Carolocup: Simulation

Studienarbeit

des Studiengangs Informatik

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Leonard Faix

Juni 2021

Matrikelnummer, Kurs	8708990, 18D
Ausbildungsfirma	ipolog GmbH, Leonberg
Betreuer	Alexander Zimmer

Erklärung

Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema: *Carolocup: Simulation* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Stuttgart, Juni 2021

_____

Leonard Faix

Inhaltsverzeichnis

Abkürzungsverzeichnis

ROS Robot Operating System

Listings

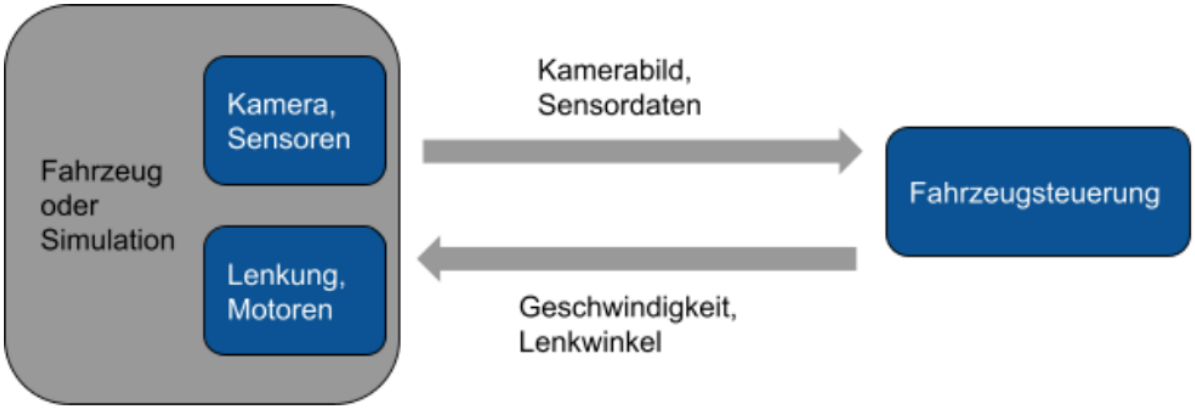
1 Aufgabe

1.1 Carolocup, Regeln

Der Carolocup ist ein jährlich veranstalteter Hochschulwettbewerb der Universität Braunschweig. Teams aus Studenten entwickeln ein 1:10 Modelauto. Dieses Auto soll in der Lage sein autonom durch einen Parkour zu fahren. Die Autos sind meist ausgestattet mit Kameras, Sensoren und einem Rechner auf dem verschiedene Algorithmen zur Auswertung der Daten laufen.

1.2 Simulation

Der Kern dieser Arbeit beschäftigt sich mit dem Aufbau einer Simulationsumgebung. Das Fahrzeug Hat einen Hardware und einen Software Teil.
Im unteren Schaubild befindet sich der Hardwareteil auf der linken Seite. Das sind zum einen die Sensoren und die Kamera und zum anderen die Fahrmechanik. Zur Fahrmechanik gehört alles was sich im Fahrzeug bewegt, d.h. Motoren, Servos, Räder...
Auf der rechten Seite des Schaubilds ist die Fahrzeugsteuerung. Die Hardware liefert ein Kamerabild und Sensordaten an die Fahrzeugsteuerung. Auf Basis dieser Daten berechnet die Fahrzeugsteuerung einen geeigneten Lenkwinkel und verändert gegebenen Falls die Geschwindigkeit des Fahrzeugs. Diese zwei Werte werden dann an die Hardware gegeben, und Motor und Servo werden auf die neuen Werte angepasst.



Das Ziel einer Simulation ist es den Linken Teil des Schaubilds zu ersetzen. Statt der Hardware kann man die Fahrzeugsteuerung mit der Simulation laufen lassen. Die Schnittstelle bleibt die Gleiche: Die Simulation empfängt Lenkwinkel und Geschwindigkeit und errechnet daraus ein Kamerabild und andere Sensordaten. Dafür müssen sämtliche Hardwarekomponenten virtualiesiert werden. Es wird eine 3D-Umgebung benötigt in der sich ein Fahrzeug auf virtuellen Fahrbahnen bewegen kann. Dadurch kann eine virtuelle Kamera ein Bild erstellen. Außerdem wird eine Logik benötigt führ die Fahrmechanik. Ein Lenkwinkel und eine Geschwindigkeit müssen umgerichnet werden in eine Transformation des Fahrzeugs in der 3D-Umgebung.

Warum Simulation?

Wenn man den linken Hardware Teil durch eine Simulation ersetzt, ist rechts und links im Schaubild Software. Man könnte dann an der Fahrzeugsteuerung entwickeln, und neuen Code testen ohne Zugriff auf die Hardware des Fahrzeugs zu haben. Das ist ein Vorteil wenn viele Leute gleichzeitig am Fahrzeug arbeiten. Wenn die Simulation Tests auf Knopfdruck bereitstellt, fällt es Entwicklern leichter den Code zu testen. Dadurch können Feedbackloops, durch öfteres testen bei der Entwicklung, kleiner werden. Somit werden Bugs oder eine Verschlechterung bestimmten Fahrverhaltens schneller erkannt.

1.3 Anforderungen

Als Basis wird hier die KIT Gazebo Simulation genutzt. Primäres Ziel der Arbeit ist es eine Schnittstelle zwischen der DHBW Smart Rollerz Fahrzeugsteuerung und der KIT Gazebo Simulation zu schaffen. Wenn dies gelingt können Funktionen der Simulation, wie die Evaluierung von Fahrten oder die Generation von Strecken genutzt werden.

Neue Sensoren testen wie ToF und Lidar.

2 Umfeld

2.1 ROS

Das Robot Operating System (ROS) ist ein Betriebssystem das speziell auf Roboter angepasst ist. Es läuft auf einem anderen Betriebssystem, in diesem Fall Ubuntu. Es stellt Funktionalitäten wie Hardware Abstraktion, Paket Management, Kommunikation zwischen Prozessen... zu verfügung. Das Ziel von Robot Operating System (ROS) ist die Wiederverwendbarkeit von Code. Dies gelingt dadurch das Robot Operating System (ROS) ein verteiltes System an Prozessen (Nodes) ist. Die Nodes kommunizieren untereinander über Topics. Topics sind die Busse auf denen Nodes Nachrichten senden. Nachrichten zu senden oder zu empfangen ist anonym, dadurch sind Sender und Empfänger komplett entkoppelt voneinander. [7] Nodes lassen sich so austauschen und wiederverwenden. [6] Die Regelung des Fahrzeugs läuft auf einer ROS Architektur.

2.2 Gazebo

Gazebo ist ein open-source Robotersimulator. Er beinhalted eine 3D-Umgebung mit Physik-Engine, Sensorsimulation und Antriebssimulation. Gazebo verfügt außerdem über ein graphisches Interface. Ein Vorteil von Gazebo ist die aktive Community und die vielen Plugins. [2]

2.3 KIT Gazebo Simulation

Als Ausgangssituation für die Simulation wird die open source Simulationsumgebung des KITs genutzt. Diese ist eine Gazebo Simulation mit verschiedenen ROS Nodes. Die wichtigsten Nodes sind folgende:

CarStateNode

Publisht sämtliche Informationen über den aktuellen Zusand des Fahrzeugs. Dazu gehören die Position, die Orientierung, die Geschwindigkeit und ein Field of View. [3]

GazeboRateControlNode

Diese Node ist nur nötig wenn die Simulation in einem Docker Container läuft. Gazebo gewärleistet keine update-Rate innerhalb eines Docker Conteiners. In dem Fall kann sich die GazeboRateControlNode darum kümmern. [3]

ModelPluginLinkNode

Erlaubt eine leichte Interaktion mit Gazebo Modellen, über bereitgestellte Topics. [3]

WorldPluginLinkNode

Erlaubt Hinzufügen und Entfernen von Gazebo Modellen, über bereitgestellte Topics. [3]

AutomaticDriveNode

Erlaubt es die Simulation ohne Fahrzeugsteuerung laufen zu lassen. Das Fahrzeug beweg sich entlang der idealen Fahrspur. [3]

Verschiedene Evaluierungs Nodes

Das Fahrzeug wird evaluiert indem über verschiedene State Machines das Verhalten aufgezeichnet wird. Außerdem werden Geschwindigkeit, Kollisionen und Spurhaltung aufgezeichnet. [4]

Groundthrouth Nodes

Stellt Informationen über die aktuelle Straße zur Verfügung. [5]

URDF Model

Das URDF Model definiert das Aussehen des Fahrzeugs. Auch Sensoren und Kamera sind im URDF Model definiert und angepasst. Das Model ist jedoch statisch. Man übergibt dem Model direkt eine Transformation um es zu bewegen. Dadurch spielen im URDF Model definierte Größen, wie das Gewicht, die Reibung oder Interaktionen zwischen Bausteinen innerhalb des Models, keine Rolle. Bisher werden diese Größen vernachlässigt da man eine externe Regelung benötigt um die Simulation zu nutzen. [3]

2.4 Fahrzeugsteuerung

Die eigentliche Fahrsimulation des Fahrzeugs ist nicht Teil der Simulationsumgebung. Um das Fahrzeug zu bewegen erwarted die Simulation ein Topic auf dem neue Position gepublished werden. Da die Simulation aber den Lenkwinkel als Input nutzen soll, muss die Simulation dahingehend erweitert werden.

Immer wenn ich Fahrzeug sage meine ich das simulierte, wenn ich echtes Fahrzeug sage meine ich das echte.

3 Implementierung

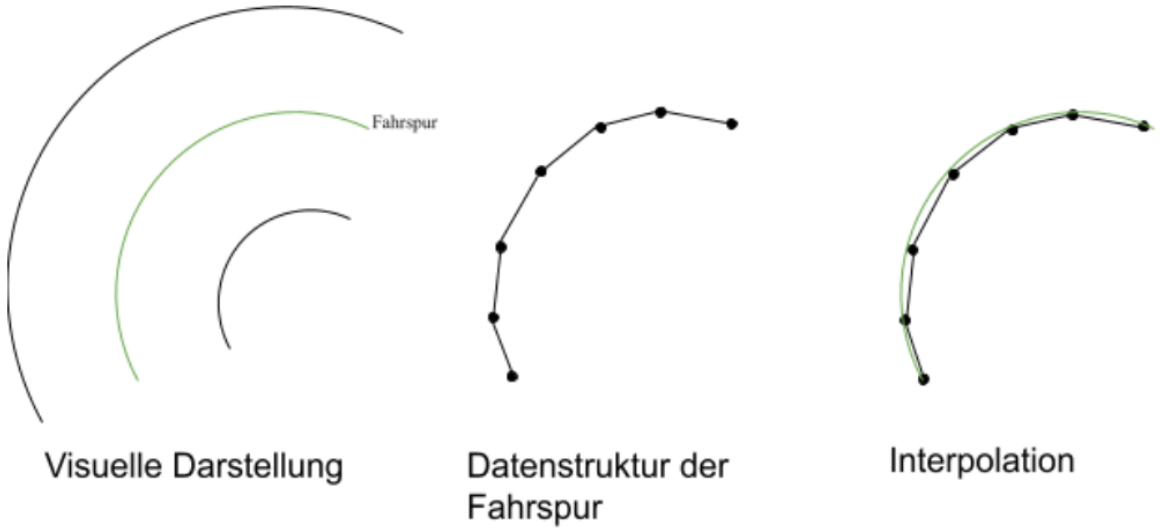
3.1 Ausgangslage

3.2 Simulation zur Datengenerierung

Die Simulation erlaubt es eigene Strecken aufbauen. Dazu gibt es eine High-Level Schnittstelle um verschiedene Streckenabschnitte zusammen zu setzten. Um eine neue Strecke aufzubauen kann man unterschiedliche Kurven, Kreuzungen und Geraden kombinieren. Dazu wird die visuelle Darstellung generiert. Visuelle Darstellung sind Seitenstreifen, Mittelstreifen, Haltelinie und andere Straßenmarkierungen, aber auch Objekte. Parallel dazu wird die Fahrspur geniert. Die Fahrspur ist die Spur in der Mitte der Fahrbahn. Das Fahrzeug sollte die Fahrspur möglichst genau entlang fahren, um nicht von der Fahrbahn abzukommen. Die Fahrspur wird für zwei Anwendungen benötigt: - Sie ermöglicht eine Fahrt ohne Fahrzeugsteuerung im AutoDrive Modus - Sie ermöglicht eine Evaluierung der Fahrzeugsteuerung. Die Fahrzeugsteuerung wird bewertet, je nach dem wie nah das Fahrzeug an der idealen Fahrspur fährt

3.2.1 Interpolation der Fahrspur

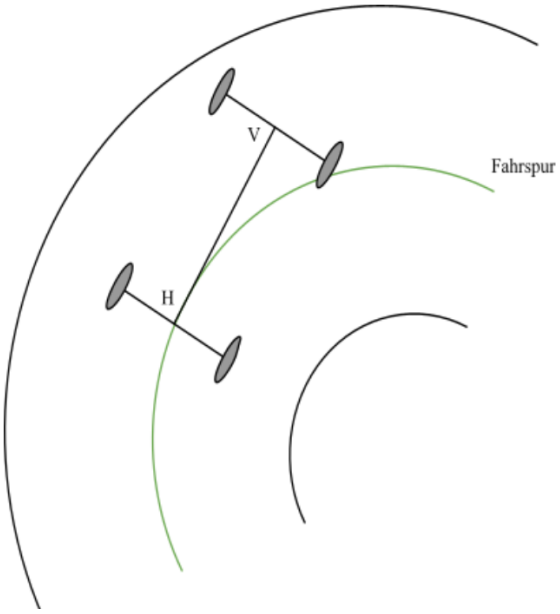
Das Modul AutomaticDrive kann genutzt werden damit das Fahrzeug entlang der Fahrspur fährt. Beim Testen dieser Funktion kam es zu einem ruckeligen Fahrverhalten in den Kurven. Dies liegt an der Speicherung der Fahrspur.



Links sieht man die visuelle Darstellung, die generiert wird. Die Fahrspur (in grün) wird dann gespeichert. Dies geschieht indem in einem regelmäsigen Abstand Punkte, die auf der Fahrspur liegen gespeichert werden. Die Fahrspur wird dann als Liste mit Punkten gespeichert. Das AutomaticDrive Modul nutzt dann jeweils die direkte Verbindung von Punkt zu Punkt. Dadurch fährt es immer kurz gerade aus, und an jedem Punkt dreht das Fahrzeug auf der Stelle. Um dieses Verhalten zu verhindern und an die Realität anzupassen wie Interpolation genutzt. Hier wird Kubische Interpolation genutzt. Dabei werden die Punkte nicht mit einer geraden Linie verbunden sondern mit einem Polynom dritten Grades. Dadurch kann auch die Steigung am Punkt betrachtet werden und es gibt keine Ecken mehr. Dies zeigt die grüne Linie im dritten Bild. Die interpolierte Fahrspur stellt nun, mit ausreichender Genauigkeit, mit der originalen Fahrspur im ersten Bild überein. In Python kann so eine Interpolation zum Beispiel mit der Klasse interp1d aus dem Modul scipy.interpolate berechnet werden.

3.2.2 AutoDrive

Die Positionierung und Orientierung funktioniert mit dem Hinterachsenmittelpunkt H als Basis. Das Fahrzeug fährt indem der Punkt H mit einer konstanten Geschwindigkeit entlang der Fahrspur fährt. Die Orientierung des Fahrzeugs ist so definiert, dass das Fahrzeug am Punkt H eine Tangente zur Fahrspur bildet. Die Tangente kann mit einer Funktioner der Datenstruktur der Fahrspur erechnet werden. In einer Kurve kann das Fahrzeug dann so aussehen:

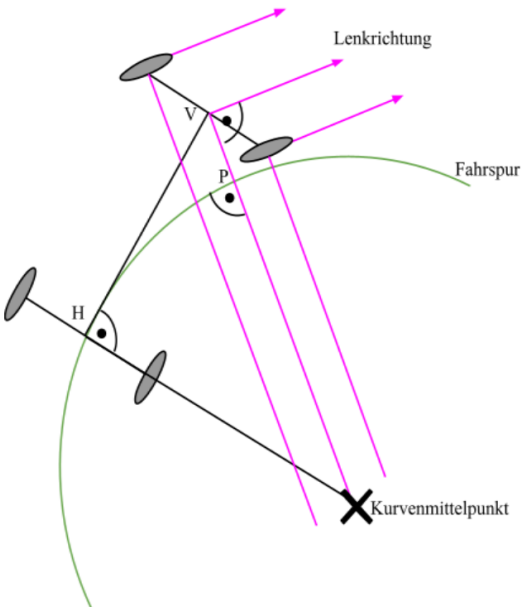


Positionierung und Rotation um Punkt H hat sich als realitätsnah erwiesen, da die Kamera, die vorne am Fahrzeug montiert ist, in den Kurven ein ähnliches Bild geliefert hat wie das echte Fahrzeug.

Kamera entlang der Strecke fahren lassen

Um Algorithmen und Model in der Spurerkennung und Situationsklassifizierung zu verifizieren und verbessern, benötigt der Auto drive noch eine Möglichkeit den Lenkwinkel auszugeben. Für den AutomaticDrive nehmen wir ein paralleles Lenksystem an. Punkt V nach H. $\vec{VH}$

Die Vorderräder sind ausgerichtet entlang der Tangente VT am Punkt V zur Fahrspur. Der Lenkwinkel β ist der Winkel zwischen $\vec{VH}$ und $\vec{Lenkrichtung}$.



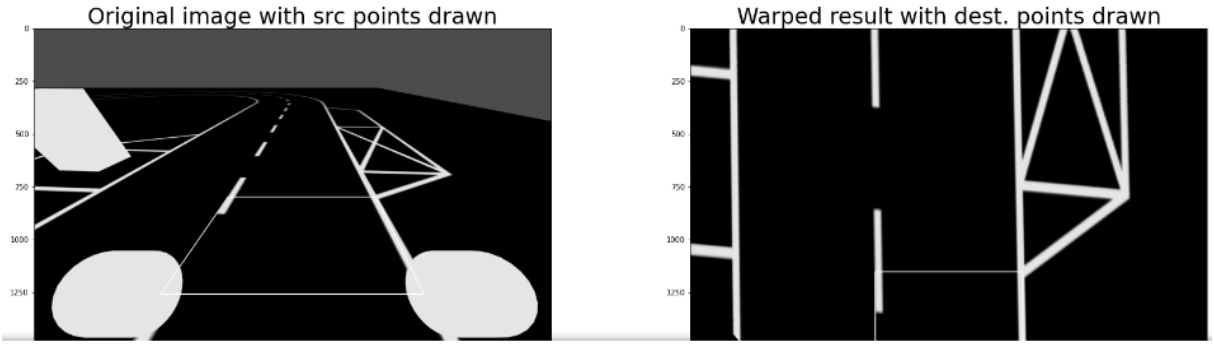
PARAMETER INCLUDE AUTODRIVE

3.3 Kamera

Um die Kamera zu simulieren wird camera_controller plugin genutzt. Dieses Plugin hat diverse Einstellungsmöglichkeiten für die Kamera wie FoV, clipping, Farbformat, Updaterate... Es verfügt außerdem über fortgeschrittenere Einstellungsmöglichkeiten wie Distortion oder Noise.

Die Auflösung der Kamera ist 2456x2054 Pixel. Die Position und Orientierung der Kamera wird genau ermittelt, um die simulierte Kamera an die selbe Position zu bringen.

```
<link name="front_camera_link">...</link>
<joint name="front_camera_joint" type="fixed">...</joint>
<gazebo reference="front_camera_link">
    <sensor name="front_camera" type="camera">
        <plugin name="camera_plugin" filename="libgazebo_ros_camera.so"
            >...</plugin>
        <update_rate>25</update_rate>
        <camera>
            <horizontal_fov>2.064911321881509</horizontal_fov>
            <image>
                <width>2456</width>
                <height>2054</height>
                <format>L8</format>
            </image>
            <clip>
                <near>0.1</near>
                <far>4</far>
            </clip>
        </camera>
    </sensor>
</gazebo>
```



Problem: Für die IPM umrechnung muss der Kamerafeed sehr genau den echten wiederspiegeln. Neue matrix wäre eine Scheiß Lösung.

3.4 statische Fortbewegung

Problem: Lenkwinkel muss irgendwie in eine Transformation des Fahrzeugs umgewandelt werden. Der genaue Lösung wäre eine dynamische Simulation der Räder und Lenkung. Dies war nicht so einfach abzustimmen, da sich die Simulation anders Verhalten hat als das echte Fahrzeug. Vorläufige Lösung: Eine Rechnung die die Transformation abschätzt, aber sehr genau ist bei kleinen Lenkwinkel und hoher Fps.

Gegeben ist der Lenkwinkel alpha Die Geschwindigkeit des Fahrzeugs v Der daraus resultierende Vektor Lenkrichtung steering Position des Fahrzeugs p Orientierung des Fahrzeugs und der daraus resultierende Vektor direction

Als erstes wird aus dem Lenkwinkel alpha der Vektor steering bestimmt. Danach wird die neue Position pNeu bestimmt mit pNeu = p + steering * (deltaTime * v) Die neue Orientierung des Fahrzeugs directionNeu wird erechnet mit directionNeu = direction rotieren um k*alpha dabei ist k Abhängig von der gefahrenen Strecke, sowie der Fahrzeuglänge.

3.5 dynamische Fortbewegung

Die statische Fortbewegung war nur eine Übergangslösung. Die dynamische Fortbewegung ist genauer. Bisher wurde einfach nur die Position des Fahrzeugs geändert. Die Physik-Engine, die Gazebo bietet, wird also weder vom AutoDrive noch von der statischen Fortbewegung genutzt. Ziel der dynamischen Fortbewegung ist es ein Fahrzeug mit vier Rädern zu simulieren. Das Fahrzeug kann fahren, indem sich die Räder drehen. Die Hinterräder sollen den Antrieb haben. Die Vorderräder sollen lenken können.

Vorteile dynamische Fortbewegung

Wenn für das Fahrzeug, die Massen und die Reibung richtig definiert ist, können bestimmte Fahrverhalten besser abgebildet werden. Zum Beispiel kann das Fahrzeug dann auch aus Kurven rausgeschläudert werden, wenn es zu schnell ist. Die zusätzliche Kompläxität, die durch die Räder und das Lenksystem kommt, ist deutlich genauer als die Methode zu statischen Fortbewegung.

URDF Definitionen

Die Räder des Fahrzeugs werden definiert innerhalb einem Link (siehe urdf unten). Jeder der vier Links besitzt eine visuelle Komponente und ein Kollisions-Komponente. Das visuelle Objekt und das Kollisions-Objekt stimmen überein, beide definieren den gleichen Zylinder. Das Kollisions-Objekt wird benötigt um mit dem Boden zu kollidieren um auf diesem zu fahren. Jeder physikalisch simulierte Link benötigt einen inertial-Tag. Dieser definiert die Masse "massünd die Massenverteilung als inertia Tensor ïnertia".

Joints

Damit sich ein Rad drehen kann, wird es mit einem Joint an dem Fahrzeug befestigt. Ein Joint ist eine Verbindung zwischen zwei Links. In einem Joint wird definiert wie sich die zwei Links zueinander verhalten. Es gibt die unterschiedliche Typen von Joints:

- revolute - Rotiert um eine Achse bis zu gesetzten Ober- und Untergrenze
- continous - Rotiert um eine Achse ohne Grenze
- prismatic - Gleitet entlang einer Achse bis zu Grenzen
- fixed - Dieser Joint verbindet zwei Links fest miteinander. Es gibt keine relative Bewegung
- floating - Erlaubt freie Positions- und Rotationsänderung zwischen zwei Links
- planar - Erlaubt Bewegung auf einer Ebene

[8]

Drehen und Lenken

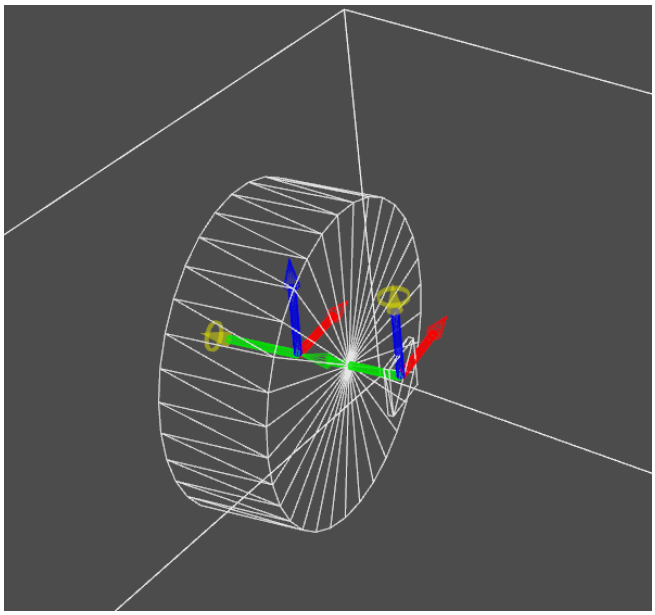
Die Räder werden alle mit einem Joint mit dem Fahrzeug verbunden. Der Typ des Joints ist für die Räder continuos. Das bedeuted, dass sich die Räder ohne Einschränkung um eine Achse drehen können. Die Hinterräder werden direkt mit der Basis des Fahrzeugs verbunden. Die Vorderräder werden mit einem Zwischenstück verbunden. Dieses Zwischenstück wird benötigt, da die Vorderräder einen zweiten Joint benötigen um eine Lenkbewegung durchzuführen. Zwei Links können nur mit einem Joint miteinander verbunden werden. Vom Mittelstück zur Basis gibt es einen Joint, der die Lenkung möglich macht. Dieser Joint ist vom Typ revolute. Der Joint dreht sich auf der Achse, orthogonal zum Boden. Er hat Limits bei +-60°.

Das Vorderrad kann sich nun um die eigene Radachse drehen, und es kann sich mit dem Mittelstück um die z-Achse drehen. Der folgende Code definiert das linke Vorderrad "reifen_front_links"mit visual, collision und inertial. Danach wird das Mittelstück "links_front_steering"definiert, welches eine einfache Box ist. Das Mittelstück wird mit der Basis (chassis) verbunden mit einem revolute Joint. Das Vorderrad mit mit dem Mittelstück verbunden mit einem continous Joint.

```
<link name="reifen_front_links">
    <visual>
        <origin xyz="0 0 0" rpy="1.57075 0 0"/>
        <geometry>
            <cylinder length ="0.02" radius="0.03"/>
        </geometry>
        <material name="reifenfarbe">
            <color> 1.0 1.0 1.0</color>
        </material>
    </visual>
    <collision>
        <origin xyz="0 0 0" rpy="1.57075 0 0"/>
        <geometry>
            <cylinder length ="0.02" radius="0.03"/>
        </geometry>
    </collision>
    <inertial>
        <inertia ixx="0.0001" ixy="0" ixz="0" iyx="0" iyy="0.0001" iyz="
            0" izx="0" izy="0" izz="0.0001"/>
        <mass value="0.05"/>
    </inertial>
</link>
<link name="links_front_steering">
    <visual>
        <origin rpy="0 0 0" xyz="0 0 0"/>
        <geometry>
            <box size =".01 .001 .01"/>
        </geometry>
    </visual>
    <inertial>
        <inertia ixx="0" ixy="0" ixz="0" iyx="0" iyy="0" iyz="0" izx="0"
             izy="0" izz="0"/>
        <mass value="0.01"/>
    </inertial>
</link>
<joint name="basis_zu_links_front_steering" type="revolute">
    <parent link="chassis"/>
    <child link="links_front_steering"/>
    <origin xyz="0.1 0.06 0.03" rpy="0 0 0"/>
    <axis xyz="0 0 1"/>
```

```
39      <limit lower="-1.1" upper="1.1" effort="10" velocity="100"/>
40  </joint>
41  <joint name="links_front_steering_to_reifen_front_links" type="
        continuous">
42      <parent link="links_front_steering"/>
43      <child link="reifen_front_links"/>
44      <origin xyz="0.0 0.02 0.0"/>
45      <axis   xyz="0 1 0"/>
46  </joint>
```

Dieser Code produziert das:



Hier ist das Linke Vorderrad zu sehen und das dazugehörige Mittelstück. Man erkennt die Zwei Joints, welche jeweils Rotation um eine Achse erlauben.

Actuator und Transmissions

Um die Räder und die Länkung zu steuern, benötigt es einen Motor. Für urdf gibt es hierfür Actuator und Transmissions. Eine Transmission verbindet einen Actuator mit einem Joint. Der Actuator, zu Deutsch Äntrieb", gibt eine Bewegung über die Transmission an den Joint weiter. In diesem Fall ist es eine Drehbewegung, da nur revolute und continous Joints genutzt werden. Der Actuator definiert das Hardware Interface. Bei den Hinterradantrieben wird ein Velocity Interface genutzt, da die zu steuernde Größe die Geschwindigkeit des Rads, bzw. des Joints ist. Für die Vorderradlenkung wird ein Effort Interface genutzt, da die zu steuernde Größe der Winkel des Rads ist. Außerdem kann der Actuator eine machanische

Reduktion der Drehgeschwindigkeit definieren. Die Definition einer Transmission mit Actuator sieht so aus:

```
<transmission name="transmission_front_left_wheel">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="basis_zu_links_front_steering">
        <hardwareInterface>hardware_interface/EffortJointInterface</
            hardwareInterface>
    </joint>
    <actuator name="motor_front_left_wheel">
        <hardwareInterface>hardware_interface/EffortJointInterface</
            hardwareInterface>
        <mechanicalReduction>1</mechanicalReduction>
    </actuator>
</transmission>
```

Controller

Um die Actuator zu steuern kann man das Package ros_control nutzten. Dies bietet eine Reihe verschiedener Controller. Controller können einen Actuator nach PID-Steuerung steuern. Um die Controller zu benutzen wird ein neues Package erstellt. Das Package */simulation/src/gazebo_controller* beinhaltet eine Launchdatei und eine Parameterdatei. Die Parameterdatei definiert die vier verschiedenen Controller. Zwei für den Hinterradantrieb und zwei für die Vorderlenkung. Die Controller für den Antrieb sind JointVelocityController, die Controller für die Lenkung sind JointPositionController. Von dem Antrieb soll die Geschwindigkeit gesteuert werden, von der Lenkung soll die Position bzw. die Rotation gesteuert werden. In der Parameterdatei werden auch die zugehörigen Joints und die PID Parameter definiert. Die Launchdatei nutzt die Parameterdatei und lädt damit die Controller über einen controller_spawner. Die Launchdatei wird in der master.launch-Datei wie folgt aufgerufen:

```
<include if="\$(eval include_automatic_drive == false)" file="\$(find
    gazebo_controller)/launch/robot_control.launch"></include>
```

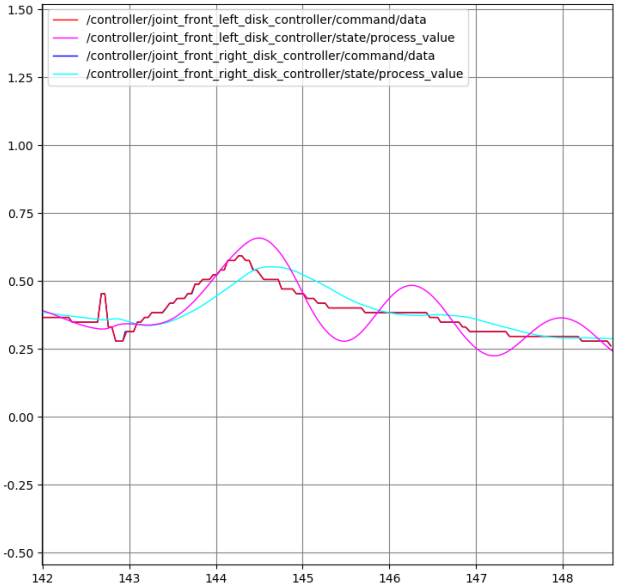
Die Controller werden nur initialisiert, wenn AutomaticDrive nicht genutzt wird.

3.5.1 PID Regelung

PID Regler werden eingesetzt um eine bestimmte Regelgröße zu regeln. Der Regler vergleicht den Soll- und Istwert der Regelgröße und versucht deren Differenz zu minimieren. Dieses Ziel verfolgt er über drei Komponenten: Die Proportional-Komponente, welche abhängig

von der aktuellen Differenz zwischen Ist- und Sollwert ist. Die Integral-Komponente, welche abhängig von der vergangenen Differenz zwischen Ist- und Sollwert ist. Die Differentail-Komponente, welche abhängig von der zukünftigen Differenz zwischen Ist- und Sollwert ist. Wie groß der Einfluss der jeweiligen Komponente ist lässt sich über die Parameter P,I und D steuern. [1]

Alle vier Controller benötigen angepasste PID-Parameter. Schauen wir uns zuerst die Vorderradlenkung an. Dafür wird die Simulation mit der Fahrzeugsteuerung verbunden. Aus der Fahrzeugsteuerung werden nun Lenkwinkel an die Simulation geschickt. Dieser Lenkwinkel ist der Sollwert, und wird durch die rote Linie im Diagram dargestellt.



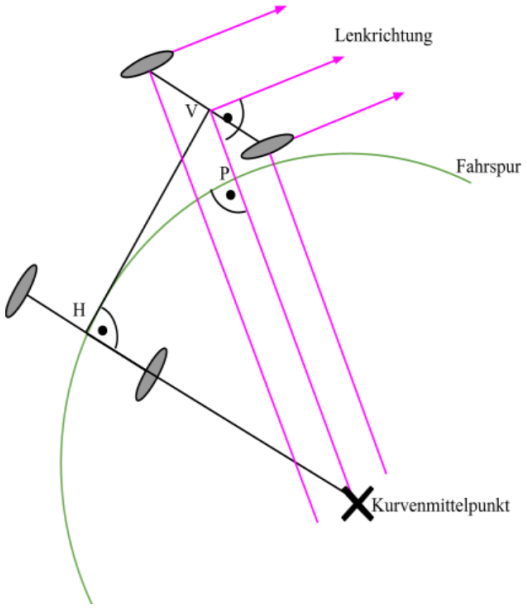
Auf diesen Sollwert hören zwei Controller. Der Controller für das linke Rad hat eine PID-Einstellung von 15-0-0. Sein Istwert ist durch die pinke Linie im Diagram dargestellt. Der Controller für das rechte Rad hat eine PID-Einstellung von 15-2-5. Sein Istwert ist durch die türkisene Linie im Diagram dargestellt.
Bis ca. t = 144 ist der pinkene Controller ein wenig näher am Sollwert als der Türkisene. Aber ab t = 144.5 überkorrigiert sich der pinke Controller ständig. Im Gegensatzt dazu ist der türkisene Controller ab t = 144.5 näher am Sollwert, da er nicht überkorrigiert. Der türkisene Controller hat einen D-Wert von 5. Dadurch überkorrigiert er nicht ständig sonder berührt die Sollwert Kurve eher.
Durch so Tests können PID-Parameter optimiert werden, sowohl für die Lenkung als auch für den Antrieb des Fahrzeugs.

3.5.2 Ackerman Lenkung

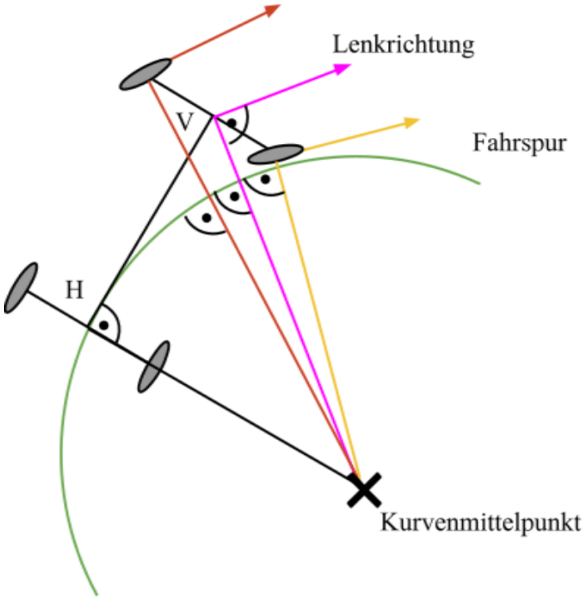
Ziel der Simulation ist es bei gegebenem Lenkwinkel und Geschwindigkeit das Fahrzeug entsprechend zu bewegen. Die Geschwindigkeit wird an die zwei Geschwindigkeitscontroller gegeben. Der Lenkwinkel wird an die zwei Lenkcontroller gegeben. Hierbei treten jedoch zwei Probleme auf:

Wenn sich beide Hinterräder in der genau gleichen Geschwindigkeit drehen, kann es zu Problemen in Kurven kommen. Das äußere Rad muss eine längere Strecke in der gleichen Zeit wie das innere Rad fahren. Dadurch kann es zu ungewünschten Kräften und Rutschen kommen. Im echten Fahrzeug wird dieses Problem durch ein Differentialgetriebe gelöst. In der Simulation hat sich das als kein großes Problem erwiesen und es bleibt erstmal bei zwei Controllern mit der genau gleichen Geschwindigkeit, auch in den Kurven.

Das zweite Problem ist die Lenkung. Wenn man den Lenkwinkel direkt an beide Controller weitergibt hat man eine Parallellenkung. Das Fahrzeug mit Parallellenkung sieht in einer Kurve so aus:

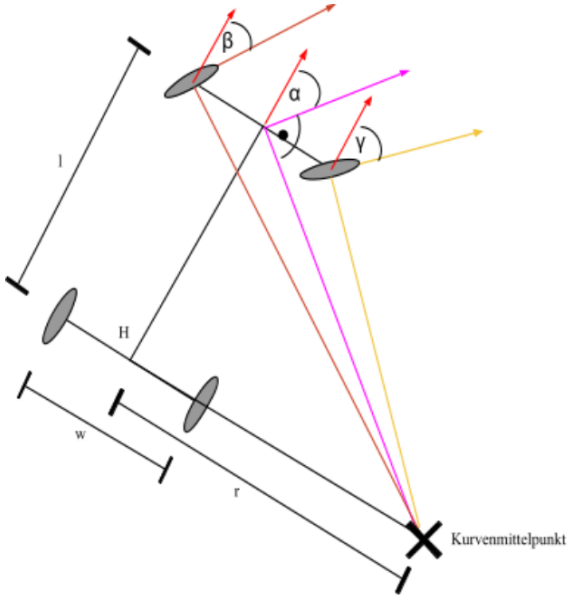


Das Fahrzeug fährt um den Kurvenmittelpunkt herum. Wenn ein Rad eine Kurve fährt, ist es so ausgerichtet, dass es entlang einer Tangenten auf dem Kreis um den Kurvenmittelpunkt fährt. Die Hinterräder des Fahrzeugs fahren um den Kurvenmittelpunkt. Die Vorderräder sind allerdings nicht entlang eines Kreises um den Kurvenmittelpunkt orientiert sondern parallel zur Lenkrichtung. Die Lenkrichtung geht aus vom Vorderachsenmittelpunkt V. Dies kann ebenfalls zu Rutschen und ungewollten Kräften auf Räder und Lenkung führen. Um das zu verhindern wird im echten Fahrzeug eine Ackermann-Lenkung verwendet. Diese sieht in einer Kurve so aus:



Im Gegensatz zu Parallellenkung sind nun auch die Vorderräder entlang einem Kreis um den Kurvenmittelpunkt ausgerichtet. Das linke Rad hat nun einen anderen Winkel wie das rechte Rad.

Wie die Ackerman-Lenkung, per Hardware, im echten Fahrzeug umgesetzt ist hier unwichtig. Um die Ackerman-Lenkung zu simulieren, können aus einem gegebenen Lenkwinkel, jeweils der Winkel für das rechte und linke Rad berechnet werden.



Ziel ist es die Winkel β und γ zu bestimmen. Gegeben ist der Lenkwinkel am Vorderachsenmittelpunkt α. Mit der Fahrtrichtung und α lässt sich die pinke Linie konstruieren. Außerdem ist gegeben dass die Hinterachse auf einer Geraden mit dem Kurvenmittelpunkt liegt. Der Schnittpunkt der pinken Linie und dieser Geraden ist der Kurvenmittelpunkt.

Die Strecke von H zum Kurvenmittelpunkt wird definiert als r. Die Strecke w ist der Radabstand zwischen den rechten und linken Rädern. Die Strecke l ist der Radabstand zwischen den vorderen und hinteren Rädern. Aus der Pinken Strecke und den Strecken r und l bildet sich ein Dreieck mit rechtem Winkel bei H. Daraus folgt

$$\tan(\alpha) = \frac{l}{r} \tag{3.1}$$

Wenn man den Punkt H um $\frac{w}{2}$ nach rechts und links verschiebt kann man zwei weitere rechtwinklige Dreiecke konstruieren. Diese beinhalten jeweils die braune oder gelbe Strecke. Aus den zwei Dreiecken gehen folgende Gleichungen hervor:

$$\tan(\beta) = \frac{l}{r - w/2} \tag{3.2}$$

$$\tan(\gamma) = \frac{l}{r + w/2} \tag{3.3}$$

Umformung Gleichung 3.1

$$r = \frac{l}{\tan(\alpha)} \tag{3.4}$$

Setzte man Gleichung 3.4 in Gleichung 3.2 ein so erhält man

$$\tan(\beta) = \frac{l}{\frac{l}{\tan(\alpha)} - w/2}$$

$$\iff \beta = \arctan(\frac{l}{\frac{l}{\tan(\alpha)} - w/2})$$

Analog dazu erhält man die Gleichung für γ

$$\gamma = \arctan(\frac{l}{\frac{l}{\tan(\alpha)} + w/2})$$

Mit diesen zwei Gleichungen lassen sich die Winkel β und γ bestimmen. Die Radabstände l und w sind fest in der Simulation definiert. Beide individuelle Lenkwinkel β und γ sind nur von einer Variablen abhängig: Dem Lenkwinkel am Vorderachsenmittelpunkt α. [9]

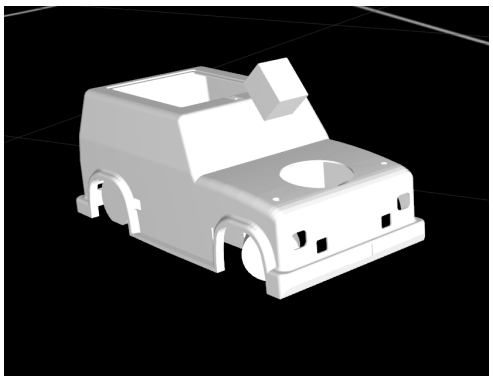
4 Ergebnis

4.1 Erfolge

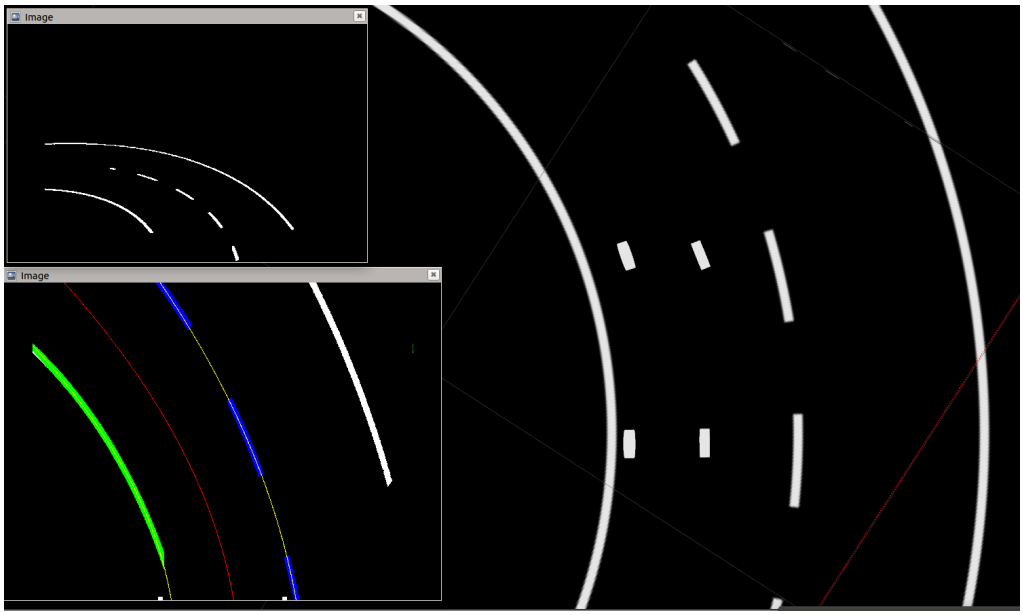
AutomaticDrive

Die Berechnung und Ausgabe des Lenkwinkels im AutomaticDrive Modus war sehr sinnvoll. Es lassen sich nun einfach Kamerabilder mit passenden Lenkwinkeln generieren. Um neue Algorithmen zu testen ist das sehr hilfreich. Es können auch datenhungrige, Machine Learning Algorithmen bedient werden, da sich die Daten schnell generieren lassen. Um solche Daten zu generieren ohne Simulation, muss man mit dem Fahrzeug im RC-Modus eine Strecke abfahren, oder man lässt die Fahrzeugsteuerung fahren. Dies ist deutlich zeitintensiefer.

Gesamtsimulation



Mit der dynamischen Fahrtsimulation, kann jetzt die Fahrtsteuerung



4.2 Herausforderungen

Testen neuer Hardware

Sensoren wie ToF und Lidar in der Simulation zu testen hat sich nicht gelohnt. Die Simulation verhält sich immer noch ein wenig anders als das echte Auto. Beim Integrieren neuer Hardware hat es sich als besser herausgestellt, Hardwaretests durchzuführen. Alles was neu in die Simulation kommt, muss in einem langwierigen Prozess immer näher an das Verhalten der echten Hardware angepasst werden.

Performance

Wenn sowohl die Fahrzeugsteuerung als auch die Simulation auf meinem Rechner laufen, ist der Rechner ganz ausgelastet. Wenn es zu wenige Resourcen gibt, berechnet die Fahrzeugsteuerung in größeren Zeitabständen einen Lenkwinkel. Dadurch kann das Fahrzeug nicht schnell fahren, da es sonst aus der Bahn fährt. Die Ursache der hohen Performancekosten ist allerdings nicht die Simulation sondern die Fahrzeugsteuerung.

Literatur

[1] DNA. *Der Regler*. 2020. URL: https://www.xplore-dna.net/mod/page/view.php?id=88.

[2] Gazebo. *Gazebo*. 2014. URL: http://gazebosim.org/.

[3] KITcar. *Gazebo Simulation Package*. 2020. URL: https://public-doc.kitcar-team.de/kitcar-gazebo-simulation/gazebo_simulation/index.html.

[4] KITcar. *Simulation Evaluation Package*. 2020. URL: https://public-doc.kitcar-team.de/kitcar-gazebo-simulation/simulation_evaluation/index.html.

[5] KITcar. *Simulation Grounndtrouth*. 2020. URL: https://public-doc.kitcar-team.de/kitcar-gazebo-simulation/simulation_groundtruth/index.html.

[6] Open Robotics. *ROS Introduction*. 2018. URL: http://wiki.ros.org/ROS/Introduction.

[7] Open Robotics. *ROS Topics*. 2021. URL: http://wiki.ros.org/Topics.

[8] Ricardo Angeli. *Joint Element*. 2018. URL: http://wiki.ros.org/urdf/XML/joint.

[9] Robert Eisele. *Ackerman Steering*. 2021. URL: https://www.xarg.org/book/kinematics/ackerman-steering/.

Anhang

(Beispielhafter Anhang)

A. Assignment

B. List of CD Contents

C. CD

B. List of CD Contents

- **Literature/**
  - **Citavi-Project(incl pdfs)/** ⇒ *Citavi (bibliography software) project with almost all found sources relating to this report. The PDFs linked to bibliography items therein are in the sub-directory 'CitaviFiles'*
    - bibliography.bib ⇒ *Exported Bibliography file with all sources*
    - Studienarbeit.ctv4 ⇒ *Citavi Project file*
    - **CitaviCovers/** ⇒ *Images of bibliography cover pages*
    - **CitaviFiles/** ⇒ *Cited and most other found PDF resources*
  - **eBooks/**
  - **JournalArticles/**
  - **Standards/**
  - **Websites/**
- **Presentation/**
  - presentation.pptx
  - presentation.pdf
- **Report/**
  - Aufgabenstellung.pdf
  - Studienarbeit2.pdf
  - **Latex-Files/** ⇒ *editable LaTeX files and other included files for this report*
    - **ads/** ⇒ *Front- and Backmatter*
    - **content/** ⇒ *Main part*
    - **images/** ⇒ *All used images*
    - **lang/** ⇒ *Language files for LaTeX template*