beginnt, v der Geschwindigkeit, die maximal nach der Beschleunigungsphase erreicht werden kann und c einem konstanten zu addierenden Wert, der als Sicherheitspuffer genutzt wird und ebenfalls zur Laufzeit angepasst werden kann. Die Herleitung ist Anhang H zu entnehmen. Es gilt:

$$s_b = \frac{s}{2} + \frac{v_0^2 - v_t^2}{4a} + c$$

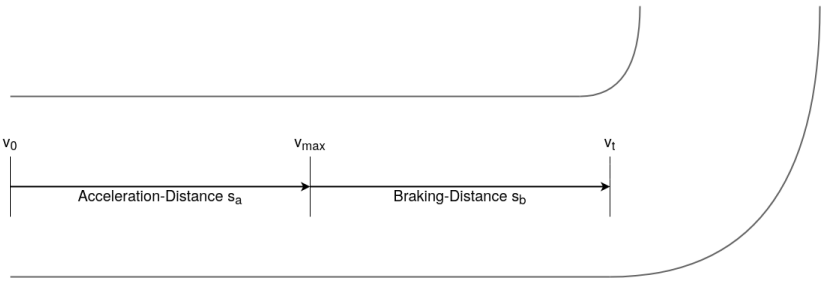


Abbildung 1: Verhalten des Autos vor einem Kurveneingang. Zuerst beschleunigt es von der aktuellen Geschwindigkeit v_0 so lange wie möglich über eine Distanz s_a und erreicht dann die hier berechnete maximale Geschwindigkeit v_max. Danach muss es über die Distanz s_b durchgehend bremsen, um dann die Zielgeschwindigkeit v_t an dem Kurveneingang zu erreichen.

2.2.2 Zielgeschwindigkeit

Die maximale Zielgeschwindigkeit v_t wird erreicht, wenn die Zentrifugalkraft F_z, die durch eine Kurve auf das Auto wirkt, genau so groß ist wie der Reibungswiderstand F_r der Reifen auf dem Boden. Wenn F_z größer wäre als F_r, würden die Reifen die Haftung zum Boden verlieren und das Auto könnte aus der Kurve fliegen. Die Zielgeschwindigkeit wird mit der Reibung μ des Autos auf der Strecke, der Fallbeschleunigung g und dem Kurvenradius r wie folgt berechnet:

$$\begin{aligned} & F_z = F_r \\ \Leftrightarrow \quad & m\frac{v_t^2}{r} = m\mu g \\ \Leftrightarrow \quad & v_t = \sqrt{\mu g r} \end{aligned}$$

2.3 Sensorik/Aktorik

Das in dieser Projektgruppe verwendete Auto basiert auf den Vorgaben der *F1/10-Challenge* [23], einer „open-source-basierten, erschwinglichen, und hochperformanten Plattform für autonome Fahrzeuge im 1:10 Format“. [23] Die F1/10-Plattform bietet Studenten und Forschungsgruppen verschiedener Universitäten weltweit die Möglichkeit, auf einer identischen Plattform die bestmögliche Software zu entwickeln und sich in Wettrennen zu messen. Dabei lassen die verschiedenen Gruppen ihre autonom fahrenden Autos auf der gleichen Stre-

Abbildung 2: Das unmodifizierte Fernlenkauto. [24]

cke fahren und vergleichen verschiedene Kriterien, wie zum Beispiel Rundenzeit oder den Fahrstil [23].

Besagte Plattform ist ausgestattet mit verschiedenen Sensoren, einer performanten CPU/GPU und einer Motorsteuerung, dargestellt in Abbildung 3. Grundlegend basiert das Fahrzeug auf einem 1:10 Modell eines Ford Fiesta ST Rally, hergestellt von *Traxxas* (siehe Abbildung 2) [24]. Dieses wird umgebaut um die Steuerung basierend auf verschiedenen Sensoren zu ermöglichen: Die mitgelieferte Motorsteuerung wird durch eine FOCBOX ersetzt, ein auf dem VESC-Projekt basierender Motor-ESC (electronic speed control). Zudem wird dem Auto zum Messen der Umwelt eine Reihe an Sensoren angebaut: einer IMU (Inertial Measurement Unit), einem LiDAR (Light Detection and Ranging) und eine ZED Cam (Stereoskopische Kamera).

Die IMU ermöglicht es, sowohl die Drehung, die Beschleunigung und das Erdmagnetfeld um die X-, Y- und Z-Achse zu messen. Anhand dessen kann bspw. unabhängig von der Motorsteuerung die aktuelle Geschwindigkeit berechnet werden, wodurch Abweichung durch Rutschen erkannt werden können. Der LiDAR-Sensor ermöglicht eine Erkennung der Umgebung auf einer Höhe. Diese Daten lassen sich in Form einer Punktwolke verarbeiten, beinhalten aber nur Positionsinformationen. Die ZED Cam hingegen ermöglicht die Umgebungserkennung zwar mit einer geringeren Auflösung als der LiDAR-Sensor, liefert aber zusätzlich noch Farbinformationen zu den Positionsinformationen, welche beispielsweise zur Segmentierung des Umfeldes in Hindernisse/Nicht-Hindernisse relevant werden können.

Diese Sensoren werden angeschlossen an ein Nvidia Jetson TX2 Board, welches über ein Orbitty Carrier Board verschiedene Schnittstellen bietet. Dieses Jetson-Board bietet sowohl mehrere ARM- und Nvidia-Kerne als auch eine Vielzahl an CUDA-Einheiten zur Berechnung aufwendiger, paralleler Rechenprozesse [22]. Das Jetson-Board gibt schließlich Steuersignale an die FOCBOX ESC

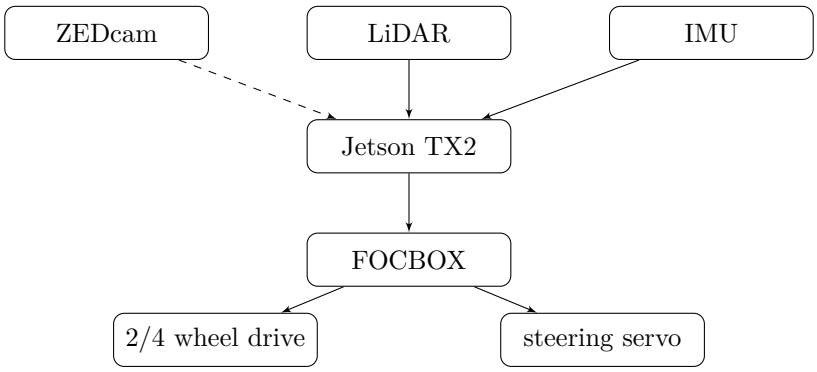


Abbildung 3: Die von der vorhergehenden Gruppe genutzten Sensoren und Aktoren im Überblick.

weiter, welche sowohl den Lenkeinschlag über einen Servo-, als auch den Antriebsmotor steuert. Eine genauere Beschreibung der Sensoren und Aktoren ist in Anhang D zu finden.

Durch die Ausstattung mit einer Vielzahl verschiedener Sensoren hat das Auto die Möglichkeit zur Erkennung der Umgebung und seiner eigenen Position. Es ist leicht erkennbar, dass diese Daten prädestiniert für gängige SLAM-Algorithmen sind (die Kombination aus IMU und LiDAR macht dies deutlich), jedoch ist auch eine Vielzahl von anderen Algorithmen denkbar, die schon von der vorherigen Projektgruppe erarbeitet wurden. Diese sollen jetzt von dieser Projektgruppe erweitert und um weitere Features ergänzt werden.

2.4 ROS und Gazebo

ROS (Robot Operating System) ist ein flexibles Framework, welches die Zusammenarbeit verschiedenster Forschungsbereiche im Bereich der Robotik vereinfachen soll. Dazu werden in ROS verschiedenste Prozessmodule, sogennante *Nodes*, erzeugt, die alle unabhängig ihre Aufgaben, wie zum Beispiel das Erkennen von Objekten oder eine Pfadplanung durch eben diese Objekte hindurch, erfüllen [18]. ROS bietet als Kernkomponente für diese Nodes eine Publisher / Subscriber-basierte Kommunikationsinfrastrukur, damit die Nodes ihre Daten untereinander über vorher definierten Schnittstellen austauschen können und somit verschiedene Nodes von verschiedenen Forschungsteams gemeinsam größere Aufgaben lösen können.

Diese Nodes sollen dabei möglichst leichtgewichtig sein, damit im Optimalfall eine Node in verschiedensten Robotern verwendet werden können. Ferner wer-

den Nodes als einzelne Prozesse ausgeführt, sodass eine gute Lastverteilung bei Mehrprozessorsystemen möglich ist. Zudem folgt ROS für die Programmierung der Nodes einer Unabhängikeit der genutzten Sprache (sofern die Schnittstellen korrekt implementiert werden), C++ und Python bilden hierbei jedoch die Basis. Dies ermöglicht einen hohen Grad an Flexibilität bei der Auswahl der Programmiersprache und dem fachlichen Anspruch an die jeweiligen Programmierer.

Die Kernkomponente von ROS bildet die Nachrichteninfrastruktur, welche die Kommunikation zwischen den verschiedenen Nodes realisiert. ROS nutzt dazu verschiedene Mechanismen, wobei wir zur Kommunikation zwischen Nodes Topics verwendet haben. Topics folgen dem Publisher / Subscriber-Prinzip, sodass Nodes ihre Daten auf verschiedene Topics verbreiten können, während andere Nodes bestimmte Topics subscriben und auf Daten, die auf diesen Topics verbreitet werden, reagieren können. Topics selbst sind nur IDs und stellen den Namespace der Daten dar [18].

Zur Simulation wird Gazebo eingesetzt, da es die Möglichkeit bietet realistische Fahrsimulationen mit den entwickelten ROS-Nodes zu testen (Gazebo ist in ROS integriert). Als weiteren positiven Punkt bietet Gazebo die Möglichkeit eine Vielzahl von Sensoren über Erweiterungen zu simulieren oder mit einfachen Mitteln zu implementieren. Ein detaillierter Einblick in die Funktionsweise von ROS und Gazebo ist in Anhang E zu finden.

2.5 CUDA

CUDA ist ein von Nvidia entwickeltes Konzept zur Abarbeitung von Programmteilen durch eine oder mehrere GPUs [20]. CUDA steht dabei für *Compute Unified Device Architecture* und umfasst neben dem Konzept auch eine Programmiersprache zur Umsetzung. Geschrieben werden die auszulagernden Programmteile in *C for CUDA* und C++ mit den entsprechenden Erweiterungen. Es existieren auch Wrapper für Perl, Python, Ruby, Java, FORTRAN, .NET, MATLAB, Mathematica und R [31]. Die CUDA Erweiterungen können auf Windows, Linux sowie macOS installiert und verwendet werden. Damit decken sie die derzeit populärsten Betriebssysteme für Workstations und Multi-Purpose Server ab. Zur Ausführung der Programmteile wird eine Grafikkarte von Nvidia ab der GeForce 8-Serie oder ab der Quadro FX5600 benötigt [21]. Trotz der Herstellerbindung hat sich CUDA als Standard für die GPU-Programmierung durchsetzen können [10].

Heutzutage sind CPUs grundsätzlich mit zwei bis acht Prozessorkernen ausgestattet, was bereits eine beachtliche Rechenleistung zur Verfügung stellt. Moderne Grafikkarten kommen jedoch selbst im preiswerten Segment auf mehrere Hundert Prozessorkerne, die viele kleine Programmteile parallel ausführen können. Die Verwendung dieser Infrastrukturart bietet sich daher vor allem

bei hochgradig parallelisierbaren Aufgaben an, wie beispielsweise Simulationsberechnungen, Video- und Bildbearbeitung, Machine Learning, Training von neuronalen Netzen und anderen KI Anwendungen. Bemerkenswert ist hierbei, dass durch das Auslagern von Rechenarbeiten an GPUs der Stromverbrauch des gesamten Systems gesenkt werden kann.

Zu beachten ist, dass CPU und GPU sich nicht einen gemeinsamen Speicher teilen. Daher müssen Daten die von der GPU verarbeitet werden sollen zunächst in aufwendigen Operationen von dem Hauptspeicher in den Speicher der GPU und im Anschluss wieder zurück in den Hauptspeicher transferiert werden. Diese Operationen verursachen in unserem Anwendungsfall bisher erheblich mehr Rechenaufwand als CUDA einspart, sodass sich zunächst dazu entschieden wurde in den Implementierungen auf CUDA zu verzichten. Verwendet wird CUDA allerdings von der Kamera des Autos. Es steht auch eine Entwicklungsumgebung auf einem Server bereit, der für die Projektgruppe aufgesetzt wurde, um selbst CUDA Code zu schreiben und zu testen.

Eine detaillierte Einführung in die genaue Funktionsweise von CUDA und das Zusammenspiel von CPU und GPU mitsamt Code-Beispielen findet sich in Anhang F.

2.6 Maschinelles Lernen

Im Folgenden wird auf Grundlagen und Konzepte des Maschinellen Lernens eingegangen. Hierbei wird sich auf solche beschränkt, die während der Projektarbeit Anwendung stattgefunden haben. Des Weiteren wurde sich bei der Modell-Wahl auf Neuronale Netze, die in Anhang G näher erläutert werden, beschränkt. Der Grund hierfür ist, dass Maschinelles Lernen ausschließlich dafür eingesetzt wurde, ein eigenständig fahrendes Modell zu trainieren und von einem komplexen Zusammenhang zwischen den Eingangsdaten des Modells, der Messwerte der eingesetzten Sensoren, und den Ausgangsdaten, dem Lenkwinkel und der Zielgeschwindigkeit, auszugehen ist.

Aufgrund der Relevanz für die Projektarbeit wird kurz auf die Methodik des überwachten Lernens und im Besonderen auf die Methodik des bestärkenden Lernens und den Q-Learning Algorithmus eingegangen.

2.6.1 Überwachtes Lernen

Beim überwachten Lernen versucht man einem Modell ein bestimmtes Verhalten mit Hilfe von zur Verfügung stehenden Trainingsdaten beizubringen. Hierbei

wird davon ausgegangen, dass sich das zu erlernende Verhalten aus den Trainingsdaten ableiten lässt. Unerlässlich ist daher, das jedes Trainingsdatum mit einem Zielwert, dem sogenannten Label, versehen ist. Gleichzeitig müssen die Trainingsdaten eine möglichst repräsentative Stichprobe der Anwendungswelt darstellen in der das Modell letztendlich zum Einsatz kommt.

Während des Training werden die Parameter des Modells optimiert. Eine gängige Optimierung ist hierbei die Minimierung des Vorhersagefehlers auf den Trainingsdaten ϵ_t. Dieser ergibt sich aus der Summe der Differenzen zwischen dem wahren Zielwert y_i jedes Trainingsdatums und dem vom dem Modell vorhergesagt Zielwert $\overline{y}_i$ für dieses Trainingsdatum.

$$\epsilon_t = \sum_i |y_i - \overline{y}_i|$$

In Anhang G wird näher darauf eingegangen, wie die Parameter und Gewichte eines Neuronalen Netzes anhand einer Optimierungsfunktion, wie der Minimierung des Vorhersagefehlers, optimiert werden.

2.6.2 Bestärkendes Lernen

Bestärkendes Lernen oder Reinforcement Learning ist neben unüberwachtem [6] und überwachtem Lernen das dritte Teilgebiet des Maschinellen Lernens. Beim bestärkenden Lernen geht es darum das Verhalten eines Agenten in einer sich verändernden Umgebung zu optimieren. Hierbei lernt der Agent eine Taktik π indem er in einem Trainingsverfahren Belohnungen (Rewards), die abhängig von der gewählten Aktion (Action) und dem aktuellen Zustand (State) sind, erhält. Gesucht wird eine Taktik π^* welche die zu erwartende Belohnung maximiert.

2.6.3 Q-Learning

Ein Algorithmus nach dem Konzept des bestärkenden Lernens ist der Q-Learning Algorithmus. Hierbei wird das zu betrachtende Problem als Markow-Entscheidungsproblem aufgefasst. Das Markow-Entscheidungsproblem wird durch das Tupel $(\mathcal{S}, A, R, \mathcal{P})$ beschrieben. Hierbei beschreibt

- $\mathcal{S}$ die Menge der Zustände S,
- A die Menge der Aktionen a,
- R die Verteilung Reward-Funktion, die jedem Tupel (S, a) eine Belohnung zuweist,

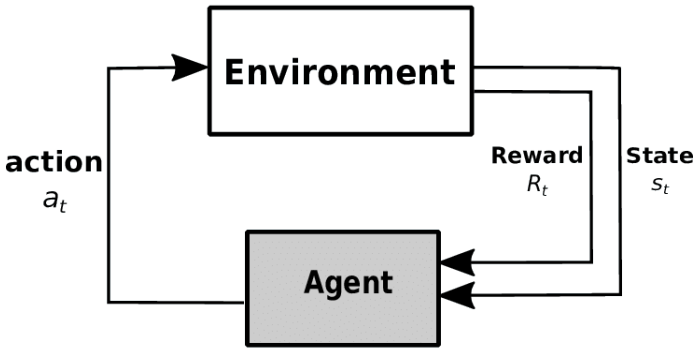


Abbildung 4: Reinforcement Learning, Agent and Environment https://www.researchgate.net/figure/Reinforcement-Learning-Agent-and-Environment_fig2_323867253

- $\mathcal{P}$ die Verteilung der Übergangswahrscheinlichkeiten in den nächsten Zustand S' bei gegebenem Tupel (S, a)

Es gilt die Markow-Eigenschaft: Die Umgebung wird zum Zeitpunkt t ausschließlich über den aktuellen Zustand S_t definiert und ist unabhängig von vergangenen Zuständen.

Es soll nun bewertet werden, wie gut ein bestimmter Zustand S ist, wenn eine bestimmte Strategie π gegeben ist. Dies gelingt in dem π auf Zustand S und die darauf folgenden Zustände angewandt und die jeweiligen Belohnungen kumuliert werden. Da sowohl die Belohnugen als auch die Zustandsübergänge aus einer Verteilung gezogen werden, können hier nur die erwarteten Werte zu Bewertung genutzt werden. So ergibt sich die Value-Funktion $V^\pi(S)$, mit der man den Wert eines Zustandes S bei zu Grunde liegender Strategie π bewertet, wie folgt:

$$V^\pi(S) = E[\sum_{t \geq 0} \gamma^t R_t | S_0 = S, \pi]$$

Hierbei ist γ ein Gewichtungsfaktor, um Belohnungen die in der Zukunft erzielt werden geringer zu gewichten.

Soll nun nicht allein der Zustand S bewertet werden, sondern ein Zustand-Aktions-Tupel (S, a), wird die Value-Funktion so angepasst, dass die Aktion a als erste Aktion im Zustand S ausgeführt wird. Daraus ergibt sich die sogenannte Q-Value-Funktion, welche eine zentrale Rolle im Q-Learning Algorithmus einnimmt:

$$Q^{\pi}(S,a) = E[\sum_{t \geq 0} \gamma^t R_t | S_0 = S, a_0 = a, \pi]$$

Die optimale Strategie π^* maximiert diese Q-Value Funktion::

$$\pi^* = \arg\max_{\pi} Q^{\pi}(S,a)$$

$$Q^*(S,a) = \max_{\pi} Q^{\pi}(S,a)$$

Da in der optimalen Strategie die gewählte Aktion in jedem Zeitpunkt hinsichtlich des in der Zukunft zu erwartenden Belohnung optimal sein muss, erfüllt die optimale Q-Value Funktion die Bellmann-Gleichung (QUELLE) und $Q^*(S,a)$ wie folgt umgeformt werden:

$$Q^*(S,a) = E[R + \gamma \max_{a'} Q^*(S',a') | S,a]$$

Wenn also die optimalen Aktionen für die nächsten Zeitschritte bekannt sind, ist die beste Strategie, die Aktion auszuführen, die den Erwartungswert von $R + \gamma Q^*(S',a')$ maximiert.

2.6.4 Lernen durch Q-Learning

Wenn die Menge der Zustände $\mathcal{S}$ und die Menge der Aktionen $\mathcal{A}$ endlich ist, kann $Q(S,a)$ durch eine sogenannte Q-Table modelliert werden. Zu Beginn des Trainingsalgorithmus wird diese zufällig initialisiert. Während des Trainings wird nun in jedem Zeitschritt entweder die laut Q-Table optimale Aktion, oder eine (mit einer im Laufe des Training abnehmenden Wahrscheinlichkeit) zufällige Aktion gewählt.

Nach jedem Zeitschritt t speichert man das Tupel (S_t, a_t) gemeinsam mit dem tatsächlich erzielten Reward R_t in einer Historie. Nach einer bestimmten Anzahl an Schritten endet eine Epoche. Am Ende jeder Epoche wird die Q-Table anhand der Historie aktualisiert. Der neue Wert v_{new} für jedes in der Historie befindliche Tupel (S_t, a_t) ergibt sich wie folgt:

$$v_{new} = R_t + \gamma \max_{a} Q(S_{t+1}, a)$$

Der Wert für $Q(S_t, a_t)$ wird jedoch nicht durch v_{new} ersetzt, sondern abhängig von einer Lernrate α an diesen angepasst:

$$Q_{new}(S_t, a_t) = Q_{old}(S_t, a_t) + \alpha(v_{new} - Q_{old}(S_t, a_t)$$

Die Historie wird oft als Heap mit fester Größe realisiert und beim Aktualisieren wird zufällig aus diesem gezogen. Die Schritte des Algorithmus ergeben sich daraus wie folgt:

1. Initialisiere $Q(S, a)$ zufällig
2. In jedem Zeitschritt wähle die laut Q-Table optimale oder eine zufällige Aktion
3. Am Ende jeder Epoche aktualisiere Q-Table anhand der Historie

2.6.5 Deep Q-Learning

Für die Modellierung der Q-Table wurde bisher davon ausgegangen, dass die Menge der Zustände $\mathcal{S}$ endlich ist. Handelt es sich aber dagegen um einen kontinuierlichen Zustandsraum, wird die Q-Table unendlich groß und einzelne Einträge in dieser werden während des Training voraussichtlich nie angepasst. Um dieses Problem zu lösen gilt es für jede mögliche Aktion a_j eine kontinuierliche Funktion $Q(S, a_j)$ zu modellieren.

Beim Deep Q-Learning modelliert man $Q(S, a)$ durch ein Neuronales Netz. Das Netz erwartet einen Zustand S als Eingabe und für jede Aktion a_j existiert ein Ausgang. Auf diese Weise wird bei der Eingabe eines Zustands ein Wert $Q'(S, a_j)$ für jede mögliche Aktion a_j vorhergesagt. Wie die Q-Table, wird auch das Neuronale Netz zufällig initialisiert. Am Ende jeder Epoche wird das Neuronale Netz wie beim überwachten Lernen trainiert. Als Trainingsmenge wird hierbei die gespeicherte Historie mit entsprechenden Zielgrößen v_{new} trainiert.

3 Vorangegangene Projektgruppe

Zuvor hat sich bereits eine andere Projektgruppe mit einer ähnlichen Aufgabenstellung befasst. Diese hat es sich zum Ziel gesetzt, eine grundlegende ROS-basierte Umgebung aufzubauen und in dieser verschiedene Algorithmen zu implementieren, die das Auto autonom auf einer Rennstrecke weitestgehend unfallfrei fahren lassen. Zudem wurde von der Projektgruppe eine Gazebo-basierte Simulationsumgebung aufgebaut, welche die Ausführung der ROS-Umgebung auf einem Auto simuliert. Die zuvor implementierten Algorithmen sind:

- Wallfollowing basierend auf zwei auf LiDAR-Daten gefitteten Kreisen
- SLAM Algorithmen basierend auf dem ROS Navigation Stack
- Deep Reinforcement Learning (Q-Learning und Policy Gradient)
- Neuronale Netzwerke mit evolutionärem Training

Dabei hat sich durch Vergleiche schnell herausgestellt, dass Wallfollowing die effizienteste Implementierung ist.

3.1 ROS

Zur Umsetzung der ROS Infrastruktur wurde die ROS Version *Kinetic Kame* gewählt. Diese ROS Version ist lose an Ubuntu 16.04 gekoppelt, welches auch auf dem Jetson-Board installiert war. Implementiert wurden Nodes, die folgende Anforderungen erfüllen:

- Automatische Notbremsung bei zu geringem Abstand zu vor dem Auto liegenden Objekten (`autonomous/emergency_stop`)
- Totmannschalter, um das Auto in Gefahrsituationen manuell zu bremsen (`car_control`, `teleoperation`)
- Fernsteuerung des Autos mit Tastatur oder Controller (`car_control`, `teleoperation`)
- Geschwindigkeitsmessung und Rundenzeitmesser zum Vergleich verschiedener Algorithmen in der Simulation (`simulation/simulation_tools`)
- Anbindung an die Hardware des Autos (`hardware`)
- Anbindung an die Gazebo-basierte Simulation (`simulation`)

- Verschiedene autonome Fahralgorithmen (`autonomous`)

Der Kontrollfluss zwischen den Nodes soll zunächst erklärt werden. Die vorhergehende Projektgruppe bietet in ihrem Wiki ein sehr gutes Übersichtsdiagramm [4]. Grundlegend wählt das System zwischen den Steuerdaten der manuellen Steuerung und des autonomen Algorithmus. Es wird sowohl die Teleoperation (`keyboard_controller`, `joystick_controller`) als auch der jeweilige Algorithmus gleichzeitig ausgeführt. Zudem wird den Daten der im `dms_controller` implementierte Totmannschalter vorgeschaltet, mit dem die Steuerung ein- und ausgeschaltet werden kann. Die Steuerdaten werden anschließend an einen Multiplexer weitergereicht.

Von diesem `drive_params_multiplexer` werden die Daten dann über den `car_controller` entweder an die `vesc_driver_node`, welche das VESC des physischen Autos anspricht, oder den `vesc_sim_driver`, der die Fahrdaten verarbeitet und an die Gazebo-Simulation sendet, weitergegeben. Je nach Ausführungsumgebung werden LiDAR-, IMU- und VESC-Daten aus Gazebo oder aus den entsprechenden Sensoren ausgelesen und auf deren jeweiligen Topic veröffentlicht.

3.2 Gazebo

Um die implementierten Algorithmen und die Softwareinfrastruktur zu testen, wurde die *Gazebo* Robotersimulationssoftware genutzt. Diese ermöglicht es, basierend auf Physik-Engines die implementierten Algorithmen zu testen während die Hardware unbeschädigt bleibt. Gazebo simuliert das Auto und dessen entsprechende Aktorik. Zudem werden auch Sensordaten wie z.B. LiDAR-Scandaten und IMU-Werte aus der virtuellen Umgebung generiert und können von den Nodes wie reale Sensordaten ausgelesen werden. Zur Simulation gibt es ein in ROS verwendbares Gazebo-Package, welches zwischen der Gazebo Simulation und der ausgeführten ROS-Umgebung übersetzt. So ist es möglich, die gleichen Implementierungen für die Simulation und das Auto zu verwenden.

Um möglichst realitätsnah simulieren zu können, wurde das Auto als 3D-Modell modelliert. Zudem wurden verschiedene Strecken konstruiert welche zur Simulation verwendet werden. So können auch verschiedene Algorithmen auf der exakt selben Strecke getestet werden, welches gute Performancevergleiche zulässt. Diese 3D-Modelle werden durch Gazebo-spezifische URDF-Dateien be-

[4] `https://github.com/Autonomous-Racing-PG/ar-tu-do/wiki/List-of-important-nodes-and-topics`

schrieben und somit um Informationen wie z.B. der Aktorik und Sensorik des Autos angereichert. Zudem werden in diesen URDF-Dateien die Eigenschaften für die Physik-Simulationen beschrieben, also Kollisionsobjekte und Reibwerte.

Viele der Sensoren konnten mit Hilfe der von Gazebo mitgelieferten Plugins simuliert werden. Eine Herausforderung war allerdings die Simulation der verwendeten Motorsteuerung. Um diese mit den gleichen Eingabewerten, wie die tatsächliche Motorsteuerung des Autos sie benötigt, anzusteuern, wurde ein *VESC Simulator* entwickelt, der diese Daten als Eingabe akzeptiert und an die Gazebo-basierte Aktorik weiterleitet. Dabei wurden auch Parameter wie die Übersetzung und weitere Eigenschaften des Elektromotors und der Lenkung beachtet.

3.3 Wallfollowing

In diesem Kapitel wird das Wallfollowing [14] der Vorgängergruppe behandelt. Dieser steht im Rahmen dieser Projektgruppe wesentlich im Fokus, da der Algorithmus im Vergleich zu anderen Verfahren, wie zum Beispiel SLAM [1], eine ausreichend schnelle Verarbeitungsgeschwindigkeit erreicht. Folgende Schritte werden dabei vom Algorithmus abgearbeitet:

1. Die empfangenen LiDAR-Daten werden von polaren in kartesische Koordinaten umgewandelt.
2. Über die Koordinaten wird iteriert und der größte Abstand zwischen zwei nebeneinander liegenden Elementen gesucht. An dem Elementindex mit diesem größten Abstand werden die Daten in rechte und linke Wand aufgeteilt.
3. Auf den jeweiligen Wänden wird eine Kreisregression ausgeführt.
4. Ein bestimmter Punkt, geradlinig vor dem Fahrzeug, dient dazu den minimal entfernten Mittelpunkt der Strecke zu bestimmen.
5. Zwischen dem Punkt und dem Mittelpunkt der Strecke wird ein Fehler berechnet, der mit Hilfe eines PID-Controllers zu der Berechnung des Lenkwinkels führt.
6. Die Zielgeschwindigkeit wird aus mehreren Parametern, wie der Größe des aktuellen Kurvenradius bestimmt.

4 Infrastruktur

Der folgende Abschnitt widmet sich der gesamten Peripherie, die für die effiziente Entwicklung und den Betrieb erforderlich ist. Im Einzelnen sind das die Entwicklungssysteme, das Fahrzeug und ein erstelltes Skript um die Bedienung und Einrichtung zu erleichtern. Zudem wird auf die besonderen Anforderungen und Maßnahmen eingegangen, die sich für die Abschlusstests durch die vorherrschende Pandemie ergeben haben.

4.1 Skript toad.sh

Um Aufgaben zu automatisieren oder auch die Ausführung des Fahr- oder des Simulationsmodus zu erreichen, wurde ein Skript entwickelt, das Zugriff auf die meisten Funktionen über einfache Parameter ermöglicht: `toad.sh`

Es befindet sich im Verzeichnis `ros_ws`. Dort findet sich ebenfalls eine Konfigurationsdatei `toad.settings`, die vor der ersten Verwendung korrekt eingestellt werden muss. Die detaillierte Dokumentation wurde im Wiki des Projektes vorgenommen und ist in Auszügen in Anhang J einzusehen. Weitere Informationen zur Konfiguration können direkt der Konfigurationsdatei entnommen werden. Das Skript bietet im Wesentlichen folgende Funktionen:

- Kompilieren, Zurücksetzen und Starten der Simulation auf dem aktuell verwendeten System und
- auf dem Auto. Zusätzlich kann ein Remote-Modus genutzt werden: Startet man den Remote-Modus per SSH (Option `-XC` setzen, damit der Keyboardcontroller auf dem SSH-Client ausgeführt werden kann) auf dem Fahrzeug, kann von beliebigen Rechnern im selben Netzwerk der Control-Modus genutzt werden: Damit läuft die RViz-Visualisierung direkt auf dem Control-System und es werden nur die Rohdaten übertragen. Dies stellt eine enorme Verbesserung gegenüber der bisherigen Lösung per VNC dar.
- Es bietet die Möglichkeit sog. Rosbags, also Mitschnitte von durch ROS erfassten und verarbeiteten Daten über Profile zu erstellen.
- Das Skript bietet die Möglichkeit, im Git gespeicherte SSH-Keys für die Anmeldung auf dem ausführenden System freizugeben.
- Es wurden zahlreiche Funktionen zur Konvertierung von aufgenommenen Videos und zur Verarbeitung von aufgezeichneten Telemetriedaten hinzugefügt.

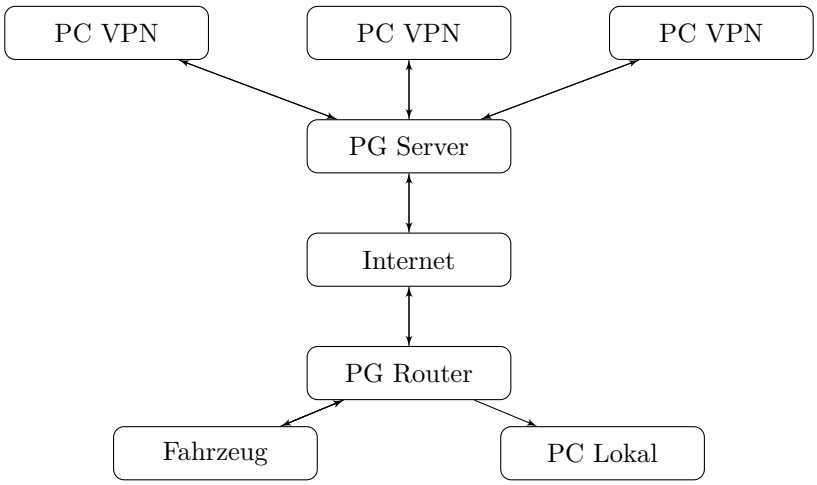


Abbildung 5: Netzwerkinfrastruktur zum Ende des Projektes

- Weitere Funktionen wie das Installieren des Systems oder benötigter Software wurden aus Gründen der Übersichtlichkeit entfernt.

4.2 Entwicklungssystem

Da die einzige offiziell unterstütze Umgebung auf Ubuntu 16.04. bzw. 18.04. beruht [9], ist die Wahl sehr eingeschränkt. Es wurde sich auf die Verwendung von Ubuntu 18.04. und ROS Melodic verständigt und eine Installationsanleitung im Wiki erstellt, die diesem Bericht in Anhang K beigefügt ist. Eine Kompatibilität zwischen den Ubuntu- und ROS-Versionen ist zwar gegeben, eine Abweichung erschwert jedoch massiv die Fehlersuche, da die für Ubuntu bereitgestellten Pakete ohnehin bereits einige Probleme aufweisen und nicht immer stabil laufen oder vollständig starten.

Ein besonderes Augenmerk ist hier auf Gazebo zu richten: Da es scheinbar für Umgebungen mit wenigen Elementen geschrieben wurde [11], kommt es bei vielen Versionen auf den meisten Systemen zu Abstürzen, wenn man die Gazebo-Server-GUI startet und damit die komplexe Welt, also die Rennstrecke, lädt. Man kann die Oberfläche zwar abschalten, hat dann aber keine Übersicht, wo sich das Fahrzeug auf der Strecke befindet. Nach langwieriger Fehlersuche und dem Testen verschiedener Versionen konnte jedoch eine ermittelt werden, die stabil läuft. Details hierzu werden in Anhang J genannt.
In der Endphase der Projektgruppe konnte jedoch der F1TENTH Racecar Si-

mulator [5] eingesetzt werden, der speziell für die F1TENTH geschrieben wurde, damit der Wettkampf virtuell ausgetragen werden kann. Dieser benutzt RVIZ als Anzeige. Gazebo und die verbundenen Probleme werden auf diesem Weg umgangen. Es wird für nachfolgende Projektgruppen empfohlen diesen Simulator zu nutzen, da zusätzlich realitätstreue Strecken ausgwählt werden können, die bei vorherigen Wettkampfveranstaltungen der F1TENTH aufgezeichnet wurden.

4.3 Entwicklungsserver

Da es leider nicht jedem Entwickler möglich ist, auf seinem eigenen Rechner die benötigte Umgebung zu installieren, sei es aus Kompatibilitäts- oder Kapazitätsgründen, wurde ein Server mit einer Nvidia GPU (zwecks Einsatz von CUDA) betrieben, den der Lehrstuhl zur Verfügung stellte. Entgegen der Planung kam es aus folgenden Gründen nicht zur Verwendung von CUDA:

- Durch die Portierung aller wichtigen Programmbestandteile von Python nach C++ konnte die Effizienz der Software und die CPU-Last im Betrieb drastisch gesenkt werden. Einer Portierung von Teilen der Software auf CUDA ist kein Mehrwert zuzurechnen und daher nicht durchgeführt worden.
- Von der dauerhaften Verwendung der Kamera wurde abgesehen, da diese keinen relevanten und messbaren Mehrwert bringt. Zudem erzeugt der Betrieb eine enorme CPU-Last und einen hohen Energiebedarf, sodass ein Akku regelmäßig nach 10 bis 15 Minuten ausgetauscht werden muss.

Dennoch blieb der Server weiter in Betrieb um Gruppenmitgliedern ohne die nötige Ubuntu-Umgebung eine Teilnahme zu ermöglichen. Neben dem gesamten für ROS benötigten Software-Stack lief dort Teamviewer für den Zugriff und BORG-Backup[6] zwecks Sicherung. Der Server führte regelmäßig einen Rebuild des Projektes und ein Update der erlaubten SSH-Keys (siehe Unterabschnitt 4.1) durch, die Sicherung erfolgte täglich, Updates wurden wöchentlich eingespielt. Da die Betreuung dieses Servers durch die Projektgruppe erfolgte, wird dieser am Ende gesichert und dann außer Betrieb genommen.

[5] `https://github.com/f1tenth/f1tenth_simulator`
[6] `https://borgbackup.readthedocs.io/en/stable/`

4.4 Continuos Integration Server

Damit im Git-Repository immer ein kompilierbarer, gut lesbarer Code liegt, wurde von der Vorgänger-PG die CI-Infrastruktur übernommen. Diese löst bei jedem Pull-Request und Push auf die Branch `development` einen Travis-Build aus, der sowohl den gesamten Code kompiliert, als auch auf korrekte Formatierung überprüft. Scheitert der Build aus soeben genannten Gründen, darf das Pull-Request nicht angenommen werden. Damit nicht jeder den Code per Hand formatieren muss, und das unnötige auswendiglernen von Formatierungs-Guidelines erspart wird, wird auf den Entwicklungsrechnern eine Pre-Commit Hook installiert, die alle veränderten Dateien automatisch formatiert, bevor sie dem Git-Index hinzugefügt werden.

Die Beschreibung aus der Travis-Konfiguration (`.travis.yaml`) dient zudem als gute Dokumentation, wie ein ROS-melodic-haltiges Ubuntu 18.04 System aufgesetzt werden kann, da die Kommandozeilenaufrufe dort die gleichen sie wie wenn man sein eigenes System aufsetzen wollen würde.

4.5 Fahrzeug

Im Folgenden wird die Hard- und Software des Fahrzeugs beschrieben.

4.5.1 Software

Im Rahmen der Einigung auf Ubuntu 18.04. als Basis für die Entwicklung wurde das Fahrzeug ebenfalls einer kompletten Neuinstallation mit identischen Versionen unterzogen. Im Prinzip erfolgte die Installation ähnlich wie bei einer normalen Arbeitsstation, aufgrund der verwendeten Plattform war jedoch an vielen Stellen ein anderes Vorgehen und bestimmte Anpassungen erforderlich auf die hier nicht näher eingegangen wird. Allerdings sei erwähnt, dass die größten Schwierigkeiten im Rahmen der Installation einer aktuellen CUDA Version, wie sie vom Framework für die ZED Kamera gefordert wird, und bei den Python-Komponenten für maschinelles Lernen auftraten.

Grundsätzlich erfordert eine saubere Installation an einigen Stellen fortgeschrittene Kenntnisse in Linux, daher sollte von einer weiteren Neuinstallation abgesehen und die Beschädigung der bestehenden Installation vermieden werden. Eine Sicherung und ein Update der Ubuntu Pakete wurde wöchentlich im Rahmen der Besprechung durchgeführt. Im Rahmen der Tests gegen Ende des Projektes erfolgte das Backup zu Beginn jeder Sitzung. Dieses ist jedoch mit der zuvor genannten Abschaltung des Servers nicht mehr nutzbar.

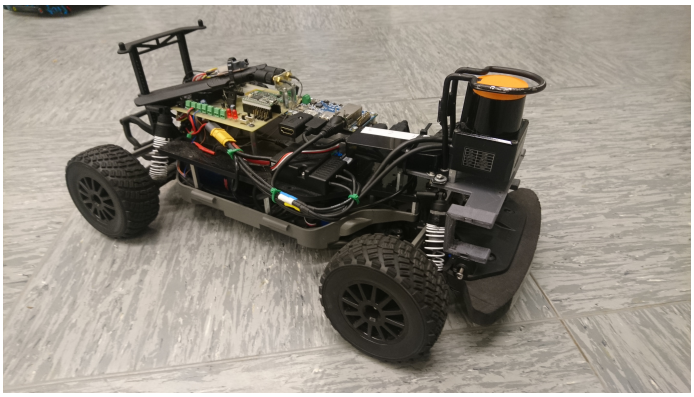


Abbildung 6: Fahrzeug in seiner endgültigen Konfiguration

Es besteht jedoch die Möglichkeit den gesamten Massenspeicher des Fahrzeugs als Image auf einen externen Datenträger zu sichern. Das Vorgehen wird im Wiki[7] näher beschrieben. Es ist empfehlenswert während der Sicherung oder dem Zurückspielen die Spannungsversorgung per Netzteil sicherzustellen, da diese Prozesse mehrere Stunden in Anspruch nehmen können.

4.5.2 Hardware

Die ursprüngliche Ausstattung des Fahrzeugs wurde bereits im Bericht der Vorgängergruppe [1] und in Anhang D beschrieben und größtenteils so belassen. Dennoch wurden einige, den Vorgaben der F1/Tenth [2] entsprechende Modifikationen vorgenommen:

- Optimierung der Positionen der Hardware um einerseits eine gute Verteilung der Masse zum anderen eine praktikable Anordnung zu erreichen. Zudem wurde die Verkabelung optimiert.
- Die Kamera und die IMU wurden fest verbaut. Letzt genannte Komponente wurde jedoch wieder außer Betrieb genommen, da derzeit keinerlei Nutzung erfolgt. Sie könnte jedoch in Zukunft bei der automatischen Optimierung einiger Parameter, wie zum Beispiel der Ermittlung des Reibwertes (durch Sie kann eine Relativbewegung zur Fahrbahn ermittelt werden was einem Stabilitätsverlust entspräche) eine Rolle spielen.
- Da es bei den eingesetzten Lithium-Polymer-Akkumulatoren unter keinen Umständen zu einer deutlich unter die Nennspannung liegenden Ent-

[7] https://github.com/arpg-sophisticated/ar-tu-do/wiki/Backup

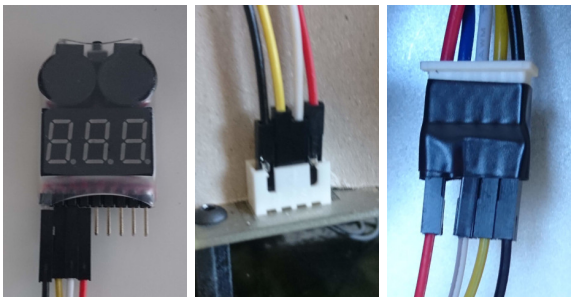


Abbildung 7: Der LiPo-Protector mit Anschlüssen

ladung kommen darf, wurde eine entsprechende Messvorrichtung (siehe Abbildung 7) verbaut. Im Falle der Unterschreitung einer vorgegebenen Spannung, in unserem Fall 10.8 V, ertönt ein nicht überhörbares Alarmsignal.

- Für die Steuerung des Fahrzeugs, sei es manuell oder zwecks Aktivierung des autonomen Fahrens (Totmannschalter) kam ein X-Box Controller zum Einsatz. Dieser ist per Bluetooth an das Fahrzeug angebunden und dient als Alternative zur Tastatur-basierten Steuerung: Zwar ist die Reichweite des Controllers beschränkter als bei einer per WLAN/Netzwerk verbundenen Tastatursteuerung, allerdings wurde festgestellt, dass die Totmannschalter-Funktion bei Einsatz der Tastatursteuerung und Verlust der SSH Verbindung nicht mehr funktioniert und das Fahrzeug seine Fahrt unvermittelt und dauerhaft fortsetzt. Dieser Effekt tritt zwar auch kurzzeitig auf wenn die Verbindung zwischen Bluetoothcontroller und Fahrzeug abreißt - das Betriebssystem benötigt einen gewissen Zeitraum um den Verbindungsabbruch zu registrieren - der Verlust wird aber nach 3-5 Sekunden erkannt und das Fahrzeug stoppt.
- In den Tests stellte sich heraus, dass jede noch so kleine Ungenauigkeit bei der Ausrichtung des LiDARs zu erheblichen Problemen führte. Es wurde daher eine kleine Wasserwaage am LiDAR angebracht um dessen Ausrichtung präzise vornehmen zu können.

4.6 Netzwerkinfrastruktur und VPN

Wie bereits erwähnt wurde eine erweiterte und dynamische Netzwerkinfrastruktur geschaffen. Zu Beginn des Projektes stand nur ein einfacher WLAN Router zur Verfügung, der lediglich an einem gezielt freigeschalteten Port innerhalb des OH16 betrieben werden konnte um eine Verbindung zum Internet herstellen zu

Abbildung 8: Router RB962UiGS-5HacT2HnT der Firma Mikrotik

können. Zwar wird für den Betrieb des Fahrzeuges kein aktiver Internetzugang benötigt, allerdings schränkt es den Workflow beim Testen und Entwickeln stark ein, wenn die Teilnehmer und das Fahrzeug jedes mal das WLAN-Netzwerk für einen Internetzugriff (Git) wechseln müssen. Da hierdurch keine Flexibilität des Ortes gegeben ist, wurde die Infrastruktur umfassend angepasst und erweitert (siehe Abbildung 5):

- Der Router wurde durch ein RB962UiGS [8] der Firma Mikrotik ersetzt. Dieser kann gleichzeitig 2 WLAN Verbindungen herstellen. Hiervon wurde eine für das interne WLAN für die Teilnehmer vor Ort und das Fahrzeug, die andere für den Uplink zum Internet über eine *Eduroam* Kennung genutzt. Da dieser Uplink überall auf dem Campus sowie am Fraunhofer-Institut für Materialfluss und Logistik (IML) zur Verfügung steht wurde die räumliche Flexibilität dadurch erhöht. Da es im Rahmen der Pandemie leider die überwiegende Zeit nicht mehr möglich war das Fahrzeug an genannten Standorten in Betrieb zu nehmen wurde zudem die Möglichkeit geschaffen den Router auch per LAN- oder WLAN-Uplink bei den Teilnehmern zu Hause in Betrieb zu nehmen.

- Da es im Rahmen der Pandemie weder ratsam und nur unter sehr strengen Auflagen erlaubt war mit mehrerer Teilnehmern physisch an einem Ort zu testen und das genehmigte Hygienekonzept (siehe Anhang I) leider kein praktikables Testen erlaubte musste eine Lösung gefunden werden, bei der die Anwesenheit einer Person ausreicht. Hierfür wurde ein VPN realisiert: Der Router stellte bei aktiver Internetverbindung einen VPN-Tunnel zu dem genannten PG-Server her. Die Teilnehmer konnten per VPN eine Verbindung zu dem Server herstellen und waren dann per L2-Bridging aus Netzwerksicht physikalisch mit dem internen Netz und damit mit dem Fahrzeug verbunden. Abseits einer leicht gesteigerten Latenz gab es somit keinen Unterschied zwischen lokalen und per VPN verbundenen Teilnehmern. Es war damit beispielsweise jedem möglich Einstellungen an den Fahralgorithmen während der Fahrt vorzunehmen und detaillierte

[8] https://mikrotik.com/product/RB962UiGS-5HacT2HnT

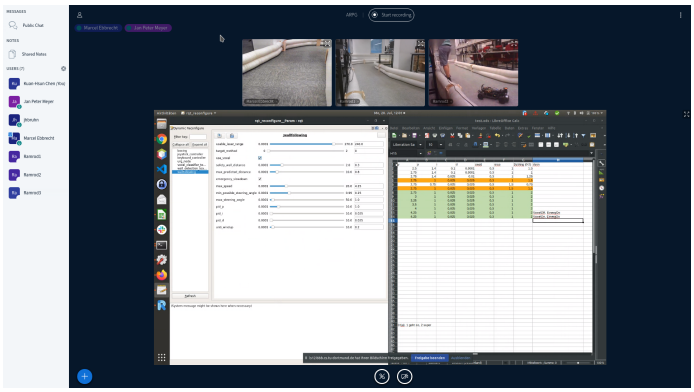


Abbildung 9: Screenshot BigBlueButton Testphase

Telemetriedaten zu erhalten. Hierdurch wurde trotz Pandemie eine effiziente Zusammenarbeit während der abschließenden Testphase erst möglich. Durch diese Erweiterung konnte die Infrastruktur praktisch überall dort eingesetzt werden, wo ein Uplink via LAN oder WLAN ans Internet gegeben war.

- Der Lehrstuhl stellte einen Zugang auf dem vorhandenen BigBlueButton Konferenzserver[9] zur Verfügung. Dieser wurde regelmäßig für Besprechungen und in der Testphase genutzt. Während der Tests wurden mehrere Notebooks mit Kameras um die Strecke positioniert, damit die per VPN verbundenen Teilnehmer einen Eindruck von der Fahrweise und damit der Auswirkung vorgenommener Änderungen am Code und an Parametern gewinnen konnten (siehe Abbildung 9).

[9] https://ls12-bbb.cs.tu-dortmund.de/b

5 Projektmanagement

Projektmanagement befasst sich mit der Planung und Steuerung von Prozessen in Projekten [7]. Wie in Abbildung 10 zu sehen ist, beginnt der Prozess des

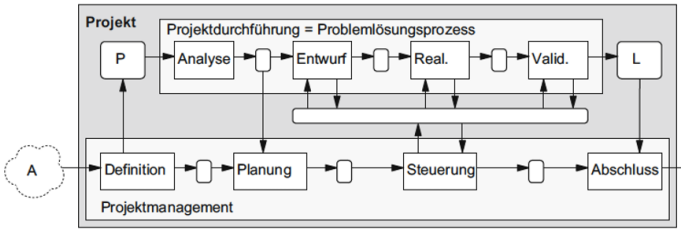


Abbildung 10: Projektablauf [15]

Projektmanagements vor der eigentlichen Projektarbeit. Nach dem Abschluss der Planung, befasst sich das Projektmanagement mit der Steuerung. Gesteuert werden Projekte in den folgenden Punkten:

- Problemlösungsprozess
- Zielsysteme
- Aufwandsschätzung
- Qualität
- Ablauf
- Risiken
- Kosten
- Projektbeteiligte

5.1 Projektorganisation

Dieses Kapitel behandelt die Organisation von Projektarbeit. Dabei wird insbesondere auf *Srumming* und *Kanban* eingegangen.

5.1.1 Scrum

Scrum ist ein sehr populäres Framework für agile Softwareprojekte [13]. Die Vorteile von Scrum liegen in den kurzen Iterationszyklen und der inkrementellen Produktentwicklung, wodurch bei korrektem Einsatz eine hohe Qualität und ein sehr dynamischer Entwicklungsprozess erreicht werden kann [19]. Die wesentlichen Elemente von Scrum lauten:

- Product Backlog: Hier werden alle Userstories die noch entwickelt werden müssen gespeichert.
- Sprint Backlog: Hier befinden sich alle Userstories die in diesem Sprint entwickelt werden.
- Sprint: Ein Sprint ist ein Zeitraum von ca. 2-4 Wochen in dem eine Iteration des Entwicklungsprozesses umgesetzt wird.
- Daily Scrum: Ein tägliches Meeting, an dem alle Projektmitglieder ihren Fortschritt und die Probleme auf die sie gestoßen sind, berichten.
- Increment: Hier werden die am Ende des Sprints fertigen Softwareprodukte gesammelt.
- Retrospective: Nach jedem Sprint werden hier positive und negative Erfahrungen festgehalten.

5.1.2 Kanban

Im Gegensatz zu Scrum definiert Kanban nicht ein genaues Vorgehensmodell, es bietet vielmehr eine Visualisierung der Arbeitsschritte und den Status aller aktuellen Aufgaben [3]. Das Hauptelement von Kanban ist das Kanbanboard. Das Kanbanboard enthält als Spalten die einzelnen Arbeitsschritte, also den Workflow. Elemente der Spalten sind jeweils Aufgaben die sich in diesem Arbeitsschritt befinden.

5.2 Vorgehensweise der Projektgruppe

In der Projektgruppe wurde sich darauf geeinigt Kanban zu nutzen. Dieses Vorgehensmodell ist sehr gut mit Github kombinierbar, da es über dort vorhandenen Funktionen direkt visualisiert werden kann. Darüber hinaus bietet es die Möglichkeit seine Arbeitszeiten frei zu verteilen, da es keine täglichen Besprechungen wie bei Scrum gibt. Meetings haben einmal in der Woche stattgefunden. Dort wurde der Fortschritt der Aufgaben besprochen, sowie neue Aufgaben verteilt. Außerdem wurden während der Meetings verschiedene Algortihmen und das weitere Verfahren diskutiert.

5.3 Phasenplan

Um ein gutes Zeitmanagement gewährleisten zu können, wurde ein Phasenplan aufgestellt. Dieser Phasenplan umfasst 10 Phasen. Für jede dieser Phasen wurde ein Zeitraum festgelegt. In der Einführung wurden folgende 3 Phasen definiert.

- Planung: Grobe Planung des Projektablaufs
- Einarbeitung: das vorhandene Verstehen
- Grundlagen schaffen: Die Grundlagen der Projektumgebung lernen

Dabei wurde darauf abgezielt, jedem Gruppenmitglied die Grundlagen der verwendeten Programmiersprachen, physikalische Konzepte sowie bestehende Algorithmen näher zu bringen.

Anschließend wurde der erste Teil des Projektgruppenziels erfasst.

- Erkennung von statischen Hindernissen
- Anpassung der Fahrweise um diesen ausweichen zu können
- Zwischenbericht

Die statische Objekterkennung ist die erste Hinderniserkennung, mit der sich die Projektgruppe beschäftigt hat. Ziel ist es, sich nicht bewegende Objekte durch die Verwendung der gegebenen Sensoren autonom als Hindernisse und nicht als Teil der Strecke zu erkennen. Anschließend soll eine Strategie zur Umfahrung dieser Hindernisse implementiert werden. Mit dem Zwischenbericht endet das erste Semester der Projektgruppe. Für das 2. Semester ist die dynamische Hinderniserkennung analog zur Vorgehensweise bei der statischen Erkennung und der Abschlussbericht und damit das Ende der Projektgruppe vorgesehen.

5.4 Gantt

Ein Gantt-Chart [12] gibt den zeitgleichen Ablauf eines Projektes wieder. In diesem Fall stellen die Ziffern die Monate dar. Das Projekt wurde in 2 Sektionen gegliedert, das erste und das zweite Semester. Die Aufgaben innerhalb dieser Sektionen sind mit Beginn und Ende im jeweiligen Monat angesiedelt. In Abbildung 11 ist das Gantt-Chart der Projektgruppe aufgeführt. Zu Beginn wurde dieses erstellt und im Laufe der Zeit angepasst, so daass Abbildung 11 nun die finale Version ist. Die Zeitplanung musste auf Grund eines nicht optimal funktionierenden Algorithmus und der Interventionen rund um das Coronavirus geändert werden. Daher konnte das Ziel der Dynamischen Hindernisserkennung nicht verflogt werden und der Fokus lag auf der Optimierung der Fahralgorithmen und dem Erkennen von Hindernissen.

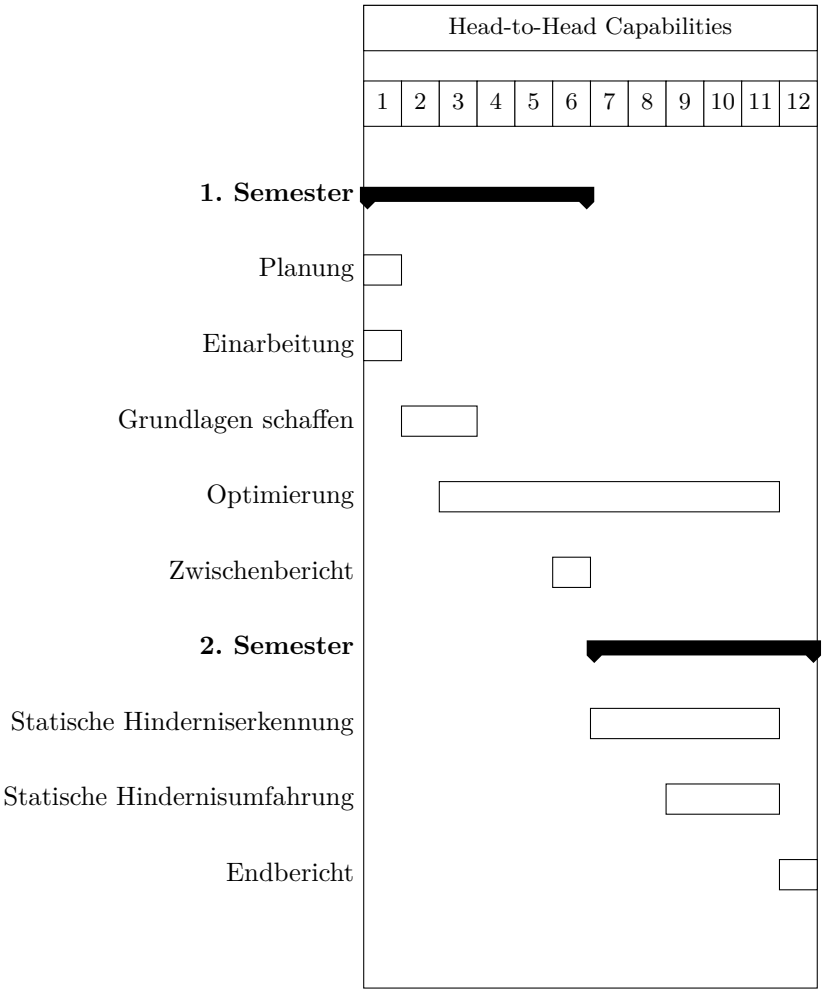


Abbildung 11: Phasenplan dieser Projektgruppe

5.5 Umgang mit dem Coronavirus

Durch die Pandemie und die daraus resultierenden Einschränkungen rund um das Coronavirus musste sich die Arbeitsweise der Projektgruppe ändern. Ab dem 13.03.2020 galt ein allgemeines Betretungsverbot der Universität für Studierende. In Folge dessen, konnten keine regulären Projektgruppen Meetings mehr stattfinden. Jedoch wurde uns ein Konferenzserver vom Lehrstuhl 12 zur Verfügung gestellt, über den die Meetings in digitaler Form stattfinden konnten. Die Inhalte und wie kommuniziert wurde, änderte sich jedoch. So waren diese Meetings nicht mehr so produktiv wie die Meetings, die in der Universität stattgefunden haben. Zudem fiel der physische Zugang zu der Teststrecke weg. Ein Aufbau einer Teststrecke bei einem der Teilnehmer konnte zwar ein minimales Testen in der Realität ermöglichen, im großen und ganzen war diese Strecke jedoch zu klein und zu eng. Die Hindernisumfahrung konnte dort nicht getestet werden, da das Auto so schon kaum durch die enge Strecke fahren konnte. Ein Ausschnitt der Teststrecke ist in Abbildung Abbildung 12 festgehalten.

Abbildung 12: provisorische Teststrecke

6 Modellbildung

Da nicht zu jeder Zeit die Gelegenheit besteht eine Rennstrecke für das Auto aufzubauen und alle Implementierungen die vorgenommen wurden real zu testen, wird eine simulierte Umgebung benötigt. Grade in Zeiten von Corona hat sich die Simulation als wichtiger denn je herausgestellt. Aktuelle Rennen der F1/10th Challenge werden nicht mehr real auf einer Strecke in einer Halle ausgetragen, sondern in einem eigens dafür entwickelten Simulator der Veranstalter. Dieser führt dann von den Teams bereitgestellte Docker-Container aus und wertet die daraus resultierenden Zeiten auf der vorgegebenen Rennstrecke aus. Bisher war die Projektgruppe auf den Gazebo Simulator angewiesen. Wie dieser bedient wird und welche Einschränkungen es gibt wird in diesem Kapitel behandelt. Gegen Ende der Projektgruppe wurde außerdem die Simulation in dem offiziellen Simulator ermöglicht.

6.1 Streckensimulation

Die Simulationsumgebung Gazebo bietet die Möglichkeit ROS direkt zu integrieren und damit zu interagieren. Hierzu muss eine entsprechende Umgebung modelliert werden. Dazu gehört das Auto selbst, verschiedene Sensoren, die es meist schon als Plugin gibt, und die Umgebung in der sich das Auto bewegen soll.

Die Modellierung des Autos konnte von der vorangegangenen Projektgruppe übernommen werden. Sie befindet sich im Ordner `simulation` unter `racer_description` und besteht aus mehreren `.dae`-Dateien welche über das Design-Tool Blender erstellt wurden. Zusammengesetzt werden die einzelnen Teile dann in der Datei `racer.xacro`. Die Steuerbefehle von ROS werden dann von den Dateien in `racer_control` verarbeitet und an das Modell in Gazebo weitergeleitet. In dem Ordner `racer_world` befinden sich zudem verschiedene Welten in der die Simulation gestartet werden kann. Diese können ebenfalls in Blender geladen und manipuliert werden. So wurde beispielsweise auch die Standardstrecke um ein Hindernis erweitert, welches das Auto erkennen und umfahren soll. Es besteht auch die Möglichkeit mehrere Modelle des Autos separat zu simulieren und diese dann gegeneinander antreten zu lassen. Des Weiteren wurde die Anzahl der Welten erhöht und die direkte Auswahl über unser Startskript ermöglicht. So stehen der Projektgruppe nun insgesamt 13 Rennstrecken zur Verfügung.

In RViz, dem ROS-eigenen Visualisierungstool, werden dem Benutzer verschiedene Telemetriewerte angezeigt. Es werden Sensorwerte dargestellt und es können aus den ROS-Nodes heraus an verschiedenen Stellen Befehle zum Zeich-

nen von Punkten, Geraden, Kurven und anderen geometrischen Formen an RViz gesendet werden. So werden verschiedene kumulierte Messwerte und Zwischenergebnisse der autonomen Algorithmen grafisch dargestellt, um besser nachvollziehen zu können wie und warum das Auto in bestimmten Situationen seine Entscheidungen trifft.

ROS bietet außerdem die Funktion, die Ausgaben vorher definierter Nodes während einer Simulation oder sogar während des Betriebes in der realen Welt in sogenannten Rosbags aufzuzeichnen. Diese Aufzeichnungen lassen sich dann in Gazebo und Rviz in Echtzeit wieder abspielen. Auch das Pausieren und verlangsamte Wiedergeben ist möglich. Besonders nützlich ist dabei die Funktion, dass auf den aufgezeichneten Sensordaten andere und angepasste Algorithmen ausgeführt werden können als es im Original der Fall war. So kann eine Fahrt aufgezeichnet werden und das Verhalten des Autos im Nachhinein analysiert, verändert und verbessert werden [27].

6.2 Physikalische Eigenschaften

Natürlich weicht die Simulation in bestimmten Bereichen von der Wirklichkeit ab. Um diese Abweichung so gering wie möglich zu halten und auf bestimmte Bedürfnisse anzupassen, ist es möglich, den verschiedenen Komponenten unterschiedliche Materialien und damit einhergehend unterschiedliche physikalische Eigenschaften zuzuweisen. So haben die aus Gummi bestehenden Reifen des simulierten Autos beispielsweise einen Reibwert von $\mu = 0.9$, damit sie sich auf der Strecke möglichst ähnlich zu echten Gummireifen auf asphaltartigem Untergrund verhalten [25]. Wie bereits in Unterabschnitt 2.2 erläutert wurden diesbezüglich eine Änderungen am Code vorgenommen. Die Aerodynamik fand keine Berücksichtigung, da bei den im Rahmen der Tests erreichten Geschwindigkeiten kein nennenswerter Einfluss festzustellen war.

6.3 Teststrecken

Dieser Abschnitt befasst sich mit den verwendeten Bauteilen und der aufgebauten Teststrecken im Rahmen eines Zwischentests am IML und des Abschlusstests im HaPra Raum im Gebäude OH16.

Abbildung 13: Lüftungsschlauch als Streckenbegrenzung

6.3.1 Bauteile

Grundsätzlich wurden für den Aufbau der Strecke zwingend zwei Komponenten benötigt, zwei weitere werden dringend empfohlen:

- Die Wände wurden aus zwei Lagen flexiblen, weißen Lüftungsschläuchen gebildet (siehe Abbildung 13) wie sie normalerweise für die Abluft von Klimaanlagen verwendet werden. Diese hatten folgende Vorteile:
  - Sie wiesen durch ihre weiße, matte Oberfläche ein gutes Reflexionsverhalten auf, sodass das LiDAR zuverlässige und genaue Entfernungsdaten liefern konnte.
  - Die Schläuche ließen sich einfach auf die gewünschte Länge ziehen und eine gewisse Flexibilität bezüglich der Streckengeometrie zu.
- Zur vertikalen Fixierung der Schläuche kamen U-förmige, durch die Vorgängergruppe selbst gebaute Holzelemente zum Einsatz um ein Verrutschen zu verhindern.
- Da es durch Kollisionen des Fahrzeugs mit der Begrenzung oder durch zu viel Spannung auf diesen immer wieder Instandsetzungen nötig waren, bewährte es sich im Rahmen der Abschlusstests die Teile mit ausreichend und qualitativ hochwertigem Klebeband zu verbinden. Stellen an denen das Fahrzeug sehr oft aus der Strecke ausbrach wurden zusätzlich mit Kisten aus Kunststoff versehen (siehe Abbildung 16 und Abbildung 17).
- Zudem hat es sich bewährt flache Styroporplatten als Streckenbegrenzung zu verwenden. Diese weisen ebenfalls eine matte Oberfläche auf und sind zudem Kostengünstig und flexibel einsetzbar. Außerdem verhindern diese besser eine Beschädigung des Autos bei einer Kollision, da das Auto bei den Schläuchen zwischen den zwei Lagen wenig geschützt ist.

6.3.2 Teststrecke IML

Im Rahmen der Projektgruppe wurde eine Halle am Fraunhofer IML bzw. dem Lehrstuhl für Förder- und Lagerwesen organisiert, in der eine Rennstrecke aufbaut werden konnte. Das Feld maß etwa $10\,\mathrm{m} \times 20\,\mathrm{m}$ und wurde von Banden umzäunt. Es stand jedoch nicht zu jeder Zeit das gesamte Feld zur Verfügung, da hier nebenher auch an anderen Projekten gearbeitet wurde und die Materialien meist am Rande des Feldes abgedeckt deponiert wurden. Es standen neben dem Feld außerdem noch andere Bandenelemente (siehe Abbildung 14) bereit. Mit diesen und den eigenen Elementen konnte eine Rennstrecke aufgebaut werden. Beim ersten Versuch wurde festgestellt, dass die schwarzen Elemente zu sehr spiegelten und das LiDAR sie somit nicht erkennen konnte. Daher wurden die Elemente mit verschiedenen schwarzen Decken abgedeckt, woraufhin eine zuverlässige Erkennung wieder möglich wurde. Daraufhin neigten die beschwerten Elemente jedoch zum Kippen. Schlussendlich konnte das Auto ein paar Runden auf der Strecke fahren, bevor wieder alles abgebaut und sauber verstaut wurde. Leider entfiel diese Testmöglichkeit mit Beginn der Pandemie.

6.3.3 Fazit IML Test

In der frühen Phase der Projektgruppe stand die Halle noch nicht zur Verfügung, somit wurde der Fokus auf die Simulation gelegt und alles was entwickelt wurde nur darin getestet. Mit dem Zugang zur Halle, fuhr das Auto in der Simulation sehr zuverlässig und schnell. Es konnte sogar einige Kurven schneiden. Beim ersten Test in der echten Welt musste allerdings sehr schnell festgestellt werden, dass es eben nur in der Simulation gut fuhr. Nachdem das Auto mit leichten Modifikationen der Streckenelemente diese endlich erkennen konnte, fuhr es dennoch sehr langsam und nahm die Kurven so scharf, dass es oft die Banden berührte bzw. in den 90-Grad Kurven so stark einlenkte, dass es mit dem hinteren Teil an der Ecke hängen blieb. Analysen ergaben, dass es sich dabei um ein generelles Problem beim zustandslosen Fahren handelt was sich im weiteren Verlauf des Projektes bestätige, da auch eine andere Gruppe dieses Problem kannte[10]. Es reichte also absolut nicht aus sich nur auf die Simulation zu verlassen, ständige Tests und Überprüfungen auf der echten Hardware auf einer Teststrecke sind für den Erfolg einer Roboterentwicklung unerlässlich.

[10] `https://www.nathanotterness.com/2019/04/the-disparity-extender-algorithm-and.html`

Abbildung 14: Versuchsaufbau am IML

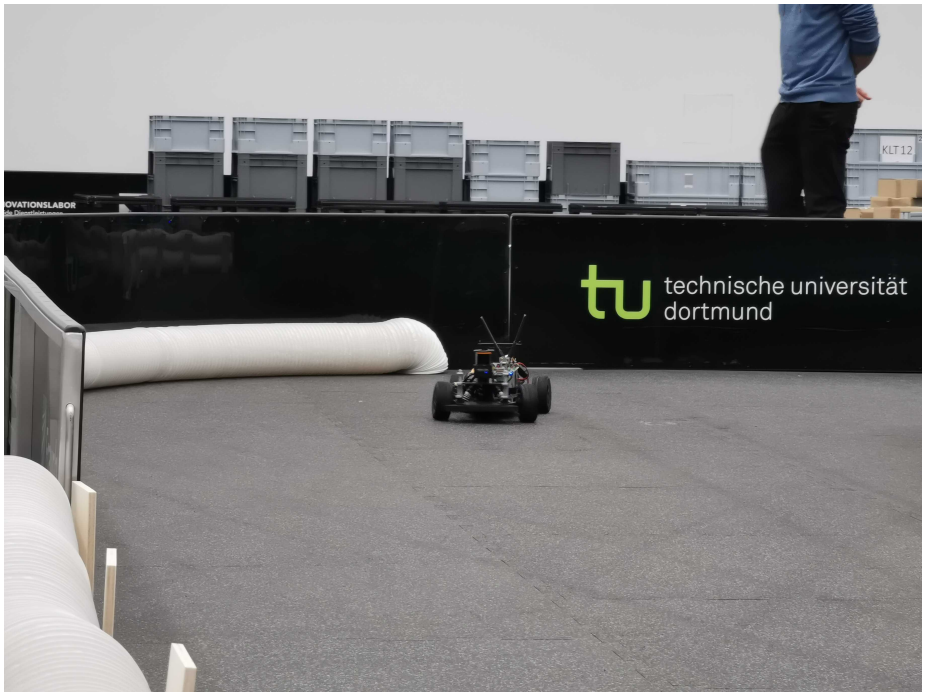


Abbildung 15: Fahrzeug auf der IML Teststrecke

Abbildung 16: Versuchsaufbau im OH16

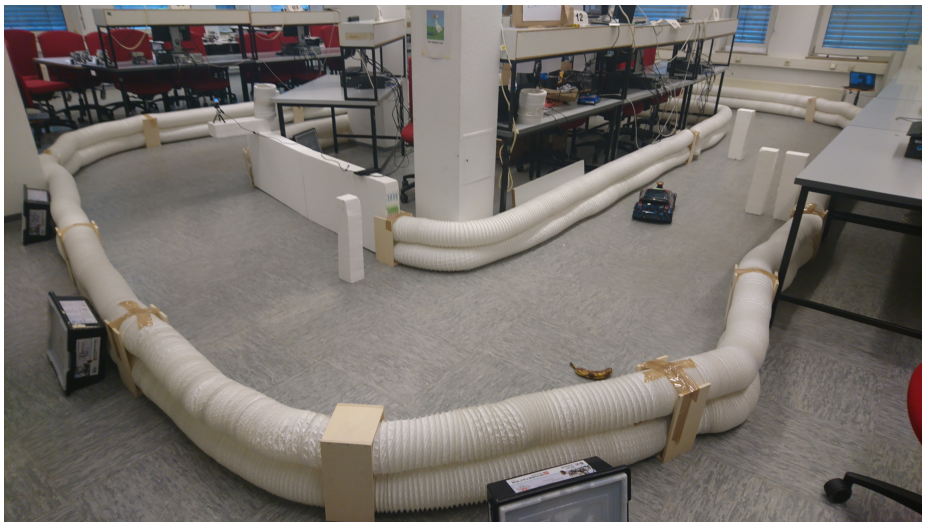


Abbildung 17: Versuchsaufbau im OH16 mit Hindernissen

6.3.4 Streckenplanung

Für die Abschlusstests stellten sich mehrere Fragen von denen hier zwei behandelt werden:

- Welche Leistung sollte erreicht werden bzw. welche Fahrgeschwindigkeit ist realistisch?
- Welcher Platzbedarf ergibt sich aus diesen Anforderungen?

Es ist erkennbar, dass beide Fragen voneinander abhängig sind. Um einen Eindruck zu gewinnen wurde ein Programm entwickelt das es ermöglicht eine Teststrecke ansatzweise zu modellieren[11]. Es bietet folgende Eingaben:

- Die Erdbeschleunigung g, welche im Normalfall bei 9.81 belassen wird
- Den zu erwarteten Reibungskoeffizienten für die Haftreibung μ
- Verschiedene Maße der Strecke
- Eine Angabe zur Auflösung, die die Größe der erzeugten Zeichnung ergibt

Aus diesen Eingaben werden verschiedene Geschwindigkeiten errechnet, die das Fahrzeug an bestimmten Punkten der Strecke erreichen kann. Hier ist allerdings zu beachten, dass hierzu ausschließlich die physikalischen Gesetze bzw.

[11] https://arpg-sophisticated.github.io/stuff/trackcalculator.html

ARPG Track Calculator

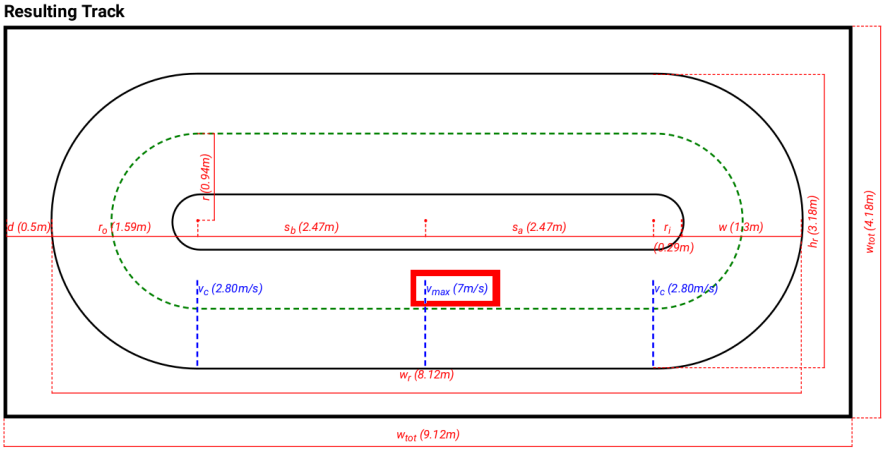
Input

9,81	m/s^2
Acceleration g	
0,85	
Coefficient of friction μ	
1.3	m
Track width w on straight	
0.5	m
Outer distance d of track	
0.4	
Speed ratio curves - 0.2 ... 1.0 (circle)	
7	m/s
Target maximum speed v_{max}	
100	px/m
Pixel per m	

Calculated

2,80	m/s	8,12	m
Speed in/out curve v_c		Width of racetrack w_r	
4,20	m/s	3,18	m
Speed gain on straight v_{gain}		Height of racetrack h_r	
0,94	m	9,12	m
Curve radius r		Overall width needed w_{tot}	
1,59	m	4,18	m
Curve radius (outer) r_o		Overall height needed h_{tot}	
0,29	m	15,79	m
Curve radius (inner) r_i		Length drive path c_d	
8,34	m/s^2	19,87	m
Effective acceleration a_{eff}		Circumfence outer wall c_o	
2,47	m	11,70	m
Acceleration/Break length s_a, s_b		Circumfence inner wall c_i	

Resulting Track



We assume CCW driving. Please scale to fit one page on printing.

Abbildung 18: ARPG Track Calculator

die Geschwindigkeit und die Beschleunigung, wie sie in Anhang C behandelt werden, genutzt werden. Eine Berücksichtigung der Fahrzeugeigenschaften oder bestimmter Software-Parameter findet nicht statt.

6.4 Teststrecke OH16

Für die Abschlusstests wurde eine Teststrecke im HaPra-Raum im Gebäude OH16 eingerichtet(Abbildung 16 und Abbildung 17). Die Strecke ähnelt prinzipiell der theoretischen Berechnung, wie sie in Abbildung 18 zu sehen ist. Leider können mit den Generator die kurzen Geraden nicht dargestellt werden, dennoch schien die maximal erreichbare Geschwindigkeit von $7\frac{m}{s}$ ein realistischer Wert für die lange Gerade zu sein, was sich im Laufe der Tests bestätigen sollte. Die Tests erfolgten mit und ohne Hindernissen.

7 Software

Die Softwareinfrastruktur des Projektes lässt sich in zwei größere Blöcke unterteilen. Der erste befasst sich mit der Sensordatenerfassung und -aufbereitung, der zweite Block mit der Nutzung dieser Daten zum Steuern des Autos. Bei der Datenerfassung und -aufbereitung geht es zu Beginn um das physische Ermitteln der Daten durch die Tiefenkamera und das LiDAR in den Nodes `LiDAR` und `Cam`. Diese Daten werden im nächsten Schritt, der Datenaufbereitung, mit den Daten der verschiedenen Sensoren miteinander fusioniert. Nachdem im ersten Block die notwendigen Daten erfasst und fusioniert wurden, wird zunächst eine Datenreduktion und eine Verminderung von Störrauschen durch sogenanntes `boxing` berechnet. Anschließend werden aus den einzelnen Datenpunkte zusammengeführte Voxel gebildet. Auf den entstandenen Voxeln werden Cluster ermittelt und klassifiziert. Das Ende der Verarbeitungspipeline bildet der Wallfollowing Algorithmus, der die erfassten und aufbereiteten Daten nutzt um die Trajektorie zu berechnen.

7.1 Implementierung mit C++

Alle in diesem Kapitel vorgestellten Algorithmen sind in C++ geschrieben. Dies ist insbesondere von Vorteil, da ressourcenbeschränkte und batteriebetriebene Hardware verwendet wird, bei der es entscheidend sein kann, möglichst effizient und ressourcenschonend zu arbeiten. Zudem lassen sich komplexere Funktionalitäten realisieren, die bei einer Implementierung mit beispielsweise Python nicht möglich wären, da eine zu hohe Latenz entstünde. Möglichst geringe Latenz zu erreichen, gehört zu den Hauptanforderungen der Fahralgorithmen. Dabei kann eine höhere Geschwindigkeit erreicht werden, wenn die Umgebung schnell genug verarbeitet wird. Ein weiterer Vorteil gegenüber Python ist die Fehlererkennung schon während des Kompilierens. Damit kann Zeit gespart werden, da Syntax- und andere Fehler nicht erst nach dem Start des Programms auftreten. Insbesondere bei der aufwendigen Simulation mit Gazebo macht sich dies bemerkbar.

7.2 Erkennungspipeline

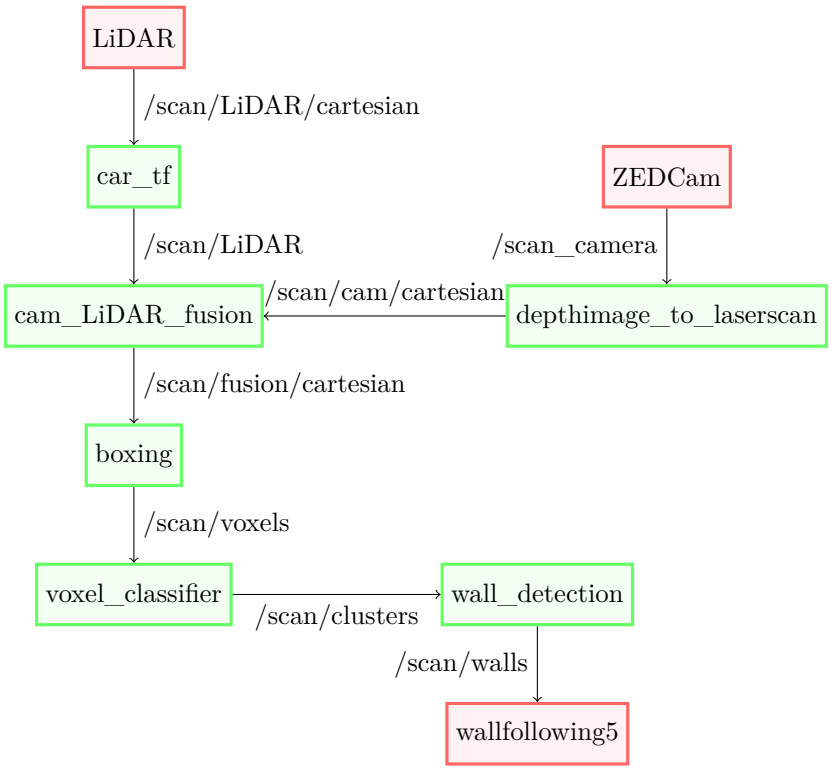


Abbildung 19: Verarbeitungspipeline für Sensordaten von LiDAR und Kamera.

Die Daten der verschiedenen Sensoren können nicht ohne Weiteres zur Steuerung verwendet werden. Bei den Tests hat sich herausgestellt, dass die LiDAR-Daten Ausreißer enthalten, die die zuverlässige Anwendung des Wallfollowing verhindern. Zudem sollen die LiDAR- und Kameradaten fusioniert werden. Abschließend sollen bei den Daten Wände und Hindernisse erkannt werden, darauf angepasst reagieren zu können.

In Abbildung 19 sind die Nodes (Knoten) und die Topics, über welche die Daten gereicht werden (Kanten) dargestellt. Der Datenfluss ist dabei von unten nach oben. In der Node `car_tf` werden zunächst die LiDAR Daten vom radialen System des LiDARs in das von ROS verwendete kartesisches Koordinatensystem weitergegeben. In `depthimage_to_laserscan` wird die 3D-Punktwolke der ZEDCam in ein 2D-Laserscan-artiges Format überführt, um mit den Daten der ZEDCam einfach weiterzuarbeiten. Dabei wird nur ein Streifen auf einer Z-Höhe aus dem 3D-Scan herausgeschnitten. Danach sollen die Kamera- und Sensorpunktwolken in der Node `cam_LiDAR_fusion` fusioniert werden. In unserem Fall ist dies eine Konkatenation der beiden Punktwolken. In der Node `boxing` werden aus den Punktwolken Voxel gebildet, welche in `voxel_classifier` mit Hilfe des DBScan Algorithmus geclustert werden. Diesen Clustern wird in `wall_detection` eine Klasse zugewiesen. Je nach Clustertyp werden dann entweder eine oder mehrere Wände erkannt, oder aus den Clustern Hindernisinformationen extrahiert. Diese Informationen werden letztendlich an die Node `wallfollowing5` (siehe Unterabschnitt 7.3) weitergegeben, welche die Steuerung des Autos übernimmt.

7.2.1 ZED-Camera

In der Simulation lieferte der LiDAR-Sensor immer korrekte Werte und die Fahralgorithmen arbeiten zu jeder Zeit wie erwartet. Leider zeigten Tests mit dem Fahrzeug, dass diese Zuverlässigkeit außerhalb der Simulation nicht immer gegeben war. So kam es zum Beispiel bei einem Fahrtest, bei dem stark reflektierende Banden als Fahrbahnbegrenzung zum Einsatz kamen, zu ständigen Fehlerkennungen seitens des LiDARs und dementsprechend unzureichenden Ergebnissen bezüglich des Fahrverhaltens, z.B. zu spätes oder zu frühes Bremsen, Nichterkennung von Hindernissen oder Schlingern. Diese Probleme wurden noch verstärkt, wenn das Material der Begrenzungen nicht homogen war. Um diesen Effekten begegnen zu können und eine höhere Zuverlässigkeit der Umgebungserkennung zu erreichen wurde geplant, die Kamera als zusätzlichen Sensor zu verwenden. Zum einen sollte durch einen Abgleich beider Datenquellen ein Ausschluss unwahrscheinlicher Messwerte erreicht werden, zum anderen sollte der Kamera bei der Erkennung dynamischer Hindernisse eine zentrale Rolle zukommen. Wie in Unterunterabschnitt 4.5.2 erläutert konnten viele dieser Probleme auf eine fehlerhafte Ausrichtung des LiDARs zurückgeführt werden wofür eine

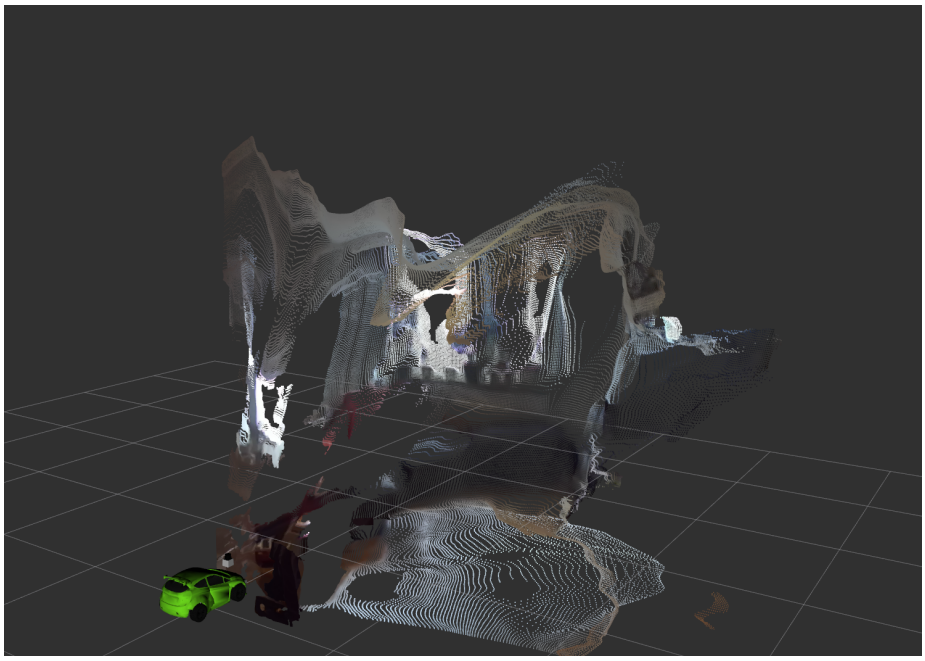


Abbildung 20: Point-Cloud der ZED Kamera

hardwareseitige Lösung entwickelt wurde. Andere Probleme erübrigten sich im Laufe des Projektes. Dennoch soll hier ein Überblick über die Erfahrungen bei der Nutzung der Kamera gegeben werden.

Die Inbetriebnahme der Kamera erfolgt per Installation von CUDA auf dem System, der Installation des ZED SDK[12] und der Integration des ZED ROS Wrappers[13]. Anschließend stehen verschiedene Nodes zur Verfügung, die sowohl Kamerabilder, als auch Tiefendaten in Form einer Depth-Cloud bzw. Point-Cloud (siehe Abbildung 20) liefern und leicht mit vorhandenen Mitteln in RViz und eigene Klassen in ROS integriert werden können. Dies ermöglicht zum einen die komfortable 3D-Erfassung der Umgebung, zum anderen auch einen Live-Stream der Bilder zum Controller, sodass man virtuell mitfahren kann. Um einen Abgleich mit den LiDAR-Daten zu ermöglichen wurde ein Transform-Filter erstellt, der einen Laserscan auf der Höhe des LiDARs aus den Tiefendaten emuliert. Grundsätzlich bietet der ZED ROS Wrapper auch die Möglichkeit, die Tiefendaten zu akkumulieren und damit während des Betriebs eine 3D Umgebungskarte zu erstellen, die man als Basis für einen Fahralgorithmus nach

[12] https://www.stereolabs.com/developers/release/
[13] https://github.com/stereolabs/zed-ros-wrapper/tree/master/zed_wrapper

dem SLAM-Prinzip [32] nutzen kann, jedoch übersteigt dies bei weitem die Rechenkapazität des verwendeten Nvidia Jetson Boards.

Leider sind die Tiefendaten der Kamera ab einer gewissen Entfernung (ca. 2 bis 3 m) ungenau: Es wird immer eine kürzere Distanz angegeben als sie tatsächlich vorliegt. Zudem handelt es sich hierbei nicht um einen festen linearen, exponentiellen oder anderen berechenbaren Fehler.

Einem weiteren Problem, steigender Ungenauigkeit am Rand des Sichtfeldes, sollte durch einen Filter begegnet werden, der die Verwendung der Daten auf einen möglichst zuverlässigen Bereich einschränken sollte. Hierbei handelte es sich nicht um ein technisches Problem der Hard- oder Software, sondern um ein immanentes Phänomen, das mit der Stereoskopie und der paralaxischen Vermessung verbunden ist: Je geringer der Abstand zwischen den Sensoren, desto geringer das Sichtfeld in dem Entfernungen sicher bestimmt werden können [33].

Um die Entwicklung von Software zur Auswertung von Tiefendaten auch in der Simulation durchführen zu können, wurde auf das ROS-Kinect Modul[14] zurückgegriffen, da es sich auf die Parameter der ZED Kamera konfigurieren lässt. Leider werden die Tiefendaten des genannten Moduls jeweils um 90° um die X- und Z-Achse verdreht ausgegeben, sodass eine Transformation erforderlich ist. Dies wurde mittels einer eigenen Node realisiert wobei hier eine bestehende Funktion von ROS genutzt wurden (siehe Listing 1).

```
<node pkg="tf" type="static_transform_publisher" name=
    "depth_rotation" args="0.18 0 0.14 -1.570796327 0
    -1.570796327 /base_link /camera 10" />
```

Listing 1: Node für die Transformation der Ausgabe des Kinect-Moduls

Die erfassten Tiefendaten werden auf Höhe der durch das LiDAR erfassten Daten nach dem in Unterabschnitt 2.1 genannten Verfahren in einen Laserscan konvertiert und bereitgestellt, sodass eine zweite, redundante Quelle für die Entfernungsmessung zur Verfügung steht.

In der Praxis wurde sich allerdings dazu entschieden, die ZEDCam nicht einzusetzen. Einerseits werden die Messdaten ab einem gewissen Abstand ungenau, andererseits ist die Kamera schlicht zu langsam. Obwohl der Hersteller von bis zu 60 FPS spricht, sind die Messwerte bei unseren Tests weitaus langsamer angekommen. Zudem benötigt die Kamera sehr viel Rechenleistung um die stereoskopischen Bilder auszuwerten. Da die Verarbeitung ohnehin schon viele Daten verarbeiten muss, wird die benötigte Rechenleistung also in der gesamten Verarbeitungspipeline nicht geringer.

[14] http://wiki.ros.org/kinect

Es wurde in den Boxing- und Cluster-Nodes auch Unterstützung von farbigen 3D-Punktwolken implementiert, um verschiedenfarbige Punkte verarbeiten zu können, wodurch Hindernisse eventuell besser von Wänden unterschieden werden könnten. Dies war jedoch so rechenaufwendig, dass der Ansatz am Ende nicht eingesetzt wurde.

7.2.2 Voxelbildung (Boxing)

Um Ausreißer zu erkennen und gut zusammenhängende Bereiche zu Clustern wird die ermittelte Punktwolke zunächst in sog. Voxel quantisiert. Die Voxel bilden dabei ein Raster über das Koordinatensystem, wobei die Auflösung für das gesamte Koordinatensystem einheitlich angepasst werden kann. Ein Voxel ist aktiv, wenn mindestens n Punkte in diesem Cluster quantisiert werden. Die Beschränkung der Anzahl ist der erste Schritt zur Erkennung von Ausreißern: Befinden sich in einem Voxel zu wenige Punkte, werden diese verworfen. Entscheidend ist es, den Parameter n richtig zu wählen: Ist er zu hoch, werden weit weg liegende Punkte zu schnell verworfen, denn diese streuen stärker als Punkte welche nah am Auto erkannt werden, ist er zu klein, steigt die Störanfälligkeit. Zudem wurde aus der pointcloud-Bibliothek eine Ausreißerentfernung, basierend auf einem statistischen Verfahren, integriert.

In Abbildung 21 ist ein Bildschirmfoto aus dem ROS-Visualisierungstool *rviz* dargestellt, in dem die berechneten Voxel um das Auto eingeblendet sind. Erkennbar wird vor allem, dass entlang der Wände viele direkt benachbarte Voxel sind. Diese Vereinfachen dem später folgendem Clustering - Algorithmus das Erkennen von Wänden und Hindernissen. Zudem reduziert das Boxing die Menge der zur verarbeitenden Daten erheblich, wodurch eine verbesserte Pipeline mit weniger Performanceeinbußen implementiert werden kann. In einem weiteren Schritt soll zukünftig jedem Cluster ein minimaler Score zugeordnet werden, der abhängig von seiner Distanz zum Ursprung, also dem Auto, ist.

7.2.3 Clustering (DBScan)

Nach Prof. Schubert ist die Definition des Begriffs Clustering nicht eindeutig [29]. Es existieren verschiedene Modelle und Prinzipien um Cluster zu ermitteln, die sich teils stark voneinander unterscheiden. Selbst die Bestimmung, ob das Ergebnis eines Clustering-Algorithmus valide ist, sei abhängig vom Betrachter.

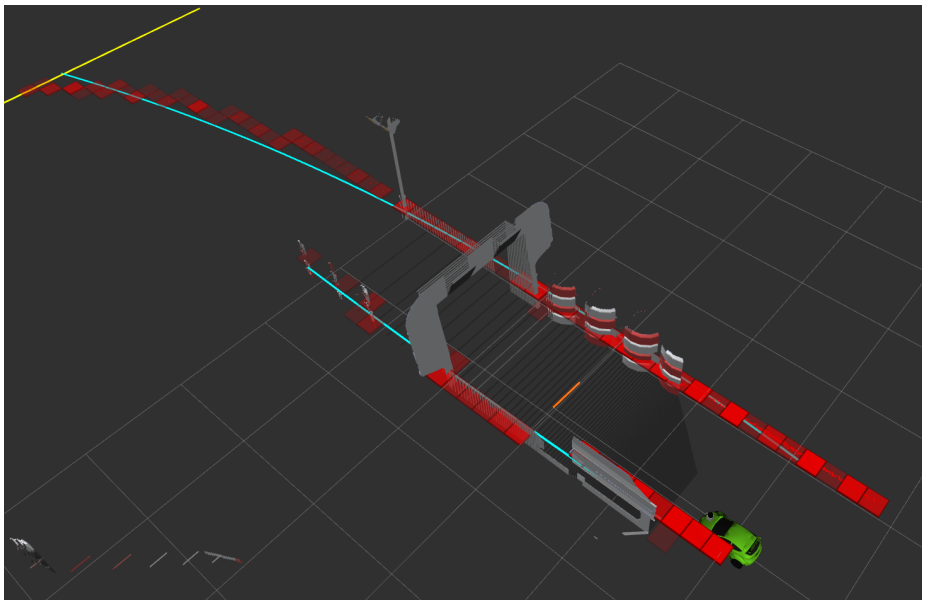


Abbildung 21: Punktwolke des LiDARs (rote Punkte) und daraus resultierende Voxel (rote Rechtecke, Score als Transparenz) gemessen um ein in Gazebo simuliertes Auto.

Für Clustering soll nach Schubert im Folgenden eine Trennung von Daten die folgende Kriterien erfüllen:

- Ähnliche Datenpunkte sollen in einem Clustern liegen
- Nicht ähnliche Datenpunkte sollten in unterschiedlichen Clustern liegen
- Die Datenpunkte sind nicht vorher definiert
- Cluster besitzen folgende Eigenschaften:
  - Connectivity
  - Separation
  - Least squared deviation
  - Density

Das Clustering zielt darauf ab die Voxel, die im vorherigen Verarbeitungsschritt entstanden sind, logisch zusammen zu führen, sodass Voxel, die Teil eines Objektes sind, z.B. einer Wand oder eines anderes Fahrzeug, dem selben Cluster angehören, um sie im nächsten Schritt klassifizieren zu können. Als Clustering-Algoritmus wird *DBScan* [8] genutzt. Hierbei handelt es sich um einen dichtebasierten Clustering-Ansatz. Der Algorithmus ist abhängig von den beiden

Parametern ϵ und *minPts*. Der Algorithmus klassifiziert alle in der Datenmenge enthaltenen Punkte in eine der folgenden Kategorien:

- Kernpunkt: Kernpunkte sind Datenpunkte deren Abstand zu mindestend *minPts* Punkten kleiner ist als ϵ.
- Dichte-erreichbarer Punkt: Ein Punkt ist dichte-erreichbar, wenn es einen Kernpunkt gibt, zu dem der Abstand kleiner als ϵ hat, der Punkt aber selbst kein Kernpunkt ist.
- Rauschpunkt: Ein Rauschpunkt ist ein Punkt der weder ein Kernpunkt, noch ein dichte-erreichbarer Punkt ist.

Anschließend können alle zusammenhängende Kernpunkte und dichte-erreichbaren Punkte zu jeweils einem Cluster zusammengefasst werden. Auf diese Weise kann eine beliebige Anzahl von Clustern erkannt werden. Die Tatsache, dass mit Hilfe des Algorithmus Cluster beliebiger Form erkannt [8] und Rauschpunkte heraus gefiltert werden können, mach DBScan für dieses Projekt besonders interessant.

7.2.4 Wall-Detection

Um den Clustern nun zuzuordnen, ob sie eine Wand oder ein Hindernis ist, wird davon ausgegangen, dass sich das Auto immer zwischen zwei Wänden befindet.

Ist der Cluster-Algorithmus gut parametrisiert und die Streckengeometrie in einem gut erkennbaren Zustand gibt es jeweils ein Cluster für die linke und rechte Wand und $n \geq 0$ Cluster, die Hindernisse sind. Da die linke Wand immer links vom Auto und die rechte Wand rechts vom Auto ist wird nun das Cluster C_i mit Punkten $p_{x,y}$ am weitesten links/rechts vom Auto, in einem Abstand a nach vorne als linke/recht Wand W gewählt:

$$W_L = \underset{C_i}{\operatorname{argmax}} \max_{p_{x,y} \epsilon C_i, 0 \leq x \leq a} +y \tag{1}$$

$$W_R = \underset{C_i}{\operatorname{argmax}} \max_{p_{x,y} \epsilon C_i, 0 \leq x \leq a} -y \tag{2}$$

Da der Cluster-Algorithmus oft keine perfekten Ergebnisse liefert, wird noch ein weiterer Schritt ausgeführt: Auf die erkannten Cluster wird jeweils ein Kreis gefittet, und alle weiteren Cluster die nah genug an der Kante des Kreises liegen werden auch zu dieser Wand hinzugezählt. Dies garantiert, dass auch weit vorne liegende Cluster mit zu der Wand gezählt werden, wenn die Wahrscheinlichkeit hoch ist, dass diese auch zu der Wand gehören. Die Werte für dieses Verfahren

sind komplett parametrisiert, sodass der Algorithmus bei übermäßigem hinzufügen von Clustern entsprechend eingestellt werden kann.

7.3 Erweitertes Wallfollowing

Während der Projektgruppe wurde der bereits bestehende Wallfollowing-Algorithmus [1] erweitert. Hierbei war das Ziel insbesondere eine bessere Planbarkeit der Geschwindigkeit und des Lenkwinkels zu ermöglichen. Der Algorithmus [1] zielt darauf ab möglichst mittig zwischen zwei Kreisen zu fahren und besitzt sonst keine Möglichkeit auf die Umgebung des Fahrzeuges zu reagieren. Unter Anderem konnten dabei folgende Probleme festgestellt werden:

- Die angepassten Kreise auf die rechte und linke Wand sind eine gute Approximation, wenn sich das Fahrzeug auf einer geraden Strecke oder Kurve befindet. Fährt es jedoch auf eine Kurve zu, so werden die Kreise ungenau.
- Die Geschwindigkeit des Fahrzeuges wird durch eine ungenaue Methode bestimmt, da physikalische Eigenschaften nicht berücksichtigt werden.
- Die Kraft durch die Beschleunigung der Reifen kann zu hoch sein, um sie auf die Fahrbahn zu übertragen.
- Es kann vorkommen, dass bei besonders scharfen Kurven, zum Beispiel einer 90 Grad Kurve, das Fahrzeug die Wand auf der Innenseite der Kurve streift.
- Beim Wallfollowing wird das Trennen der LiDAR-Datenpunkte in zwei Wände vorgenommen. Die dabei verwendete Methode ist nicht immer zuverlässig. Dieses Problem soll mit den Ergebnissen aus DBScan (siehe Unterunterabschnitt 7.2.3) umgangen werden.
- Während in der Simulation die LiDAR-Datenpunkte ohne große Abweichung die entsprechenden Wände darstellen, kann es bei realen Tests Ausreißer durch Reflexion oder andere Messfehler in den Datenpunkten kommen. Diese Ausreißer beeinflussen den Algorithmus negativ. Auch hier soll DBScan Abhilfe schaffen.
- Der PID-Controller im Wallfollowing, welcher den Lenkwinkel bestimmt, ist bei realen Tests instabil und hat zu Oszillationen im geführt.

Nachfolgend soll gezeigt werden, wie die Geschwindigkeit anhand von physikalischen Parametern bestimmt wird, das Problem einer zu hohen Reifen-

beschleunigung betrachtet und zuletzt mögliche Lösungsvorschläge präsentiert werden, die ein kollisionsfreies Lenken des Fahrzeugs ermöglichen.

7.3.1 Bremspunkt bestimmen

Da der bestehende Wallfollowing-Algorithmus [1] keine physikalisch nachvollziehbare Methode enthält, die für die Situation passende Geschwindigkeit des Fahrzeugs zu bestimmen, war es vielversprechend eine solche zu entwickeln. Für die folgende Erläuterung werden die Formeln in 2.2.1 und 2.2.2 behandelt.

Es ist das Ziel, dass das Fahrzeug eine bestimmte Strecke in möglichst kurzer Zeit zurücklegt, aber an den kritischen Stellen eine physikalische, maximale Geschwindigkeit einhält. Da das Fahrzeug eine bestimmte Reibung mit der Fahrstrecke erfährt und gleichzeitig in einer Kurve Fliehkräfte auf das Fahrzeug wirken, wird die maximale Kurvengeschwindigkeit aus dem Radius der zu durchfahrenden Kurve und dem Gleitreibungskoeffizienten der Reifen auf der Strecke abgeleitet (siehe 2.2.2). Damit die Strecke in möglichst kurzer Zeit zurückgelegt wird, soll das Fahrzeug so lange wie möglich die Geschwindigkeit halten oder weiter beschleunigen. Nach dieser Beschleunigungsphase muss abgebremst werden, damit in diesem Fall beim Einfahren in eine Kurve die maximal mögliche Kurvengeschwindigkeit erreicht wird. Dafür muss der verfügbare Weg der Beschleunigung und der Bremsweg berechnet werden (siehe 2.2.1). Damit diese Wege berechnet werden können, fehlt aber die Distanz vom Fahrzeug zu dem Kurveneingang.

Um den Kurveneingang zu bestimmen, werden unter zu Hilfenahme der im Wallfollowing-Algorithmus [1] ermittelten Kreise verschiedene Regeln verwendet:

1. Befindet sich der Mittelpunkt des Kreises auf der rechten Wand links vom Fahrzeug, fährt das Fahrzeug auf eine Rechtskurve zu.
2. Befindet sich der Mittelpunkt des Kreises auf der linken Wand rechts vom Fahrzeug, fährt das Fahrzeug auf eine Linkskurve zu.
3. In einer Rechtskurve ist der Radius des rechten Kreises größer als der des linken Kreises.
4. In einer Linkskurve ist der Radius des linken Kreises größer als der des rechten Kreises.

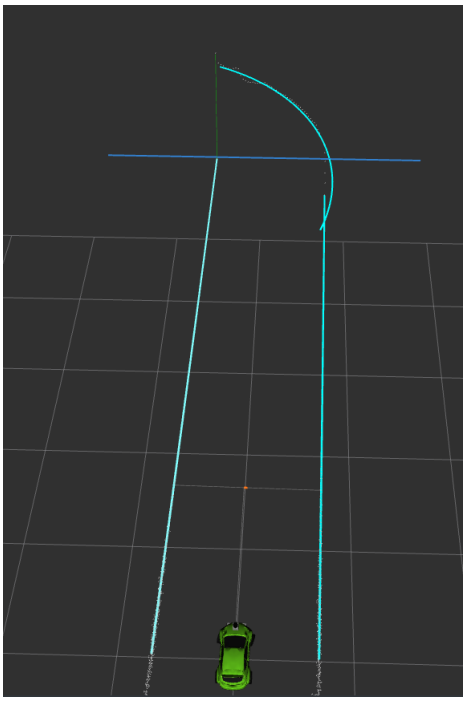


Abbildung 22: Die angepassten Kreise von rechter, linker und oberer Wand, wobei die dunkelblaue Linie den Kurveneingang darstellt und die hellblauen Linien die auf die Wände angepassten Kreise.

5. Ist der Radius des rechten Kreises größer als der des linken Kreises, obwohl sich das Fahrzeug nach Regel 1 auf eine Rechtskurve zu bewegt, so kann das Fahrzeug nicht in einer Rechtskurve sein, sondern bewegt sich auf eine zu.

6. Ist der Radius des linken Kreises größer als der des rechten Kreises, obwohl sich das Fahrzeug nach Regel 2 auf eine Linkskurve zu bewegt, so kann das Fahrzeug nicht in einer Linkskurve sein, sondern bewegt sich auf eine zu.

Wenn das Fahrzeug feststellt, dass es sich nach Regel 5 oder 6 auf eine Kurve zu bewegt, kann es den Kurveneingang bestimmen, indem es den am weitesten entfernten Datenpunkt der nach Regel 5 linken Wand und nach Regel 6 rechten Wand als Kurveneingang annimmt. Für den Bereich nach dem Kurveneingang kann anschließend als obere Wand ein oberer Kreis bestimmt werden. Vor dem Kurveneingang wird der linke und rechte Kreis neu bestimmt, indem die Datenpunkte aus der oberen Wand aus der rechten und linke Wand raus genommen werden. Abbildung 22 veranschaulicht das Ergebnis in der Simulation.

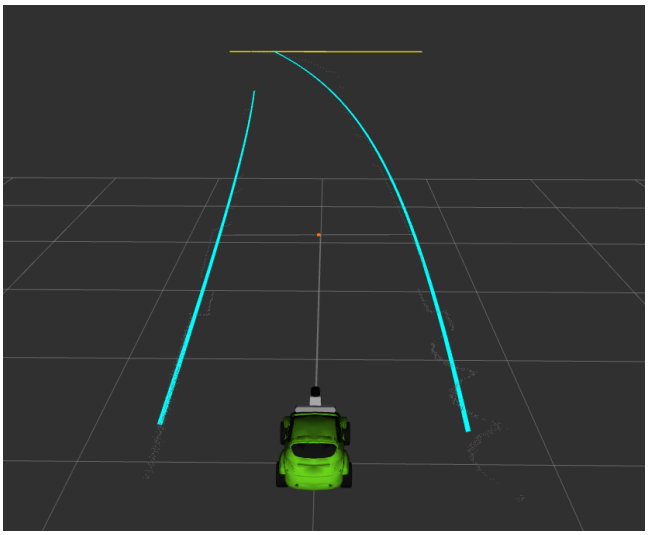


Abbildung 23: Die angepassten Kreise von rechter und linker Wand, wobei sowohl die Gerade, auf der sich das Auto befindet, als auch die Kurve, auf die es zufährt, nicht korrekt als Kreis approximiert wird. Die hellblauen Linien sind die auf die Wände angepassten Kreise.

Mit dieser Methode kann das Fahrzeug bereits eine Kurve einschätzen, bevor es selbst in dieser ist. Der Kurvenradius und die entsprechende maximale Kurvengeschwindigkeit können bestimmt werden, ebenso der Kurveneingang und damit die Information, nach welcher Distanz das Fahrzeug die Kurvengeschwindigkeit erreichen muss. Durch die weitere Unterteilung in rechte, linke und obere Wand, werden die Kreise genauer auf die Datenpunkte angepasst. Damit wird ein großer Nachteil des Wallfollowing-Algorithmus [1] beseitigt, der nur linke und rechte Wand kennt und entsprechend die Kreise nicht mehr korrekt anpasst, wenn das Fahrzeug auf eine Kurve zufährt. Das Problem wird in Abbildung 23 verdeutlicht. Wenn das Fahrzeug nicht auf eine Kurve zufährt, befindet es sich in einer Kurve. Auch eine Gerade wird als Kurve mit sehr großen Kreisradien angesehen. Das Fahrzeug strebt in diesem Fall immer die maximal mögliche Kurvengeschwindigkeit an.

7.3.2 Beschleunigung regulieren

Das Fahrzeug lässt hohe Reifenbeschleunigung zu, die aufgrund einer zu geringen Reibung nicht auf die Strecke gebracht werden kann. Dies führt zu durchdrehenden Reifen und Instabilität beim Beschleunigen und Bremsen. Um dieses Problem anzugehen wird der `AccelerationController` vorgestellt, der dafür sorgt, dass das Fahrzeug eine gleichmäßige physikalisch maximal mögliche Beschleunigung ausüben kann. Hierfür wird der Gleitreibungskoeffizient der Reifen auf der Strecke zugrunde gelegt um diese zu bestimmen. Die physikalische Formel wird in Anhang H hergeleitet.

Die Geschwindigkeit des Fahrzeuges kann gesteuert werden, indem eine Drehzahl für die Reifen eingestellt wird. Die Motoren werden die Drehzahl dann schnellstmöglich erreichen. Damit diese Beschleunigung reguliert werden kann, wird der `AccelerationController` eingesetzt: Dieser soll sicherstellen, dass die Beschleunigung innerhalb des Intervalls $[-a, a]$ bleibt. Dieser erhält die angeforderte Geschwindigkeit v_t und sendet dann nach einem Zeitintervall $\Delta t = 0.01s$ die neue Geschwindigkeit des Fahrzeuges $v_c = v_0 + a \cdot \Delta t$, wobei v_0 die bereits erreichte Geschwindigkeit des Fahrzeuges ist. Die Geschwindigkeit v_c in $\frac{m}{s}$ wird dann in die Drehzahl für die Reifen umgerechnet und an das Fahrzeug gesendet.

7.3.3 Ermittlung der Geschwindigkeit

Die Ermittlung der Geschwindigkeit erfolgt über den `AccelerationController`. Da das Auto die Geschwindigkeit als Drehzahl erhält und es intern dafür sorgt, dass diese Geschwindigkeit schnellstmöglich erreicht wird, wird angenommen, dass die angeforderte Geschwindigkeit dann auch die unmittelbar gefahrene ist. Dies kann insbesondere dadurch angenommen werden, da der `Acceleration Controller` die Geschwindigkeit in kleinen Schritten steigert. Dadurch ist zu jedem Zeitpunkt eine gute Approximation der Geschwindigkeit möglich. In der Simulation gibt es zusätzlich noch die Möglichkeit die Geschwindigkeit direkt aus den Positionsdaten zu errechnen. Da diese in der Realität nicht vorhanden sind, wurde von einer Benutzung dieser Möglichkeit außer für Testzwecke abgesehen.
Es konnte zudem festgestellt werden, dass das Auto mit den ursprünglich übernommenen Parametern in der Realität etwa um den Faktor 2,5 langsamer fuhr als in der Simulation. Es stellte sich heraus, dass die Übersetzung des Getriebes nicht korrekt in der Definition des Autos eingestellt war. Nach einer Korrektur auf die richtige Übersetzung und einer allgemeinen Korrektur der approximierten Geschwindigkeit von 10% konnte die Geschwindigkeitsmessung des Autos anhand von realen Tests validiert werden. Dabei wurde das Auto auf eine feste Geschwindigkeit gesetzt und fuhr eine bestimmte Anzahl an Runden. Die Zeit und der Weg wurden händisch gemessen und die berechnete Geschwindigkeit entsprechend mit der eingestellten abgeglichen. Dabei konnte keine signifikante Abweichung mehr festgestellt werden.

7.3.4 Lenkwinkel bestimmen

Es wurden verschiedene Methoden implementiert die Berechnung des Lenkwinkels in Bezug auf die Sicherheit des Fahrzeugs und des möglichst schnellen Fahrens zu verbessern:

- **Kreistangenten**
  Bei dieser Variante wird der Radius der über die Wände approximierten Kreise um einen Sicherheitsabstand vergrößert. Für diese vergrößerten Kreise wird die daran anliegende Tangente berechnet. Das Auto berechnet dann als Zielposition den Schnittpunkt dieser Tangente. Dies soll dazu führen, dass das Auto immer einen Sicherheitsabstand von den Wänden einhält. Zudem kostet diese Methode wenig Rechenleistung.
  Jedoch werden die Kreise nicht zuverlässig genug approximiert, so dass zu wenig Stabilität für das Auto insbesondere bei hohen Geschwindigkeiten erreicht wird.

- **Möglichst lange gerade fahren**
  Hierbei sucht das Auto den am weitesten entfernten Punkt, welchen es noch anfahren kann ohne dabei den Wänden zu nahe zu kommen. Auch hier soll immer ein Sicherheitsabstand eingehalten werden. Das Auto soll möglichst lange geradeaus fahren ohne lenken zu müssen. Dies soll zu einem schnelleren Fahren führen während gleichzeitig ein Sicherheitsabstand zu den Wänden gewahrt wird. Realisiert wird dies durch die Berechnung vieler in der Distanz liegenden Streckenmittelpunkte. Dabei wird eine Linie vom Auto zu dem zu prüfenden Streckenmittelpunkt gelegt. Anhand der Linie wird bestimmt, ob Punkte der Wände zu nah an dieser liegen. Wenn der Sicherheitsabstand unterschritten wird, wird der mögliche Ziel-Streckenmittelpunkt verworfen. Das finden des am weitesten entfernten Streckenmittelpunktes wird über eine Binäre Suche realisiert.
  Auch hier konnte keine ausreichende Stabilität bei hoher Geschwindigkeit sichergestellt werden, da die gefundenen Streckenmittelpunkte sich kurzfristig stark ändern können. Auch eine künstlich hinzugefügte Trägheit konnte keine Abhilfe schaffen.

- **Kurvenanfahrt verbessern**
  Bei diesem Ansatz fährt das Auto vor einer Rechtskurve an den linken Bereich der Strecke und bei einer Linkskurve an den rechten Bereich. Das Auto soll so besser in die nahende Kurve einfahren können.

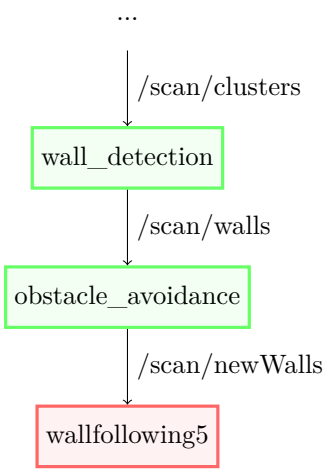


Abbildung 24: Erweiterung der Verarbeitungspipeline um Hindernisumfahrung

> Jedoch konnte auch dabei insbesondere eine Abnahme an Stabilität bei hoher Geschwindigkeit festgestellt werden.

Schlussendlich wird der ursprüngliche Lenkalgorithmus beibehalten, der immer auf die Mitte der Strecke zielt. Diese Methode führt zu gleichmäßigeren Lenkbewegungen. Dies ist von großem Vorteil bei hohen Geschwindigkeiten. Es konnte eine Verbesserung erreicht werden, indem der Lenkwinkel abhängig zur Geschwindigkeit begrenzt wird. Dadurch kann das Auto möglichst keine Lenkbewegungen ausführen, die zu einem Verlust der Haftung auf der Strecke führt. Der kleinste erlaubte Lenkradius r_{min} wird durch die Reibung μ auf dem Boden, der Fallbeschleunigung g und die aktuelle Geschwindigkeit v folgendermaßen bestimmt:

$$r_{min} = \sqrt{\frac{v^2}{\mu g}} \tag{3}$$

Der maximale Lenkwinkel ψ_{max} in Radianten kann dann mit dem Abstand d zwischen Vorder- und Hinterreifen bestimmt werden [16]:

$$\psi_{max} = \arctan \frac{d}{\sqrt{r_{min}^2 - (\frac{d}{2})^2}} \tag{4}$$

7.4 Hindernisumfahrung

Um Objekten auf der Strecke ausweichen zu können, wurde ein Ansatz gewählt der sich in die bestehende Pipeline einbinden lässt. Wie Abbildung 24 darstellt, wurde eine weitere Node vor dem Wallfollowing Algorithmus, der als Orientierung übermittelte Cluster nutzt, die als Wand erkannt worden sind. Damit der Wallfollowing Algorithmus nicht verändert werden musste, verändert der neue Zwischenschritt in der Pipeline den Verlauf der Wände.

Wenn ein Hindernis auf der Strecke erkannt wird, soll der Algorithmus entscheiden, an welcher Seite des Hindernisses das Auto dieses umfahren soll. Dies ermittelt der Algorithmus daran, auf welcher Seite der größere Abstand zwischen Hindernis und Wand ist. Angenommen der Abstand rechts zwischen Hindernis und Wand ist größer, dann berechnet der Algorithmus eine neue linke Wand, die von einem Punkt der realen linken Wand bis hin zur rechten Seite des Hindernisses verläuft. Somit wird dem Fahrzeug als einzig möglich fahrbarer Bereich ein Weg angezeigt, sodass er nicht mit dem Hindernis, kollidiert.

Aktuell existieren noch größere Probleme auf verschiedenen Ebenen, bei diesem Ansatz. Zum einem existiert innerhalb des Codes ein Bug, sodass momentan die Wand nicht korrekt entsteht, um das Objekt zu umfahren, welches aber definitiv lösbar ist und auch noch bearbeitet wird. In ersten kleineren Tests waren weitere Probleme aufgetaucht, sodass dieser Ansatz stark abhängig von sämtlichen Verarbeitungsschritten zuvor ist. Sowohl eine zeitliche Verzögerung zwischen erkennen und umfahren von einem zu großen Ausmaß vorliegt, als auch, dass bei falsch erkannten Hindernissen das Verhalten des Fahrzeuges stark gestört ist.

7.5 Dynamic Reconfigure

Da für die Operation des Autos eine Vielzahl an verschiedener Parameter anfällt und sich einige davon auch gegenseitig beeinflussen, wäre es nahezu unmöglich jeden davon einzeln zu optimieren. Nach jeder kleinen Anpassung müsste das Projekt neu gebaut und die Simulation neugestartet werden. Um das zu vermeiden, wird von ROS ein Tool namens `dynamic_reconfigure` bereitgestellt, über welches während der Laufzeit über eine leicht verständliche GUI vorher definierte Parameter angepasst werden können. Die gewählten Parameter müssen

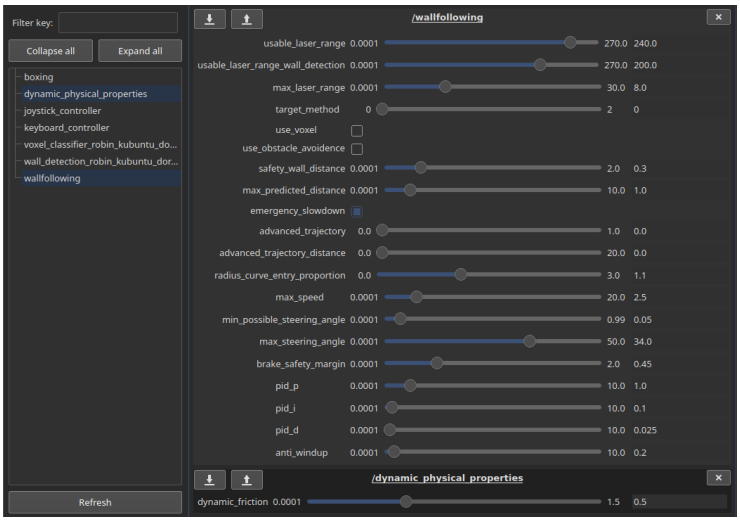


Abbildung 25: Screenshot von Dynamic Reconfigure des Wallfollowing-Algorithmus und des Reibungswertes

vorher in einer Konfigurationsdatei abgelegt und beschrieben werden. Ein Server in ROS lauscht dann bei jedem Ausführungszyklus der entsprechenden Nodes auf Änderungen an den Parameterwerten und aktualisiert diese bei Bedarf. [26]

Im Folgenden werden alle Parameter die sich zur Laufzeit in unserer Simulation und auf dem realen Auto anpassen lassen aufgelistet und kurz beschrieben:

- Wallfollowing-Algorithmus
  - `usable_laser_range`: Genutzter Sichtbereich über die LiDAR Daten in Grad
  - `usable_laser_range_wall_detection`: Genutzter Sichtbereich über die LiDAR Daten in Grad, in dem die Trennung zwischen den beiden Wänden erkannt werden kann
  - `max_laser_range`: Maximale Distanz bis zu der LiDAR-Punkte genutzt werden
  - `target_method`: Umschalten zwischen Lenkalgorithmen (0: Immer in der Mitte der Strecke fahren, 1: Kreistangenten benutzen, 2: Die am weitest entfernte Zielposition suchen mit der das Auto an den Wänden mit einem Sicherheitabstand vorbeikommt)
  - `use_voxel`: Wechsel zwischen Benutzung der LiDAR-Punktwolke oder der Voxel-Methode

  - `use_obstacle_avoidance`: Eine simple Lösung zur langsamen Hindernisumfahrung
  - `safety_wall_distance`: Sicherheitsabstand in Metern für die Kreistangentenmethode und dem Suchen der am weitesten entfernten Zielposition an der das Auto sicher an den Wänden vorbei kommt
  - `max_predicted_distance`: Maximale Distanz in Metern, die für den nähsten Streckenmittelpunkt verwendet wird
  - `emergency_slowdown`: Verlangsamt das Auto in einer unerwartet scharfen Kurve besonders
  - `advanced_trajector`: Distanz in Metern, in der das Auto in einer Rechtskurve auf die linke Seite der Strecke fährt und vice versa
  - `advanced_trajector_distance`: Distanz in Anteil von der halben Streckenbreite, die das Auto in einer Rechtskurve auf die linke Seite der Strecke fährt und umgekehrt
  - `radius_curve_entry_proportion`: Maximales Verhältnis der Kurvenradien bis zu denen ein Kurveneingang detektiert wird
  - `max_speed`: Maximal erlaubte Geschwindigkeit
  - `min_possible_steering_angle`: Minimal möglicher Lenkwinkel in Anteilen am Maximalen Lenkwinkel
  - `max_steering_angle`: Maximal möglicher Lenkwinkel in Grad, den das Auto aufgrund seiner Hardwaregegebenheiten einschlagen kann
  - `brake_safety_margin`: Abstand in Metern vor einem Kurveneingang, bei dem das Auto die Zielgeschwindigkeit für die Kurve erreicht haben muss
  - `pid_p`: p-Wert des PID-Controllers
  - `pid_i`: i-Wert des PID-Controllers
  - `pid_d`: d-Wert des PID-Controllers
  - `anti_windup`: Anti windup des PID-Controllers in Sekunden
- Physikalische Parameter
  - `dynamic_friction`: Gleitreibungskoeffizient der Reifen auf dem Boden der Strecke
- Boxing
  - `voxel_size`: Größe der Voxel in Metern
  - `lidar_point_percentage`: Prozentualer Anteil der LiDAR Punkte die tatsächlich verwendet werden
  - `filter_by_min_score_enabled`: Score Filter ein-/ausschalten
  - `filter_by_min_score`: Score Schwellwert für Score Filter

  - `sor_enabled`: Statistische Ausreißerentfernung aktivieren
  - `enable_colors`: Auswertung von Farbinformationen ein-/ausschalten
  - `adjacent_voxels`: Bei Auswertung von Farbinformationen auch diagnoal liegende Voxel beachten
  - `color_levels`: Menge an verschiedenen Werte für je R, G, B auf die quantisiert wird
  - `color_samples`: Menge an Punkten die aus der Farbwolke beachtet werden

- Voxel Classifier
  - `cluster_minimum_points`: Minimale Anzahl von Nachbarpunkten, damit ein Punkt als Kernpunkt gilt.
  - `cluster_epsilon`: Minimaler Abstand zwischen Punkten, damit diese als Nachbarn gelten
  - `color_weight`: Gewichtung für Farbwerte bei Abstandsberechnung

- Wall Detection
  - `wall_search_radius`: Abstand nach vorne in dem das Auto nach Wänden links und rechts sucht
  - `use_prediction_for_walls`: Kreisregression verwenden um Clusterwaisen zu Wänden dazuzuzählen
  - `use_prediction_for_obstacles`: Kreisregression verwenden um Cluster nicht als Hindernisse zu zählen
  - `regression_minimum_score`: Minimale Anzahl an Voxel in der Nähe einer Wand, damit dieses Cluster zu einer wand regressiert wird
  - `minimum_confidence`: Minimaler Konfidenzwert $(1 - left\_score/right\_score)$ um zu entscheiden ob dieses Cluster zu einer Wand regressiert wird

7.6 Datenerfassung und -aufbereitung

Zusätzlich zu den für das autonome Fahren verwendeten Daten wurden auch noch Telemetriedaten zu Analysezwecken aufgezeichnet. Diese werden hier vorgestellt.

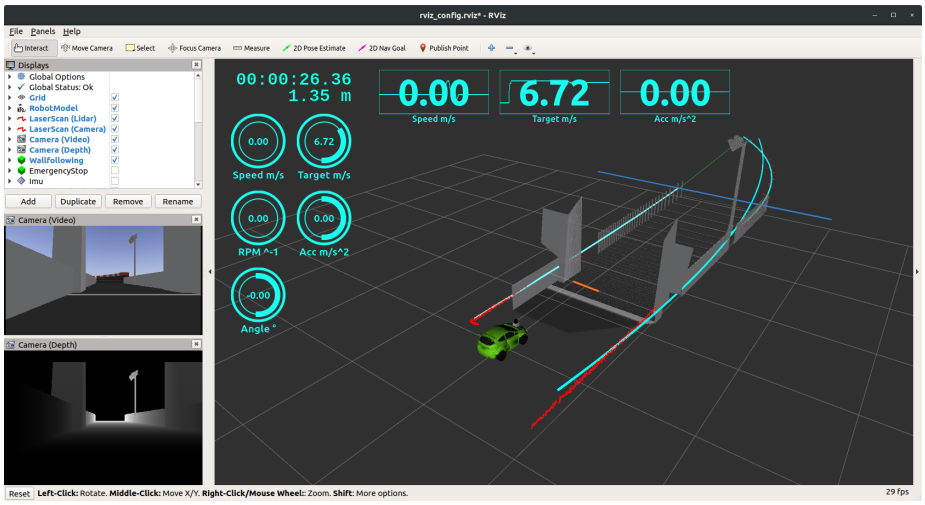


Abbildung 26: Kamerastreams und Darstellung der Pointcloud in Rviz

7.6.1 Videoaufnahme und Streaming in RViz

Zu Analyse und Demonstrationszwecken wurde das Streamen von Kamerabildern an Rviz realisiert. Die Einbindung erfolgt mittels in RViz integrierter Camera- und Pointcloud-Module. Abbildung 26 zeigt im linken Bereich die aktuellen Live-Bilder als normalen Videostream und als Tiefendaten, im Hauptfenster ist die daraus abgeleitete Punktwolke und die entsprechende 3D Rekonstruktion zu sehen.

Durch entsprechende Optionen (siehe Anhang J) beim Starten des Fahrzeugs oder der Simulation wird ein Video der Fahrt aufgezeichnet. Die Aufnahme wird im Verzeichnis `/data` des Projektes gespeichert. Diese Funktion sollte allerdings mit Bedacht genutzt werden, da sie eine erhebliche Belastung der CPU verursacht und daher nicht im Normalbetrieb Anwendung finden sollte. Zudem müssen die Aufzeichnungen anschließend bzgl. der Geschwindigkeit normalisiert werden, da die Aufnahme auf dem Fahrzeug nicht mit konstanter Framerate erfolgt. Entsprechende Parameter und Funktionen stehen im Verwaltungsskript `toad.sh` zur Verfügung und sind in Anhang J dokumentiert.

7.6.2 Capture RViz

Es besteht zusätzlich die Möglichkeit, ein Mitschnitt des RViz Hauptfensters vorzunehmen. Hierfür steht ein fertiges Modul mit der Bezeichnung *VideoCap-*

Abbildung 27: Screenshot einer Kamerafahrt

ture in RViz zur Verfügung (siehe Abbildung 28. Dort kann die Auflösung und die gewünschte Framerate definiert und mit *Enable* die Aufnahme gestartet werden. Die Aufnahme wird im ROS-Verzeichnis abgelegt und, sofern das Starten von ROS mittels `toad.sh` erfolgte, anschließend in das Verzeichnis `/data` des Projektes verschoben.

7.6.3 Telemetriedatenerfassung

Um während der Fahrt relevante Messdaten erfassen und aufzeichnen zu können, wurde die Node `log_stats` implementiert. Um Werte die vom HUD angezeigt werden nicht doppelt auslesen und berechnen zu müssen, stellt diese Node ebenfalls die Schnittstelle hierfür bereit. Sie verfügt über folgende Parameter:

- `prefix`: Sofern die Daten zwecks späterer Auswertung in Dateien geschrieben werden, kann hier ein Präfix für die Benennung angegeben werden.
- `length`: Neben Durchschnittswerten über den gesamten Zeitraum werden ebenfalls Durchschnittswerte über die hier vorgegebene Menge von Messwerten erzeugt.
- `smooth`: Um bei der Darstellung im RViz zu starke Schwankungen und damit ein optisch unschönes Verhalten der Anzeigen zu vermeiden, werden

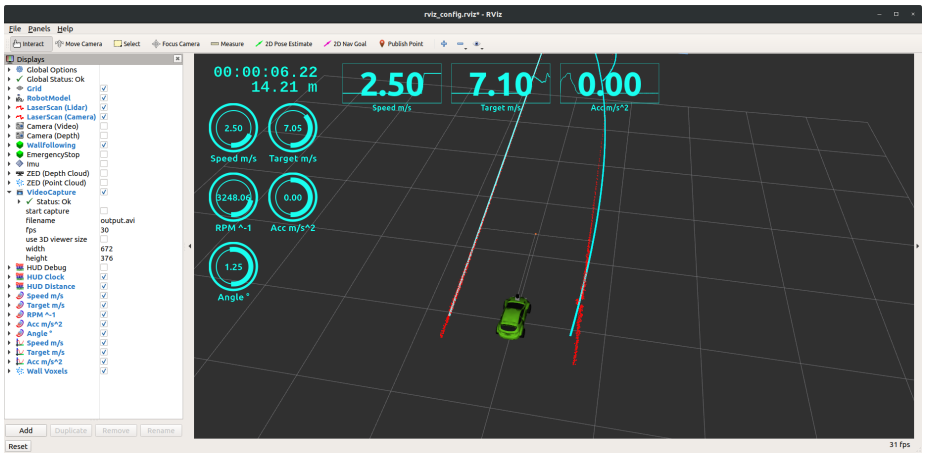


Abbildung 28: RViz mit ausgeklapptem Capture-Modul und HUD Anzeige

die HUD-Elemente mit Durchschnittswerten über die hier vorgegebene Menge von Messwerten versorgt.

- `sim`: Da die Datenerfassung in der Simulation und im realen Fahrbetrieb unterschiedlich erfolgt, muss der Node hier mitgeteilt werden, in welchem Modus sie eingesetzt wird (`true` oder `false`).
- `write`: Mit dieser Option kann das Schreiben der Protokolldateien deaktiviert werden, sofern die Daten nur für das HUD genutzt werden sollen. Die Daten werden sowohl als CSV als auch als DAT-Datei im Verzeichnis `/data` abgelegt. Erstere ermöglicht ein schnelles, manuelles Auswerten, letztere wird als Datenbasis für die Reportgenerierung mittels LaTeX*PGFPlots* genutzt.

Die Node liest folgende Daten direkt aus anderen Nodes aus (alle anderen Werte werden daraus berechnet):

- Aktuelle Geschwindigkeit v_{cur}
- Maximal zulässige Geschwindigkeit v_{max} wie sie vom verwendeten Fahralgorithmus vorgegeben wird
- Den aktuellen Lenkeinschlag als diskreten Wert aus dem Intervall $[-1...1]$

Gespeichert werden folgende Werte:

- Datapoint: Nummer des Datensatzes

- AverageTime: Der zuvor genannte Parameter `length`
- AverageSmooth: Der zuvor genannte Parameter `smooth`
- Time: Aktueller Zeitstempel
- TimeDelta: Abstand zum letzten Zeitstempel
- Speed: Aktuelle Geschwindigkeit
- SpeedDelta: Änderung seit der letzten Erfassung
- SpeedAverage: Durchschnitt über alle Werte
- SpeedAverageTime: Durchschnitt über die letzten `length` Werte
- SpeedMax: Maximalwert über alle gemessenen Werte
- SpeedMaxTime: Maximalwert über die letzten `length` Werte
- Maxspeed: Maximal zulässige Geschwindigkeit, wie sie vom verwendeten Fahralgorithmus vorgegeben wird
- MaxspeedDelta: Änderung seit der letzten Erfassung
- MaxspeedAverage: Durchschnitt über alle Werte
- Angle: Aktueller Lenkwinkel in Grad
- AngleDelta: Änderung seit der letzten Erfassung
- Acceleration: Aktuelle Beschleunigung (berechnet aus der Veränderung der Geschwindigkeit und der Zeit seit der letzten Erfassung)
- AccelerationSmooth: Durchschnitt über die letzten `smooth` Werte
- AccelerationDelta: Änderung seit der letzten Erfassung
- AccelerationMin: Minimalwert über alle gemessenen Werte
- AccelerationMax: Maximalwert über alle gemessenen Werte
- AccelerationMinTime: Minimalwert über die letzten `length` Werte
- AccelerationMaxTime: Maximalwert über die letzten `length` Werte
- Distance: Gefahrene Distanz in Metern (Summe aus dem letzten und der Veränderung seit dem letzten Messwert)
- DistanceDelta: Änderung seit der letzten Erfassung (berechnet aus der Geschwindigkeit und der Zeit seit der letzten Erfassung)
- Turn: Der aktuelle Lenkeinschlag als diskreter Wert aus dem Intervall $[-1...1]$

- TurnDelta: Änderung seit der letzten Erfassung
- RPM: Aktuelle Motordrehzahl (berechnet aus der aktuellen Geschwindigkeit und einem konstanten Faktor, vgl. Unterunterabschnitt 7.3.3)

Dies stellt den aktuellen Stand der Erfassung dar. Eine aktuelle Auflistung erfolgt stets im Wiki[15].

7.6.4 Reportgenerierung

Um eine schnelle, graphische Auswertung der gesammelten Daten zu ermöglichen, wurde das Skript `toad.sh` um Funktionen zur Generierung von Plots erweitert. Weitere Informationen zur Bedienung können Unterabschnitt 4.1 entnommen werden. Für jeden angegebenen Datensatz werden folgende Plots, jeweils nach der Zeit t und der zurückgelegten Distanz d erstellt:

- Gefahrene Geschwindigkeit v_{cur}
- Gefahrene Geschwindigkeit v_{cur} im Vergleich zur maximal zulässigen Geschwindigkeit v_{max}
- Ermittelte Beschleunigung

Bei Angabe mehrerer Datensätze erfolgt am Ende des erzeugten Dokuments ein Vergleich relevanter Werte. Die LaTeX-Quelldateien werden so erzeugt, dass sie problemlos in bestehende Dokumente eingebunden werden können. Ein Beispiel für solch einen Report ist in Anhang L ersichtlich.

7.6.5 Head-Up-Display

Zwecks Darstellung der wichtigsten Werte wurden im RViz entsprechende Anzeigen realisiert, wobei sie wie in Unterunterabschnitt 7.6.3 erläutert geglättet werden. Diese stellen im Hauptfenster (siehe Abbildung 28) folgende Werte graphisch dar:

- Die aktuelle Geschwindigkeit und deren Verlauf

[15] https://github.com/arpg-sophisticated/ar-tu-do/wiki/Telemetry-data

- Die aktuelle Beschleunigung und deren Verlauf
- Die maximal zulässige Geschwindigkeit, wie sie vom verwendeten Fahralgorithmus vorgegeben wird und deren Verlauf
- Die aktuelle Motordrehzahl
- Den aktuellen Lenkeinschlag
- Die vergangene Zeit
- Die gefahrene Distanz

Für den normalen Betrieb ist dies völlig ausreichend. Darüber hinaus kann eine weitere Anzeige *HUD Debug* aktiviert werden, die dann alle erfassten Werte (vgl. Unterunterabschnitt 7.6.3) als einfache Textansicht einblendet.

8 Anwendung Maschinellen Lernens

Während der Projektarbeit wurden verschiedene Ansätze des maschinellen Lernens evaluiert. In diesem Abschnitt wird zuerst auf die Erkenntnisse eingegangen die mit überwachtem Lernen gewonnen werden konnten. Anschließend werden die Ergebnisse vorgestellt die mit einem Q-Learning Algorithmus erzielt wurden.

8.1 Überwachtes Lernen

Zunächst wurde die Methodik des überwachten Lernens evaluiert. Die Aufgabe, die das Modell erfüllen sollte, ist hierbei die Ausgabe von Lenkwinkel und Zielgeschwindigkeit, basierend auf den zum aktuellen Zeitpunkt eingehende Sensordaten. Mit dem Ziel das Auto hinsichtlich bestimmter Kriterien möglichst optimal auf der Strecke zu bewegen.

Unentbehrlich für das Trainieren eines Modells durch überwachtes Lernen sind mit Label versehende Trainingsdaten. Für diesen Anwendungsfall müssen die Trainingsdaten das bereits erwähnte optimale Fahrverhalten repräsentieren. Vorerst lag eine solche optimale Menge von Trainingsdaten jedoch nicht vor. Daher wurde zunächst eine in der Simulation aufgezeichnete Fahrt mit dem in Abschnitt 3 beschriebenen Wallfollowing Algorithmus als Trainingsmenge verwendet. Ein Datenpunkt dieser Trainingsmenge beschreibt den Zustand des Autos im aktuellen Zeitpunkt in Form der eingehenden LiDAR-Sensordaten sowie der aktuellen Geschwindigkeit. Zudem enthält jeder Datenpunkt als Label die vom Wallfollowing-Algorithmus ausgegebenen Werte für Lenkwinkel und Geschwindigkeit für diesen Zustand. Das überwachte Lernen auf dieser Trainingsmenge zielt also zunächst darauf ab, mit dem Modell das Verhalten des Wallfollowing-Algorithmus imitieren zu können.

Grundsätzlich wurden für das überwachte Lernen zwei verschiedene Modell-Typen evaluiert. Diese unterscheiden sich in der Art und Weise wie die Sensordaten des LiDAR im Modell verwendet werden. Im ersten Modell werden die LiDAR-Daten ausgedünnt auf eine feste Anzahl Stützstellen als numerische Werte abgespeichert. Im zweiten Modell werden alle LiDAR-Daten mit dem in Unterunterabschnitt 7.2.2 beschriebenen Verfahren zu Voxeln zusammengefasst und als als PNG-Bild gespeichert. Die verringerte Auflösung der LiDAR-Daten, die aus dem Zusammenführen mehrerer Datenpunkte zu Voxeln und dem Ausdünnen auf bestimmte Stützstellen resultiert, führt zu wenig Speicherbedarf, einer kleinen Netzstruktur des künstlichen neuronalen Netzes und damit zu einer besseren Laufzeit während des Trainings und bei der Vorhersage.

Zusätzlich zu den LiDAR-Sensordaten enthält ein Datum in der Trainingsmenge folgende numerische Information:

ts [PK] timestamp without time zone	speed numeric	calc_velocity numeric	calc_angle numeric
2020-03-11 16:35:01.953853	7.13916873932	4.89540100098	-0.148901179433
2020-03-11 16:35:01.887862	7.13916873932	5.05044364929	-0.140732675791
2020-03-11 16:35:01.823652	7.13916873932	5.07436609268	-0.132523149252
2020-03-11 16:35:01.783286	7.40198040009	5.09021902084	-0.125089973211
2020-03-11 16:35:01.725023	7.40198040009	5.14343166351	-0.117472864687
2020-03-11 16:35:01.66559	7.40198040009	5.1786813736	-0.0986969098449
2020-03-11 16:35:01.659496	7.40198040009	5.25169277191	-0.0965942293406
2020-03-11 16:35:01.62077	7.49944067001	5.28661155701	-0.0890995487571
2020-03-11 16:35:01.532906	7.49944067001	5.27797079086	-0.0722576230764
2020-03-11 16:35:01.453748	7.73381328583	5.38323020935	-0.0669832751155

Abbildung 29: Auszug aus den Trainingsdaten der Postgres-Datenbank

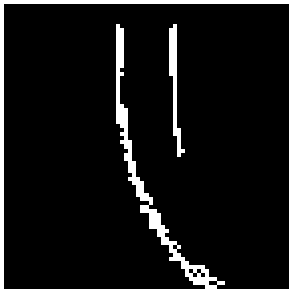


Abbildung 30: Beispiel eines abgespeicherten Voxel-Bildes

- Aktuelle Geschwindigkeit
- Lenkwinkel-Ausgabe des Wallfollowing-Algorithmus
- Zielgeschwindigkeit-Ausgabe des Wallfollowing-Algorithmus

Die numerischen Daten wurden in einer Postgres-Datenbank [16] gespeichert. Die Zuordnung zu den entsprechenden PNG-Bildern, ist durch einen mit abgespeicherten Zeitstempel möglich. Ein Auszug aus der Datenbank ist in Abbildung 29 zu sehen. Zum Vergleichen verschiedener Netz-Architekturen wurden in der Simulation insgesamt circa 300.000 Datenpunkte zum Trainieren aufgezeichnet. Ein Beispiel für ein gespeichertes PNG-Bild ist in Abbildung 30 zu sehen.

Je nach Art der eingehenden LiDAR-Daten, als Voxel oder in ausgedünnter Form, wurden entweder ein Faltungsnetz oder ein Mehrschichtenperzeptron als Netzstruktur verwendet. Beide Netzstrukturen werden in Anhang G näher erläutert.

[16] `https://www.postgresql.org/`

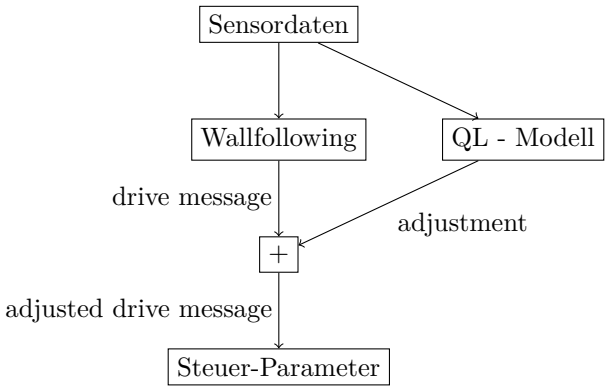


Abbildung 31: Struktur des Q-Learning Ansatzes, Hybrides Modell aus Wallfollowing und Q-Learning Modell

8.2 Q-Learning

Das überwachte Lernen ist abhängig von vorhandenen und mit Zielgrößen versehenen Trainingsdaten. Beim bestärkenden Lernen kann in dem gegeben Fall des autonomen Fahrens das Auto seine Umgebung währende des Trainings selbst entdecken. Im Folgenden wird darauf eingegangen wie Q-Learning zum bestärkenden Lernen in der Projektgruppe eingesetzt wurde.

Es ist bekannt, dass beim bestärkenden Lernen oft ein umfangreiches Training erforderlich ist, da durch die zufällige Initialisierung und die teilweise zufällig gewählten Aktionen zu Beginn des Trainings beliebig schlecht in der Umgebung agiert wird. Daher wurde ein hybrides Verfahren genutzt, das den in Abschnitt 3 beschriebenen Wallfollowing-Algorithmus mit einem - durch bestärkendes Lernen trainiertes - Modell erweitert. Hierbei wurde der Wallfollowing-Algorithmus verwendet, der von der vorhergehenden Projektgruppe übergeben wurde. Das Modell erhält, wie in Abbildung 31 dargestellt, die Sensordaten parallel zum Wallfollowing Algorithmus. Nach der Berechnung werden die Ausgaben beider miteinander addiert und als Steuer-Parameter für das Auto verwendet

Das Neuronale Netz, das für den Q-Learning Algorithmus verwendet wurde, nimmt die LiDAR-Daten ausgedünnt auf 8 Entfernungspunkte zusammen mit der aktuell gemessenen Geschwindigkeit als Eingabe entgegen. Während der Implementierung hat sich herausgestellt, dass eine höhere Auflösung der LiDAR-Daten zu einer erheblich längeren Trainingszeit des Modells führt.

Zudem kann ein Q-Learning Modell nur eine endliche Menge von Aktionen ausgeben. Hierbei wurde sich auf folgende Aktionen beschränkt: Die Zielgeschwindigkeit kann um $1m/s$ erhöht oder nicht verändert werden. Der Lenkwin-

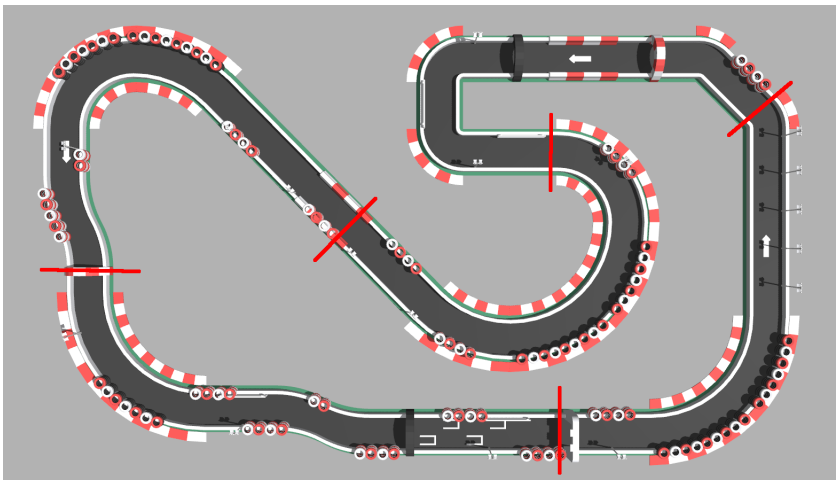


Abbildung 32: Unterteilung des Simulationsstrecke in fünf Sektoren

kel kann um 0.15 Radians nach links oder rechts oder nicht angepasst werden. Durch Kombination ergeben sich sechs mögliche Aktionen.

Die Struktur des Neuronalen Netzes wurde wie folgt gewählt, hierbei wurde für jede Schicht die rektifizierende Aktivierungsfunktion [28] benutzt:

- input-layer: 9 Neuronen
- hidden-layer: 64 Neuronen
- hidden-layer: 32 Neuronen
- output-layer: 6 Neuronen

Das Training des Modells wurde in verschiedene Phase unterteilt. Zunächst wurde wie in Abbildung 32 abgebildet die Simulationsstrecke in fünf gleich große Teile aufgeteilt, welche im Folgenden als Sektoren bezeichnet werden. In der ersten Trainingsphase wurde zufällig einer der Sektoren ausgewählt und nach dem in Unterunterabschnitt 2.6.3 beschriebenen Algorithmus befahren.

Die Belohnung der den Aktionen in der Historie hierbei zugeordnet wird, ist abhängig von der Differenz der beim Durchlauf erzielten Sektorzeit und der Sektorzeit des Wallfollowing-Algorithmus, welche vorab für jeden Sektor erfasst wurde, bestimmt.

Zudem kann es während des Trainings durchaus dazu kommen, dass es zu einer Kollision mit der Fahrbahn-Begrenzung kommt. In diesem Fall wird die

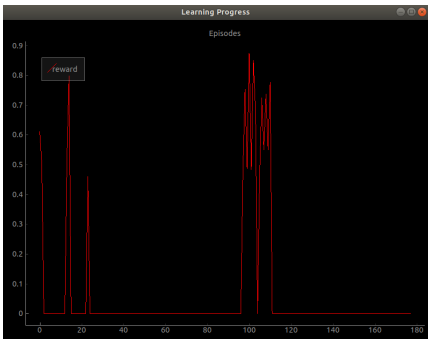


Abbildung 33: Erhaltener Reward für die ersten 180 Sektordurchfahrten bei zufällig initialisiertem Neuronalen Netz

Fahrt unterbrochen und neu gestartet. Die während der Fahrt durch den Sektor getätigten Aktionen erhalten die Belohnung 0.

Während der Entwicklung hat sich zudem heraus gestellt, dass bessere Ergebnisse erzielt werden können, wenn ein erfolgreiches Durchfahren eines Sektors, neben der Belohnung für die erzielte Zeit, zusätzlich durch einen fixen Wert für kollisionsfreies Fahren belohnt wird.

Die Belohnung-Funktion hat sich nach mehreren Tests während der Entwicklung schließlich wie folgt zusammen gesetzt:

$$R = \frac{((T_{WF} - T_{QL} + 1 - 0.05)^3) - 1}{5}$$

Hierbei steht T_{WF} für die Sektorzeit beim Fahren mit dem Wallfollowing-Algorithmus, welche vorab gemessen wurde, und T_{QL} für die Sektorzeit die bei aktuellen Durchfahrt des Sektors erzielt wurde. Die Zeiten wurden jeweils in Sekunden gemessen. Für die Verminderung der zeitlichen Differenz um 0.05 Sekunden wurde sich entschieden, um nur erhebliche Verbesserungen der Geschwindigkeit zu belohnen. Zusätzlich wurden die Potenzierung mit dem Exponent 3 und der Faktor 5 heuristisch festgelegt.

In Abbildung 33 ist der während des Trainings erhaltene Reward für die ersten 180 Sektordurchfahrten bei zufällig initialisiertem Neuronalen Netz zu sehen.

Nach der ersten Trainingsphase wurden Tests des Modells durchgeführt. Hierbei stellte sich heraus, dass zu hohe Geschwindigkeiten zu Kollisionen mit der Fahrbahnbegrenzung führen. Eine mögliche Ursache hierfür ist, dass während des Trainings Sektoren nur einzeln befahren wurden. Das Auto startet im Training jeweils am Anfang des Sektors aus dem Stand. Während einer Umrun-

dung der Strecke im Test werden folglich höhere Geschwindigkeiten erzielt, als während des Trainings. Daher wurde eine zweite Trainingsphase durchgeführt in der, neben den fünf Sektoren, eine ganze Umrundung als zufällige Option hinzugefügt wurde. Da bei einer ganzen Umrundung offensichtlich höhere Differenzen zwischen T_{QL} und T_{WF} entstehen, wurde diese Differenzen in der Belohnungsfunktion um den Faktor 0.4 verringert.

Im Anschluss an diese Trainingsphase konnten einzelne Runden kollisionsfrei im Test absolviert werden. Bei aufeinander folgenden Runden trat jedoch ein ähnliches Phänomen auf wie nach der ersten Trainingsphase. Im Training wurde eine Runde wieder stets aus dem Stand gestartet. Hohe Geschwindigkeiten zu Beginn einer Umrundung führten daher zu Kollisionen in jeder zweiten Runde im Test. Eine dritte Trainingsphase in der zufällig zwischen einer der Sektoren, einer ganzen Runde oder drei aufeinander folgenden Runden gewählt wurde, konnte auch dieses Problem lösen. Bei letzter Option wurde erzielte Differenz zwischen T_{QL} und T_{WF} um den Faktor 0.2 verringert.

9 Simulation mit Unity

Aus den Berichten der Vorgängergruppe gehen Probleme mit der Simulationssoftware Gazebo hervor, die sich während der Einarbeitung bestätigten. So läuft Gazebo auf einer virtuellen Maschine beispielsweise nur mit starken Perfomance-Einschränkungen und auch bei Desktop-Installationen kommt es immer wieder zu Abstürzen. Daher wurde sich während der Projektarbeit auch mit einem Umstieg auf die Simualtionssoftware Unity beschäftigt.

Unity ist eine der größten Plattformen zur Spieleentwicklung. Mit *Unity-Simulation*[17] kann man die Spieleplattform auch für 3D-Simulationen nutzen. Die Kommunikation zwischen ROS und dem derzeitig eingesetzen Gazebo läuft ausschließlich über Topics. Die JSON-API Rosbridge[18] kann für die Kommunikation zwischen ROS und Unity genutzt werden, indem es eine bidirektionale Kommunikation zwischen Unity und ROS ermöglicht und öffentliche Projekte, wie zum Beispiel das Toolset ROS#[19], illustrieren die Nutzungsschnittstelle. Daher wurde sich dazu entschieden parallel zur Einarbeitung in Gazebo, Unity als Alternative zu evaluieren.

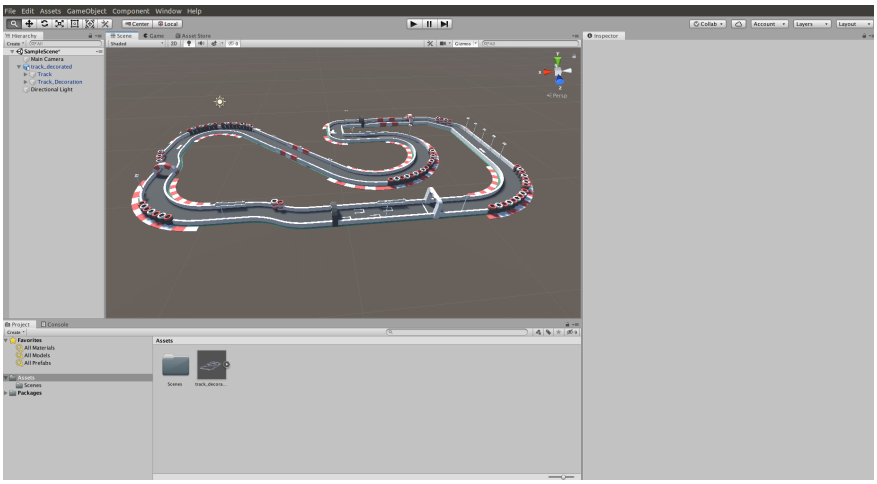


Abbildung 34: Screenshot des 3D-Modells der Teststrecke in Unity

Nach einer Evaluation wurde sich aus folgenden Gründen gegen eine Ablösung von Gazebo durch Unity entschieden:

[17] `https://unity.com/de/products/simulation`
[18] `http://wiki.ros.org/rosbridge`
[19] `https://github.com/siemens/ros-sharp`

1. Von der Vorgänger-Projektgruppe wurde eine fertige Simulationsumgebung in Gazebo übernommen. Das Nachbauen dieser würde einen erheblichen Mehraufwand neben der eigentlichen Projektarbeit bedeuten.

2. Unity ist derzeit nur für Windows verfügbar und offiziell unterstützt. Eine Bereitstellung für Ubuntu-Linux ist zwar für 2020 durch den Hersteller angekündigt [4], die Erfahrung zeigt aber, dass sich zum einen solch ein Release verzögern kann, zum anderen nicht abschätzbar ist, in welchem Umfang es zu den zuvor genannten Erweiterungen kompatibel ist. Zudem wäre eine zusätzliche Einarbeitung in Unity bei bereits fortgeschrittener Einarbeitung in Gazebo notwendig.

3. Während der Projektarbeit ist es gelungen Gazebo stabil aufzusetzen (siehe Unterabschnitt 4.2), sodass der hauptsächliche Grund für eine Alternative entfiel.

Grundsätzlich handelt es sich bei Unity um ein vielseitiges und mächtiges Werkzeug und es kann daher anderen Gruppen nur empfohlen werden, sich nach erfolgtem Release der Linux-Version damit zu befassen und es als Basis für Simulationen mit ROS in Erwägung zu ziehen.

10 Ergebnisse

Im Laufe der Projektgruppe wurden umfangreiche Veränderungen und Verbesserungen durchgeführt. Zudem wurde der Erfolg ständig im Simulator und abschließend in realen Tests überprüft. Im folgenden Abschnitt werden die messbaren Ergebnisse präsentiert und mit der Leistung der Software zu Beginn des Projektes verglichen. Die aktuelle Version wird mit **Wallfollowing 5 (WF5)**, die ursprüngliche mit **Wallfollowing 2 (WF2)** bezeichnet. Abschließend werden die Ergebnisse einer auf maschinellem Lernen basierenden Variante (**Q-Learning (QL)**) behandelt wie sie in Unterabschnitt 8.2 beschrieben wurde

10.1 Wallfollowing - Simulation

Für den Vergleich der in der Simulation erreichten Leistung wurden jeweils 10 Runden gefahren und die Daten aufgezeichnet. Wie in Abbildung 35 ersichtlich, liegt die gefahrene Geschwindigkeit beim Einsatz von WF5 oft und in Abschnitten in denen allgemein langsamer gefahren wird immer über der, die beim Einsatz von WF2 erreicht wird. Grundsätzlich ließe sich die Durchschnittsgeschwindigkeit vergleichen, da jedoch die Simulation exakt die Rundenzeiten aufzeichnen kann, bietet sich ein Vergleich dieser Aufzeichnungen an.

Die Auswertung dieser Daten zeigt, dass die **Rundenzeiten** in der neuen Version $\approx 2\,\mathrm{s}$ **geringer** sind, was einer **Verbesserung** von $\approx 9\%$ entspricht. Weitere Messwerte können Anhang L und Anhang P entnommen werden, zudem wird auf Anfrage gerne umfangreiches Daten- und Videomaterial zur Verfügung gestellt. Die hier verwendeten Messdaten stehen im GitHub bereit[20].

Es besteht die Vermutung, dass die Parameter von WF2 speziell auf die verwendete simulierte Strecke gut eingestellt wurden und daher die Geschwindigkeit von WF5 im Vergleich nicht noch mehr gesteigert werden konnte. Diese Vermutung wird durch die realen Fahrtests gestützt, in denen WF2 deutlich schlechtere Ergebnisse erzielte als in der Simulation. WF5 war wesentlich näher an der simulierten Leistung.

[20] https://github.com/arpg-sophisticated/ar-tu-do/tree/master/doc/results/report

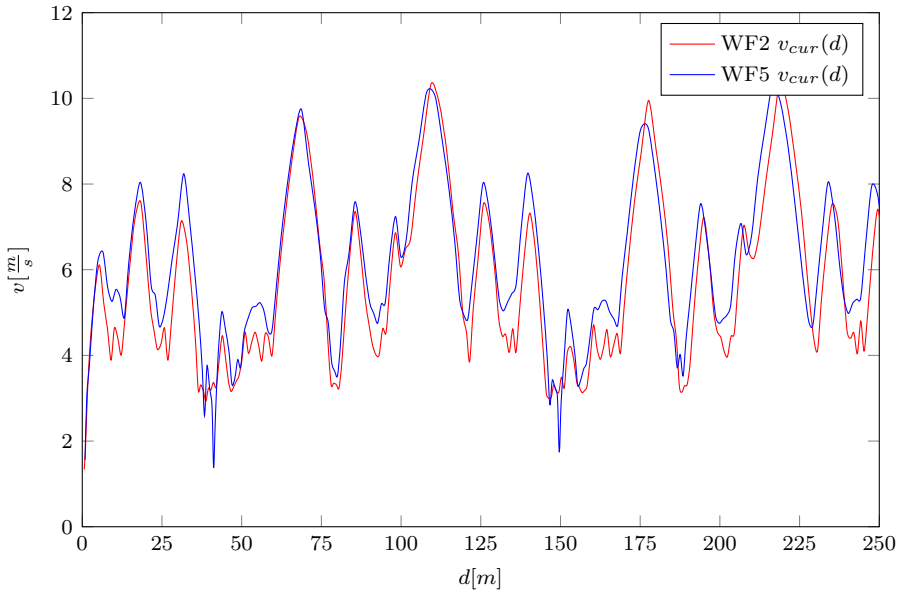


Abbildung 35: Vergleich Geschwindigkeit v nach gefahrener Distanz d, geglättet über 5 Werte (Simulation, Softwarestand v0.42g1)

```
[INFO] ... Lap 1 (forward): 0:21.36
[INFO] ... Lap 2 (forward): 0:20.96, average: 0:21.16
[INFO] ... Lap 3 (forward): 0:21.00, average: 0:21.11
[INFO] ... Lap 4 (forward): 0:20.81, average: 0:21.03
[INFO] ... Lap 5 (forward): 0:20.91, average: 0:21.01
[INFO] ... Lap 6 (forward): 0:20.85, average: 0:20.98
[INFO] ... Lap 7 (forward): 0:20.89, average: 0:20.97
[INFO] ... Lap 8 (forward): 0:20.96, average: 0:20.97
[INFO] ... Lap 9 (forward): 0:20.84, average: 0:20.95
[INFO] ... Lap 10 (forward): 0:20.93, average: 0:20.95
```

Abbildung 36: Rundenzeiten WF2 (Simulation, Softwarestand v0.42g1)

```
[INFO] ... Lap 1 (forward): 0:19.31
[INFO] ... Lap 2 (forward): 0:18.98, average: 0:19.15
[INFO] ... Lap 3 (forward): 0:18.89, average: 0:19.06
[INFO] ... Lap 4 (forward): 0:18.86, average: 0:19.01
[INFO] ... Lap 5 (forward): 0:18.83, average: 0:18.98
[INFO] ... Lap 6 (forward): 0:19.04, average: 0:18.99
[INFO] ... Lap 7 (forward): 0:19.11, average: 0:19.00
[INFO] ... Lap 8 (forward): 0:18.90, average: 0:18.99
[INFO] ... Lap 9 (forward): 0:19.08, average: 0:19.00
[INFO] ... Lap 10 (forward): 0:18.99, average: 0:19.00
```

Abbildung 37: Rundenzeiten WF5 (Simulation, Softwarestand v0.42g1)

10.2 Wallfollowing - Realer Test

Für den Vergleich der im realen Fahrbetrieb ohne Hindernisse erreichten Leistung wurden jeweils Auszüge von 500 Messwerten ausgewählt. Im Falle von WF5 wurde gefordert, dass das Fahrzeug 10 Runden mit diesen Einstellungen übersteht ohne dass es zum Crash kommt. Bei WF2 war dies leider nicht zu erreichen: Im Normalzustand erreicht das Fahrzeug nur eine sehr geringe Geschwindigkeit, wird diese auch nur geringfügig gesteigert kommt es irgendwann zum Unfall. Da mit der Optimierung der Parameter bei WF5 mehrere Tage verbracht wurden, wurden die dort gewonnenen Erkenntnisse dazu genutzt WF2 ebenfalls im Rahmen des Möglichen zu optimieren. Die hier für WF2 verwendete Messreihe stellt das ungefähre Maximum dar mit dem eine Runde sicher überstanden werden konnte. Zudem wurde die Geschwindigkeit bei der Aufzeichnung von WF2 10% zu hoch erfasst - dieser Fehler ist in den Plots und der nachfolgenden Auswertung bereits berücksichtigt.

Ein vergleichender Plot der Messdaten ist aufgrund des stark unterschiedlichen Verhaltens leider nicht möglich: Durch eine stark abweichende Trajektorie sind die zurückgelegten Distanzen unterschiedlich und durch die stark abweichende Geschwindigkeit kommt es auch beim Vergleich darüber irgendwann zu einem Auseinanderlaufen der Kurven, sodass ein vergleichender Plot keinen Vorteil bietet. Dennoch lässt sich in Abbildung 38 und Abbildung 39 erkennen, dass WF5 stark im Vorteil ist. Eine Auswertung der gesamten Protokolldatei ergibt:

- WF2: $v_{max} \approx 3.15 \frac{m}{s}$, $v_{avg} \approx 2.70 \frac{m}{s}$
- WF5: $v_{max} \approx 6.20 \frac{m}{s}$, $v_{avg} \approx 3.60 \frac{m}{s}$

Dies ergibt eine **Steigerung** von $\approx 96\%$ für die **Maximal-** und eine von $\approx 33\%$ für die **Durchschnittsgeschwindigkeit**. Berücksichtigt man hierbei,

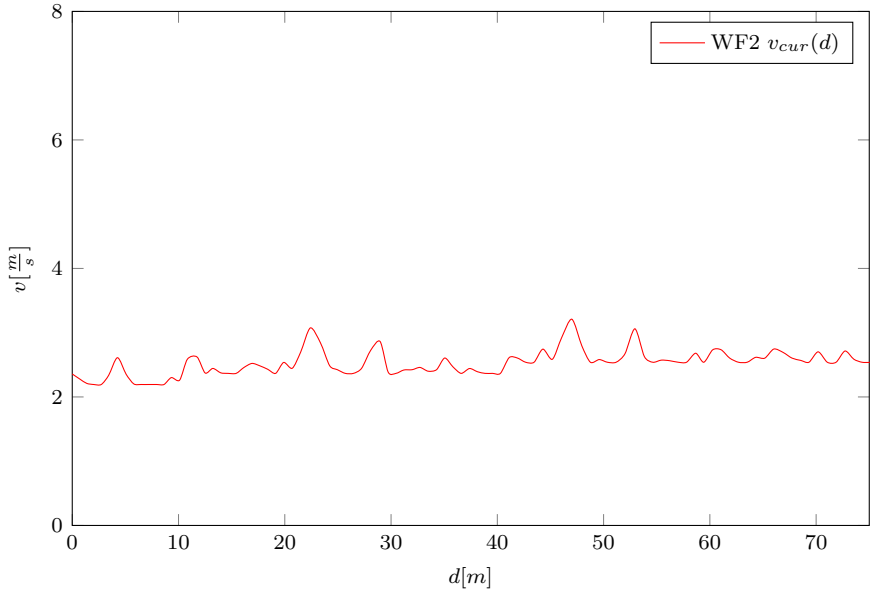


Abbildung 38: WF2: Geschwindigkeit v nach gefahrener Distanz d, geglättet über 5 Werte (Softwarestand v0.42r1)

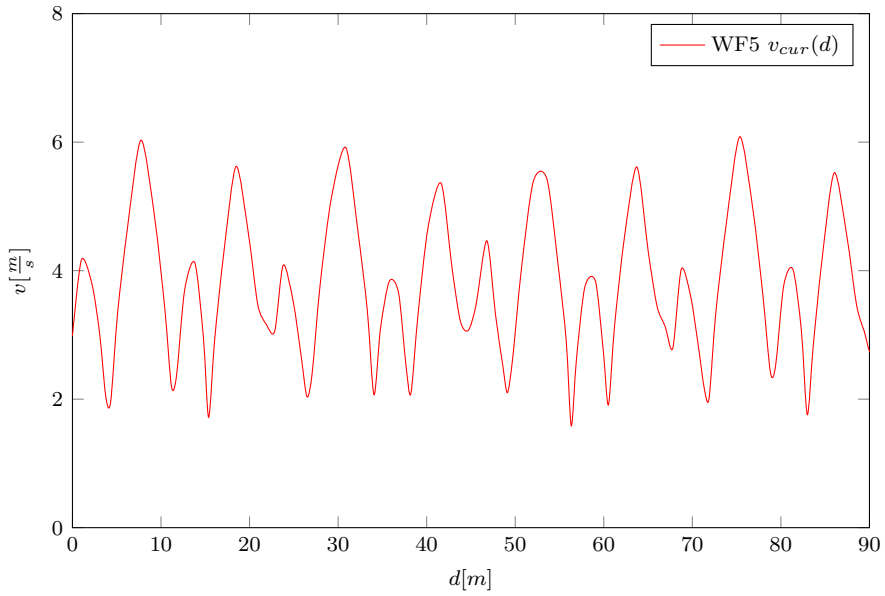


Abbildung 39: WF5: Geschwindigkeit v nach gefahrener Distanz d, geglättet über 5 Werte (Softwarestand v0.42r2)

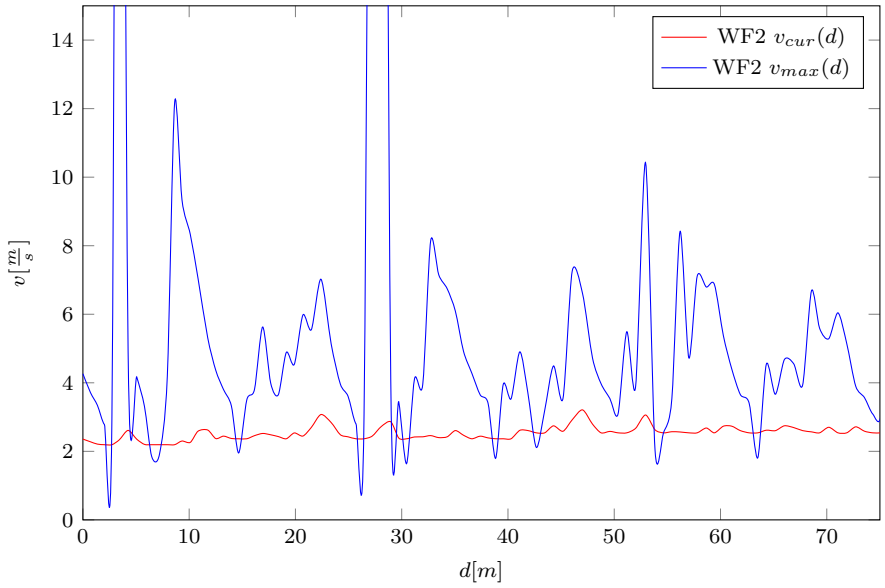


Abbildung 40: WF2: Geschwindigkeiten $v_{cur} und v_{max}$ nach gefahrener Distanz d, geglättet über 5 Werte (Softwarestand v0.42r1)

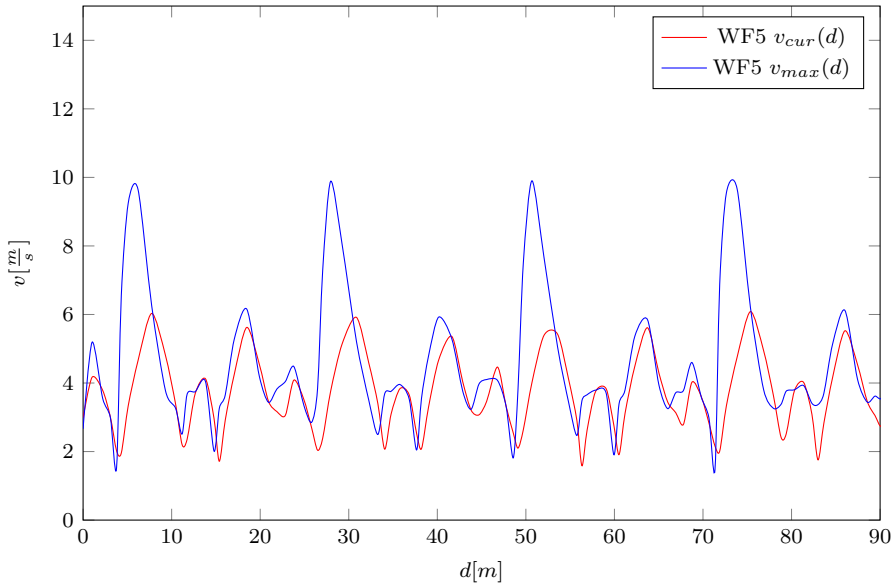


Abbildung 41: WF5: Geschwindigkeiten $v_{cur} und v_{max}$ nach gefahrener Distanz d, geglättet über 5 Werte (Softwarestand v0.42r2)

dass die berechnete Maximalgeschwindigkeit für die Teststrecke (vgl. Unterabschnitt 6.4) bei $7.00\frac{m}{s}$ liegt, kann dies als gutes Ergebnis gewertet werden, zumal beim Testen ein gewisser Sicherheitsabstand zu den Maximalwerten für die Parameter gewahrt wurde um bei diesen hohen Geschwindigkeiten keinen Crash und damit den Verlust des Versuchsfahrzeugs zu riskieren. Der Betrieb mit Hindernissen wurde ebenfalls erfolgreich getestet, wobei hier die Geschwindigkeit aus Sicherheitsgründen auf $4.25\frac{m}{s}$ beschränkt wurde. Die geringe Fahrbahnbreite von $\approx 1.5\,\mathrm{m}$ ließ keine Ausweichmanöver bei hohen Geschwindigkeiten zu.

In Abbildung 40 und Abbildung 41 werden jeweils die aktuell gefahrene und die vom Algorithmus als Maximum vorgegebene Geschwindigkeit ins Verhältnis gesetzt. Hierbei ist auffällig, dass WF5 grundsätzlich nahe dem Möglichen operiert und zudem, dass die berechneten Maximalwerte für die vorhandene Strecke realistischer sind. Die Tatsache, dass das Fahrzeug auf den langen Geraden, erkennbar an der großen Amplitude der blauen Kurven, nicht die Maximalgeschwindigkeit erreicht ist auf mangelnde Leistung des Motors zurückzuführen.

Weitere Messwerte können Anhang Q entnommen werden, zudem wird auf Anfrage gerne umfangreiches Daten- und Videomaterial zur Verfügung gestellt. Die hier verwendeten Messdaten stehen im GitHub bereit[21].

Da das Auto mit den bestehenden Fahralgorithmen nicht einer Idealline folgt, welche die kürzeste mögliche Rundenzeit ermöglichen würde, ist das Optimum noch nicht erreicht. Zudem ist die Kreisregression nicht immer völlig präzise, was ebenfalls zu einer Einschränkung der Geschwindigkeit führt.

10.3 Überwachtes Lernen

Im Folgenden werden die Ergebnisse erläutert, die mit dem in Unterabschnitt 8.1 erläuterten Ansatz des Überwachten Lernens erzielt werden konnten. Der Einfachheit halber wurde sich zunächst auf eine der beiden Zielgrößen Lenkwinkenl und Zielgeschwindigkeit beschränkt. Hierbei wurde sich für den Lenkwinkel entschieden, da dieser im anschließenden Fahrtest auch bei konstant geringer Geschwindigkeit getestet werden kann. Optimiert wurden die Modelle jeweils auf den durchschnittlichen quadratischen Fehler (MSE) [34] des normalisierten Lenkwinkels. Bei wird der Normalisierung wird der Lenkwinkel auf einen Wert zwischen -1 und 1 skaliert. Der normalisierte Lenkwinkel y_{norm} ergibt sich wie folgt:

[21] https://github.com/arpg-sophisticated/ar-tu-do/tree/master/doc/results/report

$$y_{norm} = \frac{y - \bar{y}}{y_{max} - y_{min}}$$

Hierbei ist $\bar{y}$ der Mittelwert des Lenkwinkels in der Trainingsmenge. Gleichzeitig stehen y_{max} und y_{min} für den Maximal bzw. Minimalwert aus der Trainingsmenge.

Wie in Unterabschnitt 8.1 beschrieben, wurden zwei verschiedene Netzstrukturen evaluiert. Für beide Netzstrukturen wurde jeweils ein Hyperparameter-Tuning [17] durchgeführt. Aufgrund der fehlenden Relevanz für das Resultat dieses Ansatzes wird an dieser Stelle nicht näher auf die verwendeten Netzstrukturen und die für das Tuning bereitgestellten Parameter eingegangen.

Der niedrigste MSE der im Training auf dem normalisierten Lenkwinkel erzielt werden konnte beträgt 0.000094943. Dies entspricht ungefähr einer durchschnittlichen Abweichung von $0,00974$ Radians, was wiederum einer durchschnittlichen Abweichung von $0.56°$ entspricht. Es ist zu beachten, dass es sich hierbei um die Trainingsfehler handelt.

Der erzielte Trainingsfehler lässt erwarten, dass ein autonomes Fahren mit dem trainierten Modell auf der Trainingsstrecke möglich ist. Während der Tests in der Simulation kam es jedoch häufig zu Kollisionen mit der Fahrbahnbegrenzung. Dies lässt sich darauf zurückführen, dass das Modell überangepasst auf die Fahrlinie des Wallfollowing-Algorithmus ist. Durch bereits kleine Abweichungen im Lenkwinkel wird diese Fahrlinie verlassen und das Modell befindet sich in Situationen, die es aus dem Training nicht kennt. Diesem Problem kann voraussichtlich mit einer Anreicherung der Trainingsdaten entgegnet werden. Aufgrund der Tatsache, dass mit diesem Ansatz nur eine Imitation des Wallfollowings erreicht werden kann, und der vielversprechenden Ergebnisse, die durch einen Q-Learning Ansatz erzielt werden konnten, wurde der Ansatz zum überwachten Lernen an dieser Stelle nicht weiter verfolgt.

10.4 Q-Learning

Das durch den Q-Learning Algorithmus trainierte Modell, wurde aus zweierlei Gründen ausschließlich in der Simulation getestet. Zunächst stand durch die vorübergehende Schließung der Universität nur begrenzte Zeit für reale Fahrttests zur Verfügung bei der die Optimierung des Wallfollowing-Algorithmus priorisiert wurde. Hinzukommt, dass das Modell bisher nur auf einer einzigen Strecke in der Simulation trainiert wurde. Es ist also davon auszugehen, dass bei Abweichungen von dieser keine guten Ergebnisse erzielt werden können.

In der Simulation wurde die Performance des Q-Learning Ansatzes mit der des WF2 und WF5 verglichen. Mit einer durchschnittlichen Rundenzeit von

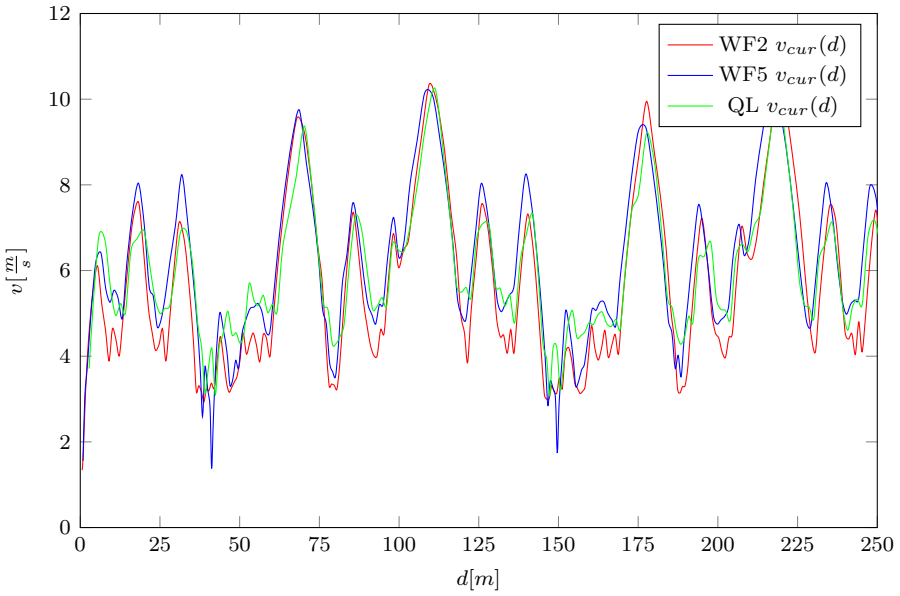


Abbildung 42: Vergleich Geschwindigkeit v nach gefahrener Distanz d, geglättet über 5 Werte (Simulation, Softwarestand v0.42q1)

18.93 Sekunden nach 10 Runden sind die Rundenzeiten mit durchschnittlich 2.02 Sekunden deutlich besser als die von WF2. Es ist sogar mit 0.07 Sekunden eine marginale Verbesserung gegenüber dem WF5 zu erkennen.

In Abbildung 42 ist zu erkennen, dass die Geschwindigkeit des Q-Learning Ansatzes vor allem in den Bereichen der Strecke in denen das Auto langsam fährt, deutlich höher ist als beim WF2-Ansatz. Der Grund hierfür ist die beschränkte Menge an möglichen Aktionen die das Neuronale Netz ausgeben kann. Hierbei ist keine Aktion vorhanden, die das Auto langsamer fahren lässt. So kann das Auto zwar eine andere Trajektorie fahren, aber es kann tendenziell nicht langsamer als der WF2-Algorithmus werden.

Des Weiteren wird deutlich, dass die Spitzengeschwindigkeit des WF5 vom Q-Learning Ansatz nicht erreicht werden können. Dies ist damit zu begründen, dass die Geschwindigkeit des WF2 durch das Q-Learning Modell nur begrenzt erhöht werden kann.

```
[INFO] ... Lap 1 (forward): 0:18.67
[INFO] ... Lap 2 (forward): 0:18.86, average: 0:18.77
[INFO] ... Lap 3 (forward): 0:18.92, average: 0:18.82
[INFO] ... Lap 4 (forward): 0:18.93, average: 0:18.85
[INFO] ... Lap 5 (forward): 0:19.02, average: 0:18.88
[INFO] ... Lap 6 (forward): 0:18.76, average: 0:18.86
[INFO] ... Lap 7 (forward): 0:19.09, average: 0:18.89
[INFO] ... Lap 8 (forward): 0:18.85, average: 0:18.89
[INFO] ... Lap 9 (forward): 0:19.26, average: 0:18.93
[INFO] ... Lap 10 (forward): 0:18.93, average: 0:18.93
```

Abbildung 43: Rundenzeiten QL (Simulation, Softwarestand v0.42g1)

11 Mögliche Aufgabenstellungen

Langfristig gibt es noch weitere Fragestellungen, mit denen sich unter Umständen eine Nachfolgegruppe beschäftigen sollte. Dies sind einerseits Probleme, die im aktuellen Stand aufgefallen sind, aber auch Ideen um von dieser Projektgruppe erarbeitete Lösungsansätze zu erweitern.

11.1 Optimierung der Parameter

Wie bereits dargestellt wird das Verhalten der einzelnen Fahralgorithmen durch Parameter bestimmt. Hier bietet es sich an eine automatische Optimierung an die jeweiligen Umstände anzustreben. Eine Möglichkeit wäre ein auf genetischen Algorithmen beruhendes Verfahren, eine andere die Optimierung durch maschinelles Lernen zu erreichen. So könnte zum Beispiel der Reibwert geändert und das Verhalten des Fahrzeugs (Stabilitätsverlust) mittels der IMU analysiert werden.
Eine weitere Variante könnte sein, den Reibwert kontinuierlich zu erhöhen und in Kurven zu prüfen, ob die berechnete Fliehkraft in einer Kurve mit dem gemessenen Wert aus der IMU übereinstimmt. Wenn der Wert aus der IMU signifikant kleiner ist als der berechnete, dann rutscht das Auto und der Reibwert sollte niedriger gewählt werden.

11.2 Austausch der Simulationsumgebung

Im Rahmen der Tests mussten wir feststellen, dass die verwendete Simulationsumgebung Gazebo zwar geeignet ist grundlegende Funktionalitäten zu entwickeln jedoch kein geeignetes Werkzeug darstellt um damit die Bedingungen einer realen Fahrt hinreichend zu simulieren: Während die Simulationen über mehrere Monate exzellente Ergebnisse lieferte fielen die ersten Fahrtests katastrophal aus. Im Rahmen der Vorbereitung zur Teilnahme an einer kommenden, virtuellen Veranstaltung[22] wurde der dort genutzt Simulator[23] bereits grundlegend in das Projekt integriert. Diese Integration sollte abgeschlossen werden, sodass auf Gazebo vollständig verzichtet werden kann.

[22] https://f1tenth.org/iros2020.html
[23] https://f1tenth.dev/

11.3 Austausch Totmannschalter

Wie in Unterunterabschnitt 4.5.2 erläutert birgt die derzeitige Lösung gewisse Risiken. Hier sollte darüber nachgedacht werden, die bisherige über eine dedizierte, in Hardware realisierte zu erweitern. So wäre es denkbar einen Funkschalter zu nutzen der die Stromversorgung zum Motor trennt, wodurch das Fahrzeug unmittelbar anhielte.

11.4 Spannungsversorgung

Derzeit besteht nur die Möglichkeit das Fahrzeug entweder per Netzteil oder einem Akku mit Spannung zu versorgen, ein Wechsel erfordert jedes mal einen kompletten Neustart des Systems. Hier empfiehlt es sich eine Schaltung zu installieren die es ermöglicht mehrere Spannungsversorgungen zu verbinden, sodass ein Wechsel der Akkus oder ein Anschluss an das Netzteil im laufenden Betrieb möglich wird.

11.5 Permanente Testumgebung

Es sollte eine permanente Testmöglichkeit zur Verfügung gestellt werden. Wie in Unterabschnitt 11.2 erläutert und die eigene Testerfahrung bestätigt reicht eine Simulation für die Realisierung von leistungsfähigen und sicheren Fahralgorithmen nicht aus. Eine permanent zur Verfügung stehende Testumgebung böte einige Vorteile:

- Es könnten regelmäßig und kurzfristig Tests durchgeführt werden.
- Da der Auf- und der Abbau der Strecke, je nach Größe eine enorme Menge Zeit in Anspruch nehmen kann entfiele dieser Verlust und die Effizienz einer Sitzung würde nachhaltig gesteigert.
- Durch eine permanent installierte Strecke ist eine Reproduzierbarkeit über mehrere Sitzungen gegeben. Während unserer Tests stellten wir fest, dass selbst kleine Veränderungen an der Strecke das Verhalten des Fahrzeugs grundlegend ändern können - ein Umstand der die Fehlersuche oder Optimierung unnötig erschwerte.

- Teile der in Abschnitt 4 beschriebenen Infrastruktur werden zum Ende der Projektgruppe außer Betrieb genommen, da sie entweder nicht weiter verwaltet werden können (Server) oder Privateigentum darstellen (Controller, Router). Hier sollte in Abstimmung mit den derzeitigen Teilnehmern entsprechender Ersatz gesucht bzw. ein Konzept erarbeitet werden um den Teilnehmern eine ähnlich komfortable Entwicklungsumgebung bieten zu können. Diese sollte durch die Leitung der Projektgruppe bereitgestellt werden, da der durchschnittliche Teilnehmer nicht über die nötigen Kenntnisse im Bereich Linux-Server oder Netzwerktechnik verfügt.

11.6 Verbesserung Hindernisumfahrung

Der aktuelle Zustand des Features Hindernisse zu umfahren ist in Work in Progress. So sollte im nächsten Schritt zunächst der Bug im Code, welcher in Unterabschnitt 7.4 erwähnt wurde. Nach Beseitigung des Bugs, sollten zunächst Tests vollzogen werden, ob dieser Ansatz eine gute Möglichkeit bietet um Hindernisse zu umfahren.

Nach der Überprüfung des bisherigen Ansatz sollten im nächsten Schritt eine Methodik entwickelt werden die mit mehreren Hindernissen umgehen kann. Als letzter Schritt sollte eine Methodik entwickelt werden, die dynamischen Systemen ausweichen kann. Dabei sollte zu beachten sein, dass ein vorher fahrendes Fahrzeug in einem Rennen zunächst aufgeholt werden muss, bis zu einem Zeitpunkt wo ein Überholmanöver möglich ist. Also muss ein Trade-Off zwischen Aufholen und der Positionierung für ein Überholmanöver beachtet werden.

12 Zusammenfassung

In den ersten sechs Monaten ist es der Projektgruppe gelungen, eine sehr gute Infrastruktur der benötigten Entwicklungssysteme aufzusetzen. Gleichzeitig wurde bereits durch die initiale Auseinandersetzung mit grundlegenden Themen der Physik, Fahrzeugtechnik, Informatik und des Projektmanagements eine unbedingt erforderliche Basis für die weitere Arbeit gelegt. Ferner ist es gelungen, den übernommenen Wallfollowing-Algorithmus erfolgreich zu verbessern und zusätzlich durch diverse Aufbauten realer Teststrecken vermehrt unter realen Bedingungen zu testen.

Die Probleme, die bei den Fahrtests aufgetreten sind, waren größer als erwartet. Dabei hat sich insbesondere die Stabilität des Fahrzeugs beim Lenken bemerkbar gemacht. Dadurch hat sich der Beginn der Bearbeitung der eigentlichen Problematik, das Fahrzeug überholen zu lassen, verzögert. Eine Ursache dafür ist wie bereits erwähnt der arbeitsaufwändige Aufbau der Teststrecken und die Leistungsfähigkeit der Hardware bei hohen Geschwindigkeiten. Durch das Aufräumen des Betriebssystems auf dem Auto und dem portieren des Pythons Codes nach C++, ist es jedoch gelungen neben der Fahrstabilität des Wallfollowing-Algorithmus auch die allgemeine Leistungsfähigkeit des Autos zu erhöhen.

In den zweiten sechs Monaten konnte der Wallfollowing soweit verbessert werden, dass neben der höheren Fahrstabilität auch eine Geschwindigkeitserhöhung stattgefunden hat. So ist der Wallfollowing 5 Algorithmus bei höherer Geschwindigkeit Fahrstabiler als sein Vorgänger. Mit der Verwendung eines Q-Leariningansatzes konnte die Rundenzeit um weitere $0.7s$ verbessert werden. Zudem konnte die Pipeline um ein Clustering Algorithmus, dem DBScan, erweitert werden. Mit dem DBScan lassen sich neben dem Erkennen von Wänden, auch Hindernisse auf der Strecke erkennen. Mit den aus dem DBScan gewonnenen Voxeln, lässt sich die Hindernisumfahrung realisieren. Die Hindernisumfahrung selbst, wurde nebenfalls implementiert. Auf Grund der Sicherheit des Fahrzeuges und seiner Umgebung, konnten Hindernisse nur bei geringer Geschwindigkeit umfahren werden.

Alles in Allem wurden die geforderten Ziele, trotz der Coronapandemie, dieser Projektgruppe erfolgreich umgesetzt. So ist neben der erhöhten Geschwindigkeit und der höheren Fahrstabilität auch das Umfahren von Hindernissen bei geringeren Geschwindigkeiten möglich. Zusätzlich konnte die Infrastuktur erweitert werden und durch den Einsatz von Skripten eine teilweise Automatisierung der Entwicklugsumgebung hergestellt werden.

Literatur

[1] 620, P. Projektgruppe 620: Autonomous Racing - Endbericht, 2018.

[2] Abbas, H., Behl, M., Brady, M., Hu, P., Hu, T., Jain, P., Kelkar, P., N, N. K., Mangharam, R., Ng, L. W. P., O'Kelly, M., and Sharer, C. F1Tenth - The Rules. https://f1tenth.org/misc-docs/rules.pdf, 2016. Zugriff: 21.10.2019.

[3] Ahmad, M. O., Markkula, J., and Oivo, M. Kanban in Software Development: A Systematic Literature Review. Software Engineering and Advanced Applications (SEAA) (09 2013), 9–16.

[4] Announcing the Unity Editor for Linux. https://blogs.unity3d.com/2019/05/30/announcing-the-unity-editor-for-linux/. Abgerufen: 24.03.2020.

[5] Bardt, H. Autonomes Fahren: Eine Herausforderung fuer die deutsche Autoindustrie. IW-Trends - Vierteljahresschrift zur empirischen Wirtschaftsforschung 43, 2 (2016), 39–55.

[6] Barlow, H. B. Unsupervised learning. Neural computation 1, 3 (1989), 295–311.

[7] Bergmann, R., and Garrecht, M. Projektmanagement. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016, pp. 231–260.

[8] Birant, D., and Kut, A. ST-DBSCAN: An algorithm for clustering spatial–temporal data. Data & Knowledge Engineering 60, 1 (2007), 208–221.

[9] F1 Tenth Build Manual V2. http://f1tenth.org/build/BuildV2.pdf. Abgerufen: 26.03.2020.

[10] Forbes: Nvidia is coming to your datacentre. https://www.forbes.com/sites/moorinsights/2019/04/12/nvidia-is-coming-for-your-data-center/#76fce4871603. Abgerufen: 26.03.2020.

[11] Gazebo Community: Limits of the Simulation. https://community.gazebosim.org/t/limits-of-the-simulation/54/9/. Abgerufen: 24.03.2020.

[12] Geraldi, J., and Lechler, T. Gantt charts revisited: A critical analysis of its roots and implications to the management of projects today. International Journal of Managing Projects in Business 5, 4 (2012), 578–594.

[13] Hossain, E., Ali Babar, M., and Paik, H.-y. Using Scrum in Global Software Development: A Systematic Literature Review. Proceedings - 2009 4th IEEE International Conference on Global Software Engineering, ICGSE 2009 (07 2009), 175–184.

[14] Imhof, A., Oetiker, M., and Jensen, B. Wall following for autonomous robot navigation. 2012 2nd International Conference on Applied Robotics for the Power Industry (CARPI) (2012), 1–4.

[15] Jakoby, W. Projektmanagement für Ingenieure. Springer Viewung, 2007, p. 30.

[16] Jazar, R. N. Steering Dynamics. Springer US, Boston, MA, 2008, pp. 379–454.

[17] Kim, Y. Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882 (2014).

[18] Koubaa, D. A. Robot Operating System (ROS), The Complete Reference (Volume 3), 2019.

[19] Moe, N., and Dingsøyr, T. Scrum and Team Effectiveness: Theory and Practice. Lecture Notes in Business Information Processing 9 (06 2008), 11–20.

[20] CUDA Zone. `https://developer.nvidia.com/cuda-zone`. Abgerufen: 17.03.2020.

[21] Nvidia Developers: CUDA GPUs. `https://developer.nvidia.com/cuda-gpus`. Abgerufen: 26.03.2020.

[22] NVIDIA Jetson TX2 Board. `https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/jetson-tx2/`. Zugriff: 21.10.2019.

[23] O'Kelly, M., Sukhil, V., Abbas, H., Harkins, J., Kao, C., Pant, Y. V., Mangharam, R., Agarwal, D., Behl, M., Burgio, P., and Bertogna, M. F1/10: An Open-Source Autonomous Cyber-Physical Platform, 2019.

[24] Rally ST Specifications. `https://traxxas.com/products/models/electric/ford-fiesta-st-rally?t=specs`. Abgerufen: 09.10.2019.

[25] Reibbeiwerte. `https://www.schweizer-fn.de/stoff/reibwerte/reibwerte.php`. Abgerufen: 09.10.2019.

[26] ROS Wiki: Dynamic Reconfigure. `http://wiki.ros.org/dynamic_reconfigure/Tutorials`. Abgerufen: 16.09.2020.

[27] ROS Wiki: Rosbag Commandline. http://wiki.ros.org/rosbag/Commandline. Abgerufen: 26.03.2020.

[28] Schmidt-Hieber, J., et al. Nonparametric regression using deep neural networks with relu activation function. Annals of Statistics 48, 4 (2020), 1875–1897.

[29] Schubert, E. Vorlesungsfolien: Maschinelles Lernen. PART III: CLUSTERING, 2018.

[30] Traxxas Ford Fiesta ST Rally - Specifications. https://traxxas.com/products/models/electric/ford-fiesta-st-rally?t=specs. Zugriff: 21.10.2019.

[31] Wikipedia: CUDA. https://de.wikipedia.org/wiki/CUDA. Abgerufen: 26.03.2020.

[32] Wikipedia: Simultaneous Localization and Mapping. https://de.wikipedia.org/wiki/Simultaneous_Localization_and_Mapping. Abgerufen: 27.03.2020.

[33] Wikipedia: Stereoskopie. https://de.wikipedia.org/wiki/Stereoskopie. Abgerufen: 27.03.2020.

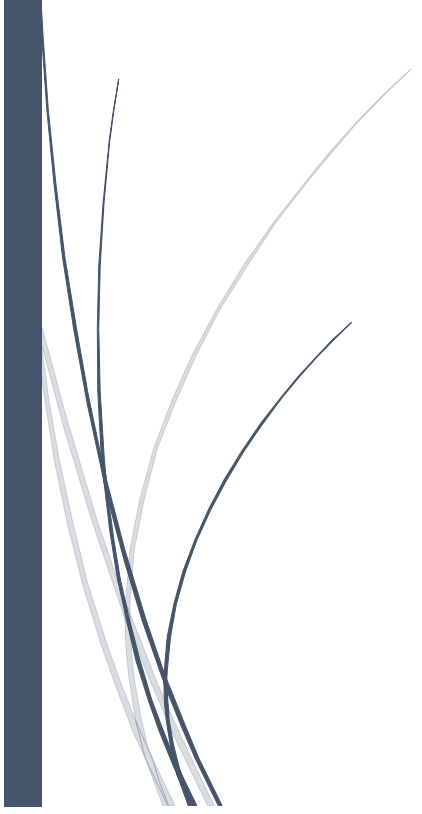
[34] Willmott, C. J., and Matsuura, K. Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. Climate research 30, 1 (2005), 79–82.

A Projektmanagement

19.10.2019

Projektmanagement

PG Autonomous Racing Head to Head Racing Capabilities



Timo Gojowczyk
TU DORTMUND

Inhaltsverzeichnis

Einleitung

Zu Beginn des Computer Zeitalters benötigte man für Software ein Minimum an Projektmanagement. Im Laufe der Jahre wurde Software immer komplexer und umfangreicher. Begünstigt wurde dieses durch immer schneller werdende Computer. Mit dem Umfang der Software wuchs auch die Anforderung an das Projektmanagement. SO wurden anfangs prozedurale Projektmanagement eingeführt, schnell wurde jedoch klar, dass sich diese als nicht sinnvoll erwiesen. Aktuell werden Software Projekte immer häufiger agil umgesetzt.

Diese Ausarbeitung soll die Grundlagen des Projektmanagements den PG Teilnehmern näherbringen. Dabei wird insbesondere auf die verschiedenen Phasen und Aufgaben, die im Projektmanagement anfallen eingegangen.

Definition

Dieses Kapitel soll Grundlegende Begriffe des Projektmanagements erläutern.

Projekt

Ein Projekt ist ein multipersonales Vorhaben zur Erreichung eines neuartigen Ziels in vorgegebener Zeit und mit begrenzten Ressourcen.[1]

Dabei Zeichnen sich Projekte durch folgende Merkmale aus:

- Zielklarheit
- Schwierigkeit
- Prozesscharakter
- Beteiligte
- Ressourcenbegrenzung,
- Terminierung

Wichtig hierbei ist u.A. das nicht immer alle Punkte gleichstark zutreffen. So kann es sich auch um ein Projekt handeln, sofern eine Ressourcenbegrenzung nicht zu 100% geben ist. Dabei werden in [1] 7 Fragen formuliert, mit denen man ein Projekt erschließen kann. Diese Fragen ziehen sich nicht nur durch die Projektdefinition, sondern finden sich auch in vielen anderen Aspekten des Projektmanagements wieder:

- Was ist das Problem?
- Was muss getan werden?
- Wie viel Aufwand ist erforderlich?
- Wer soll es tun?
- Wann soll es getan werden?
- Was könnte schief gehen?
- Läuft es nach Plan?

Projektmanagement

Management ist die Planung und Steuerung von Prozessen. [1]

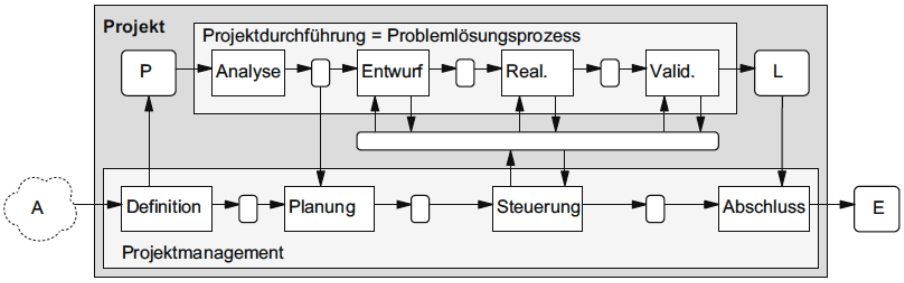


Abb. 1.16 Grobstruktur eines gemanagten Projekts

Wie in Abb. 1.16 zu sehen ist, beginnt der Prozess des Projektmanagements vor der eigentlichen Projektarbeit. Mit dem Abschluss der Planung, befasst sich das Projektmanagement mit der Steuerung. Gesteuert werden Projekte in den folgenden Punkten:

- Problemlösungsprozess
- Zielsysteme
- Aufwandsschätzung
- Qualität
- Ablauf
- Risiken
- Kosten
- Den Projektbeteiligten

Dazu findet man in [1] eine Vielzahl an Checklisten.

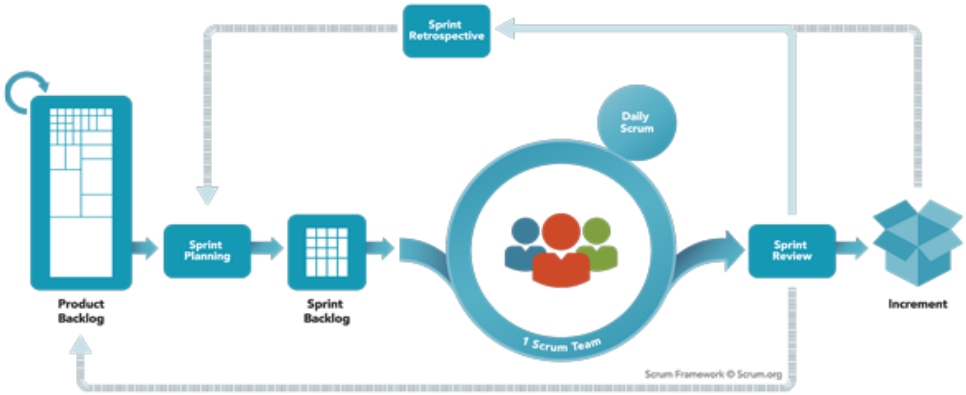
	Frage	Vertiefung in Checkliste
1.	Was ist das Problem?	Problemlösen + Projektgründung
2.	Was muss getan werden?	Strukturplanung
3.	Wie viel Aufwand ist erforderlich?	Aufwandsschätzung
4.	Wer soll es tun?	Projektorganisation
5.	Wann soll es getan werden?	Ablauf- und Terminplanung
6.	Wo sind die Risiken?	Risikomanagement
7.	Läuft es nach Plan?	Kosten- und Qualitätsmanagement Projektsteuerung + -abschluss

Projektorganisation

Dieses Kapitel beschäftigt sich mit der Organisation der Projektarbeit. Dabei wird insbesondere auf agile Vorgehensmodelle eingegangen.

Scrum

Scrum ist das am häufigsten benutzte Framework für agile Softwareprojekte. Die Vorteile von Scrum liegen in den kurzen Iterationszyklen und der inkrementellen Produktentwicklung.



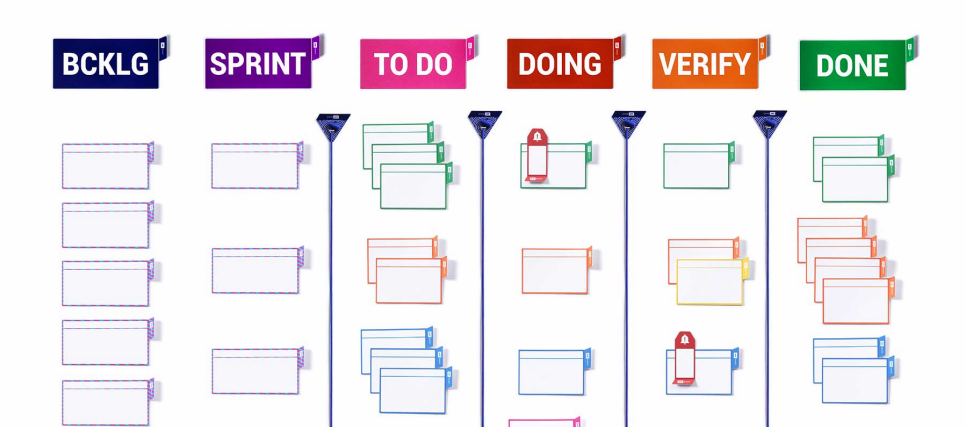
Download The Scrum Framework

Die wichtigsten Elemente von Scrum sind:

- Product Backlog: Hier werden alle Userstories die noch entwickelt werden müssen gespeichtert.
- Sprint Backlog: Hier befinden sich alle Userstories die in diesem Sprint entwickelt werden.
- Sprint: Ein Sprint ist ein Zeitraum von ca. 2-4 Wochen in dem eine Iteration des Entwicklungsprozesses umgesetzt wird.
- Daily Scrum: Ein tägliches Meeting, an dem alle ihren Fortschritt und die Probleme auf die sie gestoßen sind, berichten.
- Increment: Fertiges Softwareprodukt am Ende des Sprints.
- Retrospective: Nach jedem Sprint wird hier sprochen was gut und was nicht so gut gelaufen ist.

Kanban

Im Gegensatz zu Scrum definiert Scrum nicht ein genaues Vorgehensmodell. Es bietet vielmehr eine Visualisierung der Arbeitsschritte und den Status der Aufgaben. Das Hauptelement von Kanban ist das Kanbanboard.



Das Kanbanboard enthält als Spalten die einzelnen Arbeitsschritte, also den workflow. Elemente der Spalten sind jeweils Ausfgaben die sich in diesem Arbeitsschritt befinden.

Häufig wird eine Mischung aus Scrum und Kanban benutzt. Diese nennt sich „Scrumban“

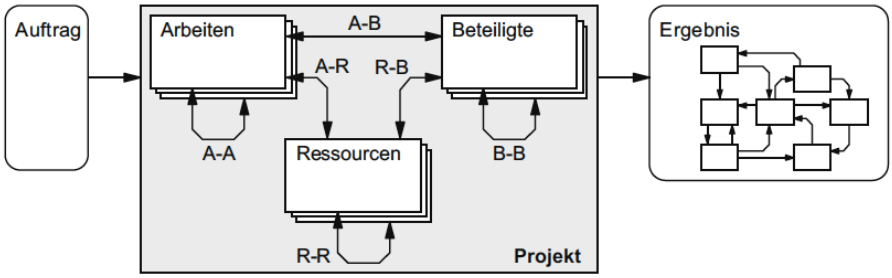
Projektplanung

Hauptaufgabe der Strukturplanung ist das Erstellen der Projektsturktur. Das Artefakt, dass hierbei entsteht ist der Projektstrukturplan. Der Projektstrukturplan definiert die Phasen, die das Projekt durchläuft und deren Aufgaben.

1	**Vorprojekt**
2	Bestandsaufnahme vor Ort
3	Grobe Bedarfsermittlung
4	Grobkonzept
5	Grobe Marktanalyse
6	Angebot erstellen
7	**Analyse und Entwurf**
8	Detaillierte Bedarfsanalyse
9	Detailkonzept ausarbeiten
10	Analgenpläne zeichnen
11	Terminierten Ablaufplan entwerfen
12	**Beschaffung**
13	**Genaue Marktanalyse**
14	Solarkollektoren (inkl. Halter + Verbdinung)
15	Solarmodul (inkl. Rohre)
16	Steuerung (inkl. Fühler + Leitungen)
17	Wasserspeicher
18	**Einholung von Angeboten**
19	Solarkollektoren (inkl. Halter + Verbdinung)
20	Solarmodul (inkl. Rohre)
21	Steuerung (inkl. Fühler + Leitungen)
22	Wasserspeicher
23	Erstellung Preisspiegel
24	Bestellung

25	**Aufbau**
26	Ausbau und Entsorgung alter Komponenten
27	Maurerarbeiten für Leitungsführung
28	Einbau der Rohr-Leitungen
29	Gerüst aufstellen
30	Montage der mech. Dachhalterungen
31	Montage der Solarkollektoren
32	Einbau Wärmespeicher
33	Anschluß aller thermischen Komponenten
34	Montage der elektr. Leitungen
35	Einbau Solarstation
36	Einbau Steuerung
37	Anschluß und Prüfung aller elektr. Komponenten
38	Gerüst abbauen
39	**Dokumentation**
40	Betriebsanleitung
41	Bedienungsanleitung
42	Wartungsvorschrift
43	**Anlagentest**
44	Befüllung und Dichtigkeitsprüfung
45	Inbetriebnahme
46	Einweisung des Betreibers

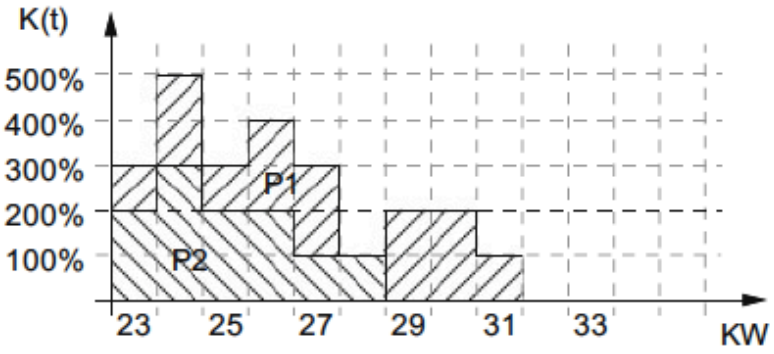
Auf Basis dieses Plans werden alle weiteren Pläne erstellt und angepasst. Wenn der PSP steht, kann er mithilfe eines Gantt-Diagramms visualisiert werden.
Unter anderem lässt sich durch den Produktstrukturplan ein Abhängigkeitsgraph entwickeln.



Dabei können durch obenstehende Abhängigkeiten die Voraussetzungen für die einzelnen Aufgaben dargestellt werden.

Ressourcenplanung

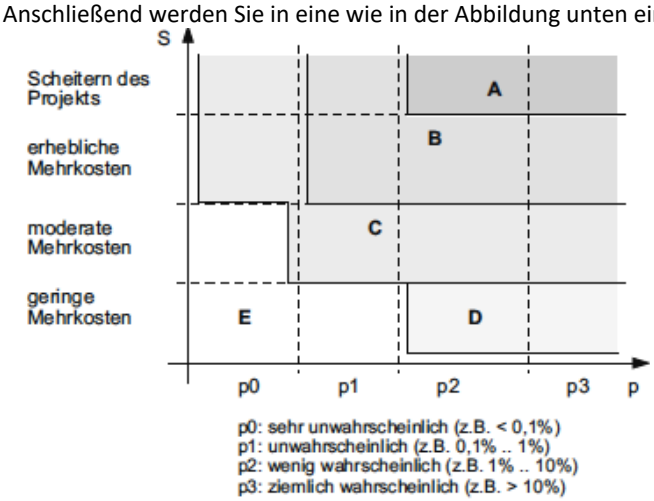
Da in Projekten nur eine begrenzte Anzahl an Ressourcen zur Verfügung steht ist eine möglichst gleichmäßige Verwendung von Nöten. Als Framework hierfür dient das Kapazitätsgebirge.



Das Kapazitätsgebirge stellt die Benutzung einer Ressource zeitlich dar. In der obigen Abbildung werden Ressourcen P1 und P2 zeitlich in Wochen dargestellt. Eine Optimierung erfolgt durch reduzieren der Spitzen erzielt. So kann durch geschickte Umplanung die Spitzenbelastung von 500 % auf 300% gewährleistet werden. Durch diese Optimierung kann Personal besser eingesetzt und das Verfügbare Material gezielt genutzt werden, um eine Neuanschaffung zu verhindern.

Risikomanagement

Projekte haben neben positiven Effekten auch immer eine gewisse Anzahl an Risiken. Diese Risiken lassen sich durch die Merkmale eines Projektes nicht ausschließen. Um diese zu managen müssen sie zunächst klassifiziert werden. Dazu wird für jedes Risiko eine Schätzung des Schadensausmaßes als auch der Eintrittswahrscheinlichkeit durchgeführt. Anschließend werden Sie in eine wie in der Abbildung unten eingefügt.



Den Schaden den ein Risiko bei Eintritt anrichtet wird durch Schadensausmaß * Eintrittswahrscheinlichkeit errechnet.

Risiken der Kategorie A und B sollten nach Möglichkeit ausgelagert werden. Risiken der Kategorie C sollten gemindert werden. Dabei lassen sich diese Risiken durch 2 Parameter beeinflussen. Entweder wird die Eintrittswahrscheinlichkeit und/oder das Schadensausmaß reduziert. Risiken der Kategorie D und E werden ertragen.

Die Komponente Mensch

Viele Menschen an einem Projekt bedeutet auch viele Individuelle Eigenschaften, Bedürfnisse und interpersonelle Kommunikation. Die Ziele der Gruppen Mitglieder müssen nicht immer deckungsgleich sein und können sich im Extremfall Widersprechen. Wichtig für den Erfolg eines Projektes ist es alle Beteiligte bei Laune und auf einem Kurs zu halten. Es bringt nichts, wenn es 7 Einzelspieler statt einer Mannschaft gibt. Dabei durchläuft jede Gruppe folgende 4 Phasen:

Phase	Engl. Bez.	Charakteristische Merkmale
Orientierungsphase	Forming	Unsicherheit, Kennenlernen, Formieren
Konfliktphase	Storming	Konflikte, Konkurrenzdenken, Machtproben
Normierungsphase	Norming	Zusammenrücken, gemeinsame Ziele, Etablierung von Regeln
Leistungsphase	Performing	Kooperation, Offenheit, Verständnis

B Koordinatensysteme und Koordinatentransformationen

Koordinatensysteme und Koordinatentransformationen

Robin Thunig
Autonomous Racing Group, LS 12, TU Dortmund
robin.thunig@tu-dortmund.de

1 EINLEITUNG

Damit sich ein Fahrzeug, wie das in der Projektgruppe verwendete, im Raum bewegen kann, ist es hilfreich die Position des Fahrzeuges auf der Rennstrecke zu kennen. Außerdem ist es erstrebsam die physikalischen Fahreigenschaften effektiv auszunutzen. Dafür ist es nötig die genaue Geschwindigkeit und Fahrtrichtung des Fahrzeuges zu kennen.
Jedoch stehen die Informationen über den Zustand des Fahrzeuges nur in dem Fahrzeug eigenen Koordinatensystem zur Verfügung. Das Problem in dieser Arbeit soll sein, die Positionskoordinaten und Richtungsvektoren in das Koordinatensystem der Rennstrecke zu transformieren, damit eine Positions-, Richtungs-, und Geschwindigkeitsbestimmung möglich wird mit der sich womöglich der Pfad des Fahrzeuges auf der Rennstrecke nachvollziehen lässt. Dies könnte ermöglichen, dass das Fahrzeug Positionsdaten über die Strecke sammeln kann und in nachfolgenden Runden so schneller auf der Strecke fahren könnte, da besser geplant werden kann, wie das Fahrzeug zum Beispiel durch Kurven fährt oder wo es Risikofrei beschleunigen kann.

2 KÖRPER- UND RAUMFESTE SYSTEME

Wie in der Einleitung erwähnt, befindet sich das Fahrzeug in einem Fahrzeug eigenen Koordinatensystem, das auch Körperfestes System genannt wird. Dieses Körperfeste System befindet sich wiederum in einem Raumfesten System, das in diesem Fall dem Koordinatensystem der Rennstrecke entspricht. In Abbildung **??** ist dieses Fahrzeug in den Koordinatensystemen visualisiert. Zudem sind die Beschleunigung und Richtung der beiden Vorderräder mit grünen Richtungsvektoren dargestellt. Ein Ziel könnte es sein diese Richtungsvektoren und die Position des Fahrzeuges, die in dem Körperfesten System bekannt sind, in das Raumfeste System zu transformieren, damit ermittelt werden kann wie sich das Fahrzeug im Raumfesten System beziehungsweise auf der Rennstrecke bewegt.

3 POSITION IM BEZUGSYSTEM

In Kapitel soll die Position des Fahrzeuges aus dem Körperfesten System in das Raumfeste System transformiert werden, damit die Position aus dem Fahrzeug eigenen Koordinatensystem in das der Rennstrecke überführt werden kann. Dafür soll in Kapitel 3.1 die Transformation von Ortsvektoren nach Rotation und in Kapitel 3.4 nach Translation des Körperfesten Koordinatensystems behandelt werden.

3.1 Rotation

In diesem Kapitel wird auf die mathematischen Grundlagen eingegangen werden wie in Kapitel 3.2 ein Vektor und in Kapitel 3.3 ein Koordinatensystem rotiert wird und wie die Koordinaten von einem Koordinatensystem in ein anderes transformiert werden können.

3.1.1 Rotation eines Vektors

Ein Vektor kann durch seine Polarkoordinaten (r, Φ) und seine kartesischen Koordinaten (x, y) beschrieben werden. Eine Umwandlung von Polarkoordinaten in kartesische Koordinaten ist durch folgende Formeln möglich:

$$\begin{aligned} x &= r\cos\Phi \\ y &= r\sin\Phi \end{aligned} \tag{1}$$

Wird ein Vektor um den Winkel θ rotiert, dann sind die neuen Polarkoordinaten $(r, \Phi + \theta)$. Durch die

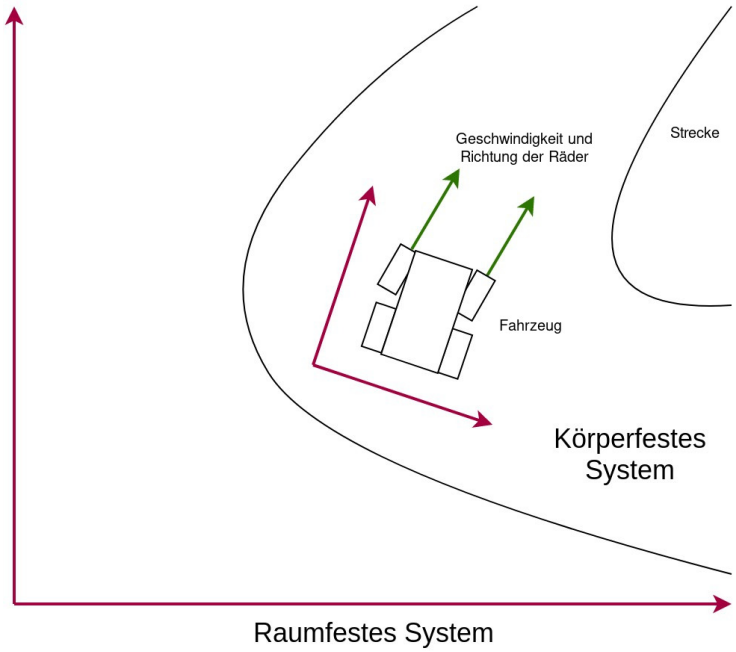


Abb. 1: Fahrzeug in seinem Körperfesten Koordinatensystem, dass sich einem Raumfesten Koordinatensystem befindet

entsprechende Umwandlung von Polarkoordinaten in kartesische Koordinaten ergeben sich die neuen kartesischen Koordinaten (x', y'):

$$\begin{aligned} x' &= r\cos(\Phi+\theta) \\ &= r\cos\Phi\cos\theta - r\sin\Phi\sin\theta \\ &= (r\cos\Phi)\cos\theta - (r\sin\Phi)\sin\theta \\ &= x\cos\theta - y\sin\theta \end{aligned}$$

$$\begin{aligned} y' &= r\sin(\Phi+\theta) \\ &= r\sin\Phi\cos\theta - r\cos\Phi\sin\theta \\ &= (r\sin\Phi)\cos\theta - (r\cos\Phi)\sin\theta \\ &= y\cos\theta - x\sin\theta \\ &= x\sin\theta - y\cos\theta \end{aligned} \tag{2}$$

Diese Formeln lassen sich auch als Matrixmultiplikation darstellen in Form einer Rotationsmatrix und einem Vektor:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{3}$$

In Abbildung **??** wird die Rotation eines Vektors um den Ursprung visualisiert.

3.1.2 Rotation eines Koordinatensystems

Nachdem in Kapitel 3.2 gezeigt werden konnte wie die kartesischen Koordinaten eines Ortsvektors nach einer Rotation um einen Winkel θ bestimmt werden, soll das Konzept auf die Transformation eines Ortsvektors von einem Körperfesten Koordinatensystem in ein Raumfestes Koordinatensystem erweitert werden.
Die Zielsetzung ist dabei den Punkt ${}^b p = ({}^b x, {}^b y)$ aus dem Körperfesten System nach einer Rotation zu dem Punkt ${}^a p = ({}^a x, {}^a y)$ des Raumfesten Systems zu transformieren. Das hochgestellt a gibt dabei an, dass der Punkt aus dem Raumfesten System betrachtet wird und b aus dem Körperfesten System.
Wenn das Körperfeste System b um den Winkel θ rotiert wird, bleiben die Koordinaten der Punkte aus b relativ zu b gleich. Jedoch verändern sich die Koordinaten aus b relativ zum Raumfesten System a. Die Koordinaten des Punktes ${}^a p = ({}^a x, {}^a y)$ können unter zur Hilfenahme der Formel 3 folgendermaßen bes-

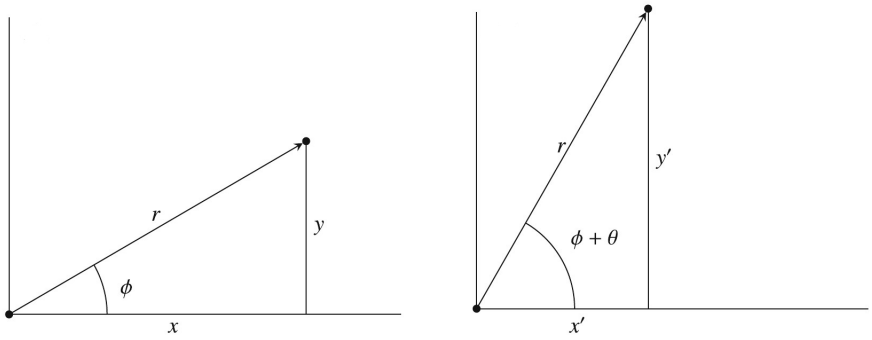


Abb. 2: Rotation eines Vektors um den Ursprung

timmt werden:

$$ {}^{a}p = {}^{a}_{b}R \cdot {}^{b}p $$

$$ bzw. \tag{4} $$

$$ \begin{bmatrix} {}^{a}x \\ {}^{a}y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} {}^{b}x \\ {}^{b}y \end{bmatrix} $$

Dabei ist ${}^{a}_{b}R$ die Rotationsmatrix, die den Punkt ${}^{b}p$ von Koordinatensystem b zu Punkt ${}^{a}p$ des Koordinatensystems a rotiert.
Eine solche Rotation wird in Abbildung **??** dargestellt. Dabei wird das Körperfeste System b in blau um den Winkel θ rotiert, während das Raumfeste System a in rot in der ursprünglichen Position von b liegt.

3.2 Translation

Neben der Rotation eines Koordinatensystem kann es auch um Δx und Δy verschoben werden. Die Berechnung der Translation von einem Punkt ${}^{b}p$ zu dem entsprechenden Punkt ${}^{a}p$ wird folgendermaßen vorgenommen:

$$ {}^{a}p = \begin{bmatrix} {}^{a}x \\ {}^{a}y \end{bmatrix} = \begin{bmatrix} {}^{b}x \\ {}^{b}y \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \tag{5} $$

In Abbildung **??** wird eine solche Translation dargestellt. Dabei wird das Körperfeste System b in rot um Δx und Δy verschoben, während das Raumfeste System a in schwarz in der ursprünglichen Position von b liegt.

3.3 Homogene Koordinaten

Nachdem in Kapitel 3.3 und 3.4 gezeigt werden konnte wie Punkte aus einem Körperfesten System nach Translation und Rotation in ein Raumfestes System transformiert werden kann, soll auch die Kombination aus beiden betrachtet werden. Dabei können beide Transformationen nacheinander angewendet werden:

$$ {}^{a1}p = \begin{bmatrix} {}^{a1}x \\ {}^{a1}y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} {}^{b}x \\ {}^{b}y \end{bmatrix} $$

$$ {}^{a}p = \begin{bmatrix} {}^{a}x \\ {}^{a}y \end{bmatrix} = \begin{bmatrix} {}^{a1}x \\ {}^{a1}y \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \tag{6} $$

In Abbildung **??** wird Translation und Rotation des Körperfesten Systems b in rot im Raumfesten System a in blau dargestellt. Dabei wird ein Hilfskoordinatensystem $a1$ in grün verwendet.
Zuerst wird die Rotation des Körperfesten System betrachtet. Der Punkt ${}^{b}x$ wird dabei auf den Punkt ${}^{a1}x$ transformiert. Das Hilfskoordinatensystem $a1$ ist nicht rotiert im Raumfesten System. Anschließend wird die Translation eingerechnet, indem der Punkt ${}^{a1}x$ auf den Punkt ${}^{a}x$ transformiert wird.

Es ist möglich dieses Zweistufige Verfahren in einen Schritt zusammenzufassen. Dabei wird eine sogenannte homogene Transformation verwendet.
Zuerst müssen die bekannte Rotationsmatrix aus Formel 4 und der Translationsvektor aus Formel 5 modifiziert werden und es ergeben sich die neuen Formeln für Rotation und Translation:

$$ \begin{bmatrix} {}^{a1}x \\ {}^{a1}y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{b}x \\ {}^{b}y \\ 1 \end{bmatrix} \tag{7} $$

$$ \begin{bmatrix} {}^{a}x \\ {}^{a}y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{a1}x \\ {}^{a1}y \\ 1 \end{bmatrix} \tag{8} $$

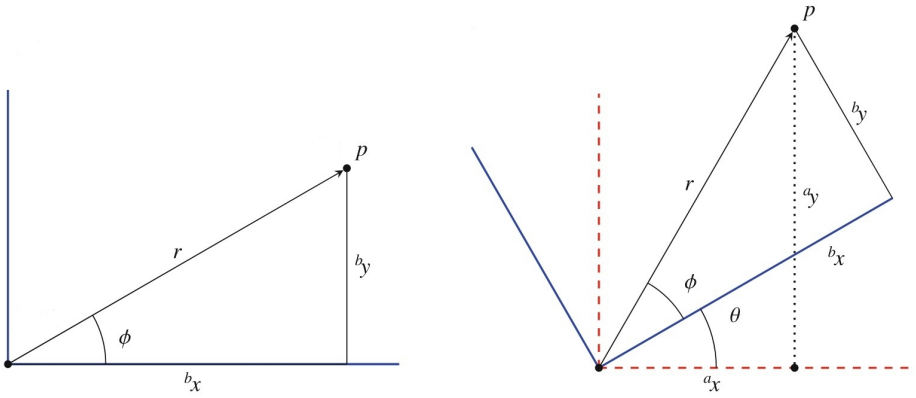


Abb. 3: Rotation des Körperfesten Koordinatensystems *b* (blau) im Raumfesten Koordinatensystem *a* (rot)

Die Rotationsmatrix wurde um eine Eins in der unteren rechten Ecke und sonst Nullen in der letzten Zeile und Spalte erweitert. Die Multiplikation mit einem Vektor, der um eine Eins erweitert wurde, ergibt immer noch die den rotierten Vektor in den ersten beiden Elementen. Der Translationsvektor wurde mit einer Matrix ersetzt. Diese ist so beschaffen, dass die Multiplikation mit einem Vektor ebenfalls immer noch den translatierten Vektor in den ersten beiden Elementen ergibt.
Diese Veränderungen mussten vorgenommen werden, damit die Rotationsmatrix und Translationsmatrix kombiniert werden können:

$$\begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & \Delta x \\ \sin\theta & \cos\theta & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \tag{9}$$

Mit der homogenen Transformation können die homogenen Koordinaten bestimmt werden, die die Koordinaten von Punkt ^ax enthalten:

$$\begin{bmatrix} ^ax \\ ^ay \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & \Delta x \\ \sin\theta & \cos\theta & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} ^bx \\ ^by \\ 1 \end{bmatrix} \tag{10}$$

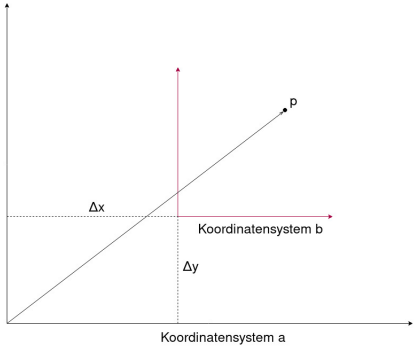


Abb. 4: Verschiebung des Körperfesten Koordinatensystems *b* (rot) im Raumfesten Koordinatensystem *a* (schwarz)

3.4 Euler-Winkel

In Kapitel 3 wurde bisher der zweidimensionale Fall betrachtet, Punkte von einem Körperfesten System in ein Raumfestes System zu transformieren. Es soll ebenfalls kurz auf den dreidimensionalen Fall eingegangen werden.
Im dreidimensionalen Fall kann statt um eine Achse um drei Achsen rotiert werden. Damit ergeben sich drei Rotationsmatrizen:

z-Achse:

$$R_{z(\Psi)} = \begin{bmatrix} \cos\Psi & -\sin\Psi & 0 \\ \sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{11}$$

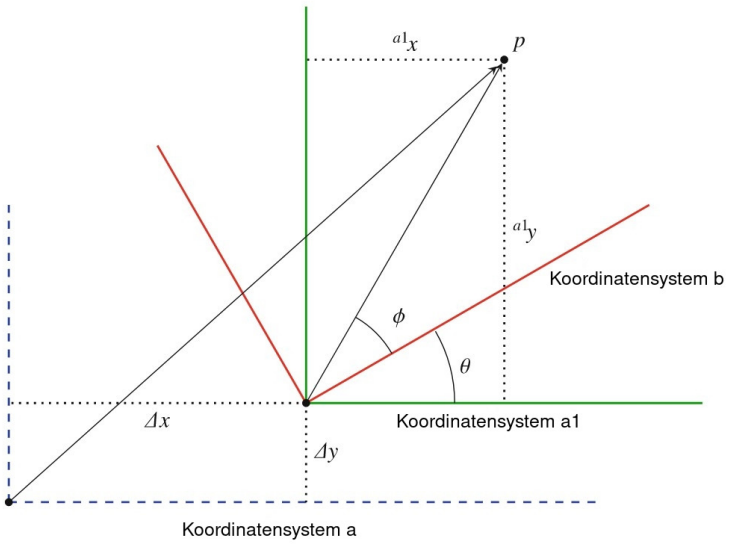


Abb. 5: Verschiebung und Rotation des Körperfesten Koordinatensystems b (rot) im Raumfesten Koordinatensystem $a1$ (blau) mit dem Hilfskoordinatensystem $a1$

y-Achse:

$$R_{y(\theta)} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{12}$$

x-Achse:

$$R_{x(\Phi)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\Phi & -\sin\Phi \\ 0 & \sin\Phi & \cos\Phi \end{bmatrix} \tag{13}$$

Jede beliebige Rotation kann durch das rotieren der x-Achse, y-Achse und z-Achse realisiert werden. Dabei genügt es jede Achse einmal zu rotieren. Somit können Rotationen aufsummiert werden, wenn sie die gleiche Achse betreffen. Die Winkel um die die Achsen rotiert werden, werden Euler-Winkel genannt. Die Gesamtrotationsmatrix ergibt sich durch die Multiplikation der Rotationsmatrizen für die einzelnen Achsen:

$$R = R_{z(\Psi)} R_{y(\theta)} R_{x(\Phi)} \tag{14}$$

In Abbildung **??** wird die aufeinanderfolgende Anwendung der Rotationsmatrizen veranschaulicht. Dabei wird aus dem Grundzustand zuerst die z-Achse, dann die y-Achse und zuletzt die x-Achse um jeweils 90° rotiert.

4 DYNAMIK IN BEWEGTEN BEZUGSYSTEMEN

Nachdem in Kapitel 3 die Transformation von Punkten aus einem Körperfesten System in ein Raumfestes System betrachtet wurden. Soll zusätzlich die Transformation von Bewegung von einem Koordinatensystem in das andere ermittelt werden. Dabei wird die Geschwindigkeit eines Ortsvektors $\vec{r}$ betrachtet, die sich aus verschiedenen Komponenten zusammensetzt. In Abbildung **??** werden diese Komponenten veranschaulicht:

- Für die Rotation eines Körperfesten Systems in einem Raumfesten System ergibt sich folgende Geschwindigkeit $v_{K_{rot}/R}$:

$$v_{K_{rot}/R} = \vec{\omega} \times \vec{r} \tag{15}$$

- Für die Bewegung eines Körperfesten Systems in einem Raumfesten System ergibt sich folgende Geschwindigkeit $v_{K/R}$:

$$v_{K/R} \tag{16}$$

- Für die Bewegung eines Ortsvektors $\vec{r}$ in seinem Körperfesten System ergibt sich folgende Geschwindigkeit $v_{r/K}$:

$$v_{r/K} : \tag{17}$$

Damit kann durch aufsummieren der Teilgeschwindigkeiten die Gesamtgeschwindigkeit $v_{r/R}$: eines Ortsvektors $\vec{r}$ in einem Raumfesten System bestimmt werden:

$$v_{r/R} = v_{K_{rot}/R} + v_{K/R} + v_{r/K} \tag{18}$$

5 MÖGLICHE REALISIERUNG

Das Fahrzeug der Projektgruppe verfügt über entsprechende Sensoren mit denen durch Koordinatentransformation ein Karte der Rennstrecke aufgezeichnet werden könnte. Dabei ist vermutlich insbesondere die sogenannte IMU relevant. Bei dieser handelt es sich um eine InvenSense MPU-9250 9-Achsen Intertial Measurement Unit, die über folgende Sensoren verfügt:

- 3-Achsen Gyroskop
- 3-Achsen Beschleunigungssensor
- 3-Achsen Erdmagnetfeldsensor

Sowohl das Gyroskop als auch der Erdmagnetfeldsensor haben das Potenzial die Rotation des Körperfesten Fahrzeugkoordinatensystems zu messen. Der Beschleunigungssensor könnte in der Lage sein unabhängig oder in Verbindung mit einem eventuellen Schrittmotor die Geschwindigkeit des Fahrzeuges zu bestimmen und damit die Translation. Somit wäre es möglich die Position und Bewegung des Fahrzeuges auf der Strecke mithilfe der zur Verfügung stehenden Sensoren zu bestimmen.

6 ZUSAMMENFASSUNG

In Kapitel 2 konnte gezeigt werden, dass ein Körperfestes System ein Koordinatensystem ist, das in einem Raumfesten System bewegt und rotiert werden kann. Im Fall der Projektgruppe handelt es sich bei dem Körperfesten System um das Koordinatensystem des Fahrzeugs und bei dem Raumfesten System um das Koordinatensystem der Rennstrecke. In Kapitel 3 wurden die Koordinaten, die in einem rotierten und translatierten Körperfesten System liegen, durch Rotation und Translation für ein umgebendes Raumfestes System ermittelt. Zudem konnte mit der homogenen Transformation Rotation und Translation in einem Operator angewendet werden. Anschließend wurde in Kapitel 4 auf die Dynamik in bewegten Bezugssystemen eingegangen. Abschließend wurde in Kapitel 5 gezeigt, dass es mit den zur Verfügung stehenden Sensoren möglich sein könnte mit Koordinatentransformationen eine Karte der Rennstrecke zu erstellen.

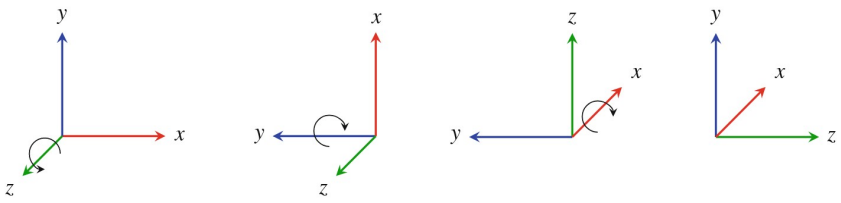


Abb. 6

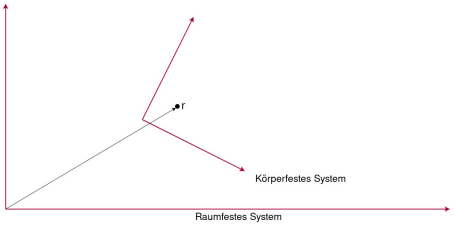


Abb. 7: Ortsvektor $\vec{r}$ in einem Körperfesten System, das sich in einem Raumfesten System befindet

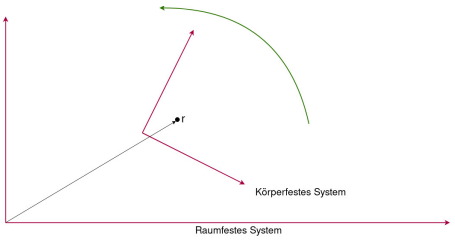


Abb. 8: Rotation des Körperfesten Systems im Raumfesten System

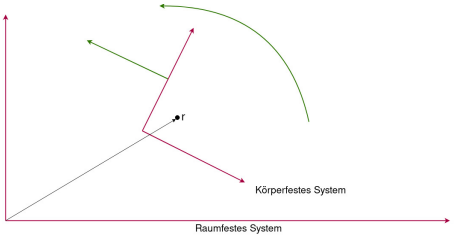


Abb. 9: Bewegung des Körperfesten Systems im Raumfesten System

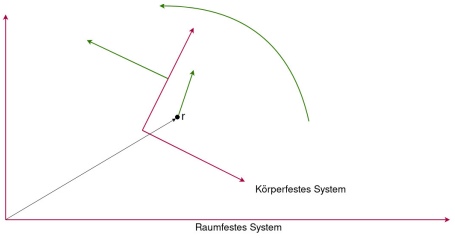


Abb. 10: Bewegung des Ortsvektors $\vec{r}$ im Körperfesten Systems

Abb. 11

C Kinematik und Dynamik des Fernlenkautos

Kinematik und Dynamik des Fernlenkautos

Wie ist die Kinematik des Fernlenkautos und welche physikalischen Kräfte spielen eine Rolle

Marcel Ebbrecht *Autonomous Racing Group*
TU Dortmund, Lehrstuhl Informatik 12
Dortmund, Deutschland
marcel.ebbrecht@googlemail.com

***Zusammenfassung*—Im Rahmen der Projektgruppe soll eine dynamische Fahrweise implementiert werden. Da die Physik dem Fahren in jeder Situation gewisse Grenzen setzt, wird in dieser Arbeit eine Teilmenge davon untersucht und mögliche Gegenmaßnahmen, die vorwiegend die Regelung, aber im Ansatz auch mechanische Kompensation beinhalten, besprochen. Der Umfang dieser Darstellung orientiert sich an der derzeit spärlichen Menge an Informationen, die zum Zeitpunkt der Erstellung vorliegen (Stand 09. Oktober 2019). Den Anfang bildet eine grundlegende Einführung, eine Betrachtung des Fahrzeugs und der Präsentation eines geeigneten Modells. Dies bildet die Grundlage für eine spezifischere Schilderung einiger Fahrsituationen und möglichen Lösungen. Am Ende werden Maßnahmen und Ideen genannt, die im Verlauf des Projektes Anwendung finden können.**

I. Grundlagen

A. Vorbemerkung

Die gesichtete Literatur [2], [5], [7], [13], [18] legt nahe, dass es sich bei der Dynamik eines Fahrzeugs, sei es auch nur in sehr grundlegenden Situationen, um eine komplexe Angelegenheit handelt. Zwar zeigt die genannte Literatur extensiv Möglichkeiten auf, das Verhalten in vielen Situationen vorherzusagen, jedoch erfordert dies ein nicht unerhebliches Maß an detaillierten Informationen über alle Komponenten des Fahrzeugs, die bei unserem Fahrzeug, grundlegend handelt es sich um ein Spielzeug, leider nicht vorliegen. Zudem drängt sich natürlich die Frage auf warum Hersteller von Kraftfahrzeugen aller Art einen nicht unerheblichen Testaufwand betreiben, wenn sich doch alles berechnen lässt. Ein Aspekt, auf den im weiteren Verlauf der Arbeit noch eingegangen wird.

Da diese Arbeit im Kern darauf abzielt ein autonomes Modellauto sicher und schnell im Rahmen von physikalischen Grenzen fahren zu lassen und nicht eine allgemeine Einführung in komplexe Sachverhalte der Mechanik zu geben, wird an vielen Stellen auf stark vereinfachte Erklärungen zurückgegriffen. Dies spiegelt sich teilweise auch in der angegebenen und verwendeten Literatur wieder.

B. Motivation

Grundsätzlich gibt es zwei wichtige Gründe, sich mit der Physik des Fahrens zu befassen:

1) **Sicherheit**: Die Physik begrenzt die Fahrweise. Das Überschreiten der Grenzen die im weiteren Verlauf dieser Arbeit erläutert werden, kann zu Schäden führen (Material, Menschen).

2) **Optimierung**: Eine gute Kenntnis eben dieser Grenzen hilft beim Ausreizen. Fragen wie „Wann geben wir wie viel Gas?“, „Wann und wo Bremsen wir (und wann nicht)?“ oder „Wie stark, wann und wo lenken wir?“ richtig zu beantworten kann enorm zu einer Steigerung der Fahrleistung beitragen. Hierbei gilt es allerdings zu berücksichtigen, dass manche Optimierungen einen enormen Rechenaufwand nach sich ziehen können - Rechenaufwand den wir unter Umständen durch Vereinfachung reduzieren können und nach derzeitigem Kenntnisstand auch müssen.

C. Kinematik und Dynamik

Die Kinematik, wie sie von Ampere [19] beschrieben wird, widmet sich der Beschreibung der Bewegung von Körpern mittels der Parameter Ort, Zeit, Geschwindigkeit und Beschleunigung. Bewegungen können unter der Verwendung von Koordinatensystemen [13], wie kartesisch, polar, oder bei komplexeren Sachverhalten, wie sie in

der Fahrzeugdynamik Anwendung finden, mittels des Dreibeins [5] beschrieben werden. Den letzten Baustein bilden Bezugssysteme (ruhend, bewegt und beschleunigt). Massen und Kräfte bleiben in der Kinematik jedoch völlig unberücksichtigt. Da die Themen Koordinaten- und Bezugssysteme bereits durch eine andere Arbeit im Rahmen der Projektgruppe abgedeckt sind und die anderen Aspekte Geschwindigkeit und Beschleunigung in der Dynamik Anwendung finden, wird nicht weiter auf das Thema Kinematik eingegangen, sie dient lediglich als Werkzeug.

Die Dynamik [18] verbindet Masse und den Einfluss von Kräften auf die Kinematik von, in unserem Fall, starren Körpern [17] und wird exemplarisch an einigen relevanten Fallbeispielen, wie Beschleunigung oder Kurvenfahrt, grundlegend behandelt. Hierzu werden wir die Themen Reibung und Kraftschluss anschneiden. Aerodynamik, Rollreibung, Schwingungsverhalten, Radstand, oder Federung bleiben außen vor, da keine hinreichenden Daten vorliegen um deren Einfluss bemessen zu können.

D. Kräfte

Für die Beschreibung von Kräften gelten die newtonschen Gesetze:

- **Trägheitsprinzip**: „Ein Körper verharrt im Zustand der Ruhe oder der gleichförmig geradlinigen Bewegung, sofern er nicht durch einwirkende Kräfte zur Änderung seines Zustands gezwungen wird.“ [12]

  $$\vec{v}(t_1) = \vec{v}(t_2) \tag{1}$$

  Die Trägheit zwingt also einen Körper, in unserem Fall unser Fahrzeug dazu seine Bewegung mit konstanter Geschwindigkeit und Richtung fortzusetzen, sofern keine weiteren Kräfte wirken.

- **Aktionsprinzip**: „Die Änderung der Bewegung ist der Einwirkung der bewegenden Kraft proportional und geschieht nach der Richtung derjenigen geraden Linie, nach welcher jene Kraft wirkt.“ [12]

  $$\dot{\vec{v}} \propto \vec{F} \tag{2}$$

  Dieses Prinzip ist vor allem beim Bremsen, Beschleunigen oder auch beim Ändern der Bewegungsrichtung maßgeblich.

- **Reaktionsprinzip**: „Kräfte treten immer paarweise auf. Übt ein Körper A auf einen anderen Körper B eine Kraft aus (actio), so wirkt eine gleich große, aber entgegen gerichtete Kraft von Körper B auf Körper A (reactio).“ [12]

  $$\vec{F}_{A\to B} = -\vec{F}_{B\to A} \tag{3}$$

  Hier werden wir sehen, dass dies eine wichtige Rolle beim Kraftschluss spielt.

- **Superpositionsprinzip**: „Wirken auf einen Punkt (oder einen starren Körper) mehrere Kräfte, so addieren sich diese vektoriell zu einer resultierenden Kraft auf.“ [12]

  $$\vec{F_1}, \vec{F_2}, \ldots, \vec{F_n} = \vec{F} \tag{4}$$

  Beispiel Kraftschluss: Alle wirkenden Kräfte werden einfach addiert, somit spielt es für die Reifen (nach dem noch vorzustellenden Modell) keine Rolle, in welche Richtung die Kräfte wirken und man kann sie einfach addieren.

II. Fahrzeug

A. Traxxas RC Rally Car

Folgende Informationen werden vom Hersteller zur Verfügung gestellt [15]:

- **Fahrzeug**: Traxxas RC rally car
- **Motor**: Velineon 3351R/3500 brushless DC
- **Reifen**: Traxxas 2.2 Pre-Glued with Foam Inserts
- **Antriebsart**: Allradantrieb
- **Differential**: Vorhanden (*Limited Slip*)
- **Bremsen**: Motorbremse

Leider sind zu den einzelnen Komponenten keine weiteren, hilfreichen Informationen zu bekommen, sodass im weiteren Verlauf entsprechende Annahmen getroffen werden müssen.

B. Masse

Die Masse des Fahrzeugs beträgt ab Werk 2770 g. Hinzu kommt die zusätzliche Hardware [4], die an Board verbaut ist. Die Masse bzw. die Kraft mit der Sie auf den Untergrund drückt wird in unserer idealisierten Betrachtung immer gekürzt und ist daher vorerst nicht wichtig. Sie wird jedoch relevant, sobald man negativen Auftrieb, oder z.B. Kraftgrenzen von Reifen berücksichtigt:

- Im ersten Fall erhöht der Anpressdruck F_D bei einer Geschwindigkeit v_t die Haftreibung, sodass höhere Beschleunigungen möglich sind. Wie genau, wird exemplarisch gezeigt.

- Der zweite Fall spielt vor allem bei hohen Gewichten und damit verbundenen Kräften eine bedeutende Rolle, da Reifen in Abhängigkeit vom verwendeten Material eine Belastungsgrenze haben und dann ausfallen (vom Verlust der Haftung bis hin zum Totalausfall). In Anbetracht des vergleichsweise geringen Gewichtes und in Ermangelung entsprechender Informationen wird davon ausgegangen, dass die Reifen den wirkenden Kräften hinreichend widerstehen können.

Ein weiterer wichtiger Punkt ist die Verteilung der Masse, also der Schwerpunkt, da diese eine enorme Auswirkung auf das dynamische Verhalten des Fahrzeugs haben kann. Da leider keinerlei Informationen hierzu vorliegen, wird der Schwerpunkt anfänglich mittig und möglichst tief angenommen. Sofern sich eine eklatante Abweichung von dieser Annahme zeigt, muss dies berücksichtigt werden. Wie genau, wird exemplarisch gezeigt.

C. Reifen

Beim Beschleunigen von Massen treten Kräfte auf. Im Falle des Fahrzeugs passiert das primär beim Beschleunigen, Bremsen (entspricht negativer Beschleunigung) und beim Ändern der Bewegungsrichtung. Diese Kräfte müssen aufgenommen und dadurch auf den Untergrund übertragen werden. Gemäß dem Reaktionsprinzip wirkt dann eine dieser im Betrag entsprechende Gegenkraft. Im Fall von Fahrzeugen ist das die Reibungskraft, die durch die Reifen aufgebracht wird [2] (Abbildung 1).

Eine korrekte Vorhersage der Reibeigenschaften gestaltet sich schwierig, da hierfür detaillierte Informationen zum verwendeten Material, Profil, Druck und vieles mehr vorliegen müssen. Daher bietet es sich an, Durchschnittswerte anzunehmen: Wir gehen von einem Reibbeiwert $\mu_h = 0.85$ für Haft- und $\mu_g = 0.50$ für Gleitreibung aus. Dies stellt einen Mittelwert zwischen reinem Gummi und Autoreifen auf trockenem Beton dar [16].

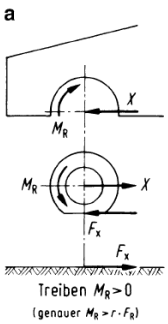


Abbildung 1: Kraftschluss Reifen [8]

Im weiteren Verlauf werden wir uns nur auf die Haftreibung beziehen, da nach Eintritt in die Gleitphase Korrekturen aufgrund mangelnder Einzelradregelung nur schwer möglich sind und somit vermieden werden sollte.

D. Motor und Antrieb

Das Fahrzeug verfügt über den oben genannten Elektromotor der gleichzeitig zum Bremsen eingesetzt wird. Da leider kein Datenblatt seitens des Herstellers zur Verfügung steht gestaltet sich eine Vorhersage über mögliche Antriebsmomente schwierig, selbst wenn zusätzlich Informationen über das Getriebe zur Verfügung ständen. Hier bietet es sich an entsprechende Messungen vorzunehmen.

Das Fahrzeug wird an allen 4 Rädern angetrieben und gebremst. Dies hat Vor- aber auch Nachteile:

- **Vorteil**: Wir können davon ausgehen, dass an allen 4 Rädern die gleichen Kräfte auftreten, wodurch wir unser Modell einfach halten können.

- **Nachteil**: Dieser Zugewinn an Haftreibung ist allerdings mit einem Verlust an Stabilität verbunden (Abbildung 2), der sich vor allem bei Kurvenfahrten bemerkbar macht. Hierdurch neigt das Fahrzeug eher zum Stabilitätsverlust worauf später eingegangen wird [9].

E. Differential

Das Differential übernimmt in der Regel die korrekte Verteilung der Antriebskraft auf die getriebenen Achsen. Auch wenn, wie bereits

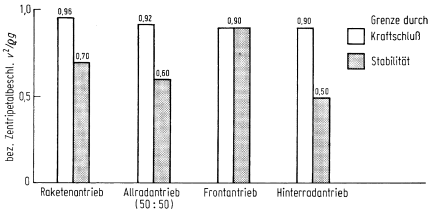


Abbildung 2: Vergleich Kraftschluss und Stabilität [9]

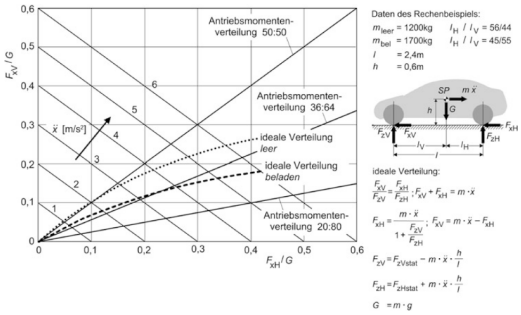


Abbildung 3: Differentialverteilung [9]

dargestellt der Schwerpunkt im Ruhezustand mittig des Fahrzeugs angenommen wird, kommt es bei Beschleunigungen, seien sie in, gegen die Fahrrichtung oder seitlich in Kurven, zu einer Verlagerung des Schwerpunktes. Da sich in diesen Momenten auch die auf die Räder wirkenden Kräfte verlagern muss ein entsprechender Ausgleich vorgenommen werden. Dies geschieht in der Regel durch ein Differentialgetriebe [10]. Ein einfaches Beispiel zeigt die Abbildung 3: Hier wird die optimale Momentverteilung im Falle einer Beschleunigung beschrieben, bei der sich das Moment stärker und in Abhängigkeit der Beschleunigung auf die Hinterachse verteilt, da durch die Trägheit der Schwerpunkt entsprechend nach hinten verlagert wird. Bei einer Bremsung verlagert sich der Schwerpunkt entsprechend nach vorne, sodass dann eine umgekehrte Verteilung erfolgen muss. Leider werden vom Hersteller keine weiteren Angaben zur Leistung des Getriebes gemacht und wir müssen davon ausgehen, dass diese Maßnahmen korrekt umgesetzt werden, da wir auch keine Möglichkeit haben, die Kraft per Regelung auf die Achsen zu verteilen.

Ein weiteres Problem stellen Kurvenfahrten dar: Wie in Abbildung 4 ersichtlich, folgen die innere und die äußere Spur verschiedenen Kurvenradien und müssen dementsprechend unterschiedlich lange Wege zurücklegen. Zusätzlich kommt es, wie wir später noch sehen werden, durch die Trägheit zu einer weiteren Momentverteilung. Die Kompensation ist wieder Aufgabe des Differentialgetriebes, genauer eines inkludierten sogenannten Ausgleichsgetriebes [6]. Auch müssen wir wieder auf eine korrekte Konstruktion seitens des Hersteller vertrauen, da uns, in Ermangelung einer Einzelradbremsung keine manuelle Korrekturmöglichkeit zur Verfügung steht.

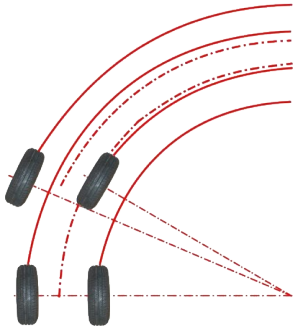


Abbildung 4: Verschiedene Weglänge und Radien in Kurven [6]

Sofern diese Probleme nicht durch das Getriebe ausgeglichen werden, ist mit einem erhöhten Hang zum Stabilitätsverlust zu rechnen, der im späteren Verlauf beschrieben wird.

F. Modell

Um einen einfachen Einstieg gewährleisten zu können, den Rechenaufwand im Fahrzeug gering zu halten und oben genannte Einschränkungen zu reflektieren, wird vorerst ein möglichst einfaches Modell verwendet (Abbildung 5).

Abbildung 5: Unser physikalisches Modell

Da, wie bereits dargestellt, die Reifen den wichtigsten Faktor für die Stabilität darstellen,

wird sich diese Arbeit im weiteren Verlauf mit dem Thema Kraftschluss und dessen Abhängigkeit von den Eigenschaften der Reifen befassen. Themen wie Rollreibung, Schwingungsverhalten, Radstand, Federung oder andere, fortgeschrittenere Betrachtungen bleiben im weiteren Verlauf unberücksichtigt. Lediglich die Aerodynamik wird an einem Punkt kurz angeschnitten.

Der Kontakt der Reifen wird punktförmig angenommen, wodurch Hebelwirkungen eliminiert werden. In diesem Fall macht es keinerlei Unterschied, ich welche Richtung die Kräfte wirken (Superpositionsprinzip). Dies setzt natürlich, bedingt durch die unzureichende Informationslage, die Annahme voraus, dass der Reifen in jeder Situation der gegebenen Belastung standhalten.

Ferner wird vorausgesetzt, dass das Fahrwerk durch entsprechend ausgelegte Federung dafür sorgt, dass jederzeit alle 4 Reifen den Kontakt zum Untergrund halten: Sollte eine Spur den Kontakt verlieren, greifen die Regeln zur Dynamik von einspurigen Fahrzeugen, die in vielen Fällen, vor allem in den noch vorzustellenden Beispielen, grundsätzlich anders funktionieren. Aus diesem Grund wird auch ein mögliches Kippen des Fahrzeugs nicht näher betrachtet.

III. Kraftschluss

Die Reifen stellen den Kontakt zum Untergrund her und müssen jederzeit die Kräfte übertragen, die auf das Fahrzeug wirken bzw. auf den Untergrund übertragen werden sollen (Reaktionsprinzip). Dies geschieht in Form von Reibungskraft. In der Fahrzeugdynamik gibt es zwei Arten, oder besser Phasen dieser Reibung [2]:

- **Haftreibung**: Sei F^B die auftretende Kraft durch Beschleunigung, F_R^h die maximale Haftreibungskraft, die die die Reifen aufnehmen können, $G = m \cdot g$ die Gewichtskraft durch die Fahrzeugmasse und μ_h der Haftreibbeiwert, es gilt: Solange die wirkende Kraft $F_B \leq F_R^h = \mu_h \cdot G = \mu_h \cdot m \cdot g$ ist, haften die Reifen am Boden und es erfolgt keine Relativbewegung zwischen Reifenoberfläche und Untergrund. Mit anderen Worten: Das Fahrzeug fährt kontrolliert und es wird die Kraft F_R^h übertragen.
- **Gleitreibung**: Sei F^B die auftretende Kraft durch Beschleunigung, F_R^g die maximale Gleitreibungskraft, die die die Reifen aufnehmen können, $G = m \cdot g$ die Gewichtskraft durch die Fahrzeugmasse und μ_g der Gleitreibbeiwert, es gilt: Sofern F_R^h überschritten wurde, tritt ein Gleiten, also eine Relativbewegung zwischen Reifen und Untergrund ein (Rutschen / Durchdrehen / Blockieren).

Um wieder den haftenden Zustand zu erreichen, muss die Gesamtbeschleunigung wieder unter F_R^g reduziert werden, wobei berücksichtigt werden muss, dass nicht nur F_B entsprechend reduziert werden, sondern nun auch die kinetische Energie, die das Fahrzeug durch den Rutschvorgang erhält, zusätzlich kompensiert werden muss. Daher sollte ein Rutschen unter allen Umständen vermieden oder nur kontrolliert (Driften) eingesetzt werden.

Im folgenden werden ein paar mögliche Szenarien anhand von Beispielberechnungen erläutert.

A. Beschleunigung

Betrachten wir nun die Kraftgrenze für die Haftreibung bei einer Beschleunigung des Fahrzeugs auf gerader Strecke. Gegeben $m = 3500\,\mathrm{g}$, $\mu_h = 0.85$, es gilt:

$$\begin{aligned} F_h^{max} &\leq F_R^h = \mu_h \cdot G \\ F_h^{max} &\leq F_R^h = \mu_h \cdot m \cdot g \\ m \cdot a^{max} &\leq \mu_h \cdot m \cdot g \\ a_h^{max} &\leq \mu_h \cdot g \approx 8.33\mathrm{m/s^2} \end{aligned}$$

Bei höherer Beschleunigung tritt Gleiten ein, sodass die die Kraftübertragung und somit die Beschleunigung auf $\frac{8.33\mathrm{m/s^2} \cdot 0.5}{0.85} \approx 4.9\mathrm{m/s^2}$ reduziert wird. Als Gegenmaßnahme muss der Motor schlagartig so weit gedrosselt werden, dass wieder Haftung eintritt, anschließend kann die Beschleunigung wieder bis maximal a_h^{max} erhöht werden (Antischlupfregelung) [7].

Wie man sehen kann, wird die Masse auf beiden Seiten der Gleichungen eliminiert, sodass sie hier keine Rolle spielt.

B. Einfluss der Aerodynamik

Wie bereits erwähnt, spielt die Masse bei dieser einfachen Betrachtung keine Rolle. Bezieht man

jedoch die Aerodynamik mit ein, ergibt sich ein anderes Bild. Im folgenden Beispiel wird davon ausgegangen, dass das Strömungsverhalten einen negativen Auftrieb erzeugt. Somit wird der Druck, mit dem das Fahrzeug und damit die Reifen auf die Straße wirken erhöht (Anmerkung: Es kann auch ein gegenteiliger Effekt eintreten, sodass tatsächlich Auftrieb entsteht, wie es bei den ersten Modellen des VW Käfers der Fall war (ohne Beleg)).

Wir nehmen also an, dass das Fahrzeug bei einer gegebenen Geschwindigkeit $v_t = 15\,\mathrm{m/s}$ einen Abtrieb entwickelt, der 10% der Ruhemasse entspricht, somit $F_A(v_t) = F_A(15\,\mathrm{m/s}) = 0.1m \cdot g$ und wir beschleunigen bei 15 m/s (Werte frei erfunden). Die durch den Abtrieb erzeugt Kraft wird nur auf der rechten Seite addiert, sodass die scheinbare Masse größer wird. Es gilt:

$$\begin{aligned} F_h^{max} \leq F_R^h &= \mu_h \cdot (G + F_A(v_t)) \\ F_h^{max} \leq F_R^h &= \mu_h \cdot 1.1m \cdot g \\ m \cdot a^{max} &\leq 1.1 \cdot \mu_h \cdot m \cdot g \\ a_h^{max} &\leq 1.1 \cdot \mu_h \cdot g \approx 9.16\mathrm{m/s}^2 \end{aligned}$$

Wie man sehen kann, geht der Abtrieb linear in die Kraftschlussgrenze ein. Allerdings muss dabei berücksichtigt werden, dass es sich um eine Funktion der aktuell gefahrenen Geschwindigkeit v_t handelt und es dementsprechend implementiert werden muss. Zudem stellt sich, in Anbetracht der Tatsache, dass uns keinerlei brauchbare Informationen zu den aerodynamischen Eigenschaften des Fahrzeugs vorliegen, die Frage, wie wir das sicher in Berechnungen umsetzen können. Grundsätzlich ist dies aber über Umfangreiche Messreihen und einer Approximation einer Funktion anhand der gemessenen Werte möglich.

Eine weitere Möglichkeit stellt natürlich auch die Implementierung per maschinellem Lernen dar: Das Fahrzeug lernt einfach von selbst, durch Erfassung der Motorwerte und dem Vergleich mit den Sensordaten, wann Instabilitäten auftreten und merkt sich diese einfach. Im Vergleich zum manuellen Messverfahren, wie zuvor angedeutet, stellt dies vermutlich den geringeren Aufwand bei höherem Nutzen dar, da auch unbekannte Situationen erlernt werden können. Ob dies dann durch eine dynamische Anpassung oder durch Einfügen weiterer Werte erfolgt, soll an dieser Stelle offen bleiben.

C. Bremsen

Da es sich beim Bremsen lediglich um eine negative Beschleunigung handelt, gelten die selben Regeln und Grenzen wie sie bereits genannt wurden.

Als Gegenmaßnahme muss die Bremskraft schlagartig so weit gedrosselt werden, dass wieder Haftung eintritt, anschließend kann die Beschleunigung wieder bis maximal a_h^{max} erhöht werden (Antiblockiersystem, kurz ABS) [7].

D. Kurvenfahrt

Ein weiteres wichtiges Szenario stellt das Fahren von Kurven dar. Hierbei ist es unerheblich, ob mehrere Kurven hintereinander gefahren werden (zum Beispiel beim Ausweichen oder Überholen) sofern wir das Fahrzeug als starren Körper betrachten und somit Schwingungen unberücksichtigt lassen. Jede Richtungsänderung kann als einzelner Prozess betrachtet werden.

Bei einer kreisförmigen Bewegung tritt, bedingt durch die Trägheit der Masse, eine Scheinkraft, die Zentrifugalkraft auf [3]. Nach dem Reaktionsprinzip steht dieser Kraft eine weitere gegenüber: Die Zentripetalkraft [12] die in unserem Fall abermals durch den Kraftschluss an den Reifen aufgebracht werden muss. Als maßgeblich gilt hier die Radialbeschleunigung:

$$a = \frac{v^2}{r} \tag{5}$$

Auch hier gilt wieder, dass solange die wirkende Kraft (Zentrifugalkraft) kleiner als die entgegengesetzte Kraft (Reibungskraft) ist, tritt keine Relativbewegung zwischen Reifen und Untergrund ein. Es lässt sich unschwer erkennen, dass mit steigendem Radius auch die mögliche Beschleunigung und damit die fahrbare Geschwindigkeit steigt. Gegeben $m = 3500\,\mathrm{g}$, $\mu_h = 0.85$ und $r = 2\,\mathrm{m}$ es gilt:

$$\begin{aligned} F = m \cdot a \leq F_R^h &= \mu_h \cdot m \cdot g \\ \Leftrightarrow m \cdot \frac{{v_{max}}^2}{r} &\leq \mu_h \cdot m \cdot g \\ \Leftrightarrow \frac{{v_{max}}^2}{r} &\leq \mu_h \cdot g \\ \Leftrightarrow v_{max} &\leq \sqrt{\mu_h \cdot g \cdot r} \\ \Leftrightarrow v_{max} &\leq \sqrt{0.85 \cdot 9.81\mathrm{m/s}^2 \cdot 2\,\mathrm{m}} \\ &\approx 4\,\mathrm{m/s} \end{aligned}$$

Die maximale Kurvengeschwindigkeit beträgt also unter den genannten Annahmen circa 4 m/s. Auch hier kann natürlich wieder ein aerodynamisch bedingter Anpressdruck einbezogen werden:

$$\begin{aligned} F = m \cdot a \leq F_R^h = \mu_h \cdot (G + F_A(v)) \\ \Leftrightarrow m \cdot \frac{{v_{max}}^2}{r} \leq \mu_h \cdot 1.1m \cdot g \\ \Leftrightarrow \frac{{v_{max}}^2}{r} \leq 1.1 \cdot \mu_h \cdot g \\ \Leftrightarrow v_{max} \leq \sqrt{1.1 \cdot \mu_h \cdot g \cdot r} \\ \Leftrightarrow v_{max} \leq \sqrt{1.1 \cdot 0.85 \cdot 9.81\text{m/s}^2 \cdot 2\,\text{m}} \\ \approx 4.2\,\text{m/s} \end{aligned}$$

Natürlich geht hier die Wirkung nur mit der Wurzel der zusätzlichen scheinbaren Masse ein und es gelten dieselben Anforderungen wie bei Beschleunigungen auf gerader Strecke.

Auch hier tritt wieder beim Überschreiten der Kraftgrenze ein Gleiten ein. Die Kompensation ist allerdings, wie noch erläutert wird, deutlich anspruchsvoller als auf gerader Strecke, da jede Korrektur nach zuvor genannten Möglichkeiten eine zusätzliche Beschleunigung darstellt und die Instabilität unter Umständen noch verschlimmert. Daher empfiehlt es sich dringend, diese Notwendigkeit gar nicht herbeizuführen, den Kurvenradius und Verlauf so gut und früh wie möglich zu bestimmen und die Kurve entsprechen anzufahren. Dabei sollte eine geringe Reserve für mögliche Radius-Verkleinerung gehalten werden. Wie hoch diese Ausfallen muss, müssen Tests zeigen.

E. Stabilitätsverlust in Kurven

In der Realität verliert ein Fahrzeug nie an beiden Achsen gleichzeitig die Haftung und kommt ins Rutschen. Sollte das der Fall sein, würde es einfach seitlich aus der Kurve treiben und die Korrektur wäre relativ einfach. Leider tritt der Effekt des Haftungsverlustes immer an einer der beiden Achsen des Fahrzeugs auf. Dies ist sowohl durch die Fahrsituation, zum anderen aber auch durch die Gewichtsverteilung und Antriebsart begründet. Während Fahrzeuge mit Frontantrieb eher zum Untersteuern neigen, kommt es bei Fahrzeugen mit Heckantrieb verstärkt zum Übersteuern. Ein Allradantrieb kombiniert diese beiden Nachteile in der Regel [14]. Im weiteren werden wir die Betrachtung auf diese Antriebsart beschränken.

F. Untersteuern

Beim Untersteuern kommt es zu einem Verlust der Haftreibung an der Vorderachse [11] und der tatsächlich gefahrene Kurvenradius wird größer als der beabsichtigte - man schiebt quasi über die Vorderreifen.

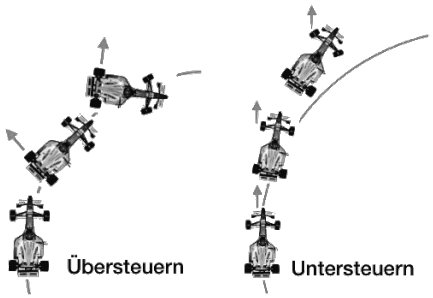


Abbildung 6: Unter- und Übersteuern [14]

Da in diesem Fall ein Drehmoment im Gegenuhrzeigersinn auftritt, wird dies in modernen Fahrzeugen durch ein elektronisches Stabilitätsprogramm, kurz ESP, mittels einer Einzelradbremsung am hinteren, inneren Rad korrigiert, sodass ein Moment im Uhrzeigersinn das Fahrzeug wieder in die angedachte Spur bringt.

Leider steht dieser Mechanismus nicht zur Verfügung, da bedingt durch die Motorbremse in Verbindung mit dem Allradantrieb nur alle Räder gleichzeitig gebremst oder beschleunigt werden können. Es gibt allerdings auch andere, wenn auch weniger effiziente Möglichkeiten:

- Die konservativste Möglichkeit stellt ein einfaches Rollen dar, also die Wegnahme jeglicher Beschleunigung [1]. Durch den Winkel zwischen Reifenspur und Fahrrichtung tritt automatisch ein Bremseffekt ein, sodass eine Stabilisierung eintreten kann. Sollte dies nicht ausreichen, kann zusätzlich leicht gebremst werden.

- Die weitaus riskantere, aber effektivere Möglichkeit besteht darin, den Lenkwinkel weiter zu erhöhen, wodurch zum einen die Lenkkraft in die angedachte Fahrtrichtung, als auch die Bremswirkung durch die Querstellung der Reifen weiter erhöht wird.

Die letztgenannte Variante birgt allerdings die Gefahr, dass es bei plötzlich erlangter Haftreibung zu extremen Kräften kommen kann, die direkt in ein Übersteuern führen und damit ein Überschlagen des Fahrzeugs auslösen können.

G. Übersteuern

Beim Übersteuern kommt es zu einem Verlust der Haftreibung an der Hinterachse [11] - die Hinterräder überholen das Fahrzeug und es beginnt sich zu drehen.

Da in diesem Fall ein Drehmoment im Uhrzeigersinn auftritt, wird dies in modernen Fahrzeugen durch ein elektronisches Stabilitätsprogramm, kurz ESP, mittels einer Einzelradbremsung am vorderen, äußeren Rad korrigiert, sodass ein Moment im Gegenuhrzeigersinn das Fahrzeug wieder in die angedachte Spur bringt.

Auch hier stehen nur folgende Möglichkeiten zur Verfügung

- Die konservativste Möglichkeit stellt wieder ein einfaches Rollen dar, also die Wegnahme jeglicher Beschleunigung [1]. Durch den Winkel zwischen Reifenspur und Fahrrichtung tritt automatisch ein Bremseffekt ein, sodass eine Stabilisierung eintreten kann. Sollte dies nicht ausreichen, kann zusätzlich leicht gebremst werden.

- Die abermals riskantere, aber effektivere Möglichkeit besteht in einem vorsichtigen Gegenlenken um ein entsprechendes Drehmoment zu erzeugen (Drift).

In beiden Fällen sind die Korrekturmöglichkeiten extrem begrenzt, sodass diese Situationen, wie bereits mehrfach erwähnt, so gut es geht vermieden werden sollten. Sollte das Fahrzeug verstärkt zu diesen Instabilitäten neigen, besteht allerdings auch die Möglichkeit mittels mechanischem Tuning, wie Anpassung der Federung, der Verwendung anderer Reifen oder Fahrwerkskorrekturen, eine eklatante Verbesserung herbeizuführen. Hierbei sollte allerdings darauf geachtet werden, dass es durch diese Modifikationen nicht zu Problemen mit dem Regelwerk des F1Tenth-Wettbewerbes kommt.

IV. Agenda

Trotz der relativ bescheidenen Möglichkeiten sollte es möglich sein, ein sicheres Fahren, auch in der Nähe von physikalischen Grenzen zu ermöglichen. Dabei sollten die Modelle und Maßnahmen vorerst so einfach wie möglich gehalten werden. Im Laufe des Projektes sollten folgende Punkte beachtet werden:

- **Annahmen überprüfen**: Inwieweit sind die Annahmen zum Schwerpunkt, der Grenzbelastung der Reifen und der Fahrzeugstatik korrekt. Dies sollte mittels einfacher Messungen realisierbar sein.

- **Messen und Experimentieren**: Durch kontrolliertes Beschleunigen, Bremsen und Kurvenfahrten auf verschiedenen Untergründen sollten die Werte für Beschleunigung, Bremskraft und Reibbeiwerte ermittelt werden. Hier bietet es sich an, die Sensoren für die Messung von Stabilitätsverlusten einzusetzen und aus den gesammelten Daten entsprechende Werte und Funktionen abzuleiten.

- **Kurvenradien**: Das Fahrzeug sollte nach Möglichkeit die Kurve kennen, die es als nächstes befahren muss, sodass diese Fahrt sauber berechnet und damit geplant werden kann.

- **Simulator**: Die Erkenntnisse aus den Messungen und die dargestellten Modelle sollten in den Simulator einfließen. Niemandem ist geholfen, wenn man zwar eine Ideallinie durch eine Kurve berechnen, diese dann aber in der Realität aufgrund von Kraftgrenzen gar nicht fahren kann.

- **Machine Learning**: Wie bereits angedacht sollte überprüft werden, inwiefern maschinelles Lernen für die Ermittlung der Kraftgrenzen zum Einsatz kommen kann. Grundsätzlich ist es dem Fahrzeug möglich einen Stabilitätsverlust selber zu erkennen. Dieses Lernen muss auch nicht im Rennbetrieb erfolgen, sondern könnte im Vorfeld auf Daten, die durch Testfahrten ermittelt werden, geschehen. Dies hätte zudem den Vorteil, dass im Rennbetrieb deutlich weniger Rechenzeit für die Fahrphysik aufgebracht werden muss.

V. Schlussbemerkung

Dieser Ausarbeitung wird vor der Verwendung im Abschlussbericht überarbeitet, da bis dahin mehr Informationen zum und Erfahrungen mit dem Auto genauere Prognosen und eine bessere Methodik zulassen.

Literatur

[1] Allsecur Versicherung Schleuderratgeber. https://www.allsecur.de/kfz-versicherung/auto-schleudert-ratgeber/. Abgerufen: 09.10.2019.

[2] Demtröder, Wolfgang: Experimentalphysik, Seite 130ff. Springer, 2015. 1. Auflage.

[3] Descartes, René: Die Prinzipien der Philosophie, übersetzt von Artur Buchenau, Seite 86ff. Felix Meiner Verlag, 1965. 7. Auflage.

[4] F1Tenth Build Manual. http://f1tenth.org/build.html. Abgerufen: 09.10.2019.

[5] Fliessbach, Torsten: Mechanik - Lehrbuch zur Theoretischen Physik I, Seiten 2–8. Springer, 2015. 7. Auflage.

[6] KFZ-Tech Ausgleichsgetriebe. https://www.kfz-tech.de/Biblio/Achsantrieb/Ausgleichsgetriebe.htm. Abgerufen: 09.10.2019.

[7] Mitschke, Manfred und Henning Wallentowitz: Dynamik der Kraftfahrzeuge, Seite 211ff. Springer, 2015. 1. Auflage.

[8] Mitschke, Manfred und Henning Wallentowitz: Dynamik der Kraftfahrzeuge, Seite 20. Springer, 2015. 1. Auflage.

[9] Mitschke, Manfred und Henning Wallentowitz: Dynamik der Kraftfahrzeuge, Seite 795. Springer, 2015. 1. Auflage.

[10] Mitschke, Manfred und Henning Wallentowitz: Dynamik der Kraftfahrzeuge, Seite 204. Springer, 2015. 1. Auflage.

[11] Mitschke, Manfred und Henning Wallentowitz: Dynamik der Kraftfahrzeuge, Seite 873. Springer, 2015. 1. Auflage.

[12] Newton, Isaac: Philosophiae naturalis principia mathematica, Seiten 2–5. Benjamin Motte, 1687. 1. Auflage.

[13] Nolting, Wolfgang: Grundkurs Theoretische Mechanik 1 - Klassische Mechanik, Seite 163. Springer, 2013. 10. Auflage.

[14] Racingpals Tuning. https://racingpals.wordpress.com/2016/05/06/ubersteuern-oder-untersteuern-tuning-tipps-zum-setups/. Abgerufen: 09.10.2019.

[15] Rally ST Specifications. https://traxxas.com/products/models/electric/ford-fiesta-st-rally?t=specs. Abgerufen: 09.10.2019.

[16] Reibbeiwerte. https://www.schweizer-fn.de/stoff/reibwerte/reibwerte.php. Abgerufen: 09.10.2019.

[17] Sayir, Mahir und Stephan Kaufmann: Ingenieurmechanik 3 - Dynamik, Seite 9. Springer, 2015. 2. Auflage.

[18] Wikipedia: Dynamik. https://de.wikipedia.org/wiki/Dynamik. Abgerufen: 09.10.2019.

[19] Wikipedia: Kinematik. https://de.wikipedia.org/wiki/Kinematik. Abgerufen: 09.10.2019.

D Sensorik und Aktorik des Fernlenkautos

Sensorik und Aktorik des Fernlenkautos

Jan-Henrik Bruhn
Autonomous Racing Group
TU Dortmund, Lehrstuhl Informatik 12
Dortmund, Deutschland
hi@jhbruhn.de

***Zusammenfassung*—Im Rahmen dieser Ausarbeitung sollen die Sensoren und Aktoren des in der Projektgruppe *Autonomous Racing* verwendeten Fernlenkautos beschrieben werden.**

Dabei werden sowohl die verschiedenen Eingaben (IMU, LiDAR, Stereo Kamera), die Recheneinheit und das Fahrgestell inklusive Antrieb und Lenkung betrachtet. Die Ausarbeitung soll dem Leser dabei einen Einblick in die verwendeten bzw. vorgegebenen Sensoren und Aktoren geben und somit die Verwendung des Fahrzeuges und der Softwareoptimierung dienen.

I. Überblick

Das in dieser Projektgruppe verwendete, von der vorherigen Projektgruppe gebaute Auto basiert auf den Vorgaben der *F1/10-Challenge* [1], einer öpen-source-basierten, erschwinglichen, und hoch-performanten Testplattform für autonome Fahrzeuge im 1:10 Format". [1] Die F1/10-Plattform bietet Studenten und Forschungsgruppen verschiedener Universitäten weltweit die Möglichkeit, auf einer gemeinsam gleichen Plattform die bestmögliche Software zu entwickeln und sich in Wettrennen zu messen. Dabei lassen die verschiedenen Gruppen ihre autonom fahrenden Autos auf der gleichen Strecke fahren und vergleichen verschiedene Kriterien, wie zum Beispiel Rundenzeit oder den Fahrstil.[1]

Besagte Plattform ist ausgestattet mit verschiedenen Sensoren, einer performanten CPU/GPU und einer Motorsteuerung.

Grundlegend basiert das Fahrzeug auf einem 1:10 Modell eines Ford Fiesta ST Rally, hergestellt von *Traxxas* unter der Produktbezeichnung 74054-4[1] (siehe Abbildung 1) [2]. Dieses Fernlenkauto wird jedoch umgebaut, um die Steuerung basierend auf verschiedenen Sensoren zu ermöglichen: Die mitgelieferte Motorsteuerung wird ersetzt durch eine FOCBOX, ein auf dem VESC-Projekt basierender Motor-ESC (electronic speed control). Zudem wird dem Auto zum Messen der Umwelt eine Reihe

[1]Der Name geht locker über die Zunge.

Abbildung 1. Das unmodifizierte Fernlenkauto. [2]

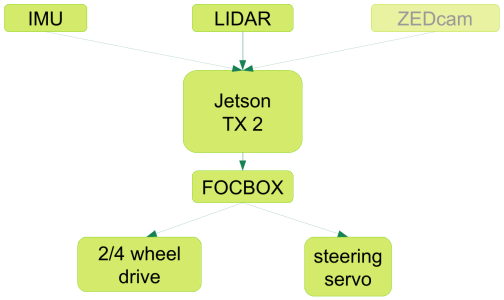


Abbildung 2. Die Sensoren und Aktoren im Überblick.

an Sensoren angebaut: einer IMU (Inertial Measurement Unit), eine LiDAR (Light Detection and Ranging) und einer ZED Cam (Stereoskopische Kamera).

Diese Sensoren werden angeschlossen an ein Nvidia Jetson TX2 Board, welches über ein Orbitty Carrier Board verschiedene Schnittstellen bietet. Dieses Jetson-Board bietet sowohl mehrere ARM-Kerne als auch eine vielzahl an CUDA-Einheiten zur Berechnung aufwendiger, paralleler Rechenprozesse.

Das Jetson-Board gibt schließlich Steuersignale an eine FOCBOX ESC weiter, welche sowohl den Lenkeinschlag über einen Servo, als auch den Antriebsmotor steuert.

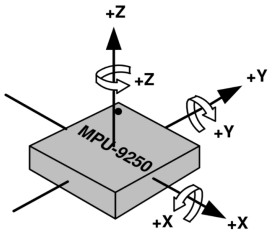


Abbildung 3. Die Achsen für Bewegung und Rotation der IMU. [3]

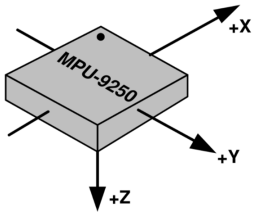


Abbildung 4. Die Achsen für die Magnetfeldmessung der IMU. [3]

II. Sensoren

A. IMU

Bei der eingebauten IMU handelt es sich um eine *InvenSense MPU-9250* 9-Achsen Intertial Measurement Unit. Diese IMU beinhaltet sowohl ein 3-Achsen Gyroskop, einen 3-Achsen Beschleunigungssensor und einen 3-Achsen Erdmagnetfeldsensor. [3]

Ausgestattet mit diesen Sensoren werden sowohl Drehung, Beschleunigung und das Erdmagnetfeld um die X-, Y- und Z-Achse gemessen (siehe Abbildung 3 und Abbildung 4).

Die Messauflösung beträgt für alle 9 Achsen 16 bit, die Skalierungen können in verschiedenen Stufen eingestellt werden.

Für das Gyroskop kann die Skala zwischen $\pm 250 \frac{\circ}{s}$ bis $\pm 2000 \frac{\circ}{s}$ betragen. Der Beschleunigungssensor kann auf Skalen zwischen $\pm 2g$ bis $\pm 16g$ konfiguriert werden. Der Erdmagnetfeldsensor misst mit einer Genauigkeit von $15 \frac{\mu T}{LSB}$. [3]

Auf der IMU ist auch eine Prozessoreinheit integriert, die die Daten der drei Sensoren fusioniert und einen einfach nutzbaren Wert für unsere Anwendungen bereitstellt.

B. LiDAR

Um die Umgebung zu erkennen wird dem Fahrzeug ein LiDAR-Sensor angebaut. LiDAR steht dabei für *light detection and ranging*.

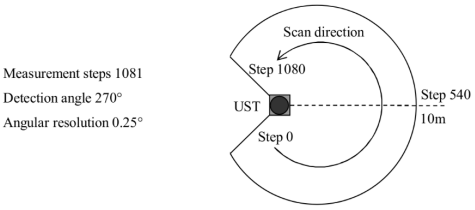


Abbildung 5. Der Messradius des Hokuyo LiDAR-Sensors. [5]

Abbildung 6. Die ZED Cam. [7]

Die Funktionsweise eines solchen Sensors ist wie folgt: Auf einem sich drehenden Körper sind sowohl ein Fotoemitter als auch -detektor angebaut. Der Emitter sendet einen Lichtstrahl aus. Nun wird die Zeit gemessen, die der ausgesendete Lichtstrahl braucht, um von einem Objekt reflektiert, und der reflektierte Lichtstrahl vom Fotodetektor gemessen wird. Anhand der Lichtgeschwindigkeit und einer Reflexivitätskonstante lässt sich somit der Abstand vom Objekt ermitteln. Dann wird sich drehende Körper weitergedreht und die Messung beginnt von vorn (vgl. Abbildung 5). [4]

Der an diesem Auto montierte LiDAR Sensor ist der *Hokuyo UST-10LX*. Er verfügt über ein Sichtfeld von $270°$, in Schritten von $0.25°$. Laut Datenblatt misst er Abstände mit einer Genauigkeit von $\pm 40mm$ und einer Frequenz von $40Hz$. Der Detektionsraum ist mit $0.06m$ bis $10m$ $(4m)$ beschrieben. Die maximale Reichweite hängt dabei von den Lichtbedingungen und den zu messenden Objekten ab. [5]

Die Kommunikation mit dem Sensor findet dabei über einen Ethernet-Port und TCP/IP statt. ROS bzw der Hersteller bieten bereits Treiber an, um die mit dem LiDAR aufgezeichneten Daten in ROS weiterzuverarbeiten. [6]

C. ZED Cam

Zur dreidimensionalen Wahrnehmung der Umgebung befindet sich an Bord des Autos auch eine stereoskopische Kamera. Ausgestattet mit zwei 4-Megapixel Kameras liefert diese nicht nur eine zweidimensionale Projektion der Umgebung, sondern zusätzlich auch aus den zwei Kamerabildern errechnete Tiefeninformationen.

Bei der hier verwendeten Kamera handelt es sich um eine *ZED Cam*, die eine Auflösung von bis zu 1080p

(bei 30 FPS) und eine Framerate von bis zu 100 FPS (bei WVGA Auflösung) ermöglicht. Die Kamera kann dabei Tiefenwerte zwischen $0.5m$ und $20m$ liefern, und errechnet anhand dieser Informationen auch Position und Rotation der Kamera auf 6 Achsen. Dabei wird eine Präzision von $\pm 0.1mm$ und $0.1°$ angegeben. [7]

Zu der ZED Cam gehört auch ein Softwarestack, der die Berechnungen der Tiefeninformation vornimmt und zu dem auch verschiedene Anbindung an z.B. ROS, TensorFlow oder OpenCV, was die Integration in dieses Projekt erleichtert. Zudem werden über diese Integrationen auch verarbeitete Daten wie z.B. Position, Rotation, aber auch eine 3D-Punktwolke und optional auch eine Umgebungskarte angeboten. Zudem verfügt die Softwareseite bereits über eine CUDA-Integration, wodurch zu erwarten ist, dass diese auf unserer Recheneinheit schon sehr performant ist. [7]

III. Recheneinheiten

Das Gehirn des Fahrzeuges ist ein Nvidia Jetson TX 2 Board, welches eine Kombination aus ARM-Kernen (Quad-Core ARM A57), zwei Denver 2 64-bit CPUs und 256 CUDA-Einheiten bietet. Diese große Menge an Kernen kann sehr gut für die Abarbeitung von parallelen Prozessen genutzt werden. Zudem bietet das Jetson-Board einen 8GB DDR4 Arbeitsspeicher und 32GB eMMC Speicher für das System. [8]

Damit das Jetson-Board in dem geringen Raum des Chassis montiert werden kann wird es auf ein Carrier-Board montiert, welches das Jetson-Board um verschiedene Schnittstellen wie z.B. USB, HDMI und GPIOs erweitert. [9]

Mit Hilfe der CUDA-Einheiten ist es denkbar, verschiedene Prozesse die möglichst effizient berechnet werden sollen, aber dabei hochparallelisierbar sind, wie z.B. Verarbeitung von Punktwolken, zu parallelisieren und somit die Performance des Systems zu maximieren.

IV. Aktoren

In Abbildung 7 ist das verwendete Chassis zu sehen. Der hier zu erkennende Elektromotor wurde ersetzt durch einen bürstenlosen Motor.

A. Antrieb

Der Antriebsmotor ist in unserem Fall ein bürstenloser Gleichstrommotor. Die Funktion eines solchen bürstenlosen Motors (BLDC) ist dabei wie folgt: in der Mitte ist ein Rotor montiert, der aus einem Magneten und einer Antriebswelle besteht. Um den Rotor herum ist der Stator montiert, der aus drei

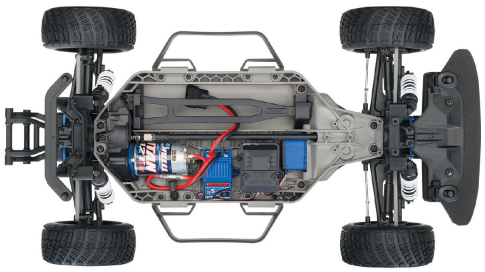


Abbildung 7. Das Chassis des genutzten Modell-Fernlenkautos. [2]

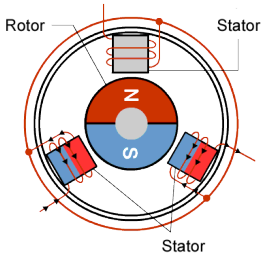


Abbildung 8. Aufbau eines BLDC. [10]

äquidistanten Spulen besteht. An diese Spulen wird reihum ein Strom angelegt, der ein Magnetfeld induziert. Je nach Reihenfolge dreht sich dann der Rotor durch das induzierte Magnetfeld (vgl. Abbildung 8). Der Vorteil eines solchen Motors im Gegensatz zu einem normalen Gleichstrommotor ist, dass es keine reibenden Teile (Bürsten) zur Stromübertragung gibt. Somit (und auch durch andere Faktoren) steigert es die Effizienz dieses Motors. [10]

Im Gegensatz zu einem normalen Gleichstrommotor benötigt ein BLDC bestimmte Steuersignale für die drei Spulen. Die Erzeugung dieser Signale übernimmt in einem solchen Fall ein ESC (*electronic speed control*). Dieses lässt sich über eine standardisierte Schnittstelle (bspw. UART, I2C, SPI, PPM, ...) ansteuern und erzeugt dann mit den richtigen Parametern die Ströme für den Motor.

In unserem Fall ist dieser ESC eine *FOCBox*, welche auf dem VESC-Projekt basiert. [11] Diese FOCBox erzeugt aus der Gleichspannung, welche von der Batterie zur Verfügung gestellt wird, die benötigten Wechselspannungen/Spannungspulse für den Motor. Dabei lassen sich viele Parameter, wie beispielsweise der maximal fließende Strom, aber auch Bremswirkung, regenerative

Bremsung etc. konfigurieren. Zudem bietet die FOCBox verschiedene Optionen, um die Performance des Motors zu beobachten.

Der Motor treibt in diesem Fernlenkauto ein Getriebe an, welches die Rotation sowohl auf die Hinterreifen als auch über eine Welle in der Mitte an die vorderen beiden Reifen überträgt. Somit fährt dieses Auto mit einem Allradantrieb.

B. Lenkung

Die Lenkung wird in dem gegebenen Fernlenkauto von einem Servomotor gesteuert. Diesem Servomotor kann man dabei eine gewissen Position bzw. einen Winkel mitteilen, welcher dann umgesetzt wird. Der Servomotor wird dabei durch ein PWM-Signal gesteuert. [2]

Die Ansteuerung des Lenkservos übernimmt in diesem Fall auch die FOCBox, der der entsprechende Lenkwinkel mitgeteilt werden kann. Der Servomotor ist mit der Vorderachse gekoppelt und schlägt je nach Ansteuerung den entsprechenden Lenkwinkel ein, womit das Auto in die gewünschte Richtung gelenkt werden kann.

V. Fazit

Die von dem F1/10-Projekt vorgegebene Plattform scheint eine gute, realitätsnahe Plattform zu sein, um ein autonomes Rennauto zu konstruieren. Es wird eine Vielzahl von Sensoren vorgegeben, die eine akkurate Erfassung der Umwelt und der eigenen Position des Fahrzeuges ermöglichen sollten.

Dadurch, dass sich das Fahrzeugmodell an einem echten Rennauto orientiert, sollten die erlernten Konzepte (zumindest zu einem gewissen Grad) auf echte Rennautos übertragbar sein. Ob die Recheneinheit dafür leistungsfähig genug wäre ist fragwürdig, jedoch geht es hier ja in erster Linie um die Entwicklung von Algorithmen unabhängig von dem Maßstab.

Literatur

[1] M. O'Kelly, V. Sukhil, H. Abbas, J. Harkins, C. Kao, Y. V. Pant, R. Mangharam, D. Agarwal, M. Behl, P. Burgio, and M. Bertogna, "F1/10: An open-source autonomous cyber-physical platform," 2019.

[2] "Traxxas ford fiesta st rally - specifications," https://traxxas.com/products/models/electric/ford-fiesta-st-rally?t=specs, zugriff: 21.10.2019.

[3] "Tdk invensense mpu 9250 imu," https://www.invensense.com/products/motion-tracking/9-axis/mpu-9250, zugriff: 21.10.2019.

[4] "Noaa ocean service — lidar explanation," https://oceanservice.noaa.gov/facts/lidar.html, zugriff: 21.10.2019.

[5] "Hokuyo ust-10lx datasheet," https://www.hokuyo-usa-com/application/files/2414/7196/1936/UST-10LX_Specifications.pdf, zugriff: 21.10.2019.

[6] "Ros hokuyo_node lidar driver," http://wiki.ros.org/hokuyo_node, zugriff: 21.10.2019.

[7] "Zed cam," https://www.stereolabs.com/zed/, zugriff: 21.10.2019.

[8] "Nvidia jetson tx2 board," https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/jetson-tx2/, zugriff: 21.10.2019.

[9] "Orbitty carrier for nvidia jetson tx2 / tx1," http://connecttech.com/product/orbitty-carrier-for-nvidia-jetson-tx2-tx1/, zugriff: 21.10.2019.

[10] http://renesas.com/us/en/support/technical-resources/engineer-school/brushless-dc-motor-01-overview.html, zugriff: 21.10.2019.

[11] "Focbox foc motor speed controller," https://enertionboards.com/FOCBOX-foc-motor-speed-controller.html, zugriff: 21.10.2019.

E ROS

Eine kurze Einführung in ROS

Woraus besteht ROS und wie verhalten sich die Komponenten untereinander

Jan Peter Meyer *Autonomous Racing Group*
TU Dortmund, Lehrstuhl Informatik 12
Dortmund, Deutschland
jan-peter.meyer@tu-dortmund.de

***Zusammenfassung*—Dieses Paper bietet einen ersten groben Überblick zur Einarbeitung in ROS. Dazu wird in diesem Paper die Philosophie und allgemeine Funktionsweise von ROS dargestellt und einzelne Begriffe von ROS eingeführt.**

I. Einleitung

ROS (Roboter Operating System) ist ein flexibles Framework zur Unterstützung von der Entwicklung komplexer Robotersystemen. Dabei steht das Vereinfachen des kollobirativen Arbeitens verschiedener Teams aus einer großen Bandbreite von Schwerpunkten, wie zum Beispiel Aktorik oder Objekterkennung, im Fokus. Als Lösungsansatz bietet ROS eine Infrastrukur, welche die Entwicklung kleiner Module ermöglicht, die durch das Kommunikationsnetz innerhalb dieser Infrastruktur zusammen größere Problemstellungen bewältigen können. Daneben bietet ROS durch dessen Community bereits viele fertige Lösungen für allgemeine Probleme und Tools, in denen eigene Module eingebettet werden können, wie zum Beispiel Gazebo, eine Simulationssoftware. [1]

II. Anwendungsbeispiel

Das kollobaratives Arbeiten im Bereich der Robotik ein zentraler Aspekt ist zeigt folgendes Anwendungsbeispiel.

In diesem Beispiel, soll ein Roboter einen Karton von Position A nach Position B bewegen. Bei dieser einfachen Problemstellung fallen bereits schon viele Aufgaben aus einer großen Bandbreite an.

Zunächst muss der Roboter im ersten Schritt den Karton erkennen können und das Objekt auf Angriffspunkte zum Greifen des Objektes analysieren. Dazu wird ein Team benötigt, dass sich mit Bilderkennung beschäftigt und eine notwendige Analyse des Objektes erstellen kann (zum Beispiel ein 3D-Modell des Objektes).

Im nächsten Schritt muss der Roboter sich zu dem Objekt hinbewegen und das Objekt greifen können. Sowohl das Hinbewegen des Roboters als auch das Greifen des Objektes sind eigene Problemstellungen. Zum einem muss beim Bewegen des Roboters die Umgebung beachtet und analysiert werden, worin sich dieser befindet, als auch für die Umsetzung des Greifens eine Sensorik und Aktorik für den Roboter entwickelt werden, welche den richtigen Druck zum Greifen des Objektes aufbringt. Zu viel Druck würde bedeuten, dass das Objekt beschädigt werden könnte, während zu wenig Druck dazu führt würde, dass es nicht angehoben werden kann.

Wenn man dieses Beispiel weiter betrachten würde, wird immer klarer das eine solch einfache Problemstellung trotzdem eine extrem hohe Komplexität besitzt und somit auch nur mit einer großen Anzahl an Spezialisten gelöst werden kann.

Um das gemeinsame Arbeiten verschiedener Teams zu vereinfachen, besitzt ROS eine eigene Philosophie.

III. ROS Philosophie

Die ROS Philosophie dreht sich im Kern um den Ausdruck „sharing and collaboration“ auf den sich die anderen Punkte der Philosophie thin, ROS-agnostic libraries und language independence zum Teil beziehen.

A. Sharing and Collaboration

Die Arbeitsstruktur innerhalb von ROS ist so aufgebaut, dass einzelne Teams an kleinen Prozessmodulen(Nodes) vereinzelt arbeiten können und diese dann über selbst definierte Schnittstellen zu einem großes System bilden können.(Collaboration)

Innerhalb der ROS-Gemeinschaft gibt es eine große Anzahl an Wegen sich gegenseitig zu unterstützen und eigene Lösungen zu präsentieren, wie zum Beispiel öffentliche Foren. Zudem existiereb bereits eingebettete Tools wie RVIZ oder Gazebo, in Repositorys von ROS.

B. Thin

Thin bezieht sich auf die vorher angesprochenen Prozessmodule (Nodes), die genutzt werden um die Umsetzung des Systems zu gewährleisten. Systeme mit ROS entwickelt werden, sollen aus einer großen Anzahl von Nodes bestehen, die miteinander kommunizieren und als einzelne Node schlank (thin) designt werden sollen.

C. ROS-agnostic libraries

Dieser Punkt der ROS-Philosophie steht für die Implementierung von definierten Interfaces innerhalb der einzelnen Nodes, damit die Kommunikation zwischen den einzelnen Nodes erfolgen kann.

D. Language Independence

Um die Entwickler nicht einzuschränken, folgt ROS dem Ansatz der Language Independence und erlaubt damit, den Entwicklern verschiedenste Programmiersprachen zur Erstellung der Nodes zu nutzen. ROS unterstützt offiziell die Programmiersprachen Python und C++, wobei C++ die weitaus Performance stärkere Programmiersprache ist.

E. Weitere Punkte

Neben den bereits genannten Punkten existiert das easy testing, dass ROS durch ein integriertes Testframework (rostest) anbietet und die gute Skalierbarkeit von Systemen die mit ROS entwickelt worden sind, da die Nodes parallel ausgeführt werden können.

IV. ROS Struktureller Aufbau

Das ROS Konzept besteht aus drei verschiedenen Ebenen. Die Filesystem Ebene, die Computational Graph Ebene und die Community Ebene, wobei sich die letzt genannte Ebene sich auf die Infrastruktur im Web bezieht, um die Kommunikation verschiedener Entwickler zu einfachen und im Weiteren nicht mehr betrachtet wird.

A. Filesystem Ebene

Das Filesystem beinhaltet verschiedene Packages, welche wiederum die verschiedensten Arten von Files enthalten können, wie zum Beispiel ROS runtime processes, ROS dependency libaries, datasets, Configuration files, etc.

Neben den normalen Packages existieren zudem Metapackeges. Die Aufgabe der Metapackages ist es Gruppen von Packages darzustellen. Die Metainformationen zu einem Package werden in einer Package Manifest Datei gespeichert. Als letztes zum Block Packages existieren Repositories, diese umfassen eine Sammlung von Packages.

Neben den Packages werden auch die Beschreibungen von Messages für Services und Topics (verschiedene Arten Nachrichten zu senden) gespeichert, welche die Strukturen der Nachrichten definieren.

B. Computational Graph Level

ROS ist im Generellen ein Peer-to-Peer Netzwerk von ROS-Prozessen(Nodes), welche zusammen Daten verarbeiten. Dieses Netzwerk besteht aus verschiedenen Komponenten: Master, Nodes, Messages, Topics, Parameter Server, Services und Bags.

Der ROS Master dient als eine Art DNS-Server in diesem Netzwerk. Es verwaltet alle Topics und Service (die beiden Arten der Kommunikation in ROS) Registrierungen der einzelnen Nodes, sodass ohne einen Master die Nodes nicht miteinander kommunizieren könnten.

Nodes sind die Prozesse, welche die eigentliche Berechnung in ROS durchführen. Sie werden nach dem Prinzip designt, dass die Aufgabe des einzelnen Nodes schlank ist (thin). Dies führt zu einem Roboter der aus einer großen Anzahl an Nodes besteht. Programmiert werden Nodes unter Einbindung der ROS Client Library der jeweiligen Programmiersprachen. Die ROS Client Libraries werden später noch einmal angesprochen.

Das Computational Netzwerk besitzt einen Parameter Server, dieser erlaubt es Daten mit einem Key zu speichern und wird im Master verwaltet. Zudem existiert damit die Möglichkeit Variablen, dynamic reconfigurable zu machen. Dies bedeutet, dass die Parameter zur Laufzeit verändert werden können, was eine große Hilfe bei der Durchführung von Tests ist.

Nodes kommunizieren mithilfe von Messages die zwischen den Nodes versendet werden. Messages sind einfache Datenstrukturen, welche definierten Typen folgen. Es existieren zwei Mechanismen, welche zur Kommunikation genutzt werden können.

Eine Asynchrone Art der Kommunikation bieten Topics. Die Kommunikation folgt dem Publish/-Subscriber Mechanismus. Die Nodes melden ihre Topics beim Master an und geben an, ob sie Daten zu diesem Topic generieren oder empfangen werden. Der Publisher wird bei diesem Mechanismus kederzeit seine Nachrichten zu seinem Topic generieren und ablegen können. Der Subscriber lauscht auf den Topic nach neuen Nachrichten und führt in

seinem Node einen Handler aus, wenn Nachrichten auf dem abonnierten Topic neu abgelegt worden sind. Die Nachrichten werden nicht über den Master gesendet, dessen Aufgabe ist nur die Organisation die Nodes mit den Topics zu vebinden. Die Kommunikation zwischen findet direkt zwischen den einzelnen Nodes direkt statt. Dieses Konzept der Kommunikation in ROS kann als many to many Kommunikation verwendet werden.

Um synchrone Kommunikation zwischen den Nodes zu erlauben wurde in ROS Services eingeführt. Diese Art der Kommunikation benötigt im Gegensatz zu der Topic Variant zwei vordefinierte Message Definitonen, während Topics nur eine benötigen. Die zwei vordefinierten Messages werden auf die Request und auf die Response Nachricht innerhalb eines Services aufgeteilt.

Als letzte Komponente existiert im Computational Graph Bags. Sie sind ein Format um die Daten von ROS Messages zu speichern und wiederholt abzuspielen.

V. ROS Kernkomponenten und Higher Level Concepts

Die Kernkomponente von ROS stellt sein Kommunikationsnetz dar und die Verwendung von sauberen Schnittstellen nach außen. Wie bereits zuvor erwähnt bietet ROS bereits Roboter-spezifische Features und Tools, die in das Komminkationsnetzwerk eingebaut werden können, bzw. das eigene Projekt um das bestehende Konstrukt herumgebaut wird. Als spezielle Tools die von besonderen Interesse für uns sind, sind RVIZ und Gazebo.

A. RVIZ

RVIZ wird als Tool genutzt um eine Visualisierung von Messdaten, wie zum Beispiel Koordinaten aus einer LiDARmessung, genutzt. RVIZ nutzt dazu einfach die Topics um die Messdaten des System zu erhalten und kann somit, einfach in die bestehende Systemumgebung eingebettet werden.

B. Gazebo

Gazabo ist eine Robotersimulation Bilbliothek. Sie bietet die Möglichkeit die Simulation in das Kommunikationssystem des Systems zu integrieren und so Sensorendaten zu simulieren oder das gesamte Verhalten eines Roboters in einer Umgebung zu simulieren. Es wird vorallem dazu verwendet erste Tests von Algortihmen zu testen.

VI. ROS Client Libraries

Bisher wurde nur das Konzept von ROS dargestellt, aber eine genaue Umsetzung in Code wurde nicht erwähnt. Umgesetzt wurden die Konzepte in den sogenannten ROS Client Libraries.

Allgemein ist ROS nach seiner Philosophie Programmiersprachenunabhängig. Der Fokus liegt aber auf der Entwicklung einer robusten C++ und Python Bibliothek. Bibliotheken in anderen Programmiersprachen existieren ebenfalls, sind aber nicht so auf Robustheit überprüft.

Literatur

[1] Documentation - ROS Wiki. http://wiki.ros.org/. Abgerufen: 12.09.2020.

F CUDA Programmierung

technische universität
dortmund

Projektgruppe Autonomous Racing
Seminarausarbeitung

CUDA Programmierung

Patrick Schmelter WS19

Projektgruppenleiter:
Dr.-Ing. Kuan-Hsun Chen
Niklas Ueter

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl Informatik XII
http://ls12-www.cs.tu-dortmund.de

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation und Hintergrund

Diese Ausarbeitung wird im Rahmen der Projektgruppe Autonomous Racing [1] im Wintersemester 2019 angefertigt und soll als Nachschlagreferenz zur weiteren Bearbeitung des Projektgruppenzieles für alle Teammitglieder dienen.

1.2 Ausgangspunkt

Die Projektgruppe führt die Arbeit an einem Projekt für Autonomes Fahren einer vorangegangenen Projektgruppe fort. Diese hat es zu Stande gebracht ein Modell eines Rally Autos im Maßstab 1/10 mit Hilfe eines LIDARs [2] vollkommen autonom um eine Rennstrecke herumfahren zu lassen. Dafür benötigt die Strecke jeweils am Rand der Fahrbahn Erhöhungen, welche vom LIDAR erfasst werden können. Das Auto versucht dann ständig in der Mitte der Abgrenzungen zu fahren. Des Weiteren wurden auch andere Ansätze verfolgt und implementiert um das Auto fahren zu lassen, darunter Neuronale Netze [3], Deep Reinforcement Learning [4] und Simultaneous Localization and Mapping (SLAM) [**SLAM**]. Zum Testen wurde eine in Gazebo [5] simulierte Umgebung verwendet. Das Auto verfügt zudem über ein automatisches Notbremssystem, einen Totmannschalter, einer Fernsteuerung über verschiedene Geräte und ein Geschwindigkeitsmesser mit Rundentimer. Das System basiert auf dem Robot Operating System (ROS) [6].

1.3 Ziele der Projektgruppe

Die aktuelle Projektgruppe wird sich mit der Thematik auseinandersetzen das autonom fahrende Auto weiter zu entwickeln. Im ersten Schritt wird der vorhandene Stand ausgiebig

geprüft und getestet, anschließend soll das Auto stehende Hindernisse auf der Fahrbahn automatisch erkennen können und diesen dann ausweichen. Anschließend wird das Problem auf sich bewegende Objekte ausgeweitet und soll schlussendlich auf ein Überholmanöver eines anderen autonom fahrenden Modelles übertragen werden. Zum Schluss soll der danach vorhandene Code dahingehend optimiert werden, dass Stecke und Hindernisse besser und schneller erkannt werden und die Rundenzeit damit minimiert wird. Am Ende der Arbeitszeit wird es einen Wettkampf gegen ein anderes Team der Universität geben, die sich mit einem ähnlichen Projekt beschäftigt. Es steht in Aussicht das Modell an der F1/10 Challenge [7] teilnehmen zu lassen, sofern die gelieferten Ergebnisse gut genug sind.

Kapitel 2

CUDA

2.1 Was ist CUDA

CUDA ist ein von Nvidia entwickeltes Konzept zur Abarbeitung von Programmteilen durch eine oder mehrere GPUs [8]. CUDA steht dabei für C̈ompute Unified Device Architectureünd umfasst neben des Konzeptes auch eine Programmiersprache zur Umsetzung eben dieses. Geschrieben werden die auszulagernden Programmteile in C̈ for CUDAünd C++ mit den entsprechenden Erweiterungen. Es existieren aber auch Wrapper für Perl, Python, Ruby, Java, FORTRAN, .Net, MatLab, Mathematica und R. Installiert werden können die CUDA Erweiterungen auf Windows, Linux sowie MacOS. Damit decken sie den Großteil aller Rechner ab. Zur Ausführung der Programmteile wird eine Grafikkarte von Nvidia ab der G̈eForce 8-Serie oder ab der Quadro FX5600 benötigt. Trotz der Herstellerbindung hat sich CUDA als Standard für die GPU Programmierung durchsetzen können.

2.2 Vorteile

Heutzutage sind CPUs in Desktoprechnern normalerweise mit zwei bis acht Prozessorkernen ausgestattet, was uns schon eine beachtliche Rechenleistung zur Verfügung stellt. Moderne Grafikkarten kommen jedoch mit Leichtigkeit auf mehrere Hundert Prozessorkerne die viele kleine Programmteile parallel ausführen können. Leider haben beide Recheneinheiten allerdings getrennte Speicher und der Kopiervorgang von einem Speicher in den anderen ist Ressourcenintensiv. Die Grafikkatenprogrammieren bietet sich daher vor allem bei hochgradig parallelisierbaren Aufgaben an, wie zum Beispiel verschiedene Simulationsberechnungen, der Video- und Bildbearbeitung, dem Machine Learning, dem anlernen von neuronalen Netzen und anderen KI Bereichen. Bemerkenswert ist hierbei, dass durch die gesteigerte Effizient durch das Auslagern von Rechenarbeiten an GPUs der Stromver-

brauch des gesamten Systems gesenkt werden kann.

2.3 Anwendungsfälle

Verwendung findet CUDA in vielen Anwendungen. Das erste bekannte Beispiel war das SETI@home Projekt [9]. Das SETI-Institut beschäftig sich mit der Suche nach außerirdischem Leben. Dazu werden riesige Radioteleskop-Arrays auf verschiedenen Teilen der Welt eingesetzt. Sie suchen den gesamten Himmel systematisch nach verschiedenen Radiosignalen ab, die auf intelligentes außerirdisches Leben schließen lassen könnten. Die Teleskope sammeln dabei täglich enorme Mengen an Daten, deren Weiterverarbeitung hohe Kosten verursacht und sehr viel Zeit in Anspruch nimmt. Mit Hilfe der noch heute aktiven SETI@home Anwendung kann seit 1999 jeder mit seinem eigenen Rechner bei der Suche mithelfen. Mit der Einführung von CUDA in die Anwendung konnte das Institut die Suche nach außerirdischem Leben auf Heinrechnern um das 10-Fache beschleunigen.

Badaboom [10] ist ein von Nvidia entwickelter Videokonverter, der im Vergleich zu Konvertierungen mittels CPU ganze 20 mal schneller arbeitet. So ist es kein Wunder, dass auch Adobe mit Photoshop, Premiere Pro, etc. auf CUDA setzt.

Kapitel 3

Installation

Für eine aktuelle Installation von CUDA 10.1 auf dem eigenen Rechner wird eine CUDA-Fähige Grafikkarte benötigt, außerdem mindestens Ubuntu 14.04, MacOS 10.13 oder Windows 7 und ein C Compiler. Auf der Internetseite von Nvidia wird ein Lokales und ein Netzwerkinstallationsprogramm angeboten [11]. Dieser kann heruntergeladen und ausgeführt werden. Nachdem die Installation erfolgreich abgeschlossen wurde, sind je nach Betriebssystem noch verschiedene Anpassungen vorzunehmen. Weitere Instruktionen sind dem Quick Start Guide oder den genauen Installationsanleitungen zu entnehmen [12].

Kapitel 4

Programmierung

4.1 Device vs Host

Bevor mit der Programmierung angefangen werden kann, müssen zwei Fachbegriffe eingeführt werden. Die CPU bezeichnet man in CUDA als den `Host`, die GPU wird `Device` genannt. Ein Host kann dabei mehrere Devices orchestrieren.

Abbildung 4.1: Device vs. Host

Der Ablauf eines typischen CUDA-Programmes sieht so aus, dass im ersten Abschnitt auf dem Host serieller Code zur Vorbereitung der parallelen Aufgabe ausgeführt wird. Es werde Speicherplätze alloziert und Daten gesetzt. Bevor das Device mit der Ausführung des parallelen Codes beginnen kann, muss der Host noch alle relevanten Variablen vom Host- in den Device-Speicher kopieren. Anschließend stößt der Host in einem sogenannten `Kernel-Launch` den parallelen Code aus einer speziellen CUDA Funktion auf dem Device an. Dieser Aufruf finden asynchron statt und der Host führt seine Programmanweisungen sofort weiter aus, währen das Device mit der Abarbeitung seines Programmteiles beschäftig ist. Sobald die Berechnungen auf dem Device abgeschlossen sind, muss sich ebenfalls der Host darum kümmern die Daten wieder zurück in den Host-Speicher zu kopieren. Hier können sie dann weiterverarbeitet oder ausgegeben werden.

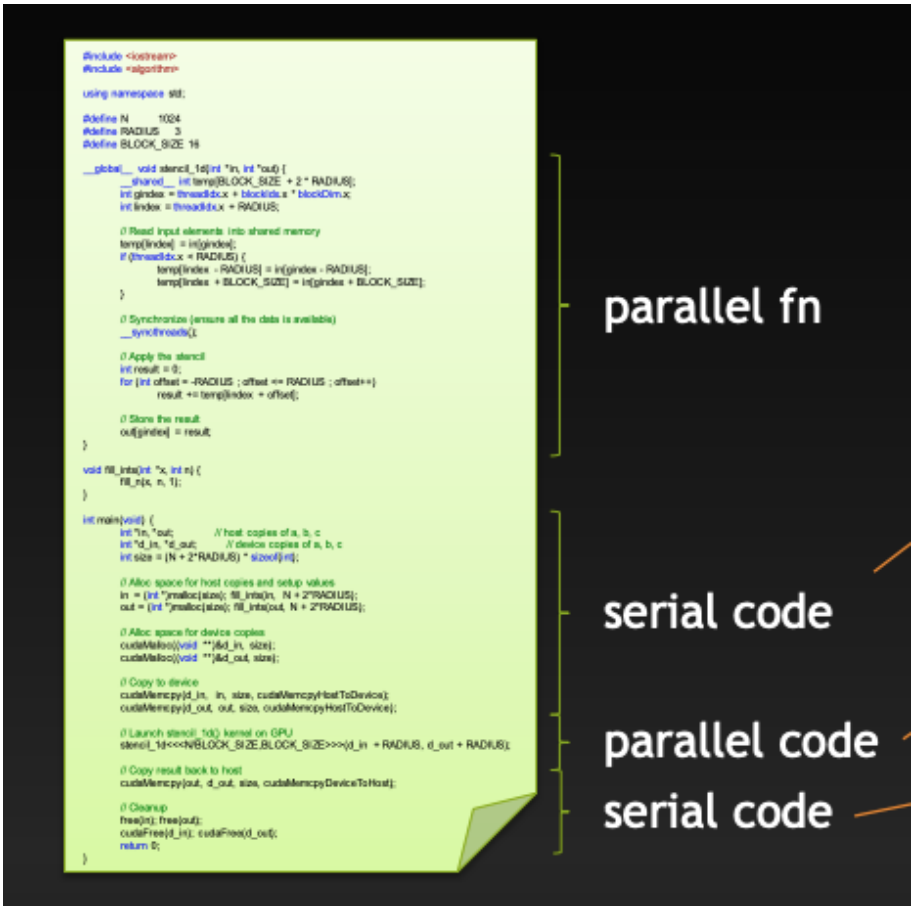


Abbildung 4.2: Ablauf eines CUDA Programmes [13]

4.2 Hello World!

```
1 int main(void) {
2   printf("Hello World!\n");
3   return 0;
4 }
```

Quellcode 4.1: Hello World! [13]

Hier zu sehen ist ein Ḧallo World!¨-Beispiel in C, welches mit Hilfe des Nvidia CUDA Compilers übersetzt wird. Erwähnenswert ist, dass der C-Code des Programmes mit dem normalen auf dem System vorhanden C Compiler übersetzt wird, während nur CUDA spe-

zifischer Code wirklich vom CUDA Compiler übersetzt wird [13].

4.3 Device Code

```
__gloabl__ void mykernel(void) {
}

int main(void) {
  mykernel<<1,1>>();
  printf("Hello World!\n");
  return 0;
}
```

Quellcode 4.2: Device Code [13]

In diesem Codeschnipsel wurde das Ḧello-World!¨-Programm um eine CUDA-Funktion erweitert. Markiert wird diese Funktion durch das `__global__` Keyword davor [14]. Alle mit diesem Keyword markierten Funktionen werden auf dem Device ausgeführt nachdem sie vom Host aufgerufen wurden. Diese Funktionen werden Kernel genannt und beinhalten den sogenannten Device Code. Aufgerufen werden solche Kernel wie in der `main`-Funktion zu sehen ist durch einen Kernel Launch mit zwei Parametern in den spitzen Klammern. Was die Parameter bedeuten, wird im weiteren Verlauf dieser Ausarbeitung erläutert werden.

4.4 Speicherverwaltung

```
__gloabl__ void add(int *a, int *b, int *c) {
  *c = *a + *b;
}
```

Quellcode 4.3: Kernel Additionsfunktion [13]

Nachdem klar ist was Device Code ist, folgt ein Beilspiel eines etwas sinnvolleren Kernels. Diese Additionsfunktion erwartet drei Pointer als Eingabeparameter. Diese Pointer sollen auf Speicheradressen des Devices verweisen und dort Addiert werden.

```
int main (void) {
  int a, b, c;       //host copies of a,b ,c
  int *d_a, *d_b, *d_c;  //device copies of a, b, c
  int size = sizeof(int);

  //allocate space for device copies of a, b, c
  cudaMalloc((void **)&d_a, size);
  cudaMalloc((void **)&d_b, size);
```

```
9   cudaMalloc((void **)&d_c, size);
10
11  //setup input values
12  a = 2;
13  b = 7;
14
15  [...]
```

Quellcode 4.4: Main Additionsfunktion Teil 1 [13]

Die Funktionen `malloc`, `free` und `memcpy` zur Speicherverwaltung auf dem Host sollten bekannt sein. Es existieren äquivalente Funktionen dafür auf dem Device, genannt `cudaMalloc`, `cudaFree` und `cudaMemcpy` [15]. Im Code-Beispiel werden zunächst die Variablen `a`, `b` und `c` auf dem Host und Pointer zu den Kopien derer auf dem Device deklariert. Mit `cudaMalloc` wird dann auf dem Device Speicherplatz für die Variablen in Größe eines Integers alloziert. Danach werden die Eingangsvariablen gesetzt.

```
1   [...]
2
3   //copy inputs to device
4   cudaMemcpy(d_a, &a, size, cudaMemcpyHostToDevice);
5   cudaMemcpy(d_b, &c, size, cudaMemcpyHostToDevice);
6
7   //launch add() kernel on GPU
8   add<<<1,1>>>(d_a, d_b, d_c);
9
10  //copy result back to host
11  cudaMemcpy(&c, d_c, size, codaMemcpyDeviceToHost);
12
13  //cleanup
14  cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
15  return 0;
16 }
```

Quellcode 4.5: Main Additionsfunktion Teil 2 [13]

Die Eingangsvariablen müssen nun auf den dafür vorgesehenen Speicher auf dem Device kopiert werden. Dazu wird `cudaMemcpy` verwendet. Anschließend kann der Kernel gestartet werden, welcher die beiden Werte für `a` und `b` auf dem Device addiert. Danach können wir das Ergebnis der Operation wieder mit `cudaMemcpy` vom Device auf den Host kopieren und den Device-Speicher mit `cudaFree` wieder freigeben [15].

4.5 Blöcke

```
add<<<1,1>>>();
      v
add<<<N,1>>>();
```

Quellcode 4.6: Launch N Blocks [13]

Bisher wurde der Kernel jeweils nur ein einziges Mal ausgeführt. In diesem Abschnitt wird betrachtet wie der Device Code einer einzigen Funktion mehrfach parallel ausgeführt werden kann. Der erste Parameter in den spitzen Klammern des Kernel Launches gibt die Anzahl der Blöcke an mit dem der Code ausgeführt werden soll. Soweit die Kapazität des Devices es zulässt werden alle Blöcke parallel von jeweils einem sogenannten `Streaming Multiprocessor` des Devices ausgeführt, welcher aus mehreren `Cuda Cores` besteht [16]. Die Gesamtheit der Blöcke werden als `Grid` bezeichnet. Innerhalb der Devices Codes kann man mittels `blockIdx.x` auf den jeweiligen Blockindex zugreifen [14]. Mit dieser Modifikation wird die einfache Additionsfunktion im Handumdrehen zu einer parallel berechnenden Vektoradditionsfunktion.

```
__gloabl__ void add(int *a, int *b, int *c) {
  *c[blockIdx.x] = *a[blockIdx.x] + *b[blockIdx.x];
}
```

Quellcode 4.7: Vektoradditionsfunktion [13]

4.6 Threads

Der zweite Parameter innerhalb der spitzen Klammern des Kernel Launches gibt an, mit wie vielen `Threads` ein Block jeweils gestartet werden soll. Äquivalent zu dem Blockindex weiß der Device Code mittels `threadIdx.x` auch in welchem Thread der Code momentan ausgeführt wird [14]. Threads werden physikalisch auf den Cuda Cores eines Streaming Multiprocessors ausgeführt [16].

4.7 Dimensionen

In den oberen beiden Abschnitten wurde auf den Index jeweils mit dem Suffix `.x` zugegriffen, was der ersten Dimension entspricht. Soweit das beim Kernel Launch nicht anders angegeben wird, ist das der Standard. Der Kernel kann mit bis zu drei Dimensionen auf Blocks und Threads gestartet werden. Auf die zweite und dritte Dimension kann jeweils mit `.y` und `.z` zugegriffen werden [17].

4.8 Datenaustausch

Innerhalb eines Blocks können Threads Daten untereinander austauschen. Das passiert dank sehr schnellem on-chip Speicher der Streaming Multiprocessors. Dadurch ist der Speicher allerdings wiederum nicht von anderen Threads in anderen Blocks sichtbar. Der Speicher wird in dem Device Code durch das `__shared__` Keyword gekennzeichnet [13].

```
1 __gloabl__ void stencil(int *in, int *out) {
2   __shared__ int temp[BLOCK_SIZE + 2 * RADIUS];
3   [...]
4 }
```

Quellcode 4.8: Stempelfunktion [13]

4.9 Race Conditions

Innerhalb eines Blocks besteht die Möglichkeiten die Threads zu synchronisieren um Race Conditions aller Arten zu vermeiden (RAW, WAR und WAW) [13]. Das geschieht indem im Device Code die `__syncthreads();` Funktion aufgerufen wird [13].

4.10 Zusammenfassung

Alle Threads führen das selbe sequenzielle Programm mit ggf. unterschiedlichen Daten aus. Die Threads werden jeweils parallel von verschiedenen Cuda Cores ausgeführt.
Ein Block beinhaltet eine Gruppe von Threads und wird auf einem Streaming Multiprocessor ausgeführt. Die Threads innerhalb eines Blockes können untereinander kommunizieren. Alle Blocks werden zu einem Grid zusammengefasst und auf verschiedenen Streaming Multiprocessors ausgeführt. Unter den Blocks gibt es keine direkte Kommunikation, so ist Synchronisation und Datenaustausch nur über ineffiziente Umwege möglich.
Ein Grid kann in bis zu drei Dimensionen arbeiten.

Abbildungsverzeichnis

Quellcodeverzeichnis

Literatur

[1] N Ueter KH Chen. *AUTONOMOUS RACING*. URL: https://ls12-www.cs.tu-dortmund.de/daes/de/lehre/lehrveranstaltungen/49-autonomous-racing/467-autonomous-racing-seminar1920.html.

[2] Spektrum. *LIDAR*. URL: https://www.spektrum.de/lexikon/physik/lidar/9033.

[3] Spektrum. *Neuronale Netze*. URL: https://www.spektrum.de/lexikon/neurowissenschaft/neuronale-netze/8653.

[4] AWS. *Overview of Reinforcement Learning*. URL: https://docs.aws.amazon.com/deepracer/latest/developerguide/deepracer-how-it-works-overview-reinforcement-learning.html.

[5] Open Source Robotics Foundation. *Gazebo*. URL: http://gazebosim.org/.

[6] Open Source Robotic Foundation. *Robot Operating System*. URL: https://www.ros.org/.

[7] PRECISE. *F1/10*. URL: http://f1tenth.org/.

[8] Nvidia. *CUDA Zone*. URL: https://developer.nvidia.com/cuda-zone.

[9] University of California. *SETI@home*. URL: https://setiathome.berkeley.edu/.

[10] Nvidia. *Badaboom*. URL: https://www.nvidia.de/object/badaboom_de.html.

[11] Nvidia. *CUDA Toolkit 10.1 Update 2 Download*. URL: https://developer.nvidia.com/cuda-downloads.

[12] Nvidia. *CUDA Toolkit Documentation v10.1.243 - Installation Guides*. URL: https://docs.nvidia.com/cuda/#installation-guides.

[13] Nvidia. *Standard Introduction to CUDA C Programming*. URL: https://www.olcf.ornl.gov/wp-content/uploads/2013/02/Intro_to_CUDA_C-TS.pdf.

[14] Mark Harris. *An Even Easier Introduction to CUDA*. URL: https://devblogs.nvidia.com/even-easier-introduction-cuda/.

[15] Mark Harris. *An Easy Introduction to CUDA C and C++*. URL: https://devblogs.nvidia.com/easy-introduction-cuda-c-and-c/.

[16] cibercitizen1. *Understanding CUDA grid dimensions, block dimensions and threads organization*. URL: https://stackoverflow.com/questions/2392250/understanding-cuda-grid-dimensions-block-dimensions-and-threads-organization-s.

[17] Udacity. *Configuring the Kernel Launch Parameters 2 - Intro to Parallel Programming*. URL: https://www.youtube.com/watch?v=yNs8B1VnMAA.

G Neuronale Netze

Neuronale Netze
Autonomous Racing Group, LS 12 TU Dortmund

Marvin Langenkämper, Matr.-Nr.:160847
Fakultät für Informatik
TU Dortmund

8. September 2020
im WS 2019/20

1 Einleitung

In der Projektgruppe 'Autonomous Racing: Head-To-Head Racing Capabilities' soll es darum gehen, ein Auto möglichst schnell in einem Rundkurs autonom fahren zu lassen. Hierbei stehen unter anderem ein LIDAR Radar Sensor und eine Stereo Kamera als Sensoren zur Verfügung. Neuronale Netze haben in den letzten Jahren aufgrund ihrer guten Resultate in der Bilderkennung vermehrt an Bedeutung gewonnen. Diese Ausarbeitung erklärt die grundlegende Idee von Neuronalen Netzen, um anschließend die Anwendung dieser und weiterer Machine Learning Verfahren in der Projektgruppe im Zusammenspiel mit den vorhandenen Sensoren zu diskutieren.

Neuronale Netze werden oft mit aktuellen Themen wie künstlicher Intelligenz, automatischer Bilderkennung und autonom fahrenden Autos in Verbindung gebracht. Die Idee hinter den künstlichen Netzen, mit denen man versucht die Arbeitsweise des menschlichen Gehirns nachzustellen, stammt jedoch bereits aus den 1940-er Jahren [4]. Der Grund dafür, dass Neuronale Netze erst seit wenigen Jahren in aller Munde sind und bei vielen Machine Learning Anwendungen benutzt werden, sind verbesserte Rechenleistungen von Grafikkarten und Prozessoren, die es ermöglichen die aufwendigen Berechnungen in annehmbarer Zeit durchzuführen.

Genau wie das menschliche Gehirn, besteht auch ein Neuronales Netz aus einer Vielzahl von Neuronen. Mit den mehreren Milliarden natürlichen Neuronen oder auch Nervenzellen des im Kopf gelegenen Teil des zentralen

Nervensystems von uns Menschen können die künstlichen Neuronen, die in den künstlichen Netzen benutzt werden, bisher jedoch weder in Quantität noch in Komplexität mithalten.

2 Klassifikation oder Regression

In der Informatik spricht man von einem Klassifikationsproblem, wenn es darum geht eine bestimmte Eingabe $\boldsymbol{x}$ zu einer von $n \in \mathbb{N}$ möglichen gegebenen Klassen korrekt zuzuordnen. Beispiele für Klassifikationsprobleme sind die Diagnose einer bestimmten Krankheit bei gegebenen Symptomen oder das Erkennen, ob ein Hund oder eine Katze auf einem Bild zu sehen ist. Klassifikationsprobleme sind einer der größten Anwendungsgebiete für Machine Learning Verfahren und Neuronaler Netze.

Beim Regressionsproblem dagegen, geht es nicht darum basierend auf der Eingabe eine von $n \in \mathbb{N}$ möglichen Klassen vorherzusagen, vielmehr wird hierbei versucht ein kontinuierlicher Wert $y \in \mathbb{R}$ vorherzusagen. Ein Anwendungsbeispiel für das autonome Fahren wäre hierbei beispielsweise die Bestimmung des optimalen Lenkwinkels basierend auf der Messung gegebener Sensoren.

Die Art der Eingabe $\boldsymbol{x}$ ist von Problem zu Problem unterschiedlich. Die meißten Machine Learning Verfahren setzen einen numerischen Merkmalsvektor als Eingabe voraus. In diesem Kapitel gehen wir davon aus, dass ein solcher numerischer Merkmalsvektor vorliegt. Auf das Thema der Merkmalsextraktion im Allgemeinen wird in dieser Ausarbeitung nicht eingegangen. In Abschnitt 5.2 gehen wir jedoch auf Faltungsnetze, einer speziellen Art der Neuronalen Netzen, ein bei denen es im Speziellen um die Merkmalsextraktion geht, wenn Bilder als Eingabe vorliegen.

2.1 Das Zwei-Klassen-Problem

Das Zwei-Klassen-Problem ist ein Klassifikationsproblem bei dem nur zwei mögliche Klassen gegeben sind. Ein Beispiel hierfür ist die Klassifikation eines Bildes in ein Hund- oder Katzenbild. Des Weiteren haben wir bereits eingegrenzt, dass wir ausschließlich Probleme betrachten bei denen die Eingabe als numerischer Merkmalsvektor vorliegt. Um das Problem grafisch anschaulich verdeutlichen zu können, betrachten wir nun ein Zwei-Klassen-Problem mit einem Merkmalsvektor der Länge zwei. Der sogenannte Merkmalsraum wird dadurch durch die beiden Merkmale x_1 und x_2 aufgespannt. Wir gehen nach der Merkmalsextraktion davon aus, dass sich im Merkmalsraum Gebiete finden lassen, die sich möglichst eindeutig zu jeweils einer der beiden Klas-

sen zuordnen lassen. Diese Gebiete die sich eindeutig einer Klasse zurordnen lassen, nennen wir Klassengebiete. In Abbildung 1 ist ein zweidimensionaler Merkmalsraum mit zwei beispielhaften Klassengebieten zu sehen.

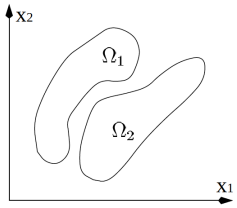


Abbildung 1: Beispielhafte Klassengebiete der Klassen Ω_1 und Ω_2 im zweidimensionalen Merkmalsraum

Bei der Lösung des Klassifikationsproblemes geht es nun darum die Klassengebiete im Merkmalsraum zu finden und diese durch eine Trennfunktion möglichst öptimalßu trennen. In der Praxis ist nie bekannt wie sich die Klassengebiete im Merkmalsraum wirklich gestalten, da in der Regel nur eine endliche Stichprobe jeder Klasse gegeben ist von der man versucht auf die Gesamtheit aller möglichen Ausprägen zu schließen. Eine Trennfunktion kann daher hinsichtlich bestimmter Kriterien, die auf der bekannten Stichprobe basieren, optimal sein.Es ist jedoch nicht möglich eine Aussage darüber zu treffen, ob diese Trennfunktion wirklich hinsichtlich der Gesamtpopulation optimal ist. In Abbildung 2 ist eine mögliche Trennfunktion abgebildet, die die beiden Klassengebiete von einander trennt.

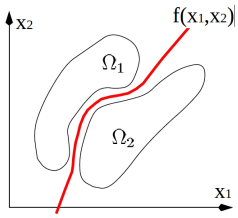


Abbildung 2: mögliche Trennfunktion von zwei Klassengebieten in einem zweidimensionalen Merkmalsraum

2.2 Das Regressionsproblem

Beim Regressionsproblem wird ein Zusammenhang zwischen dem Eingangsvektor $\boldsymbol{x}$ und dem Label y angenommen. Dieser Zusammenhang kann als

Funktion $f(\boldsymbol{x}) = y$ beschreiben werden. Wenn wir nun ein Modell trainieren, versuchen wir diese Funktion so gut wie möglich zu modellieren. Diese Trainingsdaten können als zu approximierende Stützstellen von f interpretiert werden. Die Ausgabefunktion soll nun diese Stützstellen einerseits so genau wie möglich approximieren, gleichzeitig aber auch die Funktion nicht zu speziell auf die Stützstellen anpassen, um auch nicht in den Trainingsdaten enthaltene Datenpunkte gut abzubilden.

3 Einführung in künstliche Neuronale Netze

Neuronale Netze bestehen aus mehreren Bestandteilen auf die im Folgenden eingangen wird. Zur Einführung wird zunächst die einfachste Art der Neuronalen Netze, das Mehrschichtenperzeptron, betrachtet. Dieses besteht aus mehreren hintereinander liegenden Schichten, die wiederum jeweils aus mehreren nebeneinander liegenden Neuronen bestehen. In diesem Kapitel wird zunächst auf Neuronen eingegangen, die das kleinste Element eines künstlichen Neuronalen Netzes darstellen. Anschließend wird eine einzelne Schicht sowie ein gesamtes Mehrschichtenperzeptron eingegangen. Die Funktionsweise der einzelnen Elemente wird zunächst anhand eines Klassifikationsproblems erklärt.

3.1 Neuronen

Die Funktionsweise eines künstlichen Neurons ist gut anhand einer Grafik wie Abbildung 3 zu erklären. An der linken Seite des Neurons liegen eine feste Anzahl von numerischen Eingänge x_i an. Jeder dieser Eingänge hat ein Gewicht w_i mit $i > 0$. Zusätzlich zu diesen Eingängen gibt es einen weiteren Eingang, an dem immer der Wert 1 anliegt. Dieser Eingang wird auch als Bios bezeichnet. An dem Bios liegt das Gewicht w_0 an.

Das künstliche Neuron wird bereits vollkommen definiert durch den Gewichtsvektor $\boldsymbol{w}$ und die Aktivierungsfunktion θ. Die Anzahl der Einträge des Vektors $\boldsymbol{w}$ geben hierbei die Anzahl der Eingänge des Neurons an. Hierbei ist zu beachten, dass w_0 immer für das Gewicht des Bios reserviert ist. Bei der Berechnung der Ausgabe des künstlichen Neurons wird zunächst jeder Eingangswert mit seinem anliegenden Gewicht multipliziert. Anschließend wird die Summe über alle Produkte gebildet. In Vektorschreibweise lässt sich diese Berechnung daher wie folgt darstellen: $y = \boldsymbol{x}^T\boldsymbol{w}$

Der letzte Schritt der Berechnung ist die Anwendung der Aktivierungsfunktion auf die berechnete Summe, sodass sich die Ausgabe des Neurons wie folgt berechnen lässt: $u = \Theta(\boldsymbol{x}^T\boldsymbol{w})$

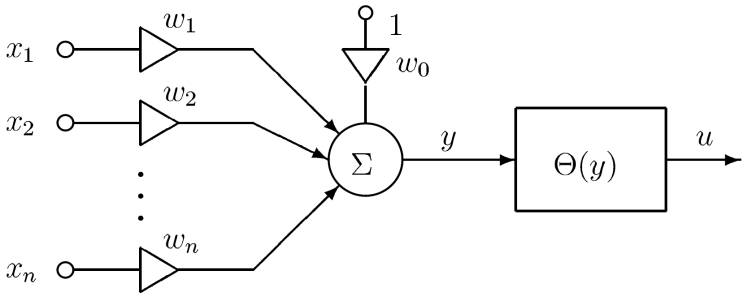


Abbildung 3: Aufbau eines künstlichen Neurons

3.2 Aktivierungsfunktion

Für die Aktivierungsfunktion können verschiedene Funktionstypen benutzt werden dessen Wertebereich zwischen 0 und 1 liegt. Ein einfaches Beispiel ist die Schwellwert-Funktion:

$$\phi(v) = \begin{cases} 0 & \text{für } v < s \\ 1 & \text{für } v \geq s \end{cases} \tag{1}$$

Die Funktion wird also genau dann 'aktiviert', wenn ein bestimmter Schwellwert s erreicht wird.

3.3 Neuronen als Klassifikatoren

Wie in 2.1 beschrieben, versucht man beim Lösen eines Klassifikationsproblems eine Trennfunktion zu finden, um die Klassengebiete im Merkmalsraum möglichst 'optimal' von einander zu trennen. Da diese 'optimale' Trennfunktion beliebig im Merkmalsraum liegen kann, wird nun betrachtet wie gut sich ein einzelnes Neuron eignet, um eine beliebige Funktion darzustellen.

Ohne die Aktivierungsfunktion hängt die Ausgabe des Neurons linear von der Eingabe ab. Die Ausgabefunktion würde in diesem Fall nur von der Vektormultiplikation des Eingabevektors und des Gewichtsvektors abhängen. Durch die Anwendung einer nicht-linearen Aktivierungsfunktion auf diese Vektormultiplikation können auch nicht-lineare Funktionen dargestellt werden.

3.4 Schichten

Auf dem Weg zum Mehrschichtenperzeptron schauen wir uns nun das nächst größere Element, die Schicht, an. Eine Schicht besteht aus mehreren nebeneinander gelegten Neuronen. Genau wie die Aktivierungsfunktion ist die Anzahl Neuronen pro Schicht heuristisch festzulegen. In der Regel werden alle Neuronen auf einer Schicht mit der gleichen Aktivierungsfunktion versehen. Zudem haben alle Neuronen einer Schicht den gleichen Eingabevektor. Die Ausgabe einer Schicht ist der Vektor, der sich aus allen Ausgaben der in der Schicht enthaltenen Neuronen ergibt.

3.5 Mehrschichtenperzeptron

Bei einem Mehrschichtenperzeptron legen wir nun mehrere Schichten hintereinander. Der Eingangsvektor jeder Schicht ist dabei der Ausgangsvektor der vorhergehenden Schicht. Damit bekommt jedes Neuron einer Schicht jeden Ausgang der vorherigen Schicht übergeben. Die Anzahl der Schichten wird genau wie die Anzahl der Neuronen pro Schicht heuristisch festgelegt. Oft wird hierbei eine Anzahl von mindestens drei Schichten gewählt, da hiermit jede beliebige Trennfunktion im Merkmalsraum gelernt werden kann. Ein beispielhafter Aufbau eines Mehrschichtenperzeptron ist in Abbildung 4 skizziert.

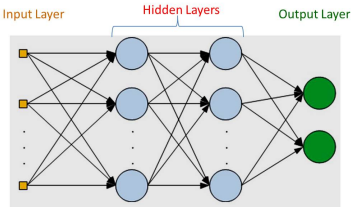


Abbildung 4: Beispielhafter Aufbau eines Multilayer-Perceptron. Quelle:[1]

3.6 Ausgabeschicht

Am Ende jedes neuronalen Netzes befindet sich die Ausgabeschicht. Die Modellierung der Ausgabeschicht ist stark abhängig von dem zu lösenden Problem. Bei einem Klassifikationsproblem wird durch die Ausgabeschicht oft ein sogenanntes 'one-hot-encoding' realisiert. Hierbei modelliert man für jede mögliche Ausgabeklasse i eine Ausgabe $\overline{y}_i = \{0, 1\}$. Man trainiert das Netz nun so, dass $\overline{y}_i = 1$, wenn der Eingabevektor der Klasse i zuzuordnen

ist. Beim Regressionsproblem reicht dagegen eine einzige Ausgabe $\overline{y}$. Es gibt verschieden Optimierungsmöglichkeiten für die Gewichte in einem Neuronalen Netz während des Trainings. Eine Möglichkeit ist die Minimierung der Summe der quadratischen Differenz $(\overline{y} - y)^2$ der Trainingsmenge.

4 Trainieren eines Neuronalen Netzes

In Kapitel 3 wurde bereits erläutert, wie ein Mehrschichtenperzeptron aufgebaut ist. Hierbei ist zu beachten, dass es die Aktivierungsfunktion, die Anzahl der Neuronen pro Schicht sowie die Anzahl der Schichten heuristisch festzulegen gilt. Des Weiteren ist bekannt, dass ein einzelnes Neuron durch seine Aktivierungsfunktion und den Gewichtsvektor definiert ist. Wenn nun also das Grundkonstrukt eines Mehrschichtenperzeptron aufgestellt wurde, gilt es im Training die Gewichte für jedes Neuron zu wählen.

4.1 Gewichte als Parameter

Neuronale Netze müssen, genau wie alle anderen Machine Learning Modelle, trainiert werden. Beim Trainieren eines Modells werden in der Regel bestimmte Parameter des Modells angepasst. Bei einer linearen Regression sind dies beispielsweise die einzelnen Koeffizienten des linearen Modells. Im Falle der Neuronalen Netze sind diese anzupassenden Parameter die Gewichte jedes einzelnen Neurons. Diese werden bei der Initialisierung des Neuronalen Netzes zunächst zufällig initialisiert. Man startet also mit einer beliebig schlechten Lösung. Hinzukommt, dass im Vergleich zu den meißten anderen Machine Learning Verfahren, eine um ein vielfaches größere Anzahl an zu trainierenden Parametern vorliegt. Daher wird für das Trainieren eines Neuronalen Netzes in der Regel eine weitaus größere Menge an Trainingsdaten benötigt als für einfachere Modelle.

4.2 Backtracking

Um die Gewichte des Neuronalen Netzes zu trainieren, wendet man das sogenannte Backtracking an. Hierbei wird für eine Teilmenge der Trainingsdaten eine Vorhersage durch das Neuronale Netz mit seinen aktuellen Parametern berechnet. Anschließend wird für jedes Trainingsdatum die Differenz zwischen der Vorhersage des Neuronalen Netzes und dem eigentlich erwarteten Ergebnis gebildet. Diese Differenz bezeichnet man dann als Fehler der Vorhersage. Beim Backtracking wird dann dieser Fehler auf die einzelnen Gewichte

im Neuronalen Netz zurück verfolgt. Da dies von hinten nach vorne beginnend mit der letzten Schicht geschieht, spricht man vom Backtracking. Die Funktionsweise des Backtrackings wird in dieser Ausarbeitung nicht näher erläutert ist aber beispielweise in [3] näher beschrieben.

5 Deep Learning

Von Deep Learning spricht man, wenn Neuronale Netze mit einer großen Anzahl von hintereinander gelegten Schichten vorliegen. Ab wie vielen vorliegenden Schichten man von Deep Learning spricht, ist jedoch nicht genau definiert. Ein sehr populäres Einsatzgebiet des Deep Learnings ist die Bilderkennung, auf die im Folgenden eingegangen wird.

5.1 Bilderkennung

Bisher wurde davon ausgegangen, dass für ein Neuronales Netz ein numerischer Vektor als Eingabe vorliegt. Soll jedoch die Klassifikation eines Bildes vorgenommen werden, ergibt sich ein solcher numerischer Vektor nicht ohne vorverarbeitende Schritte. Für ein möglichst optimales Modell, wäre also zunächst eine Vorverarbeitung notwendig, um einen numerischen Vektor zu extrahieren, der die vorliegenden Bilder möglichst optimal beschreibt.

Bei der Bilderkennung mit Neuronalen Netzen, lässt man diese sogenannte Merkmalsextraktion meißt implizit vom Modell selbst lösen. Die Merkmalsextraktion findet in zusätzlichen Schichten des Neuronalen Netzes statt. Das Problem der gesteigerte Komplexität wird also mit mehr Schichten im Modell begegnet.

5.2 Faltungsnetze

Um in einem Neuronalen Netzwerk aus einem Bild implizit Merkmale zu generieren, nutzt man sogenannte Faltungsnetze. In Faltungsnetzen oder auch Convolutional Neural Networks nutzt man in der Regel abwechselnd sogenannte Convolutional und Pooling Schichten. In den Convolutional Schichten wird pro Ausgangsschicht ein Fenster fester Größe über das Eingangsbild geschoben. Die daraus resultierende Ausgangsschicht ergibt sich daraufhin durch die Iteration diese Fensters über das Eingangsbild. Jedes Pixel ergibt sich aus der Summe der Multiplikationen der Eingangspixel mit den entsprechenden Gewichten des Fensters. Die Gewichte werden nicht festgelegt gesetzt sondern wie die Gewichte eines Mehrschichtenperzeptrons während des Trainings des Neuronalen Netzes optimiert.

In den Pooling-Schichten wird der Output der Faltungsschichten komprimiert. Auch hier wird ein Fenster fester Größe über den Input geschoben. Diesmal wird jedoch je nach Art des Poolings auf eine unterschiedliche Art und Weise aggregiert. Beliebte Aggregationsfunktionen sind die Minimal-, Maximal- oder Mittelwertbestimmung.

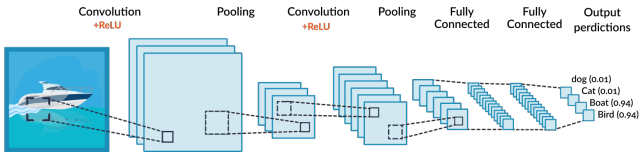


Abbildung 5: Beispielhafter Aufbau eines Convolutional Neural Network. Quelle:[2]

Die letze Pooling-Schicht bildet dann die Eingabe für ein Mehrschichtenperzeptron. Sodass sich letztendlich ein wie in Abbildung 5 dargestellter Aufbau ergibt.

6 Anwendung von Neuronalen Netzen in der PG

In der Projektgruppe wollen wir Neuronale Netze nutzen, um das Rennauto autonom möglichst schnell die auf der Rennstrecke fahren zu lassen. Dazu stellt sich die Frage, wo die entsprechenden Trainingsdaten gesammelt werden können, mit denen das Neuronale Netz trainiert werden kann. Hierbei ist besonders die Tatsache zu beachten, dass wir vor dem Rennen bei der F1/10 challange nur drei Test-Runden haben, um das Auto zu trainieren. Weiter stellt sich die Frage für welche Entscheidungen während der Fahrt das Neuronale Netz eingesetzt werden soll. Beispielsweise kann das Neuronale Netz entscheiden wie gelenkt werden soll, wie stark beschleunigt wird oder sogar die komplette Steuerung übernehmen.

Literatur

[1] basics-of-multilayer-perceptron. `https://kindsonthegenius.com/blog/2018/01/basics-of-multilayer-perceptron-a-simple-explanation-of-multilayer-perceptron.html`. Accessed: 2019-011-15.

[2] convolutional-neural-network. `https://missinglink.ai/guides/convolutional-neural-networks/convolutional-neural-network-tutorial-basic-advanced/`. Accessed: 2019-011-15.

[3] F. Günther and S. Fritsch. neuralnet: Training of neural networks. *The R journal*, 2(1):30–38, 2010.

[4] N. Hjort. *Pattern recognition and neural networks*. Cambridge university press, 1996.

H Herleitung Bremspunkt

Gegeben:

- Distanz s zum Kurveneingang
- Aktuelle Geschwindigkeit v_0 des Autos
- Zielgeschwindigkeit v_t des Autos
- Maximale Beschleunigung a des Autos

Gesucht:

- Distanz s_a vom Auto zum Bremspunkt in der beschleunigt wird
- Distanz s_b vom Bremspunkt zum Kurveneingang in der gebremst wird
- Geschwindigkeit v, die maximale erreicht werden kann nach der Beschleunigungsphase

Herleitung:

$$\begin{aligned} & & v &= at \\ \Leftrightarrow & & t &= \frac{v}{a} \end{aligned} \tag{5}$$

$$\begin{aligned} & s &= \frac{a}{2}t^2 & \quad | \text{ Setze (5) für } t \text{ ein.} \\ \Leftrightarrow & s &= \frac{a}{2}\frac{v^2}{a^2} \\ \Leftrightarrow & s &= \frac{v^2}{2a} \\ \Leftrightarrow & a &= \frac{v^2}{2s} \end{aligned} \tag{6}$$

$$\begin{aligned} & s_a &= \frac{a}{2}t^2 + v_0 t & \quad | \text{ Setze (6) für } a \text{ ein.} \\ \Leftrightarrow & s_a &= \frac{v^2 - v_0^2}{2a} \end{aligned} \tag{7}$$

$$\begin{aligned} & s_b &= \frac{a}{2}t^2 + v_t t & \quad | \text{ Setze (6) für } a \text{ ein.} \\ \Leftrightarrow & s_b &= \frac{v^2 - v_t^2}{2a} \end{aligned} \tag{8}$$

$$
\begin{aligned}
& s = s_a + s_b && \mid \text{Setze (7) und (8) für } s_a \text{ und } s_b \text{ ein.} \\
\Leftrightarrow \quad & s = \frac{v^2 - v_0^2}{2a} + \frac{v^2 - v_t^2}{2a} \\
\Leftrightarrow \quad & s = \frac{2v^2 - v_0^2 - v_t^2}{2a} \\
\Leftrightarrow \quad & \frac{2v^2}{2a} = s + \frac{v_0^2 + v_t^2}{2a} \\
\Leftrightarrow \quad & v = \sqrt{as + \frac{v_0^2 + v_t^2}{2}} && (9)
\end{aligned}
$$

$$
\begin{aligned}
& s_a = \frac{v^2 - v_0^2}{2a} && \mid \text{Setze (9) für } v \text{ ein.} \\
\Leftrightarrow \quad & s_a = \frac{as + \frac{v_0^2 + v_t^2}{2} - v_0^2}{2a} \\
\Leftrightarrow \quad & s_a = \frac{as + \frac{v_0^2 + v_t^2 - 2v_0^2}{2}}{2a} \\
\Leftrightarrow \quad & s_a = \frac{as + \frac{v_t^2 - v_0^2}{2}}{2a} \\
\Leftrightarrow \quad & s_a = \frac{as}{2a} + \frac{\frac{v_t^2 - v_0^2}{2}}{2a} \\
\Leftrightarrow \quad & s_a = \frac{s}{2} + \frac{v_t^2 - v_0^2}{4a} && (10)
\end{aligned}
$$

$$
\begin{aligned}
& s_b = s - s_a && \mid \text{Setze (10) für } s_a \text{ ein.} \\
\Leftrightarrow \quad & s_b = \frac{s}{2} + \frac{v_0^2 - v_t^2}{4a} && (11)
\end{aligned}
$$

Anschließend wird eine Konstante c zu dem Bremsweg s_b hinzu addiert, damit Messungenauigkeiten ausgeglichen werden können und ein Sicherheitsabstand hergestellt werden kann:

$$
s_b = \frac{s}{2} + \frac{v_0^2 - v_t^2}{4a} + c \qquad (12)
$$

Die Konstante c muss experimentell ermittelt werden.

I Hygienekonzept OH16

Hygienekonzept für die Nutzung des Raums OH16/U12 für Experimente der Projektgruppe 629

Lehrstuhl Informatik 12

1 Hygienekonzept

Vorbemerkung Die Lehrveranstaltung Projektgruppe ist eine zweisemestrige praktische Pflichtveranstaltung in den Masterstudiengängen Informatik und Angewandte Informatik.

Die Projektgruppe „PG 629: F1/10 - Autonomous Racing: Head-To-Head Racing Capabilities“ experimentiert mit Modellrennwagen und benötigt einen Raum mit ausreichender Fläche zum Aufbau der Rennstrecke sowie Zugang von Teilnehmerinnen und Teilnehmer, da eine Fernsteuerung via Internet zu große Verzögerungen verursacht.

In der Regel arbeiten die Studierenden von zu Hause aus und kommunizieren online. Nur für notwendige Experimente werden die Studierenden das Gebäude betreten.

Abbildung 1: Ansicht aus dem separaten Raum.

Abbildung 2: Seitenansicht des separaten Raums.

Raum Der Raum OH16/U12 steht der Projektgruppe bis auf Weiteres exklusiv zur Verfügung. Der Raum ist als ADV Periph. Geräte Raum deklariert und hat ca. 170 m^2 Grundfläche. Des weiteren hat der Raum an drei Seiten Fenster, sodass eine gute Belüftung sichergestellt werden kann. WCs befinden sich in unmittelbarer Nähe des Raums und sind exklusiv für das Untergeschoss des Gebäudes.

Jeder anwesenden Person (Teilnehmer und Aufsichten) steht ein Tisch mit Stuhl in ausreichendem Abstand (mind. 1,5 m) zu den Tischen der anderen und zur Experimentierfläche zur Verfügung. Der Mindestabstand zwischen den Tischen und der Experimentierfläche ist am Boden markiert.

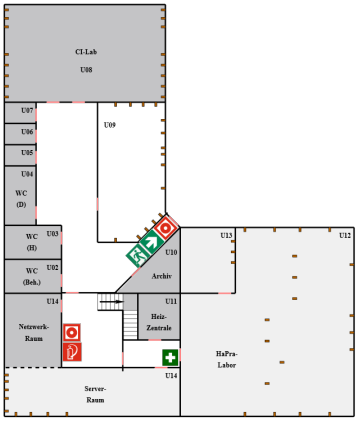


Abbildung 3: Grundriss des Untergeschosses des OH16.

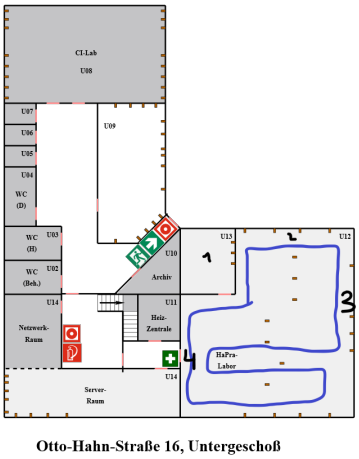


Abbildung 4: Eingezeichnete Rennstrecke und Positionen für die Teilnehmer.

Buchung des Raums Der Raum wird vom Lehrstuhl 12 verwaltet. Eine Buchung ist daher nicht nötig. Die Nutzung des Raums für die Projektgruppe wird dem Dekanat Informatik (pandemie@cs.tu-dortmund.de) eine Woche zuvor angezeigt. Das Dekanat prüft, ob weitere Veranstaltungen oder mündliche Prüfungen geplant sind, und informiert das Dezernat 6 wegen einer Reinigung des Raums.

Eine Session dauert voraussichtlich 1–3 Stunden.

Gruppenbildung Die Teilnehmerinnen und Teilnehmer werden in Zweier- bis Dreiergruppen aufgeteilt, die ggf. den Raum gemeinsam nutzen. In der Regel wird der Raum von je einer Zweier- bis Dreiergruppe genutzt, maximal von zwei Zweiergruppen. Wenn möglich, soll nur ein Teilnehmer die Experimente durchführen. Eine Durchmischung der Gruppen ist nicht zulässig.

Beaufsichtigung der Studierenden Aufsichtspersonen können sein: Prof. Dr. Jian-Jia Chen, Dr.-Ing. Kuan-Hsun Chen, Niklas Ueter, M.Sc. oder die wissenschaftliche Hilfskraft Marcel Ebbrecht. Es ist zu jeder Zeit sichergestellt, dass mindestens eine Aufsichtsperson anwesend ist um die Einhaltung der Hygieneregularien sicherzustellen und zu verantworten.

Einlass der Studierenden Die Teilnehmer finden sich zum verabredeten Zeitpunkt am Vordereingang des Gebäudes Otto-Hahn-Straße 16 ein. Sie wurden vorher angewiesen, insb. bereits beim Warten außerhalb des Gebäudes den Sicherheitsabstand einzuhalten, eine Maske zu tragen, die allgemeinen Hygieneregeln zu beachten und mit Symptomen

Abbildung 5: Skizze der Strecke und der Position eines Teilnehmers. Die rot makierten Teile sind nicht für die Teilnehmer zugänglich.

Abbildung 6: Die Mittelgänge zwischen den Tischen sind nicht für Teilnehmer zugänglich.

einer Atemwegserkrankung nicht teilzunehmen, siehe Merkblatt für Studierende. Wenn Marcel Ebbrecht die Studierenden beaufsichtigt, erfolgt der Einlass durch Claudia Graute oder eine der o. g. Aussichtspersonen.

Die Teilnehmer werden einzeln eingelassen und aufgefordert, sich in den naheliegenden WCs die Hände zu waschen oder im Eingangsbereich zu desinfizieren (Möglichkeiten zur Desinfektion sind sichergestellt). Sie werden unter Beachtung des Mindestabstands in den Raum U12 zu dem ihnen zugewiesenen Tisch bzw. Platz geführt. Die Aufsicht führende Person betritt als erstes den Raum und öffnen alle Fenster bevor die anderen Teilnehmer den Raum betreten dürfen. Die Teilnehmer werden über die Verhaltensregeln belehrt, siehe Merkblatt für Studierende und der Nachweis über die Belehrung wird bei jedem Treffen dokumentiert.

Dokumentation der Anwesenheitszeiten und Kontaktdaten Zur Erfassung der Anwesenheitszeiten und Kontaktdaten wird das Online-Formular der Fakultät genutzt. `https://dekanat.cs.tu-dortmund.de/corona/anwesenheit`

Nutzung des Equipments

- Jede Person nutzt einen eigenen Laptop am zugeteilten Tisch.
- Gemeinsam genutztes Equipment (Rennwagen und Rennstrecke) wird mit bereitgestellten Gummihandschuhen angefasst.
- Eine Desinfektion des Rennwagens könnte die Elektronik beschädigen (siehe Abbildung).

Auslass der Studierenden Die Studierenden verlassen das Gebäude durch den Ausgang im Untergeschoss.

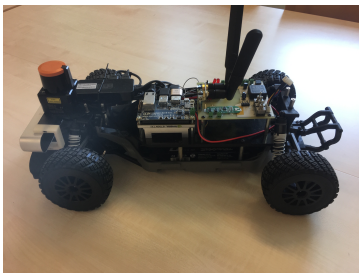


Abbildung 7: Versuchtsfahrzeug in der Seitenansicht.

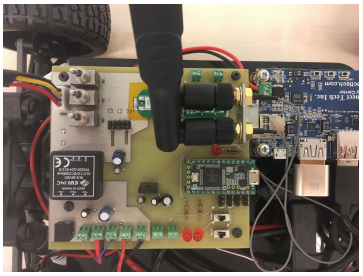


Abbildung 8: Nahfaufnahme der Boardelektronik.

2 Merkblatt für Studierende

Teilnahme Sie dürfen *nicht teilnehmen,* wenn

- Sie selbst an Covid-19 erkrankt sind oder waren, bis die durch das zuständige Gesundheitsamt verordnete Quarantäne beendet ist,
- als Kontaktperson einer an Covid-19 erkrankten Person, bis die durch das zuständige Gesundheitsamt verordnete Quarantäne beendet ist, oder
- nach einem mehrtägigen (mehr als 72 Stunden) Auslandsaufenthalt für 14 Tage nach Ihrer Einreise nach Deutschland gemäß CoronaEinreiseVO NRW.

Bei (auch milden) Symptomen eines Atemwegsinfekts sollen die Studierenden nicht an dem Treffen teilnehmen.

Sie sind nicht zur Teilnahme verpflichtet, insb. wenn Sie zu einer Risikogruppe gehören.

Alle Teilnehmer müssen die allgemeinen Hygieneregeln beachten, insb.

- gründliches Händewaschen
- Hände aus dem Gesicht fernhalten
- Husten und Niesen in ein Taschentuch oder die Armbeuge
- ständig Abstand halten
- keine Begrüßungsrituale

Ihre Vorbereitung Bringen Sie eine textile Mund-Nasen-Bedeckung oder -maske mit. Sie sind verpflichtet, diese zu tragen, solange sie den Ihnen zugeteilten Platz nicht eingenommen haben. Ohne die Mund-Nasen-Bedeckung dürfen Sie nicht teilnehmen.

Bitte seien Sie pünktlich. Achten Sie während der Wartezeiten auf einen Abstand von mindestens 1,5 m zu anderen Personen. Tragen Sie Ihre Mund-Nase-Bedeckung bereits während der Wartezeit.

Folgen Sie den Anweisungen der Aufsichtspersonen.

Verhalten im Raum Wenn Sie den Ihnen zugewiesenen Platz eingenommen haben und belehrt wurden, dürfen Sie Ihre Mund-Nasen-Bedeckung abnehmen. Es wird geraten, sie aufzubehalten.

In der Regel sitzen Sie an dem Ihnen zugewiesenen Tisch. Sie benutzen Ihren eigenen mitgebrachten Laptop.

Die Experimentierfläche betritt grundsätzlich nur ein Teilnehmer gleichzeitig. Die Aufsichtsperson entscheidet, wer die Experimentierfläche betritt. Sollten ausnahmsweise und sehr kurz zwei Teilnehmer die Experimentierfläche

Die Rennwagen und die Rennstrecke werden nur mit geeigneten Handschuhen angefasst. Die Handschuhe werden gestellt.

Folgen Sie den Anweisungen der Aufsichtspersonen.

J Dokumentation toad.sh

J.1 About toad.sh

This CLI tool contains the most important commands for useful tasks. It must be called from inside the ros_ws directory. To use it, copy toad.example to toad.settings and configure that file as needed. Then run

```
toad.sh
```

to get some help about the usage.

J.1.1 Prerequisites

For sure, you need a properly configured system with 18.04 running. Also you need 3D support enabled. For more detailed installation instructions, please refer to the installation manual.

Important: To run the simulation via ssh on the remote system's X session, you'll need gdm3 and a gnome session running (and therefore a, no, ONE gdm-x-session process running), so that the system can guess the DISPLAY to use correctly. Sorry folks, Ubunutu/Wayland/??? massacre. It could work with other display managers if you issue `export DISPLAY=:0` prior starting the simulation.

This was tested with Ubuntu 18.04 from local machine and remote system. If the can't recognize the DISPLAY correctly, please send me the output of `w -sh`, the output of `env` and the output of the script.

J.1.2 Important remark on Gazebo

Depending on your system, date and the attitudes of the ubuntupeople and fanboys, that prefer newer over stable versions, you might get a very new gazebo version and, oh wonder, that won't load the simulation. After hours of debugging the simulation it become clear, that anything other than

```
9.0.0+dfsg5-3ubuntu1+ppa2
```

will crash, or just load sometimes or starts never, a.s.o. If you experience problems with gazebo, DOWNGRADE to the above mentioned version by putting

`http://packages.ros.org/ros/ubuntu bionic/main amd64 Packages`

into your sources.list and do a manual downgrade with aptitude for example.

Important: Please mark gazebo9 package after downgrade as manually hold (= in aptitude) and proceed that for it's dependencies. Otherwise your version will be upgraded on next system upgrade!

J.1.3 toad.sh system

Commands to use on a normal computer or on the build server. Branch is configured in BRANCHBUILD.

`toad.sh system sshkeys [cron]`

This updates the local and current user's authorized_keys file with the keys located in the ssh directory of the repository from the configured branch. Please don' use this on your own machine, because all group member will gain access to your system and your own authorized_keys file is lost. You have been warned.

Arguments:

- `cron` - this removes the confirmation (useful for cron)

`toad.sh system rebuild [cron]`

This removes the build directory and rebuild the whole project from the configured build branch.

Arguments:

- `cron` - this removes the confirmation (useful for cron)

`toad.sh system resetbuild [cron]`

This will reset the working dir to the current state of the configured branch and proceed like rebuild explained before. This reset will lead to a loss of all unpushed changes. You habe been warned.

Arguments:

- `cron` - this removes the confirmation (useful for cron)

`toad.sh system run [nogui,fast,drive,manual,customtrack,videohd,record] [trackname]`

This will run the gazebo simulation on the track which is specified in toad.settings.

Arguments:

- `nogui` - this disables the local gazebo client (more stable and performing better on our systems)
- `fast` - enforce fast mode
- `drive` - enforce autonomous driving
- `manual` - enforce manual driving
- `customtrack` - start simulation with custom track as configured in toad.settings, you may add a track name as 4th parameter to override track in settings
- `record` - record video
- `videohd` - enable 720p mode for cameras

J.1.4 toad.sh car

Commands to use on the car. Branch is configured in BRANCHCAR.

`toad.sh car sshkeys [cron]`

This updates the local and current user's authorized_keys file with the keys located in the ssh directory of the repository from the configured branch. Please don' use this on your own machine, because all group member will gain access to your system and your own authorized_keys file is lost. You have been warned.

Arguments:

- `cron` - this removes the confirmation (useful for cron)

`toad.sh car rebuild [cron]`

This removes the build directory and rebuild the whole project from the configured build branch.

Arguments:

- `cron` - this removes the confirmation (useful for cron)

`toad.sh car resetbuild [cron]`

This will reset the working directory to the current state of the configured branch and proceed like rebuild explained before. This reset will lead to a loss of all unpushed changes. You have been warned.

Arguments:

- `cron` - this removes the confirmation (useful for cron)

`toad.sh car run [drive,manual,record,videohd]`

This will run all the software on the car.

Arguments:

- `drive` - enforce autonomous driving
- `manual` - enforce manual driving
- `record` - record video
- `videohd` - enable 720p mode for cameras

`toad.sh car remote [drive,manual,record,videohd]`

This will run all the software on the car, but without rviz.

Arguments:

- `drive` - enforce autonomous driving
- `manual` - enforce manual driving
- `record` - record video
- `videohd` - enable 720p mode for cameras

`toad.sh car control`

This will run rviz software on your machine connecting to the car.

J.1.5 toad.sh video

This is for converting videos recorded by ros and rviz.

`toad.sh video list`

This lists all video files available for conversion.

`toad.sh video cam DATASET`

Converts camera videos from given DATASET.

`toad.sh video rviz DATASET`

Converts rviz videos from given DATASET.

`toad.sh video stitch DATASET`

Stitches rviz and camera videos from given DATASET.

`toad.sh video move`

Will copy all videos from car to current system.

J.1.6 toad.sh telemetry

This is for creating several reports based on telemetry data. Please ensure proper configuration in toad.settings.

`toad.sh telemetry list`

List available data-sets for reports to include in configuration.

`toad.sh telemetry report`

Create tex/pdf report with plots from collected telemetry data.

`toad.sh telemetry move`

Will copy all telemetry data from car to current system.

K Installationsanleitung

K.1 Installation

This article covers a basic installation manual to install all required software to run the simulation. The installation on the car is nearly equal but got some cornercases that are not covered here. After install and checkout of the project, please read the manual of toad.sh how to run the software.

The following install steps are taken from out travis configuration, you'll find in the base directory of the project.

K.1.1 Basic steps for Ubuntu 18.04 and ROS Melodic

First install some packages

```
sudo apt-get install libeigen3-dev python-wstool libsdl2-dev clang-format-3.9
libyaml-cpp0.5v5 ca-certificates
```

Now run basic installation of ROS and CUDA

```
PATHROS="/opt/ros/melodic/setup.bash"

# install keys and package sources
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" \
  > /etc/apt/sources.list.d/ros-latest.list'
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' \
  --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable \
  `lsb_release -cs` main" > /etc/apt/sources.list.d/gazebo-stable.list'
wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
sudo apt-get update -qq
sudo apt-get upgrade -y

# install os packages
sudo apt-get install -y python-catkin-tools libsdl2-dev ros-melodic-ackermann-msgs \
  ros-melodic-serial ros-melodic-desktop-full gazebo9 libgazebo9-dev \
  ros-melodic-gazebo-ros-control mplayer ffmpeg mencoder netcat \
  ros-melodic-rviz-imu-plugin ros-melodic-depthimage-to-laserscan \
  ros-melodic-jsk-rviz-plugins
sudo apt-get install -y libignition-math2-dev
sudo apt-get install -y python-rosinstall python-rosinstall-generator \
  python-wstool build-essential python-rosdep
```

```
# install ros, reset pip and install dependencies
sudo rosdep init
sudo rosdep update
source $PATHROS
sudo python -m pip uninstall -y pip
sudo apt-get install -y python-pip
sudo apt-get install -y libsdl2-dev clang-format python-pyqtgraph
sudo python2 -m pip install --upgrade pip --force
sudo python2 -m pip install --no-cache-dir torch autopep8 cython circle-fit
sudo pip install autopep8
wstool init
wstool up

# we install cuda
source /etc/lsb-release
sudo wget https://developer.download.nvidia.com/compute/cuda/repos/\
  ubuntu1804/x86_64/cuda-ubuntu1804.pin -O /etc/apt/preferences.d/cuda-repository-pin-600
sudo apt-key adv --fetch-keys https://developer.download.nvidia.com/\
  compute/cuda/repos/ubuntu1804/x86_64/7fa2af80.pub
sudo add-apt-repository "deb http://developer.download.nvidia.com/\
  compute/cuda/repos/ubuntu1804/x86_64/ /"
sudo apt update -qq
sudo apt install -y cuda-10-2

# install zed
CUDA_HOME=/usr/local/cuda-10.2
LD_LIBRARY_PATH=${CUDA_HOME}/lib64:${LD_LIBRARY_PATH}
PATH=${CUDA_HOME}/bin:${PATH}
wget https://download.stereolabs.com/zedsdk/3.1/cu102/ubuntu18 \
 -O /tmp/zed_sdk.run
chmod +x /tmp/zed_sdk.run
/tmp/zed_sdk.run -- silent skip_tools

# setip a lib
cd ~/ARPG && git clone http://github.com/kctess5/range_libc
cd ~/ARPG/range_libc/pywrapper && ./compile.sh

# clean up
sudo rosdep fix-permissions
sudo apt-get -y autoremove
sudo apt-get clean

echo
echo
echo "Base installation done, now proceed with clone.sh"
```

```
echo
```

Now clone and enter directory. Then run

```
GITREPO="https://github.com/arpg-sophisticated/ar-tu-do.git"
GITBRANCH="development"

mkdir -p ~/ARPG/
cd ~/ARPG && git clone $GITREPO
cd ~/ARPG/ar-tu-do && git submodule init
cd ~/ARPG/ar-tu-do && git submodule update --recursive
cd ~/ARPG/ar-tu-do && rosdep update
cd ~/ARPG/ar-tu-do/ros_ws && rosdep install -y --from-paths \
  ./src --ignore-src --rosdistro melodic
cd ~/ARPG/ar-tu-do/ && git checkout $GITBRANCH
cd ~/ARPG/ar-tu-do/ && git pull
cd ~/ARPG/ar-tu-do/ros_ws && catkin_make
cd ~/ARPG/ar-tu-do/ros_ws && cp -a toad.example toad.settings

echo
echo
echo "Setup of repository done, now configure toad.settings and have phun!"
echo
```

L Ergebnisse Simulation WF2 vs. WF5 (v0.42g1)

M Auswertung Wallfollowing 2 - 1000

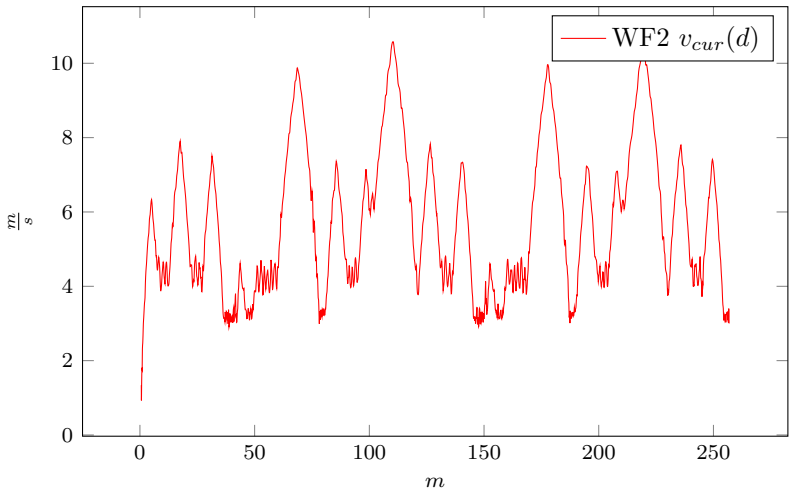


Abbildung 44:
WF2: Geschwindicgkeit v nach Distanz d - 1000 Werte

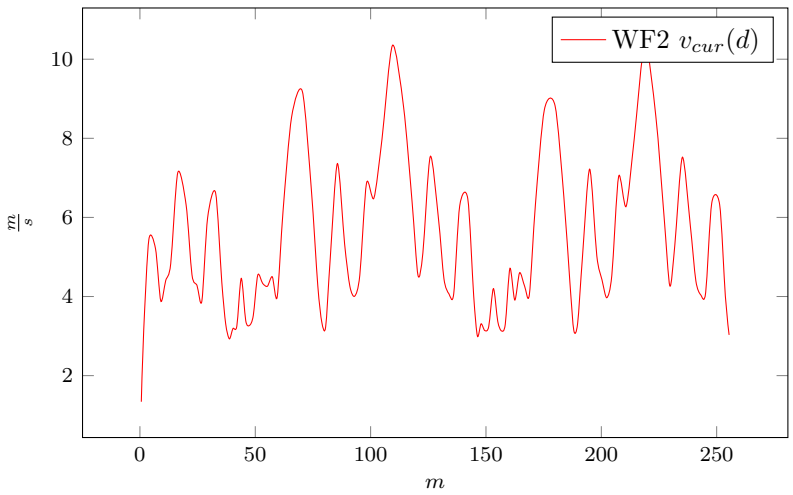


Abbildung 45:
WF2: Geschwindigkeit v nach Distanz d - 1000 Werte, Glättung 10

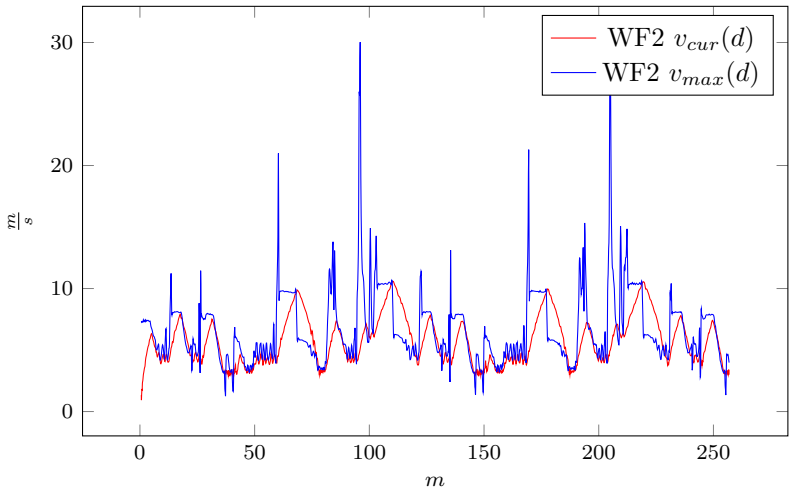


Abbildung 46:
WF2: Geschwindigkeit v nach Distanz d - 1000 Werte

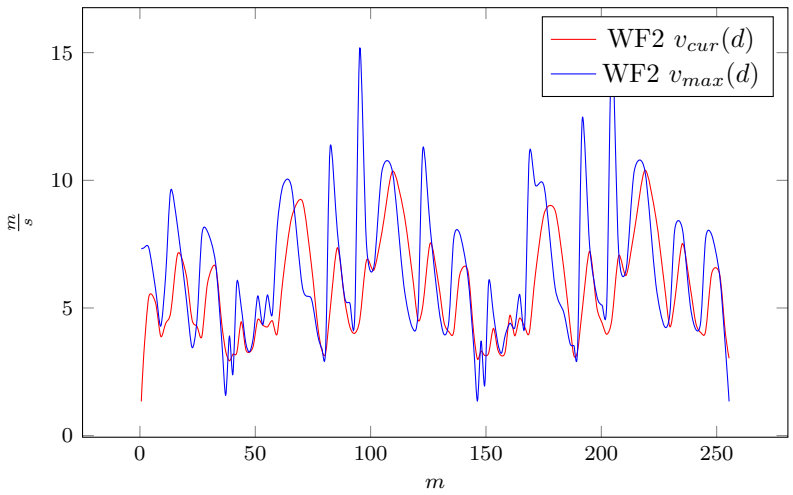


Abbildung 47:
WF2: Geschwindigkeit v nach Distanz d - 1000 Werte, Glättung 10

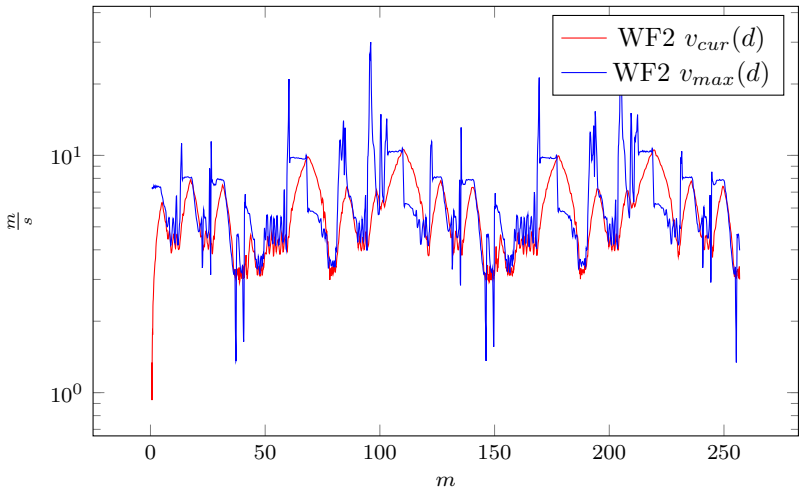


Abbildung 48:
WF2: Geschwindigkeit v nach Distanz d - 1000 Werte (logarithmisch)

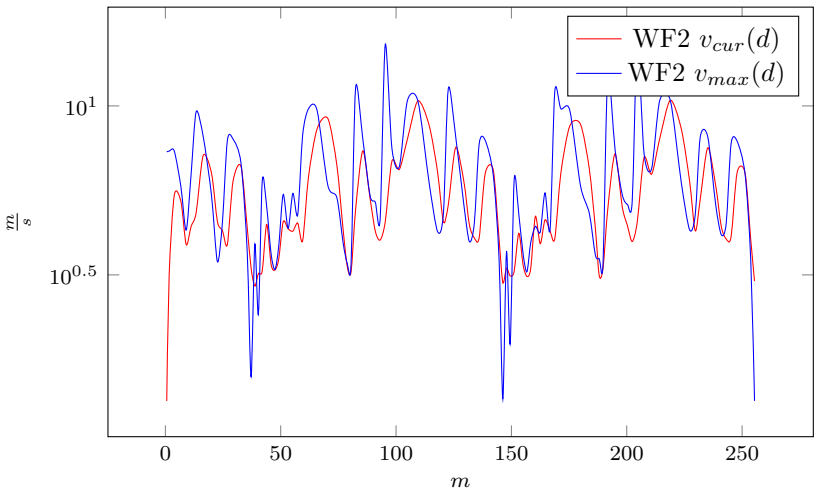


Abbildung 49:
WF2: Geschwindigkeit v nach Distanz d - 1000 Werte (logarithmisch), Glättung 10

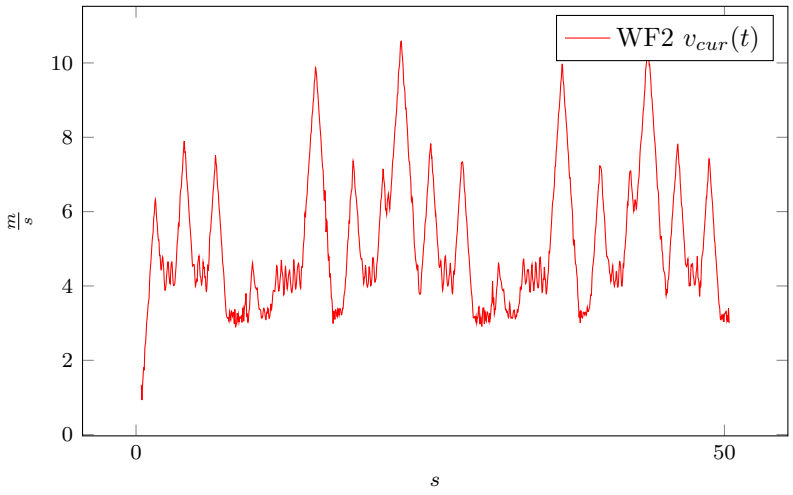


Abbildung 50:
WF2: Geschwindigkeit v nach Zeit t - 1000 Werte

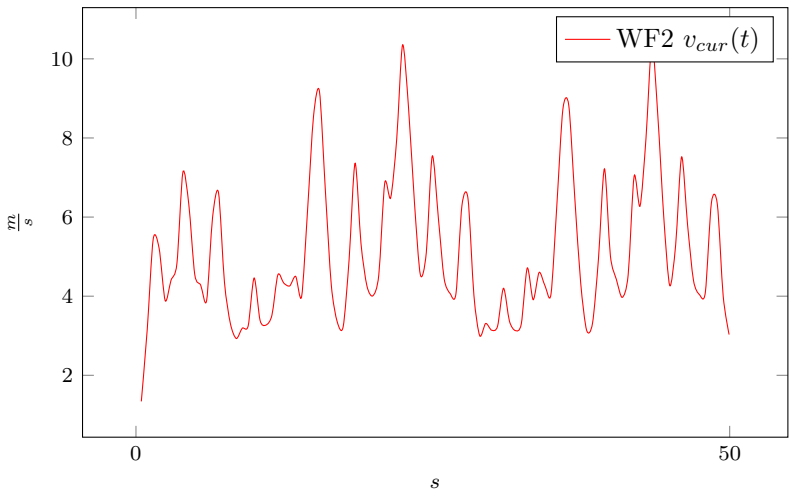


Abbildung 51:
WF2: Geschwindigkeit v nach Zeit t - 1000 Werte, Glättung 10

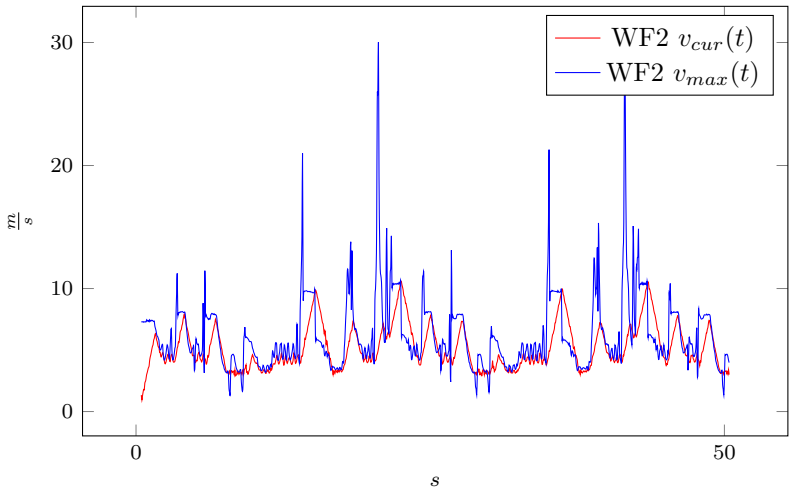


Abbildung 52:
WF2: Geschwindigkeit v nach Zeit t - 1000 Werte

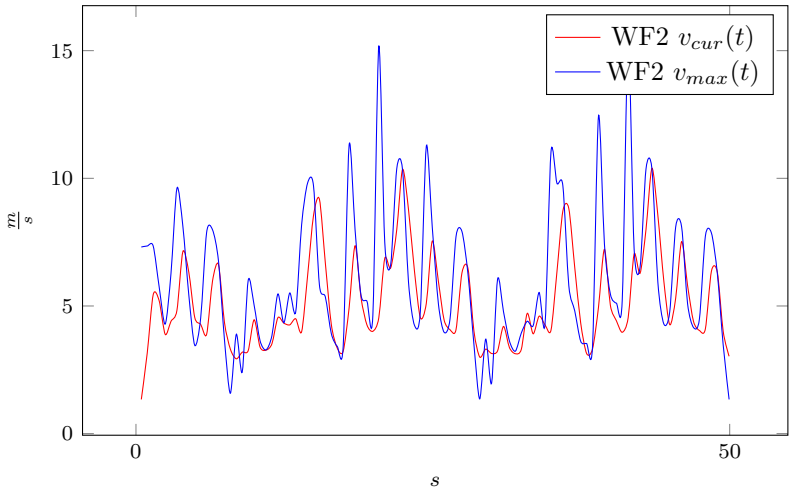


Abbildung 53:
WF2: Geschwindigkeit v nach Zeit t - 1000 Werte, Glättung 10

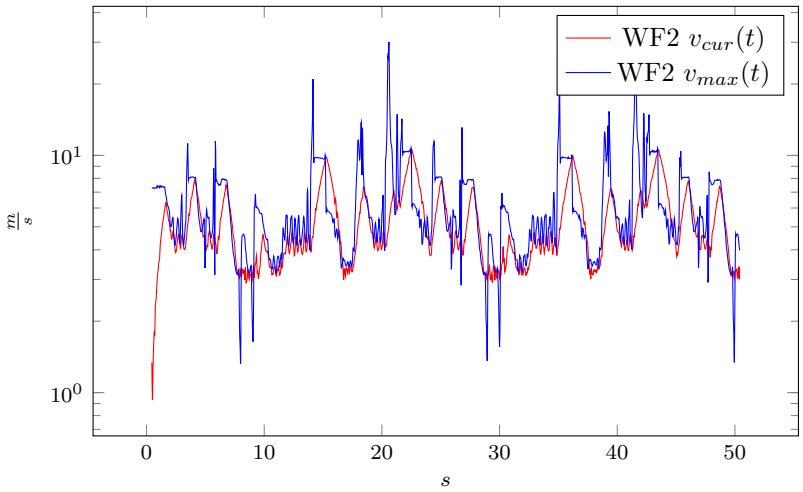


Abbildung 54:
WF2: Geschwindigkeit v nach Zeit t - 1000 Werte (logarithmisch)

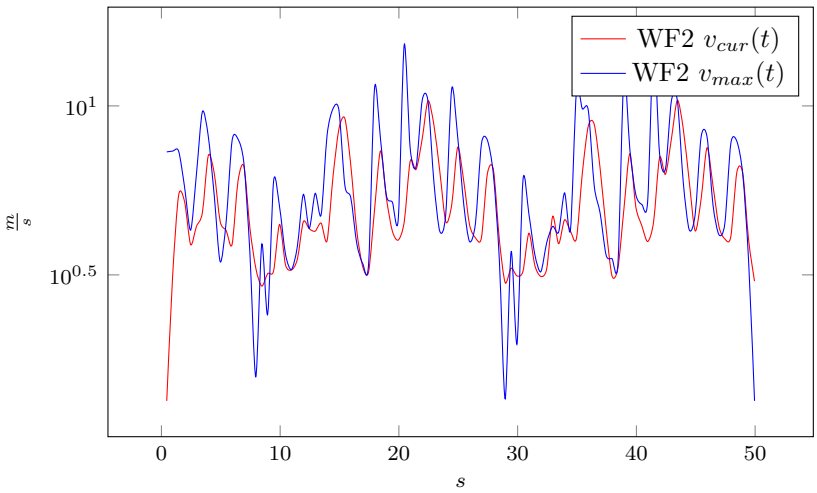


Abbildung 55:
WF2: Geschwindigkeit v nach Zeit t - 1000 Werte, Glättung 10 (logarithmisch)

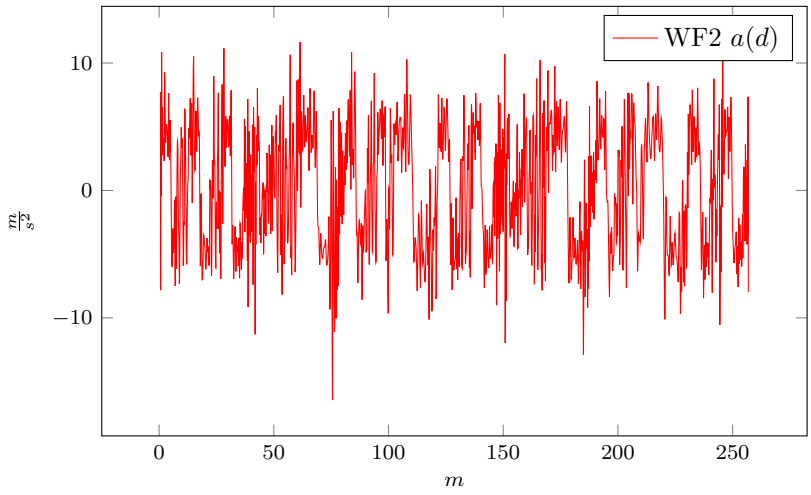


Abbildung 56:
WF2: Beschleunigung a nach Distanz d - 1000 Werte

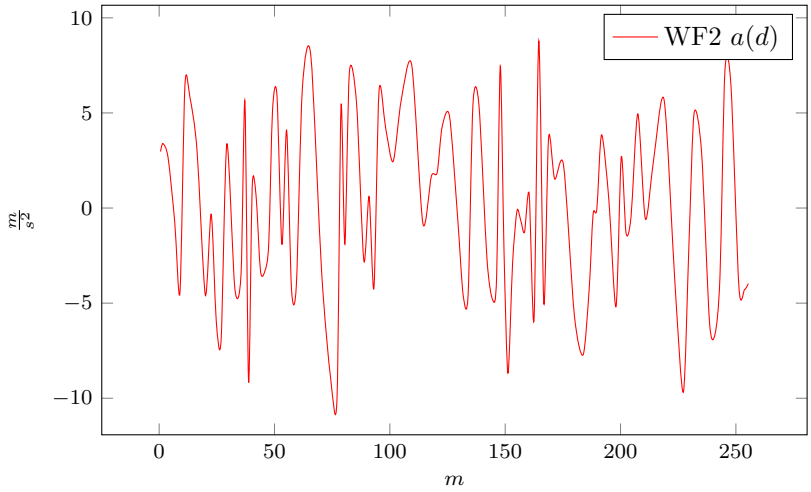


Abbildung 57:
WF2: Beschleunigung a nach Distanz d - 1000 Werte, Glättung 10

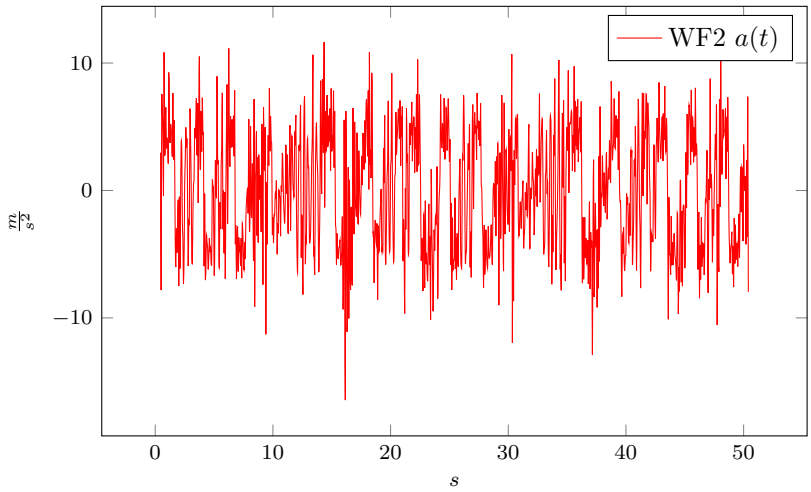


Abbildung 58:
WF2: Beschleunigung a nach Zeit t - 1000 Werte

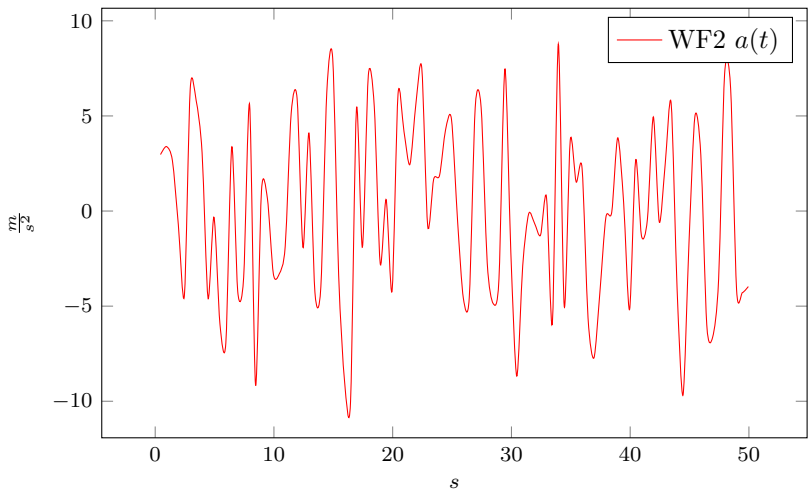


Abbildung 59:
WF2: Beschleunigung a nach Zeit t - 1000 Werte, Glättung 10

N Auswertung Wallfollowing 5 - 1000

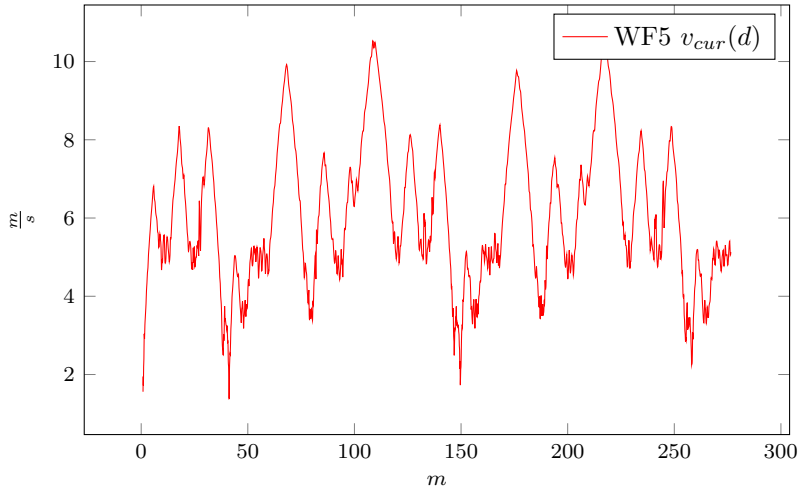


Abbildung 60:
WF5: Geschwindicgkeit v nach Distanz d - 1000 Werte

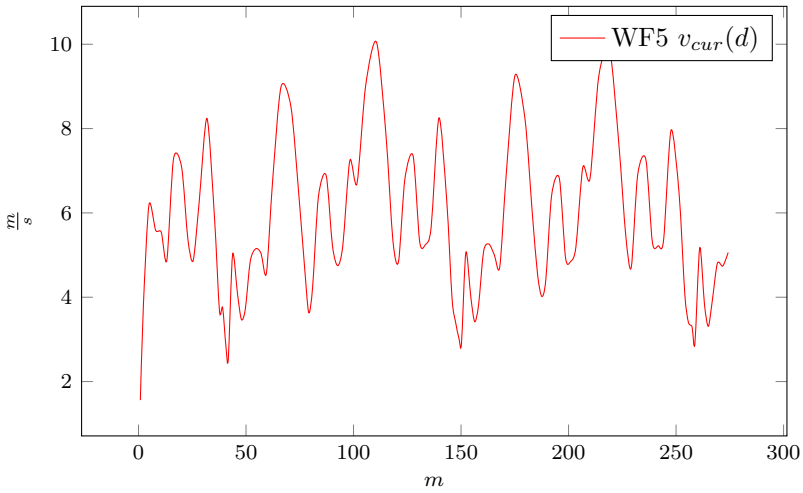


Abbildung 61:
WF5: Geschwindigkeit v nach Distanz d - 1000 Werte, Glättung 10

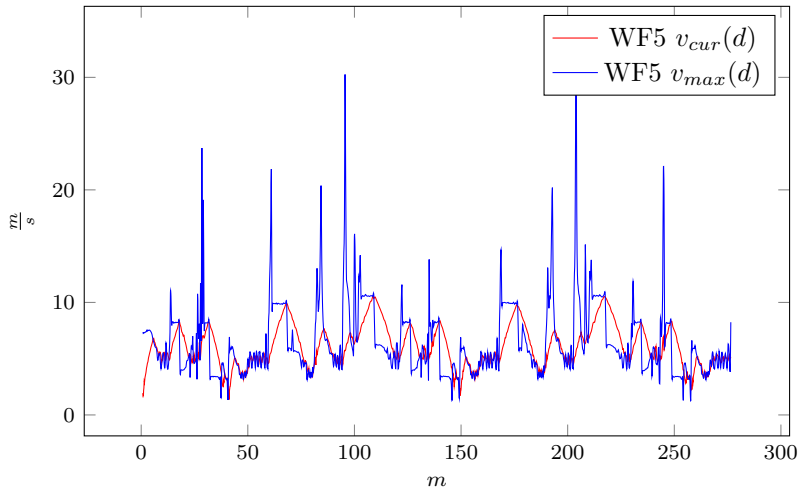


Abbildung 62:
WF5: Geschwindigkeit v nach Distanz d - 1000 Werte

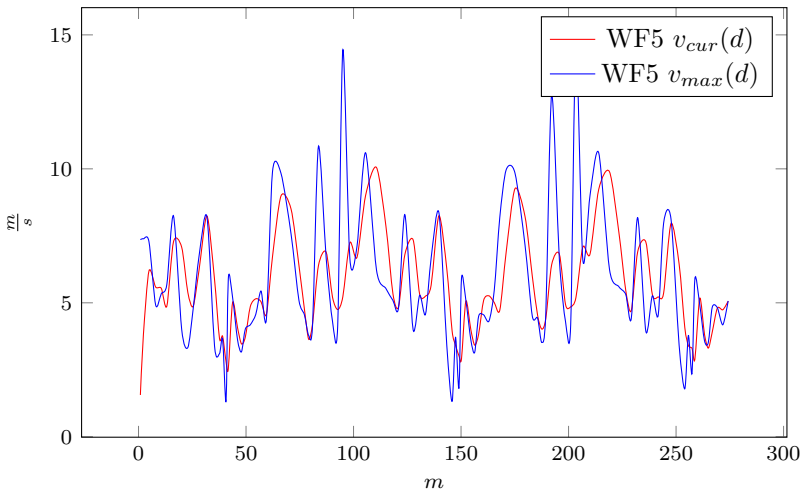


Abbildung 63:
WF5: Geschwindigkeit v nach Distanz d - 1000 Werte, Glättung 10

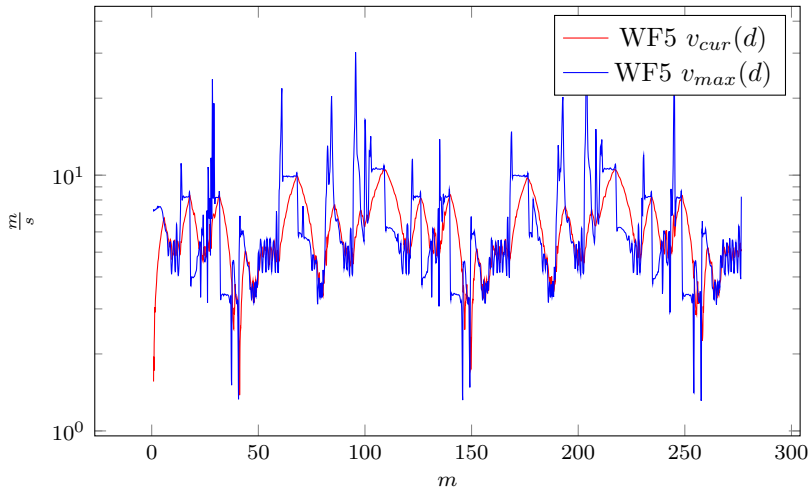


Abbildung 64:
WF5: Geschwindigkeit v nach Distanz d - 1000 Werte (logarithmisch)

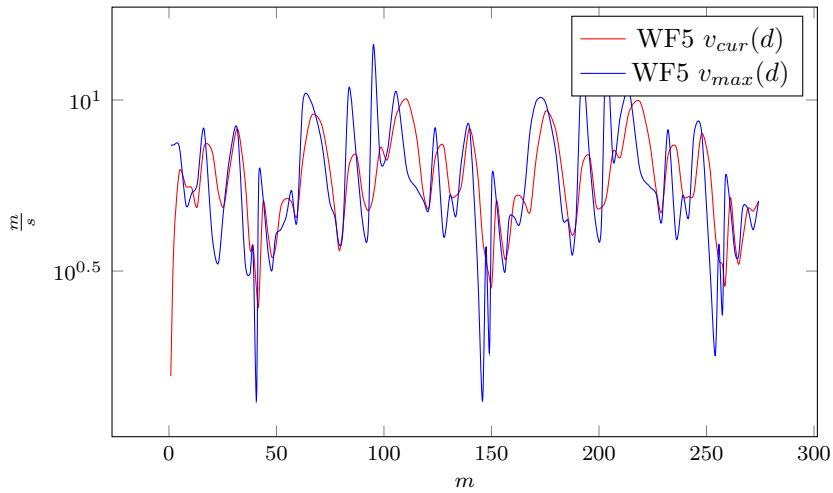


Abbildung 65:
WF5: Geschwindigkeit v nach Distanz d - 1000 Werte (logarithmisch), Glättung 10

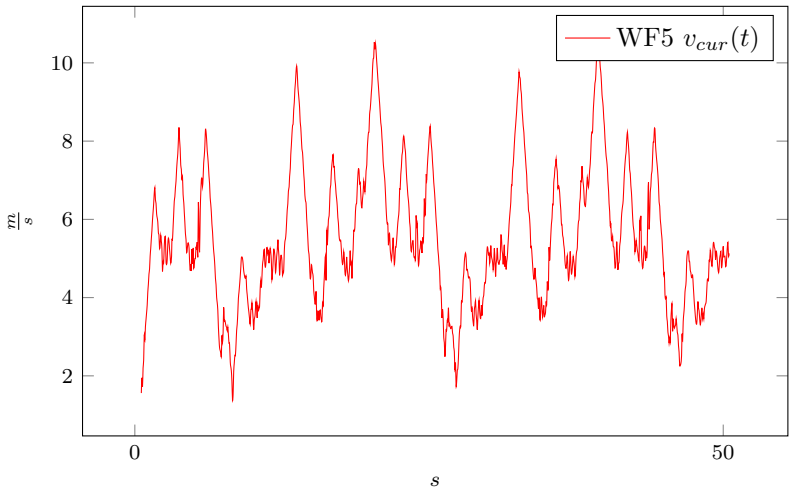


Abbildung 66:
WF5: Geschwindigkeit v nach Zeit t - 1000 Werte

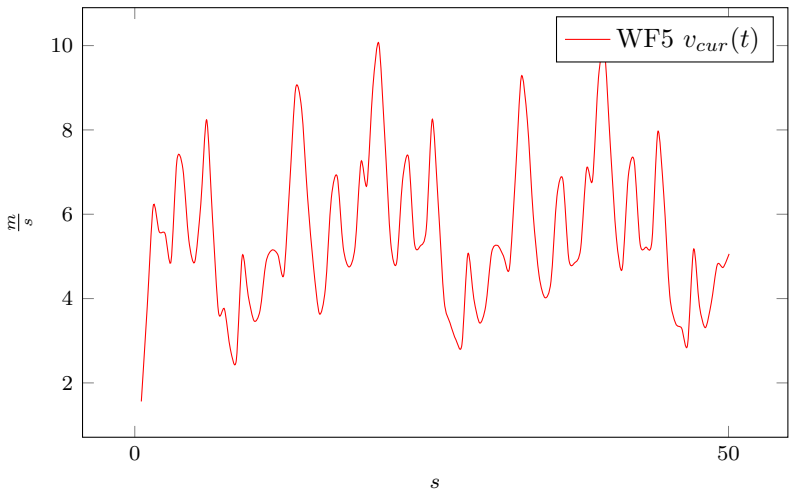


Abbildung 67:
WF5: Geschwindigkeit v nach Zeit t - 1000 Werte, Glättung 10

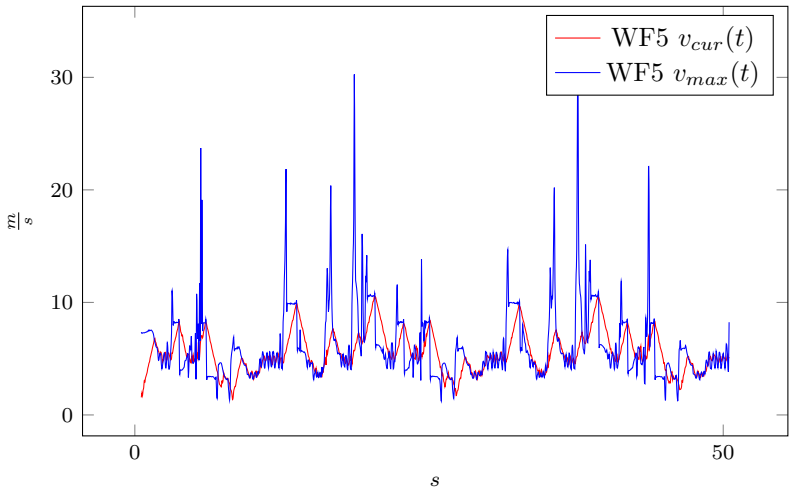


Abbildung 68:
WF5: Geschwindigkeit v nach Zeit t - 1000 Werte

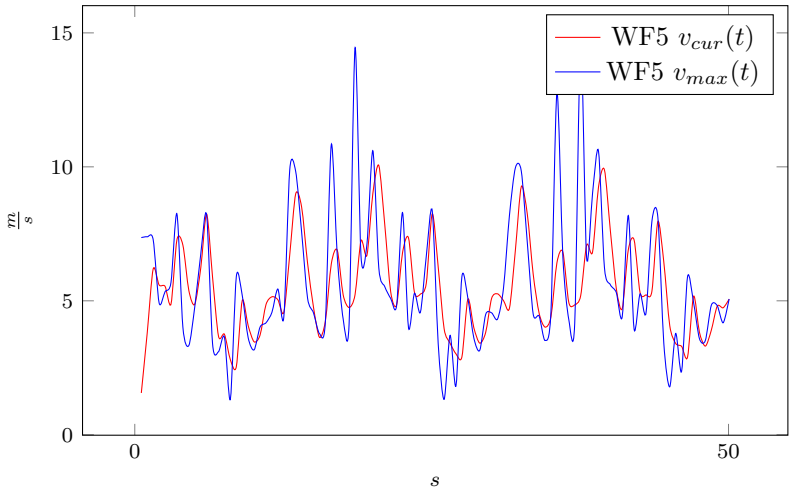


Abbildung 69:
WF5: Geschwindigkeit v nach Zeit t - 1000 Werte, Glättung 10

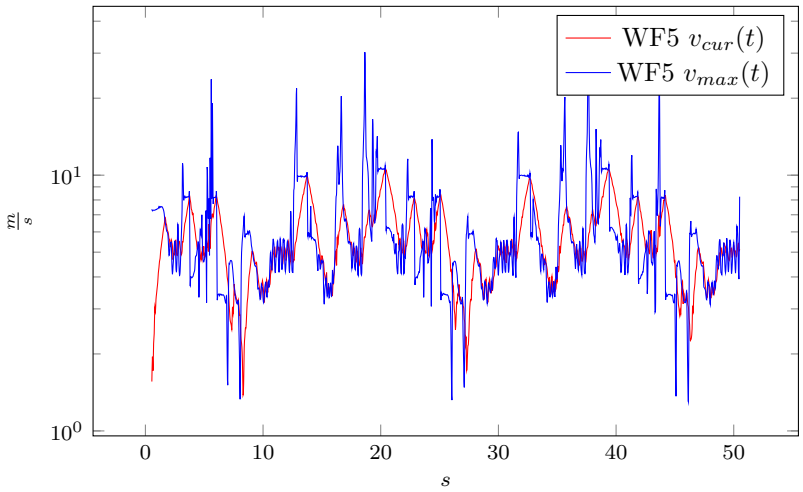


Abbildung 70:
WF5: Geschwindigkeit v nach Zeit t - 1000 Werte (logarithmisch)

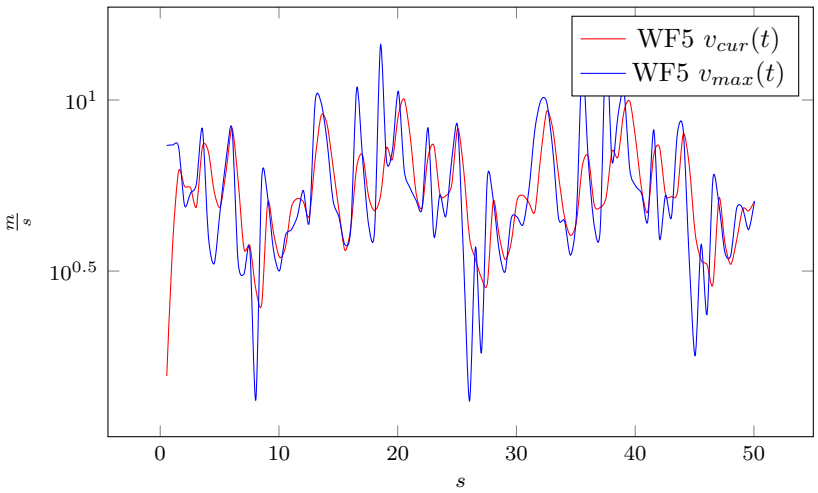


Abbildung 71:
WF5: Geschwindigkeit v nach Zeit t - 1000 Werte, Glättung 10 (logarithmisch)

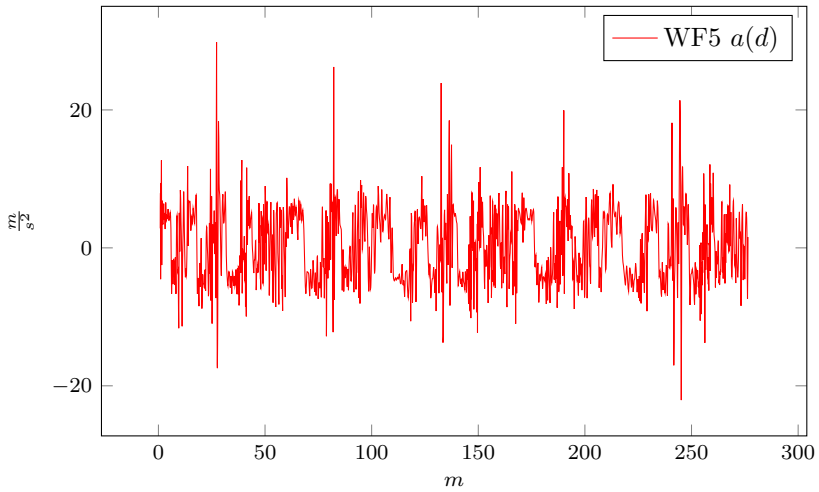


Abbildung 72:
WF5: Beschleunigung a nach Distanz d - 1000 Werte

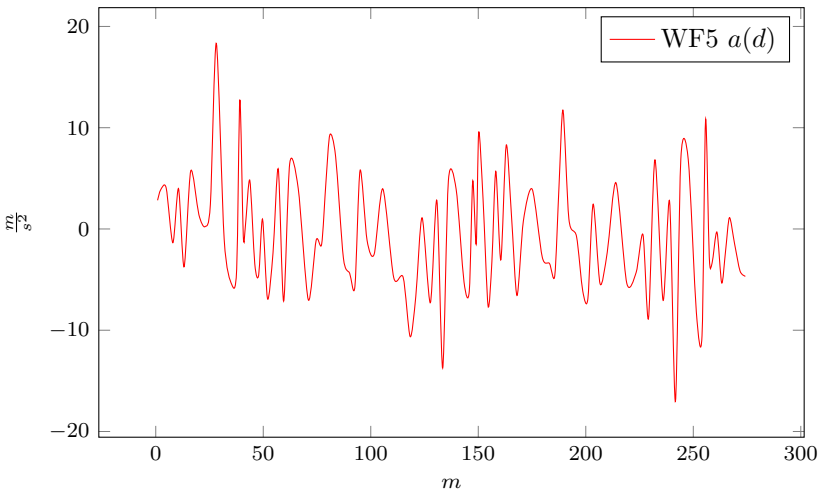


Abbildung 73:
WF5: Beschleunigung a nach Distanz d - 1000 Werte, Glättung 10

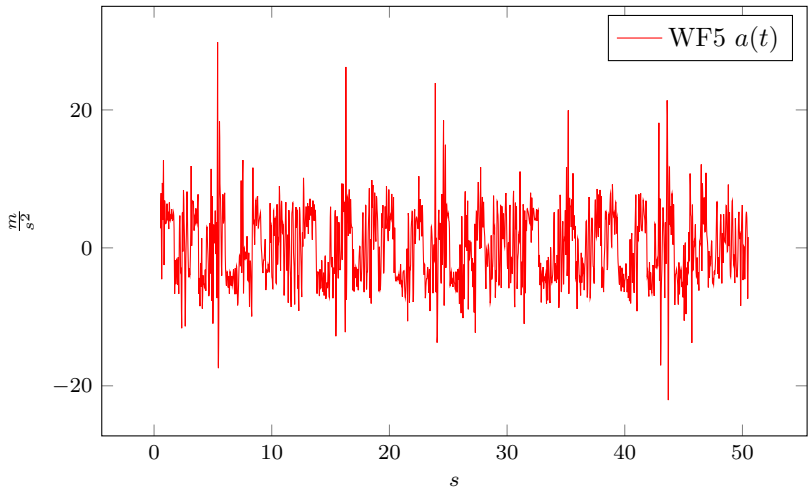


Abbildung 74:
WF5: Beschleunigung a nach Zeit t - 1000 Werte

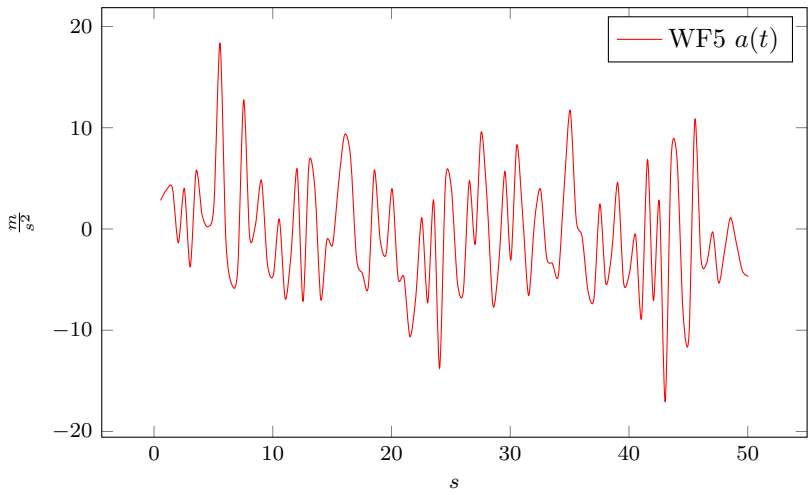


Abbildung 75:
WF5: Beschleunigung a nach Zeit t - 1000 Werte, Glättung 10

O Vergleich - 1000

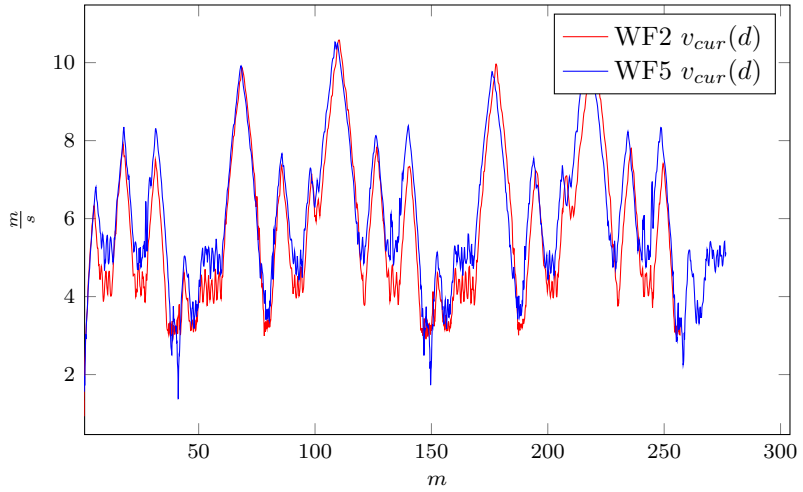


Abbildung 76:
WF5: Geschwindigkeit v nach Distanz d - 1000 Schritte

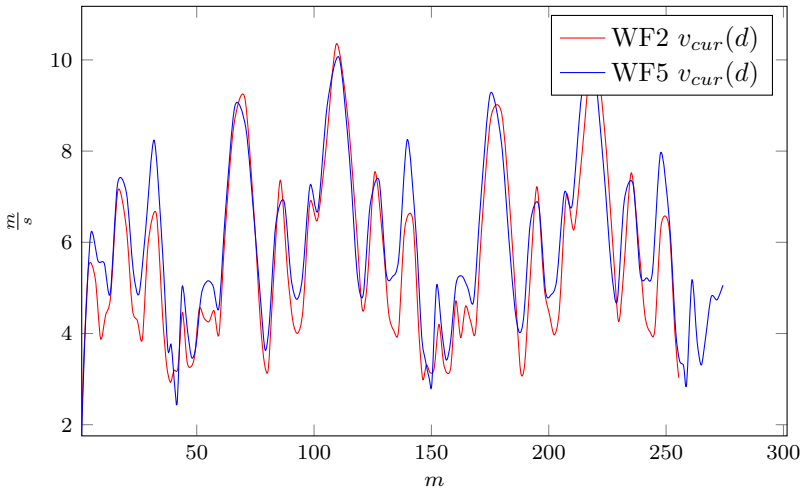


Abbildung 77:
WF5: Geschwindigkeit v nach Distanz d - 1000 Werte, Glättung 10

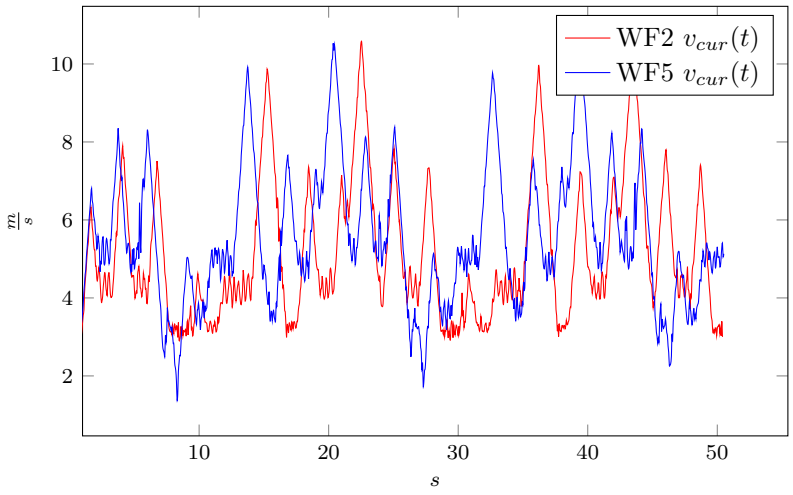


Abbildung 78:
Vergleich: Geschwindigkeit v nach Zeit t - 1000 Schritte

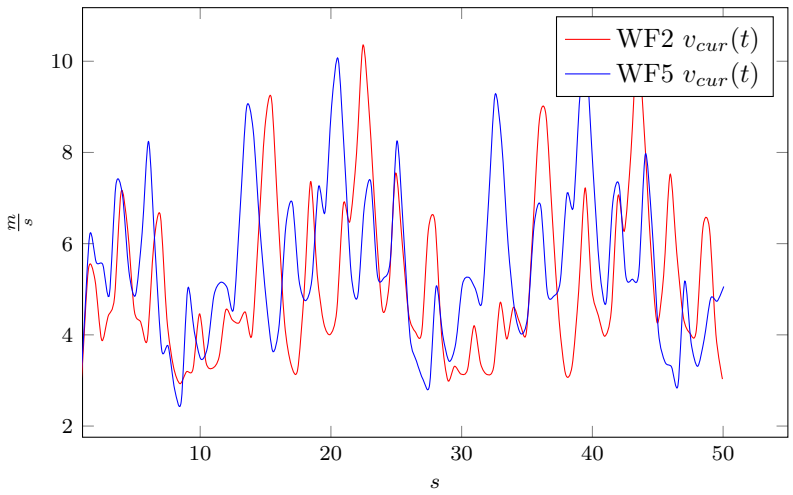


Abbildung 79:
Vergleich: Geschwindigkeit v nach Zeit t - 1000 Werte, Glättung 10

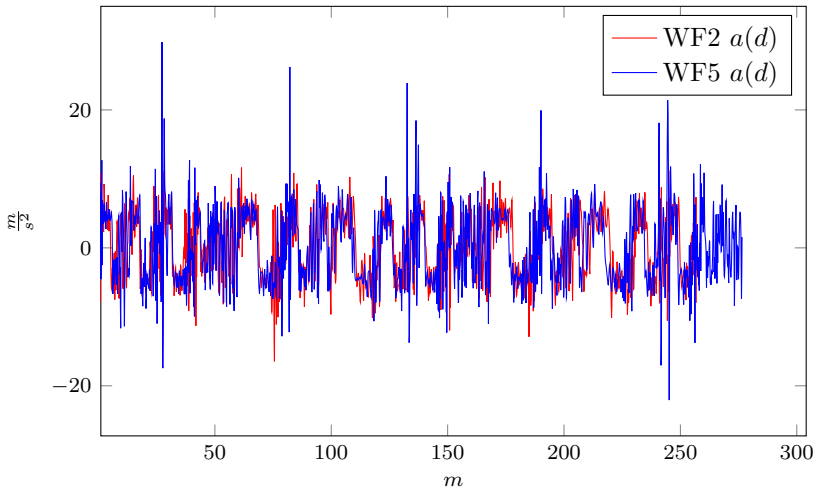


Abbildung 80:
Vergleich: Beschleunigung a nach Distanz d - 1000 Schritte

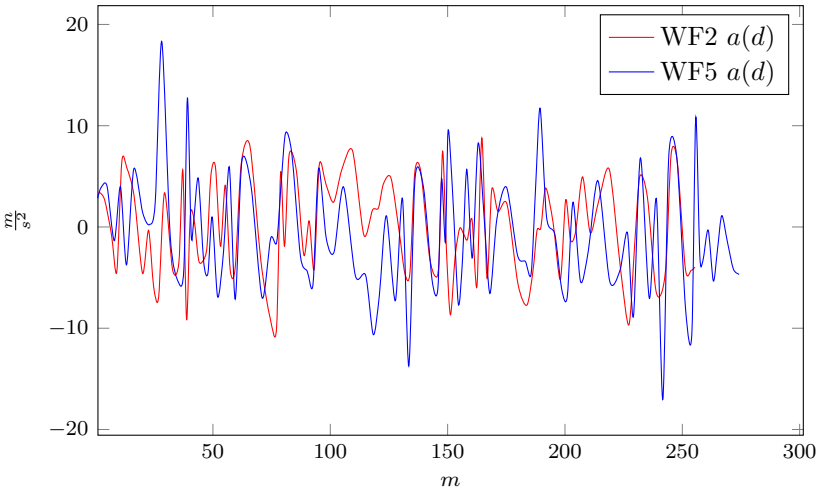


Abbildung 81:
Vergleich: Beschleunigung a nach Distanz d - 1000 Werte, Glättung 10

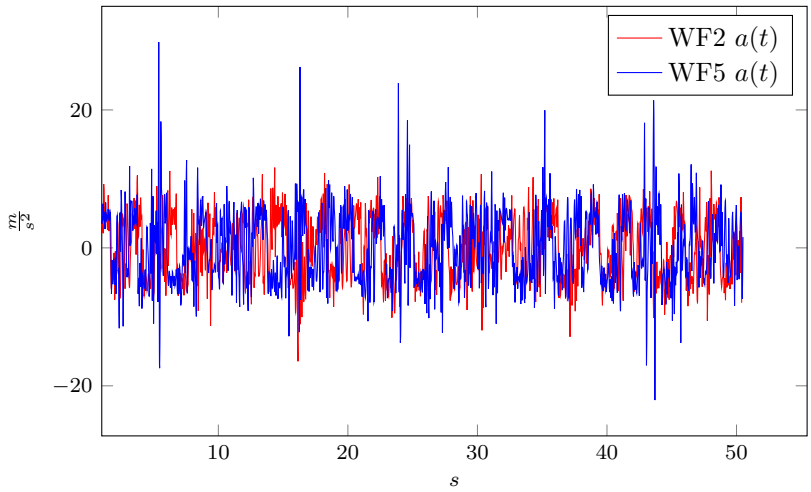


Abbildung 82:
Vergleich: Beschleunigung a nach Zeit t - 1000 Schritte

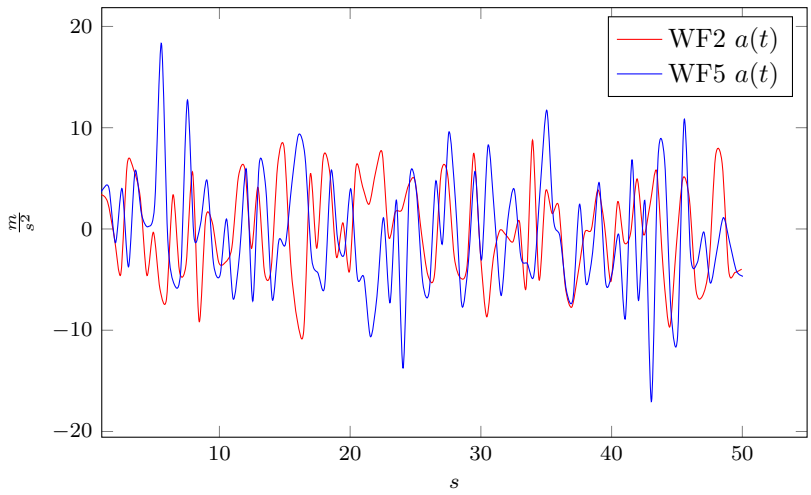


Abbildung 83:
Vergleich: Beschleunigung a nach Zeit t - 1000 Werte, Glättung 10

P Ergebnisse Simulation Q-Learning (v0.42q1)

P.1 Auswertung Q-Learning - 1000

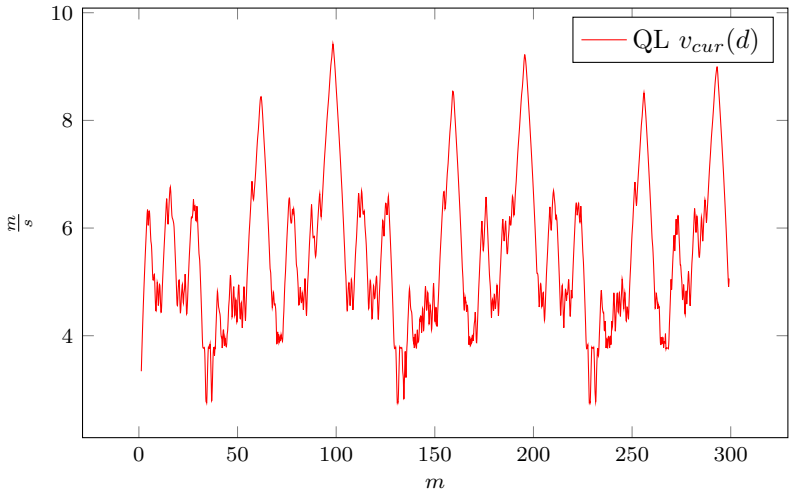


Abbildung 84:
QL: Geschwindigkeit v nach Distanz d - 1000 Werte

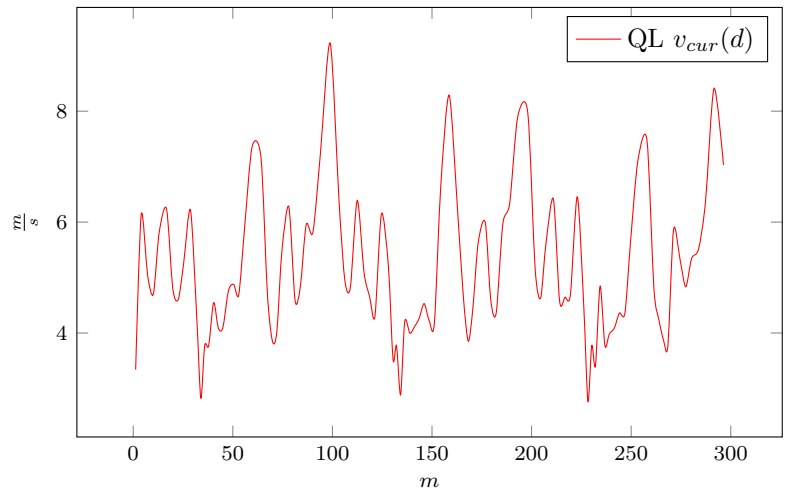


Abbildung 85:
QL: Geschwindigkeit v nach Distanz d - 1000 Werte, Glättung 10

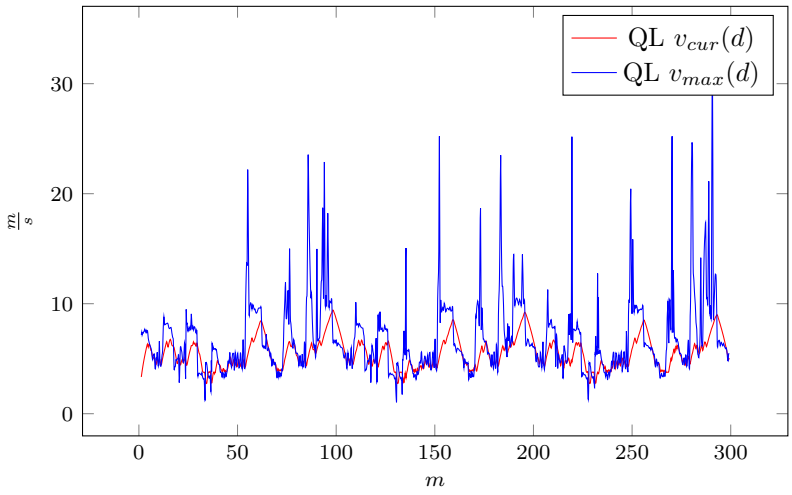


Abbildung 86:
QL: Geschwindigkeit v nach Distanz d - 1000 Werte

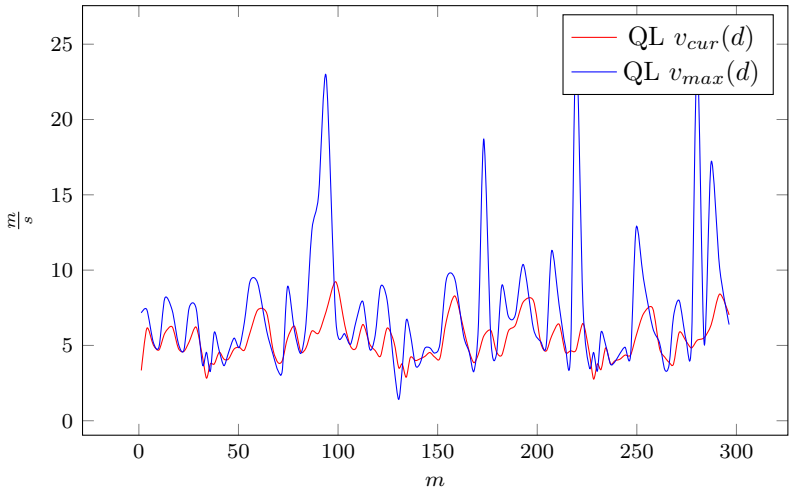


Abbildung 87:
QL: Geschwindigkeit v nach Distanz d - 1000 Werte, Glättung 10

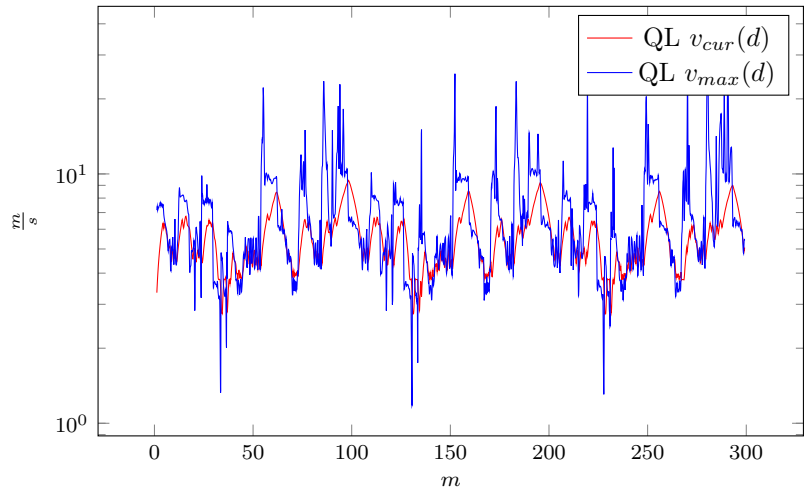


Abbildung 88:
QL: Geschwindigkeit v nach Distanz d - 1000 Werte (logarithmisch)

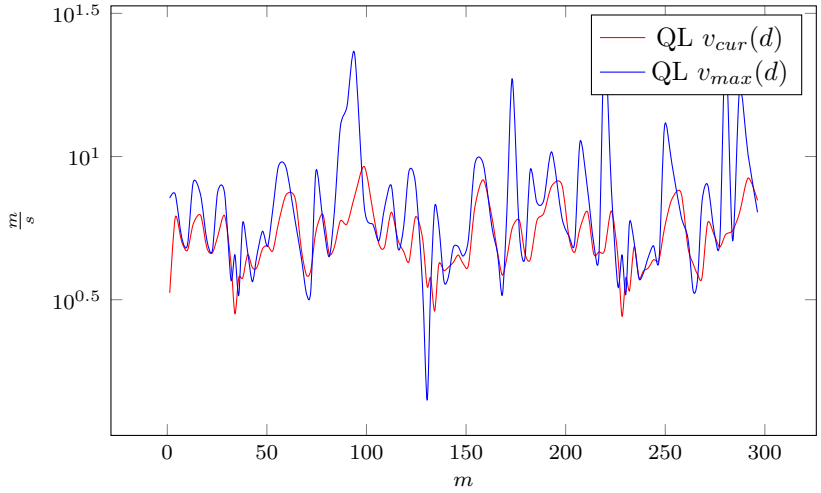


Abbildung 89:
QL: Geschwindigkeit v nach Distanz d - 1000 Werte (logarithmisch), Glättung 10

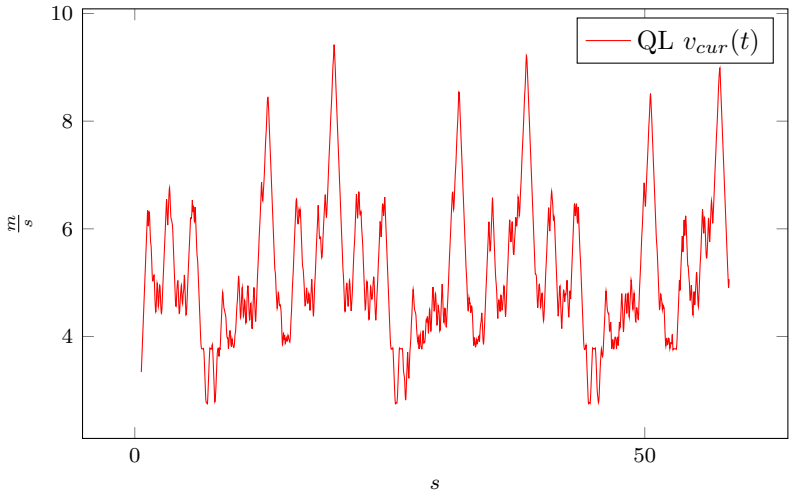


Abbildung 90:
QL: Geschwindigkeit v nach Zeit t - 1000 Werte

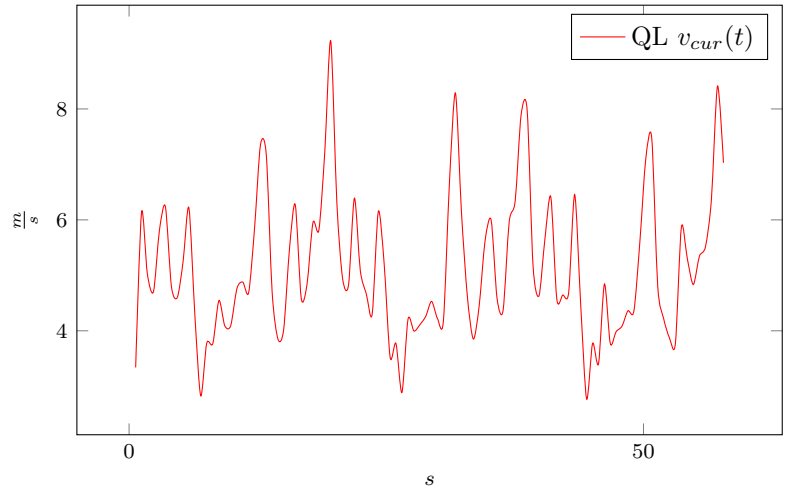


Abbildung 91:
QL: Geschwindigkeit v nach Zeit t - 1000 Werte, Glättung 10

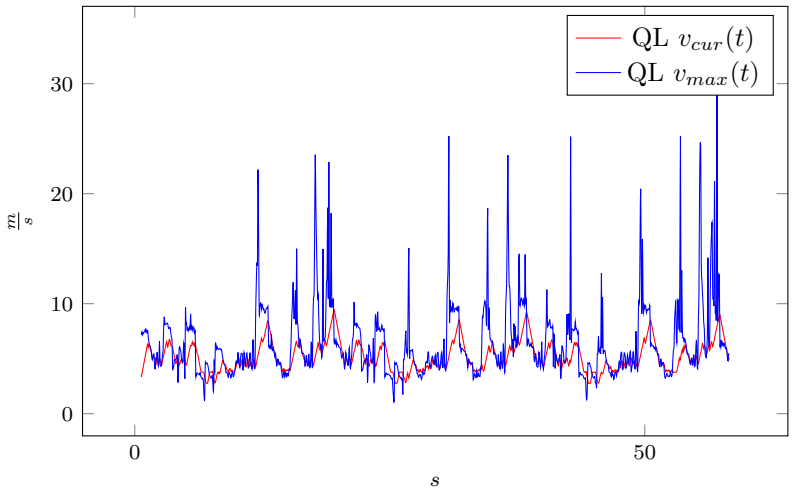


Abbildung 92:
QL: Geschwindigkeit v nach Zeit t - 1000 Werte

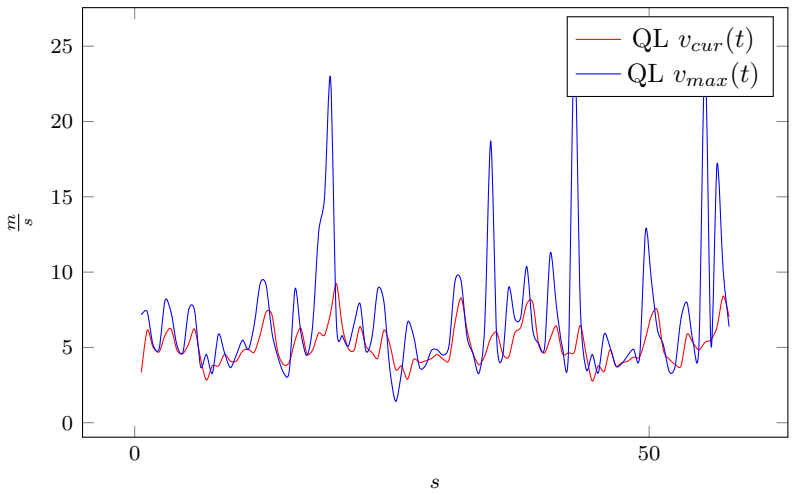


Abbildung 93:
QL: Geschwindigkeit v nach Zeit t - 1000 Werte, Glättung 10

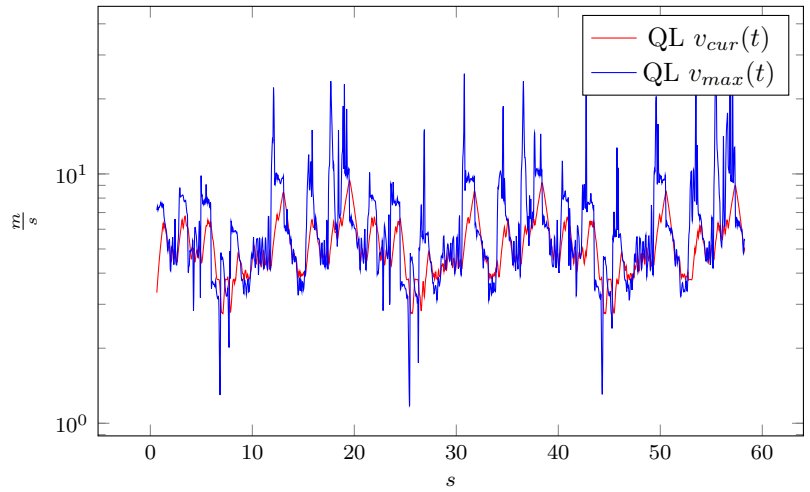


Abbildung 94:
QL: Geschwindigkeit v nach Zeit t - 1000 Werte (logarithmisch)

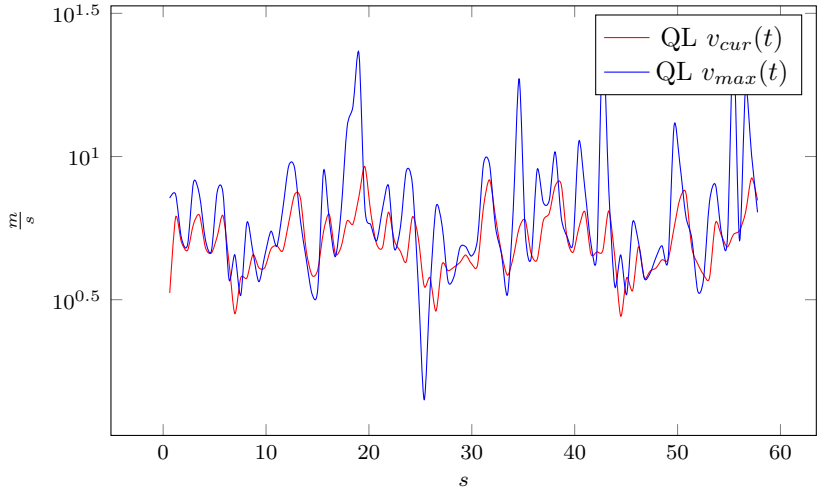


Abbildung 95:
QL: Geschwindigkeit v nach Zeit t - 1000 Werte, Glättung 10 (logarithmisch)

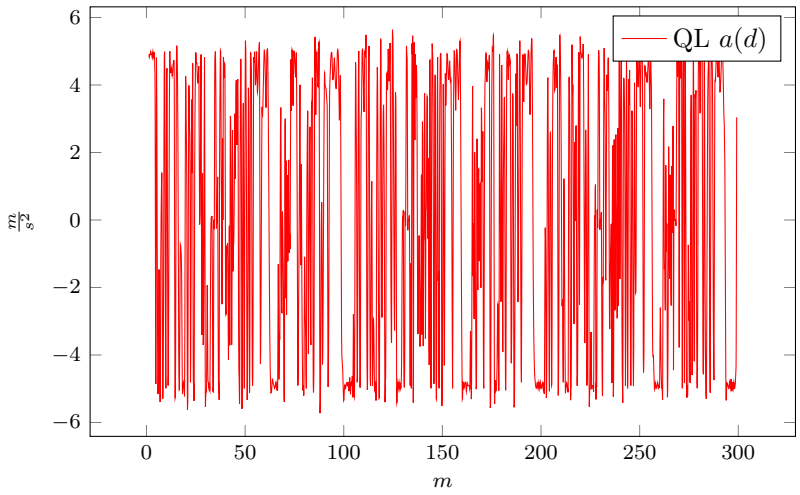


Abbildung 96:
QL: Beschleunigung a nach Distanz d - 1000 Werte

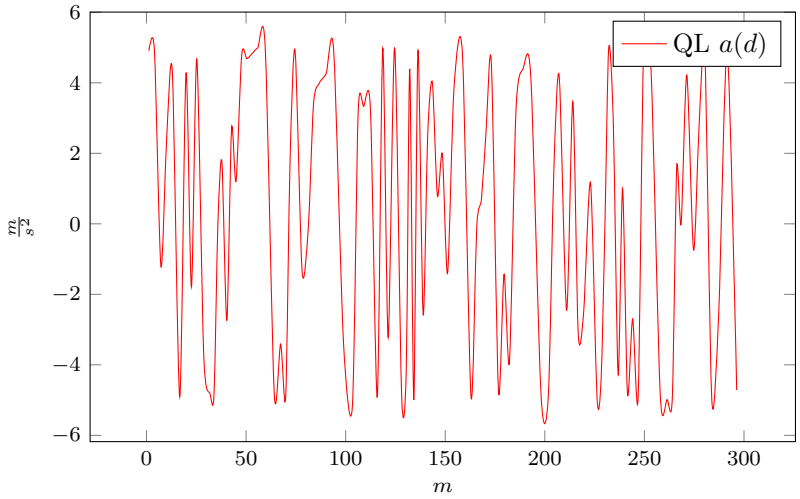


Abbildung 97:
QL: Beschleunigung a nach Distanz d - 1000 Werte, Glättung 10

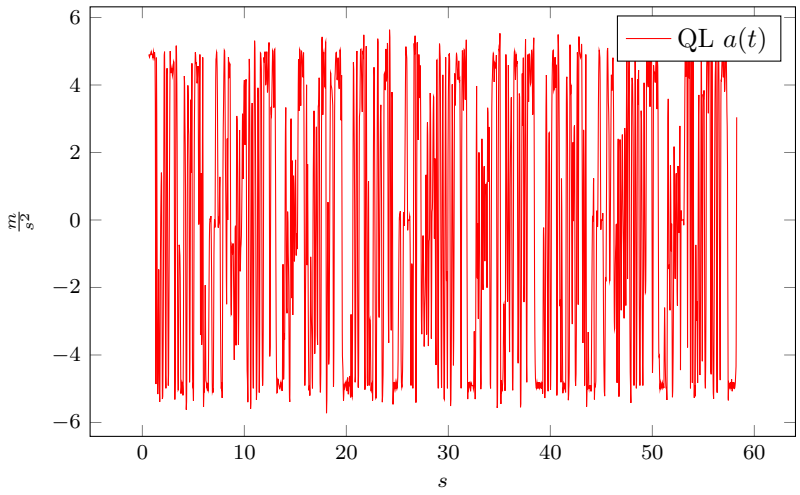


Abbildung 98:
QL: Beschleunigung a nach Zeit t - 1000 Werte

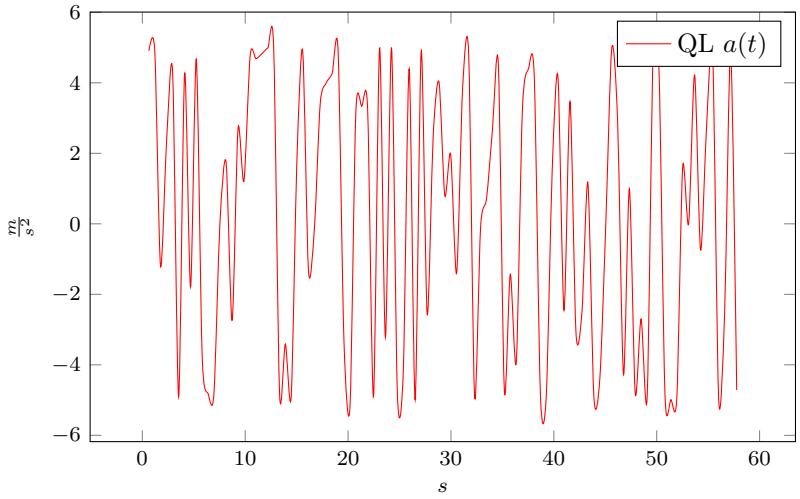


Abbildung 99:
QL: Beschleunigung a nach Zeit t - 1000 Werte, Glättung 10

Q Ergebnisse Test WF2 (v0.42r1) vs. WF5 (v0.42r2)

R Auswertung Wallfollowing 2 - 500

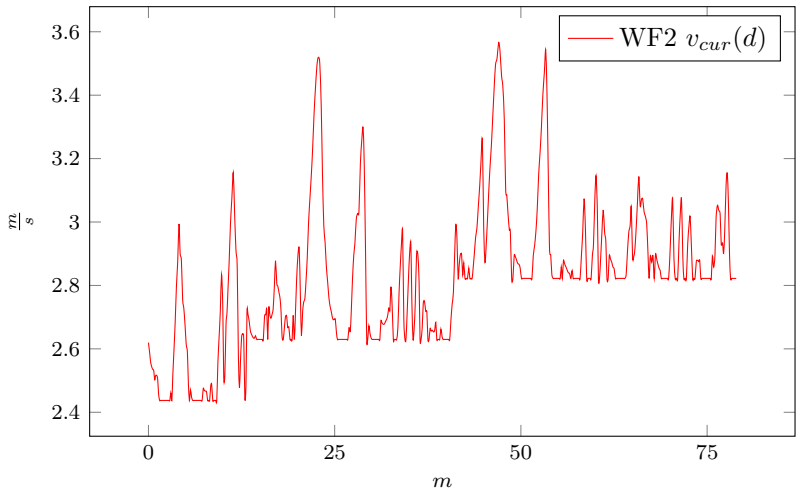


Abbildung 100:
WF2: Geschwindigkeit v nach Distanz d - 500 Werte

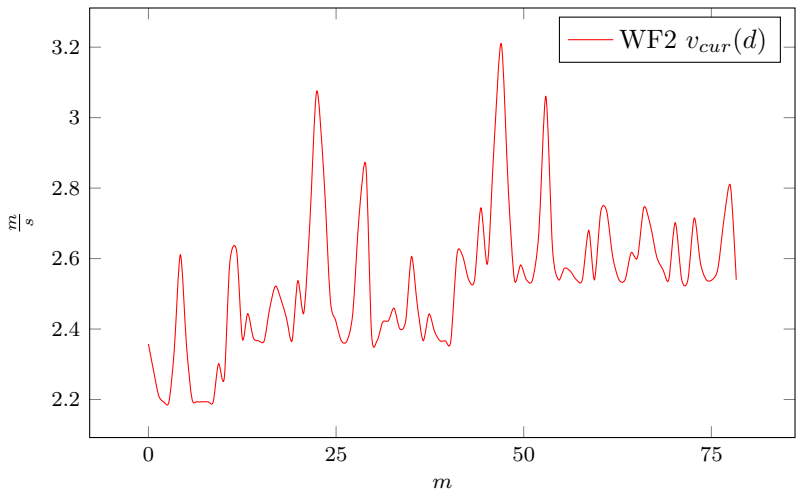


Abbildung 101:
WF2: Geschwindigkeit v nach Distanz d - 500 Werte, Glättung 5

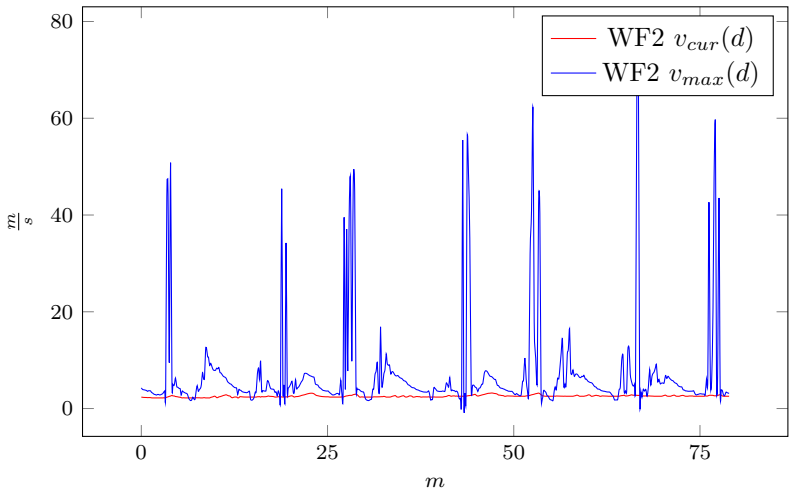


Abbildung 102:
WF2: Geschwindigkeit v nach Distanz d - 500 Werte

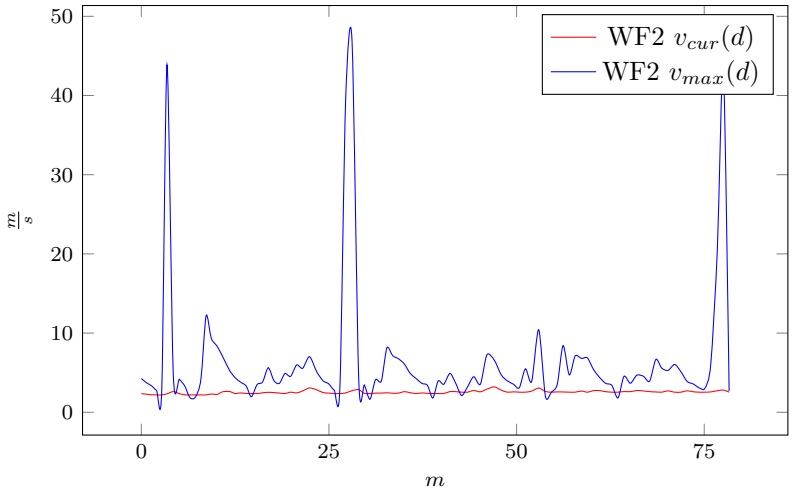


Abbildung 103:
WF2: Geschwindigkeit v nach Distanz d - 500 Werte, Glättung 5

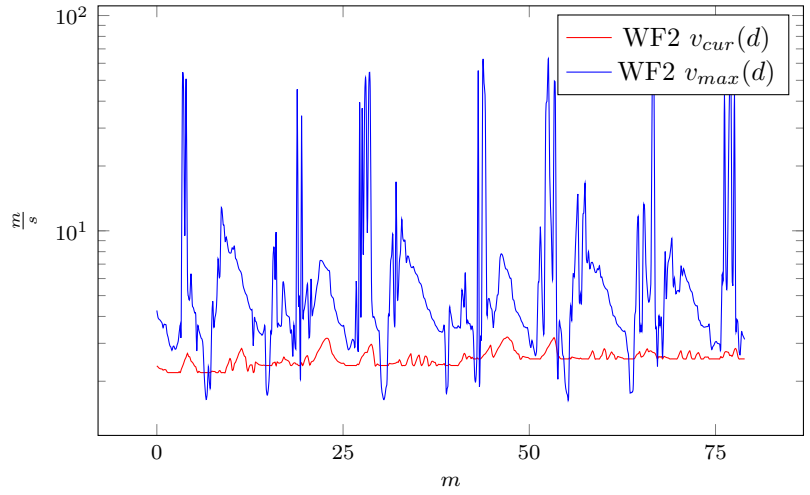


Abbildung 104:
WF2: Geschwindigkeit v nach Distanz d - 500 Werte (logarithmisch)

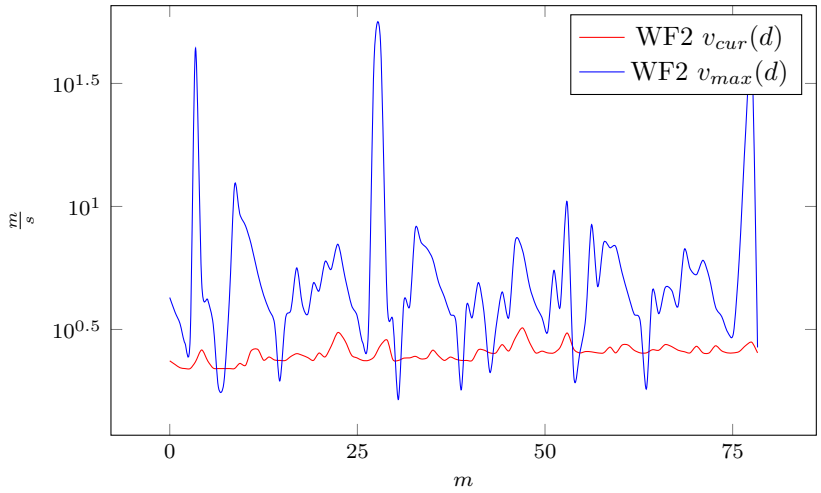


Abbildung 105:
WF2: Geschwindigkeit v nach Distanz d - 500 Werte (logarithmisch), Glättung 5

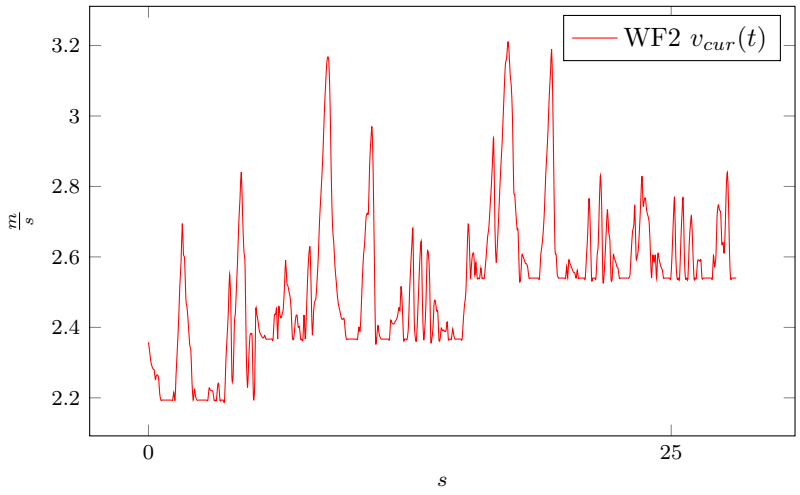


Abbildung 106:
WF2: Geschwindigkeit v nach Zeit t - 500 Werte

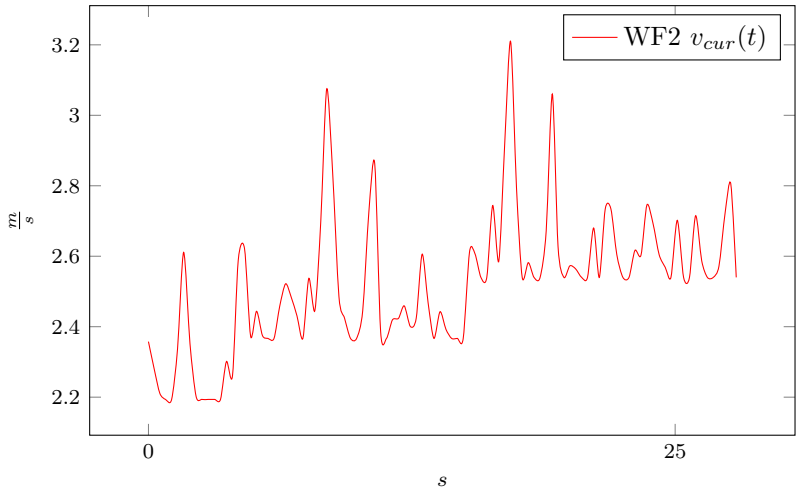


Abbildung 107:
WF2: Geschwindigkeit v nach Zeit t - 500 Werte, Glättung 5

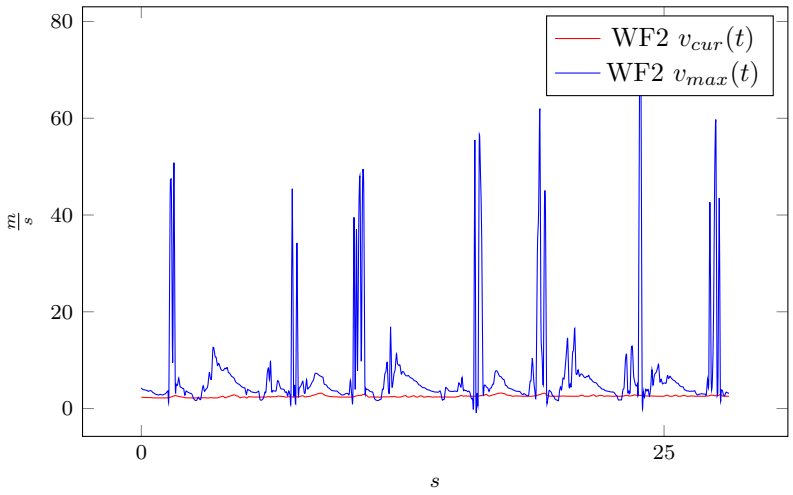


Abbildung 108:
WF2: Geschwindigkeit v nach Zeit t - 500 Werte

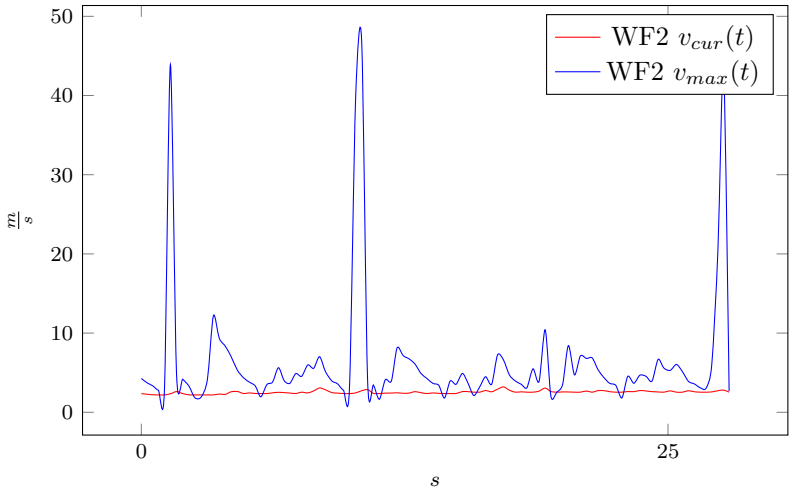


Abbildung 109:
WF2: Geschwindigkeit v nach Zeit t - 500 Werte, Glättung 5

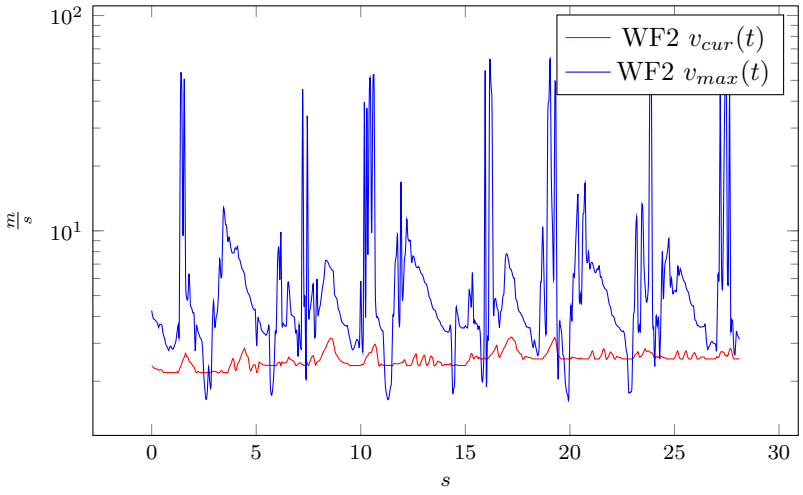


Abbildung 110:
WF2: Geschwindigkeit v nach Zeit t - 500 Werte (logarithmisch)

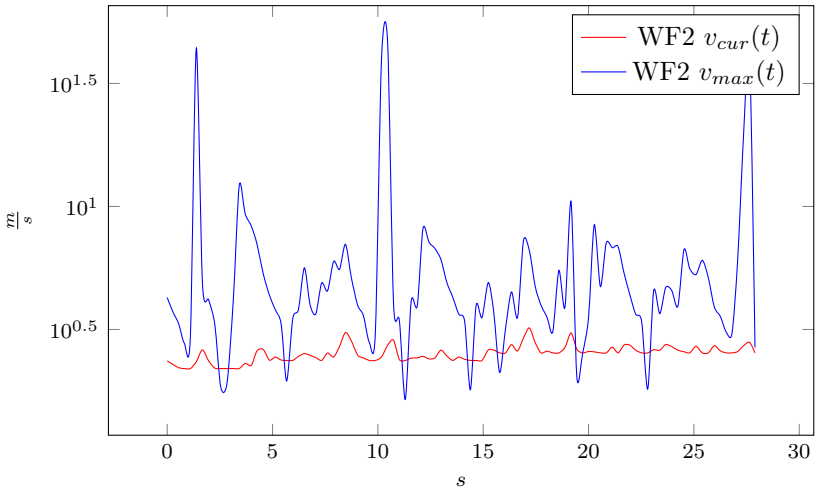


Abbildung 111:
WF2: Geschwindigkeit v nach Zeit t - 500 Werte, Glättung 5 (logarithmisch)

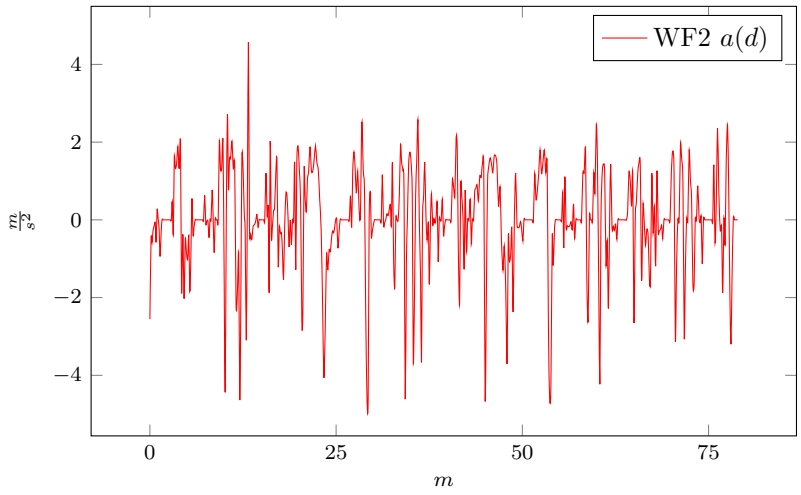


Abbildung 112:
WF2: Beschleunigung a nach Distanz d - 500 Werte

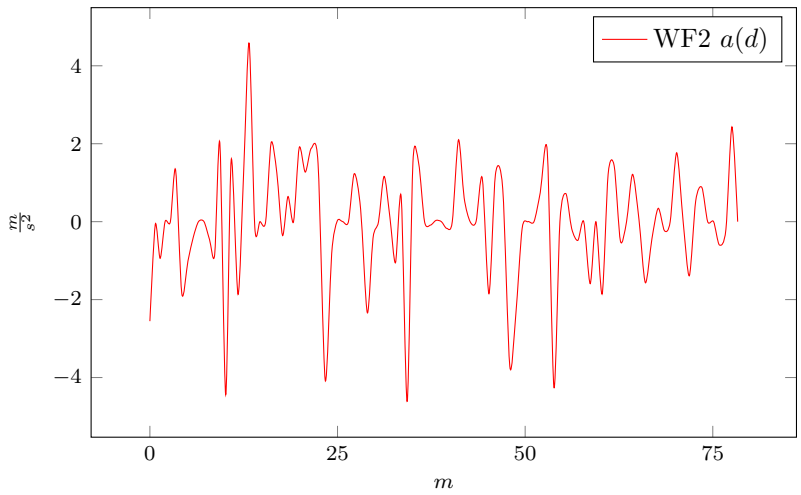


Abbildung 113:
WF2: Beschleunigung a nach Distanz d - 500 Werte, Glättung 5

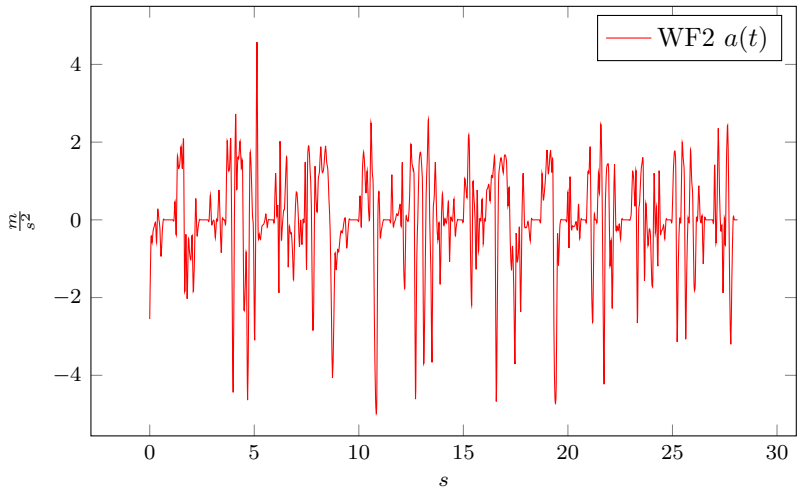


Abbildung 114:
WF2: Beschleunigung a nach Zeit t - 500 Werte

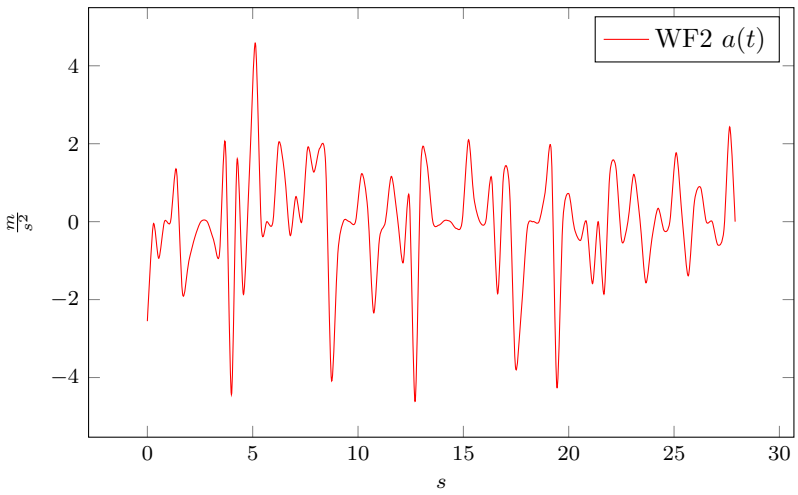


Abbildung 115:
WF2: Beschleunigung a nach Zeit t - 500 Werte, Glättung 5

S Auswertung Wallfollowing 5 - 500

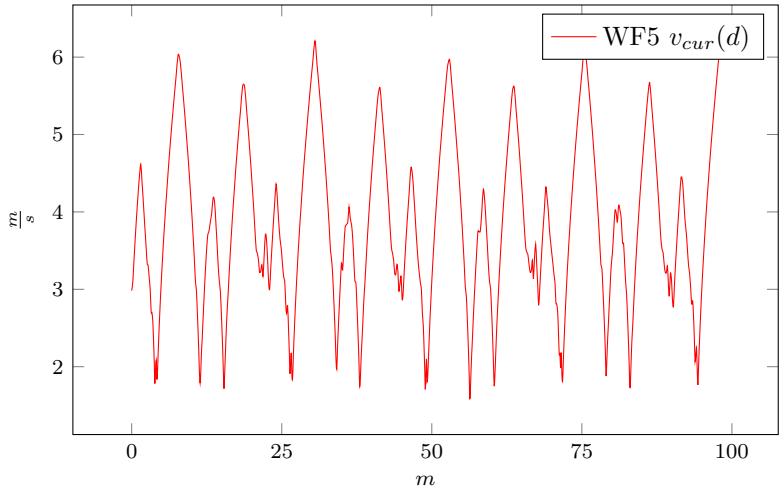


Abbildung 116:
WF5: Geschwindigkeit v nach Distanz d - 500 Werte

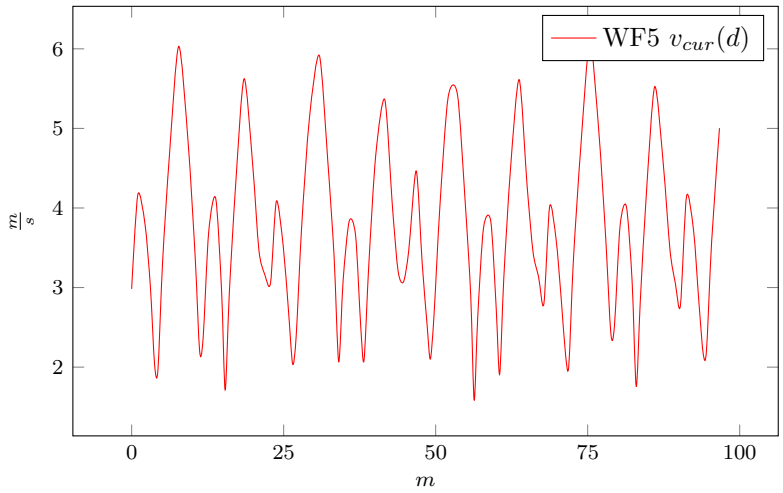


Abbildung 117:
WF5: Geschwindigkeit v nach Distanz d - 500 Werte, Glättung 5

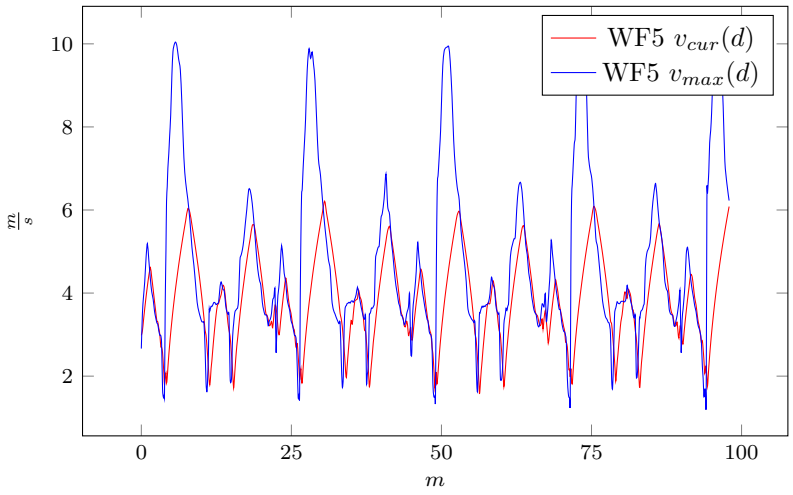


Abbildung 118:
WF5: Geschwindigkeit v nach Distanz d - 500 Werte

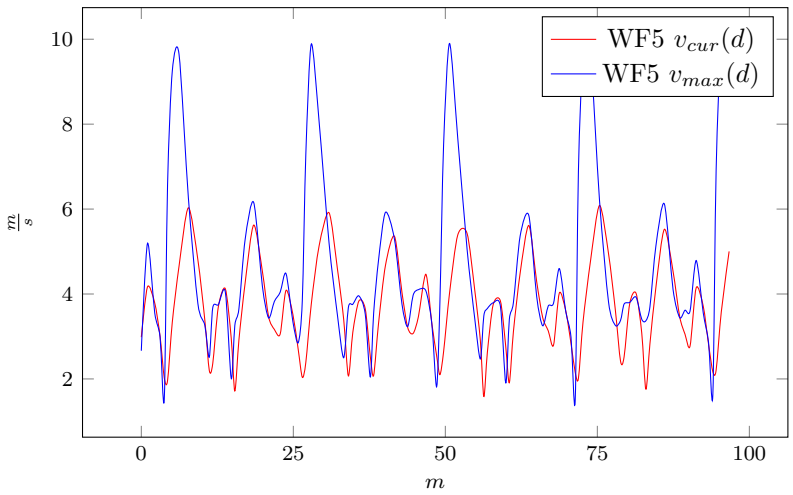


Abbildung 119:
WF5: Geschwindigkeit v nach Distanz d - 500 Werte, Glättung 5

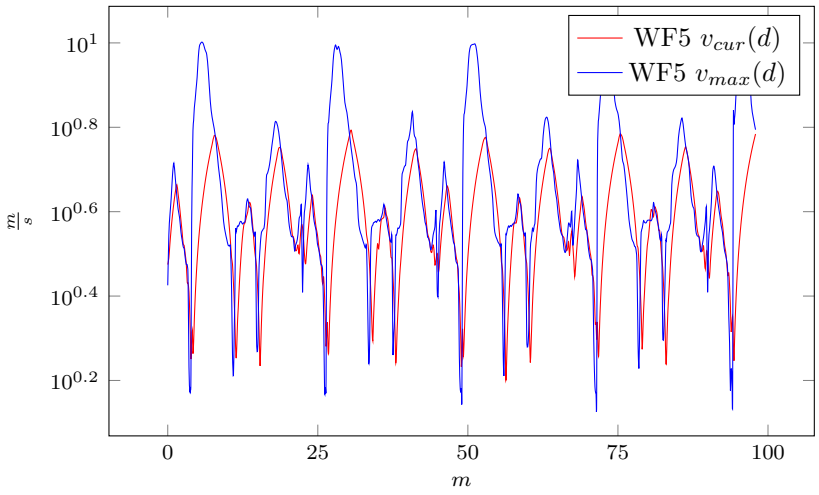


Abbildung 120:
WF5: Geschwindigkeit v nach Distanz d - 500 Werte (logarithmisch)

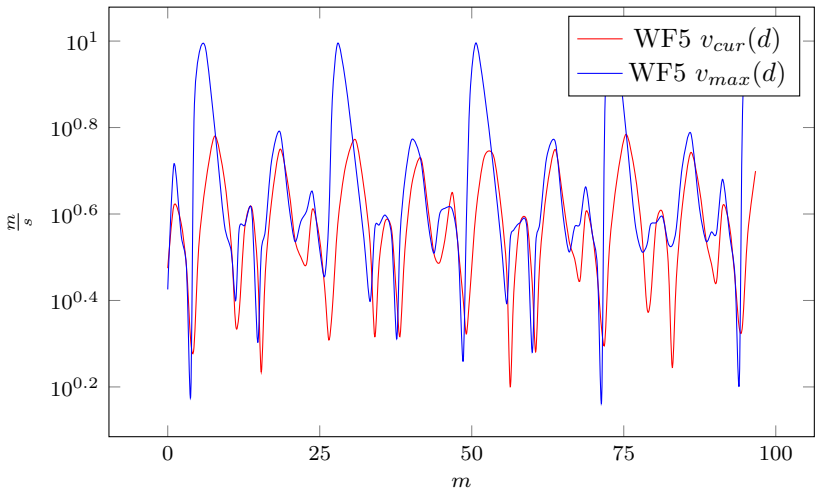


Abbildung 121:
WF5: Geschwindigkeit v nach Distanz d - 500 Werte (logarithmisch), Glättung 5

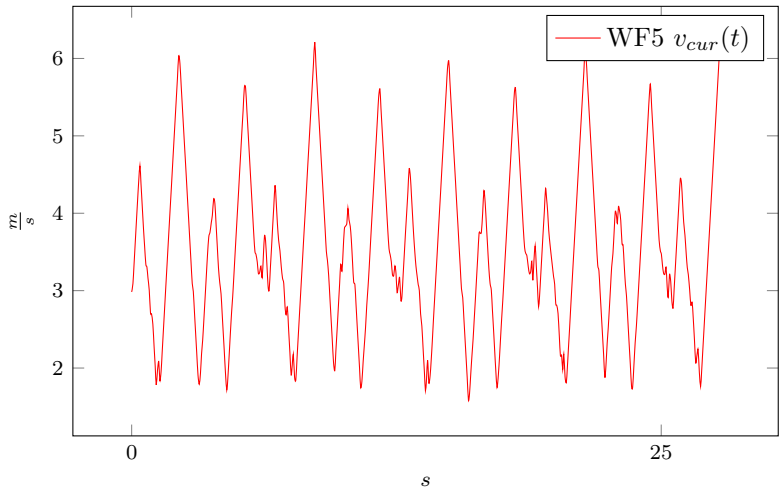


Abbildung 122:
WF5: Geschwindigkeit v nach Zeit t - 500 Werte

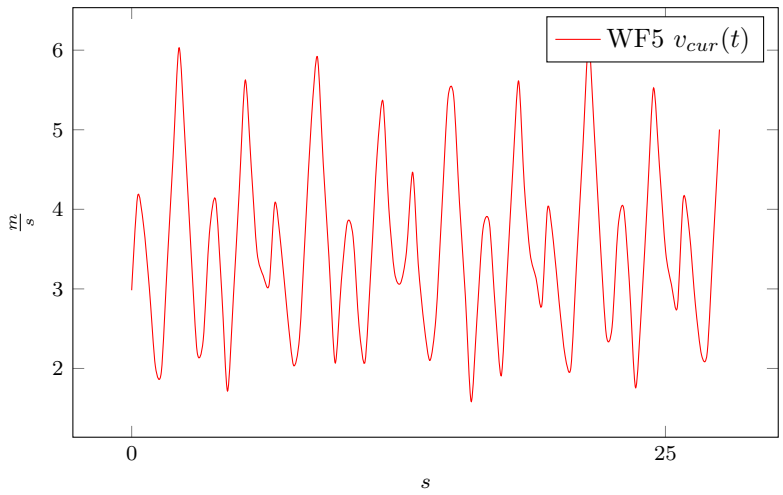


Abbildung 123:
WF5: Geschwindigkeit v nach Zeit t - 500 Werte, Glättung 5

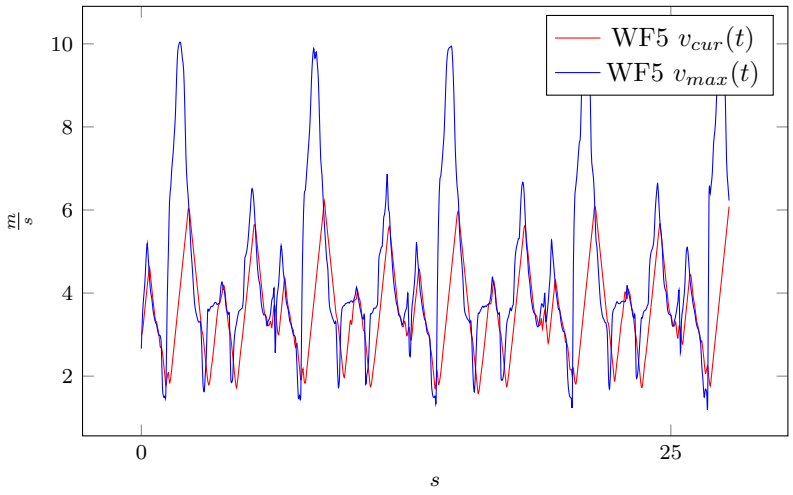


Abbildung 124:
WF5: Geschwindigkeit v nach Zeit t - 500 Werte

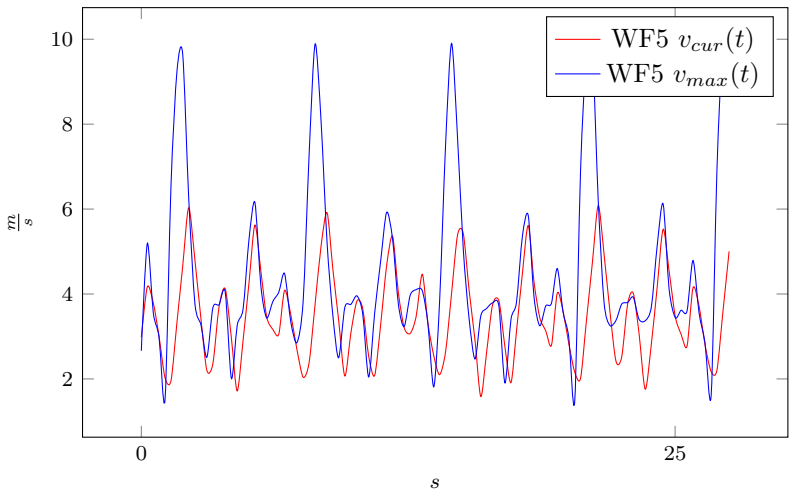


Abbildung 125:
WF5: Geschwindigkeit v nach Zeit t - 500 Werte, Glättung 5

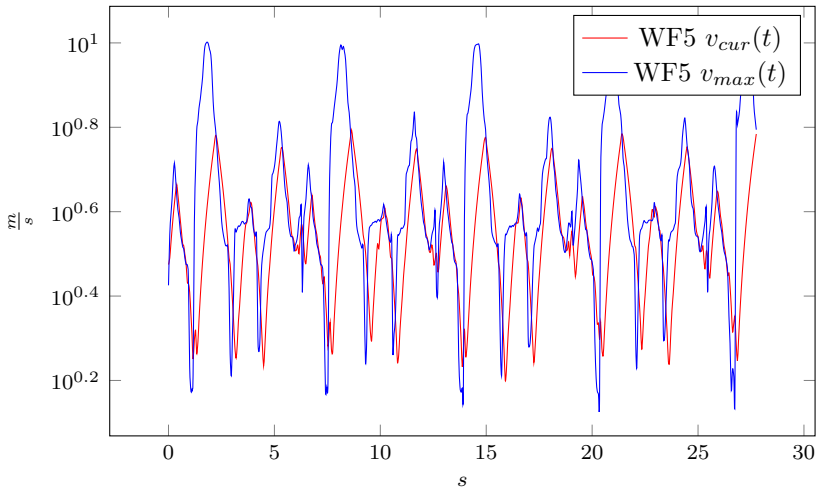


Abbildung 126:
WF5: Geschwindigkeit v nach Zeit t - 500 Werte (logarithmisch)

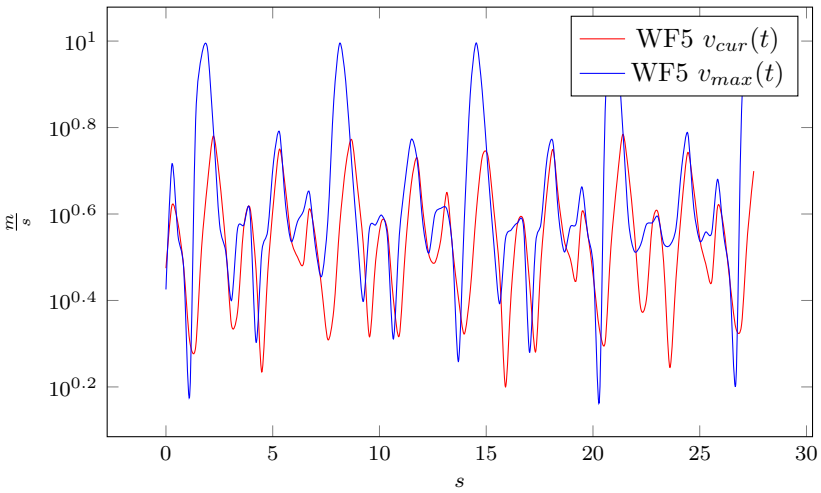


Abbildung 127:
WF5: Geschwindigkeit v nach Zeit t - 500 Werte, Glättung 5 (logarithmisch)

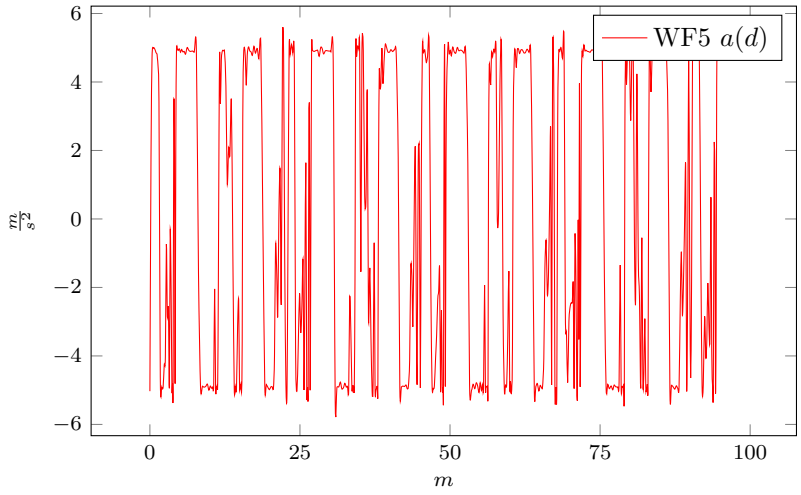


Abbildung 128:
WF5: Beschleunigung a nach Distanz d - 500 Werte

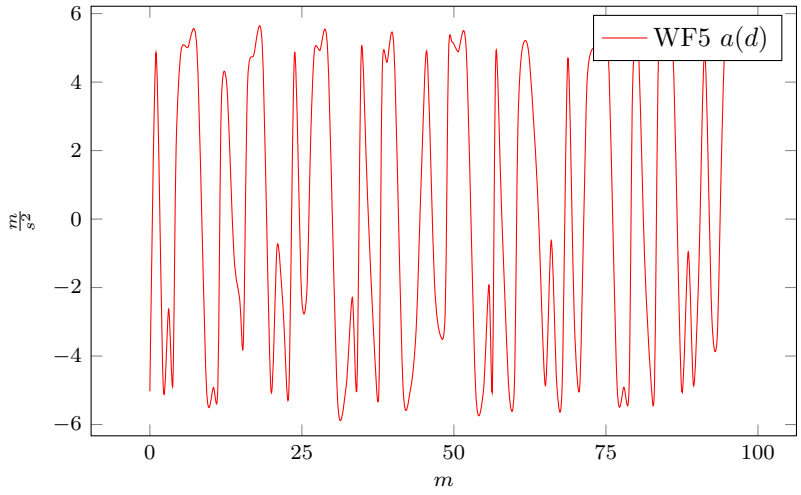


Abbildung 129:
WF5: Beschleunigung a nach Distanz d - 500 Werte, Glättung 5

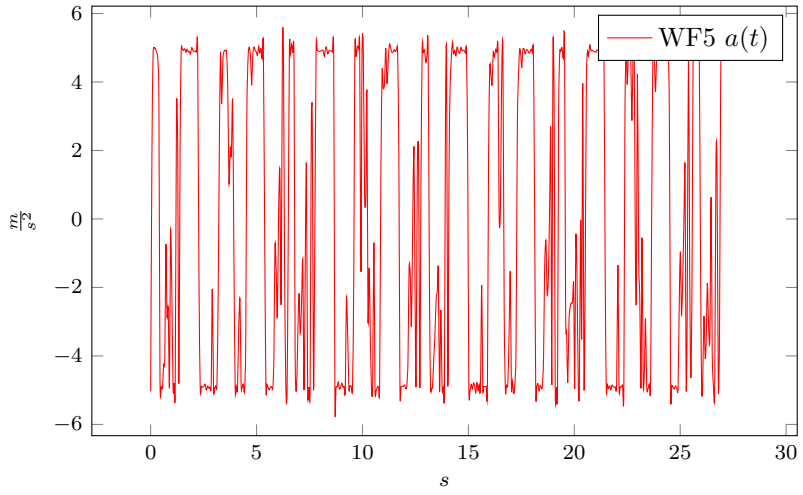


Abbildung 130:
WF5: Beschleunigung a nach Zeit t - 500 Werte

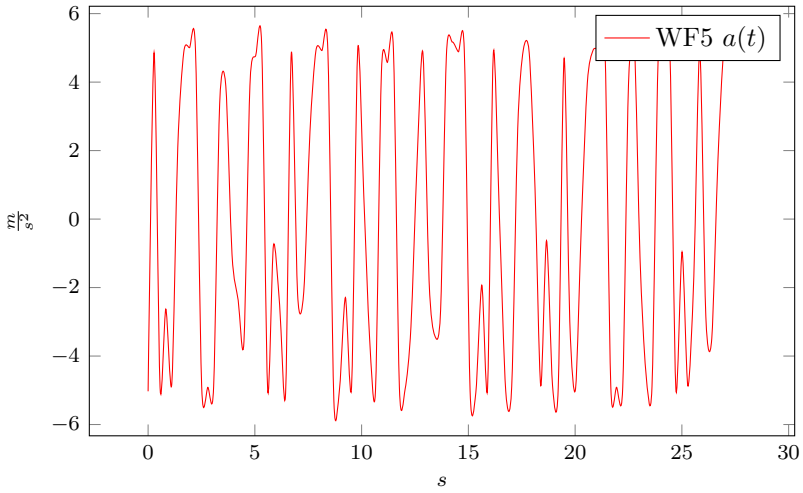


Abbildung 131:
WF5: Beschleunigung a nach Zeit t - 500 Werte, Glättung 5

T Vergleich - 500

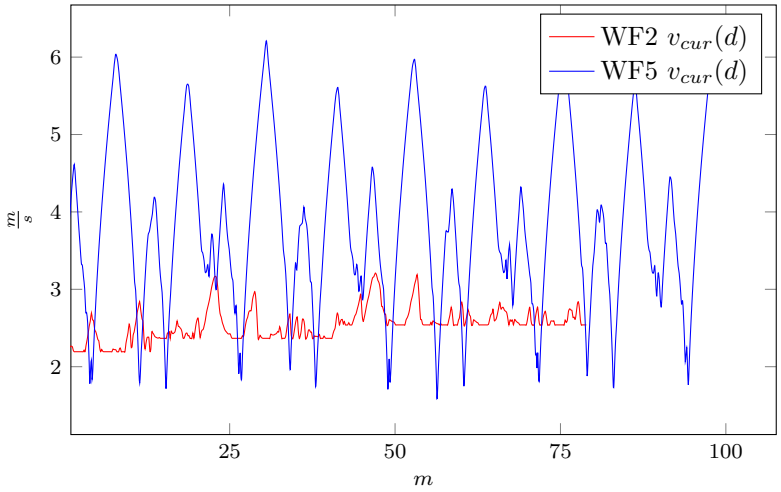


Abbildung 132:
WF5: Geschwindigkeit v nach Distanz d - 500 Schritte

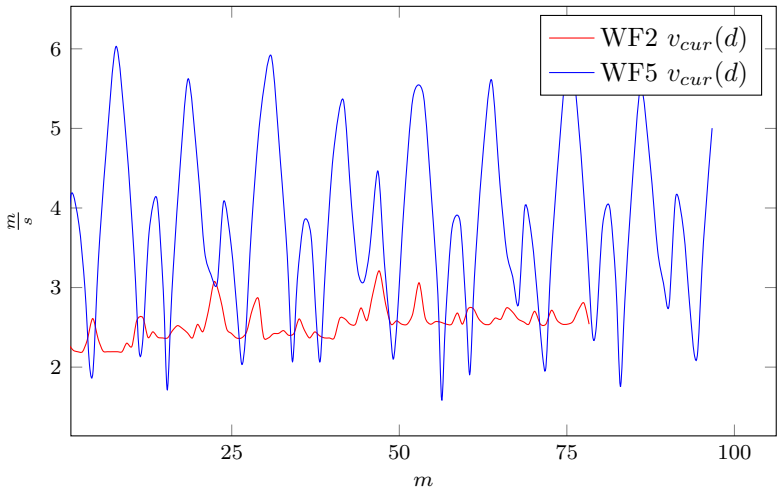


Abbildung 133:
WF5: Geschwindigkeit v nach Distanz d - 500 Werte, Glättung 5

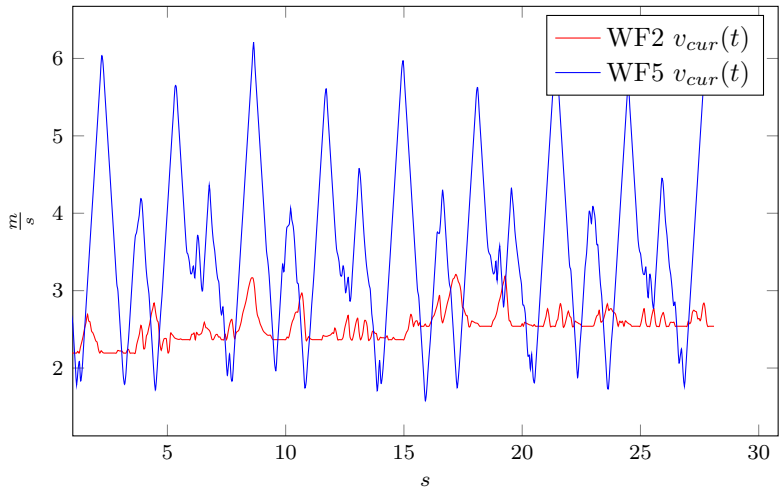


Abbildung 134:
Vergleich: Geschwindigkeit v nach Zeit t - 500 Schritte

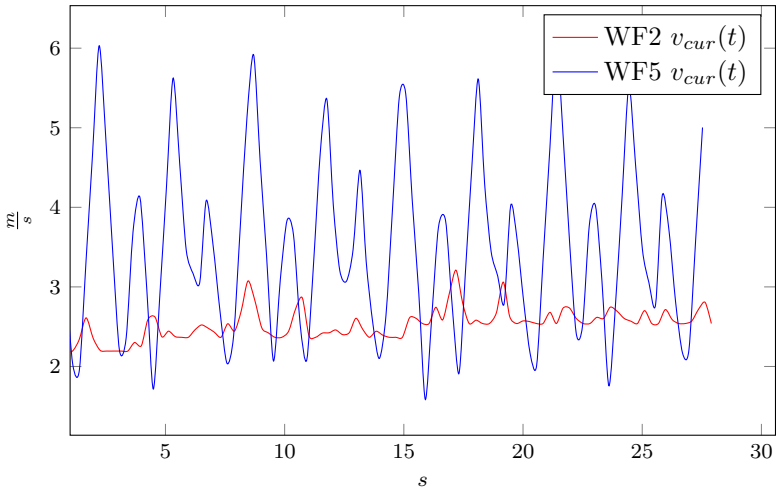


Abbildung 135:
Vergleich: Geschwindigkeit v nach Zeit t - 500 Werte, Glättung 5

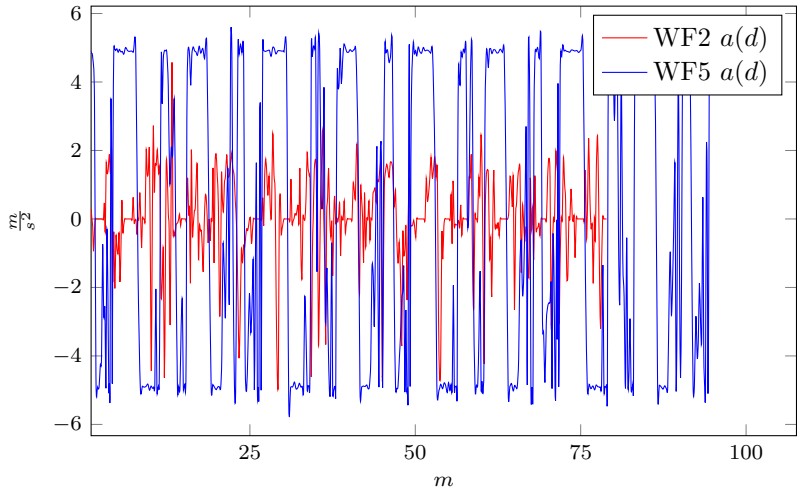


Abbildung 136:
Vergleich: Beschleunigung a nach Distanz d - 500 Schritte

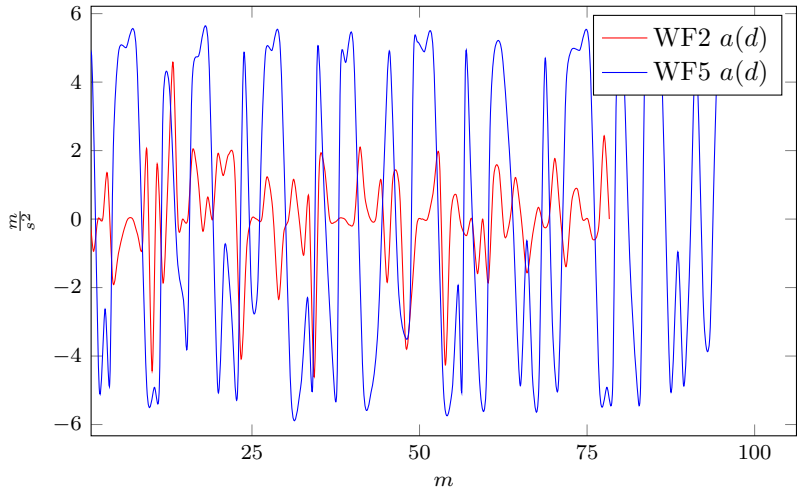


Abbildung 137:
Vergleich: Beschleunigung a nach Distanz d - 500 Werte, Glättung 5

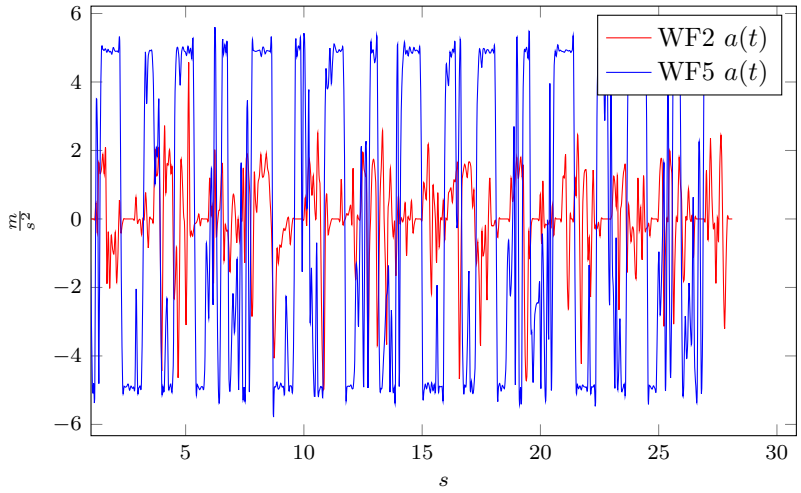


Abbildung 138:
Vergleich: Beschleunigung a nach Zeit t - 500 Schritte

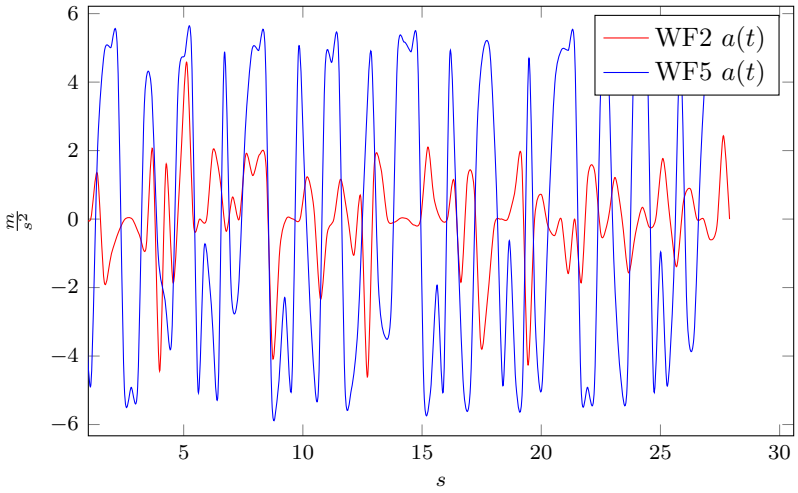


Abbildung 139:
Vergleich: Beschleunigung a nach Zeit t - 500 Werte, Glättung 5