FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Physical and Graphical Simulation of an Ackermann Steered Vehicle

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Technical Physics

by

Eugen Kaltenegger
Registration Number 1125205

to the Faculty of Physics

at the TU Wien

Advisor: Markus Bader, Univ.Ass. Dipl.-Ing. Dr.techn.
Supervisor: Martin Gröschl, Ao.Univ.Prof. Dipl.-Ing. Dr.techn.

Vienna, 26th September, 2016

Eugen Kaltenegger

Markus Bader

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Physikalische und Graphische Simulation eines Fahrzeuges mit Ackermannsteuerung

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Technische Physik

eingereicht von

Eugen Kaltenegger
Matrikelnummer 1125205

an der Fakultät für Physik

der Technischen Universität Wien

Betreuung: Markus Bader, Univ.Ass. Dipl.-Ing. Dr.techn.
Supervision: Martin Gröschl, Ao.Univ.Prof. Dipl.-Ing. Dr.techn.

Wien, 26. September 2016

Eugen Kaltenegger

Markus Bader

Erklärung zur Verfassung der Arbeit

Eugen Kaltenegger
Haizingergasse 46/1, 1180 Wien, Austria

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 26. September 2016

Eugen Kaltenegger

Abstract

The Institute of Computer Aided Automation at the Technical University of Vienna founded an interest group for autonomous driving. The idea is to attract students as well as teachers to this topic and provide them with easy access to the required hardware and software. Therefore a fleet of autonomous robots will be built. A prototype based on a RC-model car equipped with a Raspberry Pi as computation unit and an Arduino Uno as a micro controller to control the vehicle was already created [5]. In the future, this robot will be extended with sensors to facilitate autonomous driving. In order to achieve this goal, validation of systems and algorithms is imperative.
For that purpose, a physical simulation and visualization of the prototype is created. The creation of the real and the simulated robot as well as their common interface was presented within a student paper [3] at the ARW (Austrian Robotics Workshop) 2016. The focus of this thesis is on the creation of this simulation and its accuracy. Since the primary interest of the research is in the motion of the robot, it is sufficient to only simulate the chassis. Specifically, the simulation of the steering which is the defining part of Ackermann robots is explained. ROS [7] (Robot Operating System) is used as middle ware for the simulation and visualization software [10]. A ROS node able to communicate with the simulation and visualization tool is used to control the simulated robot. This node is responsible for the steering, powering the wheels, and the necessary calculations. Additionally, it enables communication between the robotic tools used. On the one hand, the node receives motion commands and on the other hand it publishes velocity motion model [9, 12] based odometry data as well as the ground truth based odometry data.
Different ways to visualize these kinds of odometry data will be introduced. A ROS node is created to visualize the estimated pose and its uncertainty as well as the ground truth pose. A pose by definition holds both the position and the orientation of the robot. The comparative advantage of this node over previously existing nodes is the visualization of the pose uncertainty. This is required to determine whether the estimated pose is credible in comparison with the ground truth pose. Since the simulation and the tracking system aim to be used for self localization, this is a important issue.
Conclusively, the comparison of the velocity motion model odometry data with the ground truth odometry data will show that the simulation meets the requirements of accuracy with respect to the tracking model. Ways to improve the accuracy of the simulation will be introduced as well.

Kurzfassung

Das Institut für Rechnergestützte Automation an der technischen Universität in Wien gründete eine Forschungsgruppe für autonomes Fahren. Die Grundidee dieses Vorhabens ist es, Studenten wie auch Lehrkräften das Thema nahe zu bringen und ihnen einfachen Zugang zu Hardware und Software zu verschaffen. Dafür soll eine Flotte selbstfahrender Roboter gebaut werden. Ein Prototyp basierend auf einem RC-Model ausgestattet mit einem Raspberry Pi als Recheneinheit und einem Arduino Uno als Microcontroller, welche die Steuerung des Fahrzeuges übernehmen, wurde bereits gebaut[5]. Es ist geplant, diesen Roboter mit Sensoren zu erweitern um selbständiges Fahren in der Zukunft zu ermöglichen. Dafür ist die Überprüfung von diversen Algorithmen und Methoden unumgänglich. Aus diesem Grund wurde eine physikalische Simulation und Visualisierung des Prototypen erstellt. Der Aufbau des Fahrzeuges und dessen Simulation sowie deren gemeinsames Interface wurden im Zuge eines Student Papers[3] auf dem ARW (Austrian Robotics Workshop) 2016 präsentiert.
Diese Bachelorarbeit konzentriert sich auf die Erstellung der Simulation sowie ihrer Überprüfung. Da das Forschungsinteresse der Bewegung des Fahrzeuges gilt, ist es ausreichend, nur das Fahrgestell zu simulieren und visualisieren. Besondere Aufmerksamkeit gilt der Ackermann Steuerung des Fahrzeuges. ROS [7] (Robot Operating System) fungiert als Zwischenanwendung für die Software [10] die die Simulation und Visualisierung übernimmt. Ein ROS Node übernimmt die Lenkung des Fahrzeuges, den Antrieb der Räder und desweiteren alle notwendigen Berechnungen. Eben dieser Node ist für die Kommunikation der verwendeten Anwendungen verantwortlich. Einerseits empfängt er Bewegungskommandos, andererseits stellt er zweierlei Odometrie Daten zur Verfügung. Diese Odometrie Daten basieren zum einen auf dem Velocity Motion Model [9, 12], zum anderen auf der tatsächlichen Position und Orientierung des Fahrzeuges.
In dieser Arbeit werden verschiedene Wege, die Odometrie Daten darzustellen, vorgestellt. Um die berechnete Position und Orientierung des simulierten Fahrzeuges und dessen Unsicherheit darzustellen, wurde ein weiterer Node erstellt. Die Möglichkeit, die Unsicherheit von Position und Orientierung darstellen zu können, zeichnet eben diesen Node gegenüber anderen Nodes aus. Dies ist wichtig um die Glaubwürdigkeit der berechneten Daten im Vergleich zur tatsächlichen Position und Orientierung prüfen zu können. Abschließend wird anhand der Odometrie Daten des Velocity Motion Models und der Ground Truth Odometrie gezeigt, dass die Simulation den Ansprüchen auf Genauigkeit im Bezug auf das Tracking Model gerecht wird. Außerdem werden Möglichkeiten, die Genauigkeit der Simulation weiter zu steigern, vorgestellt.

Contents

CHAPTER 1

Introduction

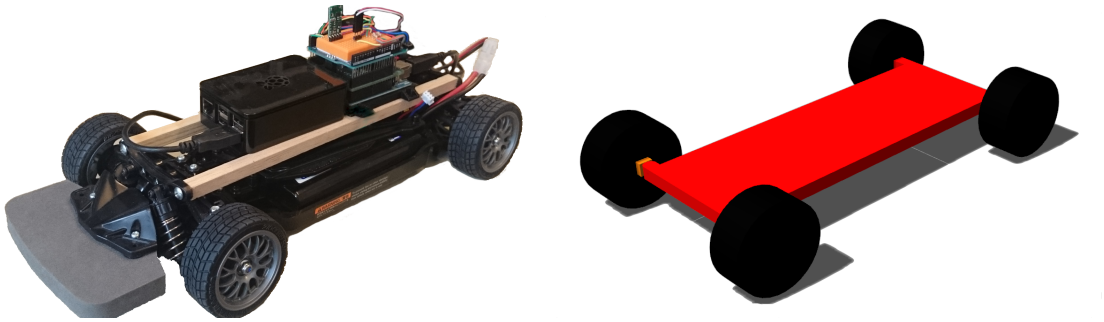
Autonomous driving is currently a research topic of automobile manufacturers like Volvo, Ford or Nissan and newcomers to the topic of automobiles such as Google [11]. At the DARPA Urban Challenge[1], universities like the Massachusetts Institute of Technology and the Stanford University presented their research accomplishments [8]. The Institute of Computer Aided Automation at the Technical University of Vienna founded a research group for autonomous driving robots as well. It targets students as well as researchers and provides them with the opportunity to conduct studies without needing the actual robot. The group intends to create a fleet of autonomous robots with Ackermann steering based on the prototype, see Figure 1.1a, created by a student in the group [5]. The Ackermann steering is a geometrical design commonly used in automobiles, pictures of an Ackermann steering will be shown in Chapter 3.
The prototype, a modified RC car, is equipped with a Raspberry Pi[2] and an Arduino Uno[3] to run the required software, read sensor data and control the actuators. The RC car as well as the Raspberry Pi and the Arduino Uno are cost efficient hardware.
A Raspberry Pi is a small computer used for computation of the robot and for running ROS [7] (Robot Operating System), see Subsection 3.1.1. An Arduino Uno is a micro controller used to control the actuators of the vehicle based on the commands it receives from the Raspberry Pi. One actuator is a BLDC (Brushless Direct-Current) motor, the other one is a servo motor.
The main software used to run this vehicle is the open source software ROS.
A simulation and visualization of this robots chassis is required to verify motion systems and motion algorithms. Simulation is a necessary development tool for several robotic tasks. This thesis presents the creation of such a simulation for robotic motions. It allows to analyze the correct functionality of motion systems and algorithms without using the

[1]DARPA Urban Challenge: http://archive.darpa.mil/grandchallenge/ (20.07.2016)
[2]Raspberry Pi: https://www.raspberrypi.org/products/ (23.04.2016)
[3]Arduino Uno: https://www.arduino.cc/en/Main/ArduinoBoardUno (23.04.2016)

real robot. The work presented in this thesis is extendable to multiple robot simulation as well as sensor simulation which is required for self localization. In order to achieve this, the real car and the simulation each calculate their estimated pose and its uncertainty based on the velocity motion model [9, 12]. A pose by definition holds both the position and the orientation of the robot.
The robot introduced above is simulated with the open source robotic simulation and visualization tool Gazebo. To keep the robot and its simulation compatible, Gazebo is used in combination with ROS. ROS is used to allow communication with the same sort of messages for the real and the simulated robot. This is necessary since the real and the simulated robot share a common interface.
The simulation only contains the robots chassis since this thesis focuses mainly on motion simulation and research. Simulating additional parts would not provide any useful impact on motion research but would exceed the computational effort.
In order to include the impact the non-simulated parts would have on the motion, their weight is taken into account. To create a model of the chassis two different file types may possibly be used, the first of which is URDF (Unified Robot Description Format) primarily used in association with ROS and the second of which is SDF (Simulation Description Format) primarily used in association with Gazebo, for more details see Section 3.1. Both are XML[4] (Extensible Markup Language) files which allow to describe robot models and their environment by defining their links, which specify bodys, and joints, which describe the connections of the links.
Even though URDF is compatible with ROS it can be rendered compatible with Gazebo if additional tags, which hold physical parameters like a mass and inertia of a link, are included. SDF is compatible with Gazebo and at this moment not supported by ROS. Because of this, URDF is the file format chosen for this project, even though SDF would provide benefits allowing easier simulation of the chassis. When using a URDF file for the robot model a plugin which simulates the mechanisms of Ackermann steering is required. In comparison, using a SDF file would allow to simulate the steering just by assembling links to an Ackermann steering with out an additional plugin. Since a plugin is required for communication with the simulation anyway, it is well within the scope of this work and does not require any additional programs. The reason why an Ackermann steering can not be operated using URDF without a plugin will be discussed in Subsection 3.3.1. The URDF file imported into Gazebo is shown in Figure 1.1b.
A ROS node which operates as a Gazebo plugin is responsible for the management of the simulated robot, the computation required for this task and for the tracking. The node controls the steering and powers the wheels with the same kind of messages that the real robot is controlled by. Therefore the steering angles have to be calculated and applied to the simulated vehicle. Additionally, the node fulfills the computation required for the tracking system of the simulated robot in the same way the real robot does.
Both the estimated pose and its uncertainty gained this way as well as the ground truth pose according to the simulation are published as ROS topics which are named buses hosted by this node. The format of this odometry data is the same that the real robot

[4]XML: https://en.wikipedia.org/wiki/XML (23.04.2016)

publishes. The node also hosts the topic from where the simulated vehicle receives its motion commands. This is necessary to ensure the simulations compatibility with different navigation nodes.
To show the accuracy of the robot model created this way the ground truth pose with the estimated pose and its uncertainty are compared. A node is created to visualize the ground truth pose during run time and the estimated pose with its uncertainty. Since this node lacks the possibility to visualize the trajectory, RViz and MATLAB are used for this task, see Section 3.5. RViz has the benefit of visualizing the trajectory during run time, but it can not visualize the uncertainty of a pose. MATLAB can visualize both the trajectory and the uncertainty of a pose, but it can not fulfill these tasks during run time.



(a) The robot aimed to simulate. (b) The simulation of the robot in Gazebo.

Figure 1.1: Comparison of the real and the simulated robot.

Two specific test are carried out to ensure the simulations accuracy in comparison with the tracking model used. MATLAB was chosen to visualize the tests because it is important to compare the difference between the calculated trajectories with the pose uncertainty of the simulation. The first test is driving a straight line and the second one a curve. The difference between the calculated and the simulated trajectory will be discussed and it will be shown that the simulation meets the requirements of accuracy. The same tests with the calculated trajectory and the trajectory of the real car are discussed in the thesis of another student [5]. Because of the close relation of the tests, the calculated, the simulated and the real trajectories are visualized.
The creation of the real and the simulated robot as well as their common interface was presented at the ARW (Austrian Robotics Workshop) 2016 within a student paper[3] . This thesis primarily treats the creation of the simulation and the knowledge required for it.
Firstly, the two robotic tools used and their main properties are introduced.
Secondly, required knowledge of the chassis and the approach to simulate it is described.
Finally, the quality of the simulation is ascertained based on the comparison of the calculated trajectory and the ground truth trajectory. It will be shown that the simulation meets the mandatory requirements while still leaving room for improvement. Some of

these improvements will be introduced.

CHAPTER 2

Related Work

This chapter is organized as follows. Firstly, robotic challenges leading the way for the development of robotics and specifically autonomous robotics are mentioned. Secondly, the basis for the interface of the robot and its simulation are introduced. Thirdly, different simulation tools as well as robots simulated with Gazebo are introduced.

2.1 Robotic Challenges

During the last years, many studies have been conducted in the field of autonomous driving [11, 4]. In the last ten years, competitions in various robotic disciplines were established. A famous operator of robotic challenges is DARPA (Defense Advanced Research Projects Agency) which hosts multiple challenges for different kinds of robots. The DARPA robotics challenge focuses on semi autonomous robots aimed to fulfill complex tasks in dangerous human engineered environments. The robotics challenge splits into three parts, a virtual robotics challenge, the VRC and two live hardware challenges. The DARPA grand challenge is a race of autonomous vehicles in different environments. The third grand challenge is known as urban challenge since it was the first grand challenge including an urban course. Universities like the Massachusetts Institute of Technology and the Stanford University participate in the DARPA challenges and present their research accomplishments [8].
Events of a smaller scale like the Freescale Cup[1] or the Carolo Cup[2] and others are created to interest students in participating. The task is to develop and build an autonomous car based on a RC car which has to pass some classic road traffic tasks as fast and accurate as possible. Conducting research for such challenges pushes the progress in robotics while also motivating students.

[1] Freescale Cup: https://community.freescale.com/welcome (23.04.2016)
[2] Carolo Cup: https://wiki.ifr.ing.tu-bs.de/carolocup/ (23.04.2016)

2.2 Interface

Both the prototype and the simulation share a common boundary, the so called interface, to exchange information with ROS. A typical ROS interface for differential drive and holonomic robots uses twist messages which, are geometry messages created by the OSRF (Open Source Robotics Foundation). For Ackermann drive robots these messages are not sufficient since they do not provide required information such as the acceleration and the jerk which are necessary for autonomous automobiles. Moreover, an Ackermann drive robot is not able to fulfill all motions that a differential drive or holonomic robot could. If twist messages are to be used for a differential drive robot, additional calculations are required.
The disadvantages of twist messages in use with Ackermann drive robots compelled the ROS Ackermann interest group[3] to create a new kind of message solely for Ackermann robots. Based on messages used by the ART (Austin Robot Technology) autonomous vehicle they developed the Ackermann messages for ROS which are used for the interface of both the real and the simulated robot. The ART messages[4] are part of the Marvin project of the University of Texas at Austin. Marvin is an unmanned ground vehicle based on a sports utility vehicle which participated at the DARPA challenge. Members of the Marvin-Team ported the software of the autonomous car to ROS and share it with the robotics community through the Ackermann interest group. This software is no longer supported since Marvin was retired after many years of operation.
Beside the ART and the Ackermann messages package the Ackermann interest group also created an Ackermann QT[5] operation package and a HKS operation package for the HKS game controller.
An alternative Interface not based on ROS could be created with Player[6]. Player is a cross-platform robotic network interface to a variety of sensors and hardware. Like the interface created for the Ackermann robot fleet, Player also allows to control different vehicles via the same interface. Adept Mobilerobots[7] Pioneer2 as well as RWI (Real World Interface) drivers use the same Player interface to control robot movement, to name a few examples.
Both ROS and Player are compatible with multiple simulation tools, some of which will be mentioned in the following section.

2.3 Simulation

Robotic simulations are used to create and verify embedded applications without the use of the real robot. Thus, robotic simulations save time and money in the development of robotic systems. For that reason multiple robotic simulation tools with a different focus

[3] Ackermann Interest Group: http://wiki.ros.org/Ackermann Group (17.03.2016)
[4] ART Messages: http://wiki.ros.org/art_nav (23.04.2016)
[5] QT Library: https://www.qt.io/ (23.04.2016)
[6] Stage Project: http://playerstage.sourceforge.net/ (17.03.2016)
[7] Adept Mobilerobots: http://www.mobilerobots.com/ (23.04.2016)

each developed.
A few commonly used simulation tools are:

- Stage, a two dimensional simulation software for mobile robots, sensors and objects. Stage is frequently used with the Player network server.
- Gazebo[8] [10], a three dimensional open source simulation and visualization tool for the robotic middle ware ROS. Gazebo was developed to be fully compatible with the Player device server [10] from the beginning. This Simulation tool is used for the work presented within this thesis.
- V-REP[9], a three dimensional cross platform simulation tool similar to Gazebo.
- Webots[10], a three dimensional simulation and visualization tool liable to pay and compatible with ROS.

DARPA recognized the importance of simulations in the growing field of robotics and dedicated one third of the DARPA robotic challenge to robotic simulation known as VRC (Virtual Robotics Challenge). DARPA announced that the standardized simulation environment is based on Gazebo [1][2]. This has pushed the focus of the robotic community onto Gazebo.
The humanoid robot Atlas[11], created by Boston Dynamics, is used by multiple teams for the DARPA robotics challenge. A tutorial[12] showing how to spawn this robot into a Gazebo world is available online.
Other robots supported by Gazebo are the previously mentioned Pioneer2 and the Pioneer3AT, which are both research robots. The Pioneer2 is a differential drive and the Pioneer3AT a four wheel drive robot. Their simple shapes allow to easily create models of them using boxes and cylinders [10]. At the Vienna University of Technology, more advanced models containing meshes and simulations of their sensors are used for research and education.
An example for a commercial robot simulated with Gazebo is Robotnik's Autonomous mobile robot called AGVS[13], which stands for automated guided vehicles. AGVS is a four wheeled transport robot running ROS. A lot of the software used for this robot is shared through a package[14] on the website of the Ackermann interest group. This package includes an model of the robot, multiple messages for navigation and path planning as well as SLAM (Simultaneous Localization and Mapping) tools. To ensure detailed simulation and especially visualization, the robot model is created with URDF and mesh

[8]Gazebo: http://gazebosim.org/ (17.03.2016)
[9]V-REP: http://www.coppeliarobotics.com/index.html (17.03.2016)
[10]Webots: https://www.cyberbotics.com/overview (17.03.2016)
[11]Boston Dynamics Atlas: http://www.bostondynamics.com/robot_Atlas.html (23.04.2016)
[12]Gazebo with URDF: http://gazebosim.org/tutorials at Connect to ROS - URDF in Gazebo (23.04.2016)
[13]AGVS Robot: http://www.robotnik.eu/mobile-robots/autonomousagvs/
[14]ROS AGVS: http://wiki.ros.org/agvs (23.04.2016)

files. AGVS is equipped with a front and a rear laser range finder for SLAM and safety reasons.
These are only a few of the robots simulated with Gazebo. There is a long list of different kinds of robots simulated with Gazebo but mentioning them all would be beyond the scope of this chapter.

CHAPTER 3

Approach

The following chapter is about the creation of the Ackermann robot simulation. Therefore the necessary functions of the robotic tools ROS and Gazebo used to achieve this will be explained. Thereby the model description formats URDF and SDF will be introduced. Also, the Ackermann steering and its geometrical basics are described. The main focus is on the implementation of the Ackermann steering into the simulation tool Gazebo in usage of URDF in combination with ROS. After that, the basis for the calculation of odometry data will be defined along with ways to visualize the odometry data.

3.1 Facility

The work presented within this thesis is software based, therefore the used software facilities will be introduced here. Both main software components ROS and Gazebo are projects of OSRF (Open Source Robotics Foundation). The defined mission of OSRF with respect to ROS and Gazebo is to '...support the development, distribution, and adoption of open source software for use in robotics research and education...' according to their website[1].

3.1.1 The Robot Operating System ROS

The open source software ROS [7] is a flexible platform for robotics. Since ROS Jade Turtle[2] is used for the robot, it is also used for its simulation to ensure their compatibility. A ROS node is an executable that can fulfill calculation and computation tasks. ROS nodes can communicate by services in terms of request and replay communication or by topics where the nodes which are named buses can publish and receive messages. Types of messages used for this project are:

[1] ORSF: http://www.osrfoundation.org/ (17.03.2016)
[2] http://wiki.ros.org/jade (20.07.2016)

- Ackermann Messages[3]
- Geometry Messages, specifically Twist Messages[4]
- Navigation Messages, specifically Odometry Messages[5]

These messages are included in packages which are free to download. Beside the packages, many nodes are free to download, one of them being RViz[6] (ROS Visualizaton) which is a useful visualization tool. Nodes can be written in Python which is supported by the client library rospy, or in C++ which is supported by the client library roscpp, the latter is used exclusively for nodes created within this project. An important feature of ROS are publishers which are used to publish messages into topics and subscribers which are used to receive messages from topics. The publisher and subscriber system allows to combine different nodes by communicating via messages like the ones mention above.
An essential executable is `roscore`, which ensures the possibility for all other nodes to communicate by tracking nodes to topics as well as services.
An important reason for the use of ROS is its ability to communicate with the simulation and visualization tool Gazebo.

3.1.2 The Simulation and Visualization Tool Gazebo

In 2015, Gazebo established itself as a stand alone open source software for robot simulation and visualization. Since its emancipation, Gazebo no longer has any direct ROS dependencies, but it can still be integrated into ROS. ROS Jade Turtle, which is the version used for the robot and its simulation, works with the Gazebo 5 series due to the installation of the Gazebo ROS package[7]. The interaction between ROS and Gazebo is achieved using ROS messages and services.
Gazebo provides a three dimensional robot physics simulation and visualization based on the open source library ODE[8] (Open Dynamics Engine). It offers the ability to simulate realistic sensor feedback and physically plausible interactions between objects. Additionally Gazebo provides a GUI (Graphical User Interface), which allows to change simulation parameters and to manipulate the robot as well as the environment. Gazebo splits into three executables:

- The `gzserver` which is responsible for the physical simulation and the generation of sensor data. By default this command starts a world which only contains a solid infinite flat surface, known as empty world.

[3] Ackermann Messages: http://wiki.ros.org/ackermann_msgs (17.03.2016)
[4] Geometry Messages: http://wiki.ros.org/geometry_msgs (17.03.2016)
[5] Navigation Messages: http://wiki.ros.org/nav_msgs (17.03.2016)
[6] RViz: http://wiki.ros.org/rviz (17.03.2016)
[7] Gazebo Ros Package: http://wiki.ros.org/gazebo_ros_pkgs (17.03.2016)
[8] Physics Engine ODE: http://www.ode.org/ (20.07.2016)

- The `gzclient` which is responsible for the three dimensional visualization and the GUI.
- The executable `gazebo` which runs `gzserver` as well as `gzclient`.

Since the visualization of Gazebo is computationally expensive, it is reasonable to run the visualization only when it is required. Since it hosts the necessary topic the `gzserver` is the essential executable for the communication between ROS and Gazebo, see Figure 3.1.

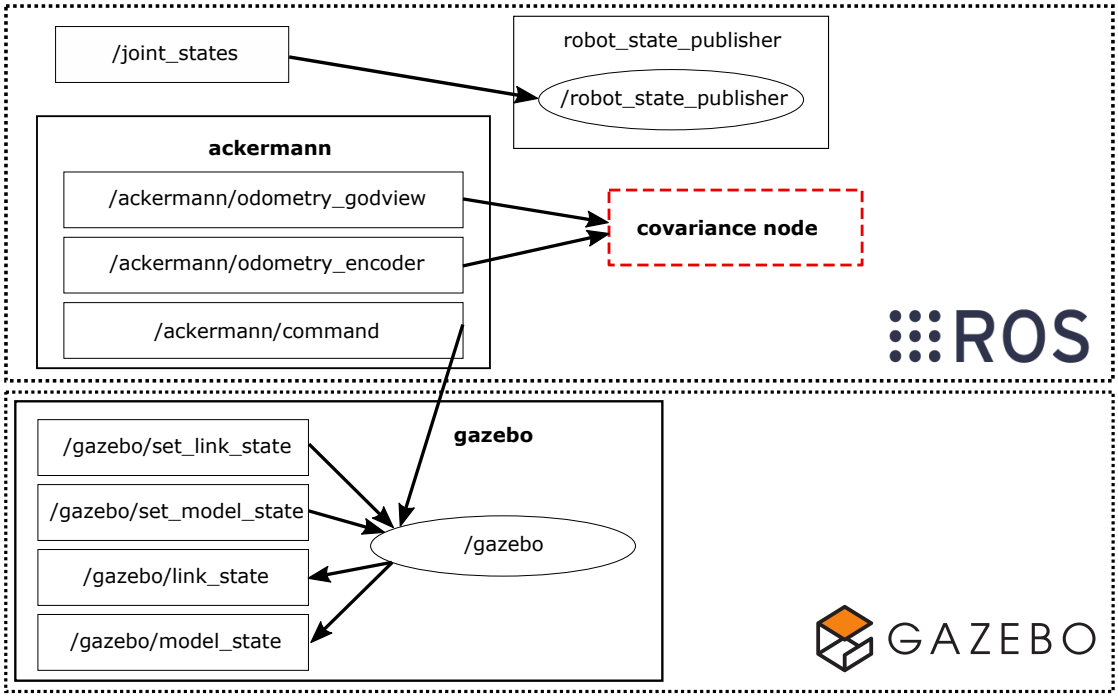


Figure 3.1: Diagram of the connection between the different Gazebo and ROS nodes and topics. The bold letters indicate name-spaces.

The GUI (Graphical User Interface) allows to change physics and computation parameters, like the real time update and the maximum step size, while the `gzserver` is running. Another feature of the GUI is to create and modify both the simulated environment as well as the robot. For this project, the robot is created and imported to Gazebo with an XML (Extensible Markup Language) file.

3.1.3 URDF and SDF

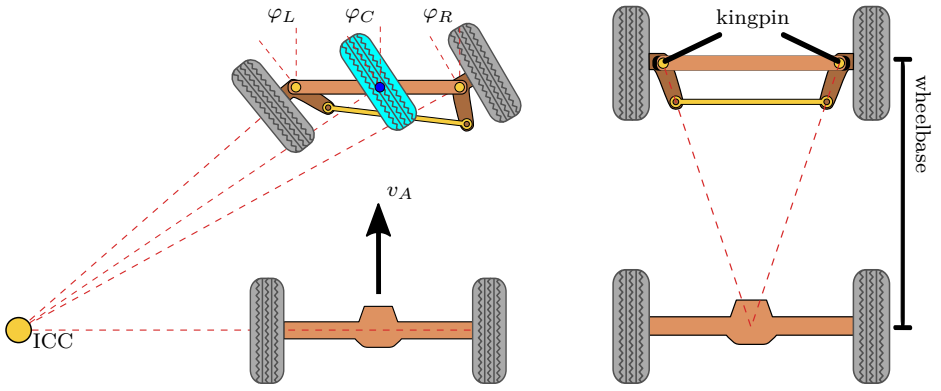
URDF[9] (Unified Robot Description Format) as well as SDF[10] (Simulation Description Format) are XML files used to define a robots structure. URDF and SDF both use links,

[9]URDF: http://wiki.ros.org/urdf (23.04.2016)
[10]SDF: http://sdformat.org/spec (23.04.2016)

which are bodies, and joints, which are link connections, to define the simulations visuals and collision, but differ in their use of additional tags.
URDF is created to be used in combination with ROS and consists of a number of different ROS packages. It is possible to use URDF files with Gazebo if the URDF file is extended with additional physics tags, see Section 4.3. Additional ROS packages like RViz and Graphviz can be used to visualize URDF models and their structure.
SDF is the file format created for Gazebo describing robots and their environment. In order to use SDF files with ROS they need to be converted into URDF files first. Therefore the chassis simulation is created with an URDF file.
A graphic created with Grapviz illustrating the URDF file is shown in Figure 3.3.

3.2 Ackermann Robot

In order to do research on autonomous automobiles, it is necessary to guarantee the equality of their motions with those fulfilled by the robot and its simulation. Therefore, an Ackermann steering similar to the one found in automobiles is implemented in the robot. In the further, robots with an Ackermann steering will be mentioned as Ackermann drive robots in relation to the differential drive robots. To understand the strategy of simulating the Ackermann drive robot, a closer look at the geometry of steered vehicles and specifically the Ackermann steering is necessary.



(a) The ICC of the Ackermann steering.

(b) The geometry of the Ackermann steering.

Figure 3.2: Schematic illustration of the Ackermann steering[11].

The perpendicular line to each wheel of a steered vehicle should intersect at one point to avoid skidding wheels. The point of intersection is the ICC (Instantaneous Center of Curvature). The curve radius for each wheel is defined by the distance between the wheel and the ICC. For Ackermann drive robots, a curve radius for the whole vehicle can

[11] Ackermann Illustrations: https://commons.wikimedia.org/wiki/File:Ackermann.svg (17.03.2016)

be defined by adding an imaginary third front wheel, centered between the two real front wheels. This curve radius will be defined as R_C and the dedicated steering angle φ_C. The curve radii for the wheels can assume positive and negative values depending on the chosen coordinate system. Usually, a Cartesian coordinate system with x in the facing direction of the robot and y to its left is defined. Using this coordinate system, a left turn assumes a positive and a right turn a negative radius. In case of driving straight forward, the curve radius and thus the ICC goes to plus or minus infinity. If the ICC is plus or minus infinity the perpendicular lines on all wheels are parallel.
These geometrical fundamentals are considered by the Ackermann steering, see Figure 3.2a. Therefore, the Ackermann steering uses an arrangement of four links to ensure the correct steering angle for the front wheels, see Figure 3.2b.
The steering angle has a maximum and minimum value that causes the smallest possible curve radius for each wheel to be higher that zero. This in turn makes it impossible for Ackermann drive robots to spin on one spot, unlike differential drive robots. To render the Ackermann drive robots compatible with differential drive robots, a conversion between the used motion commands will be introduced. This is important for when systems or algorithms created for differential drive robots are used with Ackermann drive robots. For the conversion, the knowledge of the structure of the motion commands used for Ackermann drive robots and differential drive robots is required. Ackermann drive robots receive Ackermann messages which hold a velocity and a steering angle. These messages will be defined as $\mathbf{u_A}$, see Equation 3.1.

$$\mathbf{u_A} = \begin{pmatrix} v_A \\ \varphi_C \end{pmatrix} \tag{3.1}$$

Differential drive robots commonly use twist messages, which are geometry messages. Twist messages can hold six parameters, three linear velocities and three angular velocities. A differential drive robot has three DOF (Degrees of Freedom) in a two dimensional space but only two parameters are required to define its motions. Thus, one linear velocity and one angular velocity of the twist messages is used. These are a linear velocity v_D in x direction and a angular velocity ω_D around the z axis. This kind of twist messages will be defined as $\mathbf{u_D}$ because of their usage for differential drive robots, see Equation 3.2

$$\mathbf{u_D} = \begin{pmatrix} v_D \\ \omega_D \end{pmatrix} \tag{3.2}$$

The linear velocity of the Ackermann drive motion command equals the linear velocity of the differential drive motion command, see Equation 3.3.

$$v_A = v_D \tag{3.3}$$

Because of the equality of these velocities, they will be handled as one velocity v. To calculate the steering angle of the Ackermann drive motion command, first the curve radius needs to be calculated, based on the differential drive robots motion command.

For that, the knowledge of the wheelbase w_{wb} is required.

$$\varphi_C = \arctan\left(\frac{w_{wb}}{R_C}\right) \quad \text{where} \quad R_C = \frac{v_D}{\omega_D} \tag{3.4}$$

Equation 3.4 allows values for φ_C ranging from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$. A common steering does not allow such a big range for the steering angle. If φ_C is higher than the maximum given by the vehicles steering, or lower than the minimum given by the vehicles steering, φ_C will be set to the maximum or minimum value.
Based on the knowledge presented within this section, the URDF file and an executable for the simulation are created.

3.3 Simulating the Robot

The simulation of the robot is designed to test motion algorithms and systems for the Ackermann robot. Therefore, only the parts vital for the robots motion are simulated and visualized, as seen in Figure 1.1b:

- The wheels establish physical contact between the robot and its environment. To keep the computational effort low, the wheels are approximated by cylinders. The friction parameter of the wheels is approximated to result in the expected behavior.
- The Ackermann steering, to ensure the correct steering angle of the left and the right front wheel. The maximum and minimum steering angle are approximated to $\pm 0.4\,\text{rad}$. These φ_{MAX} and φ_{MIN} values allow the simulated robot to have the same minimal curve radius as the real robot.
- A simple base link to visualize a connection between the before mentioned parts. The base link also holds the weight of the parts that are not simulated, because contrary to their function, their weight is essential for the motion. The total weight of the prototype is $2.5\,\text{kg}$.

When simulating the above mentioned parts, steering is the focus point and can be difficult when using URDF.

3.3.1 Four Linkage Problem

During the first attempt to simulate the Ackermann steering, a closed loop of four links connected with joints was created as shown in Figure 3.2b. The parent-child structure of URDF joints makes is impossible to create such a loop. A joint connects one parent link with one child link. If a second parent link is defined for a child link, only the connection to the first parent link will be considered.
This allows only treelike structures of links connected with joints, where the joints are branches and links are contact points, as shown in Figure 3.3. The mimic function of joints can solve this problem for loops created with equal sized opposite links. Since the

Ackermann steering requires an isosceles trapezoid, see Figure 3.2b, this solution is not usable for this project. A SDF file would allow to create such a loop but SDF is not compatible with ROS and therefore no solution for this problem either. Because of this, another way to simulate the Ackermann steering is needed.

3.3.2 Plug-in

A ROS node, functioning as Gazebo plug-in, is an executable which can read and manipulate Gazebo and will be referred to as plug-in in this thesis. The plug-in is used to control the Ackermann steering and power the wheels according to the motion command it receives. This is achieved by reading and manipulating joints.
The plug-in handles the calculation of the estimated pose according to the velocity motion model, see Subsection 3.4.1. It is also responsible for hosting topics to receive Ackermann messages and to publish both the velocity motion model odometry messages and the ground truth odometry messages.
Since the URDF file does not have to simulate the Ackermann steering, the kingpins as shown in Figure 3.2b, are the only simulated part of the steering. The remaining simulation of the Ackermann steering is fulfilled by the plug-in. The structure of the resulting URDF file looks is shown in Figure 3.3
Based on the motion commands the plug-in receives, it calculates the steering angle for the left φ_L and the right φ_R front wheel. The radii of both front wheels differ from the curve radius of the imaginary third front wheel by the offset of their kingpin with respect to the imaginary wheel. In the following, the distance between the left and the right kingpin will be mentioned as kingpin width w_{kp}. It is important not to use the track width instead of the kingpin width, because this causes an offset in the steering angle, which will cause an offset between the estimated and the ground truth pose. In Equation 3.5 the calculation of both steering angles is shown.

$$\varphi_L = \arctan\left(\frac{w_{wb}}{R_C - \frac{w_{kp}}{2}}\right) \quad \text{and} \quad \varphi_R = \arctan\left(\frac{w_{wb}}{R_C + \frac{w_{kp}}{2}}\right) \tag{3.5}$$

These calculated steering angles are applied to the kingpins by the plug-in. This motion is controlled by the 'P' part of a 'PID' controller. The controller causes the steering to act as expected, but this is a simplification since the real robots steering is powered by a servo motor. Contrary to the real robot, the simulated steering velocity is not constant. This simplification can be applied because the steering velocity is fast compared to the velocities driven by the car. Since the update rate of the plug-in is $100Hz$, the calculation is fast enough to avoid visible and consequential overdrive.
The rear wheels are simplified as well, as the plug-in does not consider their different curve radii. In the real robot, this is considered using a differential between the left and the right wheel. The four linkage problem of URDF, explained in Subsection 3.3.1, does not allow to simulate such a differential. A differential would require its own plug-in, but this would be beyond the scope of this thesis. To decrease the consequences of the simplification, only the simulated robots rear wheels are powered. The real robot has all

wheels powered.
To measure the accuracy of the simulated robots motions with this plug-in, motions of comparison are required. Therefore, a tracking system is used.

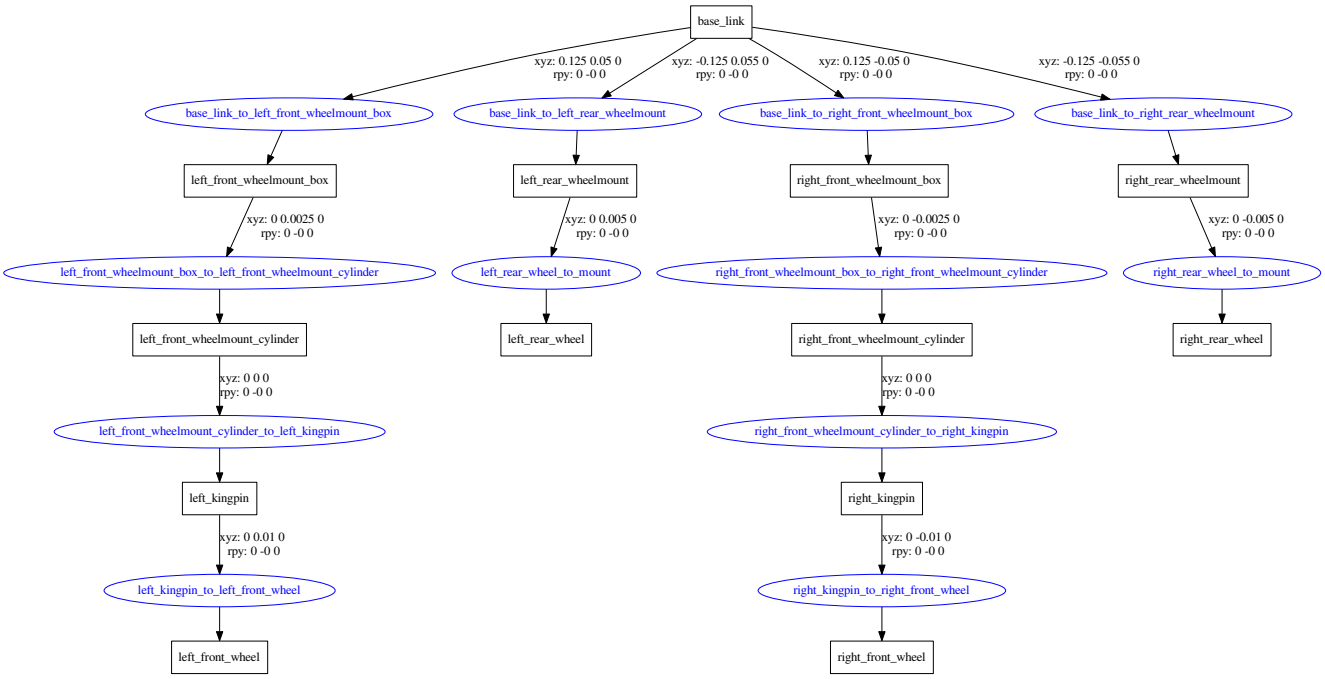


Figure 3.3: The structure of the URDF file used to simulate the robot. Links are visualized with black boxes and joints with blue ellipses. The coordinates of the joint in respective of the origin of its parent link are written above the joints.
What this file looks like in Gazebo is shown in Figure 1.1b.

3.4 Odometry

Robot motions are error afflicted because of factors like wheel slips, bumps or inaccuracies within the robot. This causes the estimated robots pose to differ from the ground truth pose. The robot simulation presented within this thesis is aimed for self localization and path planning, whereby both benefit from an accurate pose estimation. Different ways to calculate the estimated pose will be introduced in the following.

3.4.1 Velocity Motion Model

The velocity motion model is a tracking system to calculate the estimated pose of a robot based on the motion commands it receives. The kind of velocity motion model as

ascribed to Thrun [9] is designed for differential drive robots. Its simple structure allows for it to be used with the transformation defined in Section 3.2.
In the following a two dimensional space will be assumed. In this case the pose holds three parameters, see Equation 3.6.

$$\mathbf{x}_t = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} \tag{3.6}$$

A motion command for an Ackermann steered vehicle is shown in Equation 3.7.

$$\mathbf{u} = \begin{pmatrix} v \\ \varphi \end{pmatrix} \tag{3.7}$$

The velocity motion model recursively calculates the estimated pose and its uncertainty which is represented by a covariance matrix P_t, see Equation 3.8.

$$P_t\left(Cov\left(\mathbf{x}_t\right)\right) = \begin{pmatrix} Cov\left(x,x\right) & Cov\left(x,y\right) & Cov\left(x,\theta\right) \\ Cov\left(y,x\right) & Cov\left(y,y\right) & Cov\left(y,\theta\right) \\ Cov\left(\theta,x\right) & Cov\left(\theta,y\right) & Cov\left(\theta,\theta\right) \end{pmatrix} \tag{3.8}$$

The uncertainty of the robots pose is a consequence of motion errors. The robots pose at any time t is a function depending on the previous pose and the motion commands, see Equation 3.9 .

$$\mathbf{x}_t\left(\mathbf{x}_{t-1}, \mathbf{u}\right) = \begin{pmatrix} x_t = x_{t-1} + v \cdot \cos\left(\theta_{t-1}\right) \cdot \Delta t \\ y_t = y_{t-1} + v \cdot \sin\left(\theta_{t-1}\right) \cdot \Delta t \\ \theta_t = \theta_{t-1} + \frac{v \cdot \tan(\varphi)}{w_{wb}} \cdot \Delta t \end{pmatrix} \tag{3.9}$$

The initial pose and its covariance matrix need to be defined. The values in the diagonal of the initial covariance matrix must be different from zero. In case of sure placement, these values can be much smaller than one but need to be greater than zero. The change of the pose is represented by the Jacobian matrix G, which is the derivative of the state $\mathbf{x}_t$ with respect to the pose $\mathbf{x_{t-1}}$, see Equation 3.10.

$$G = \frac{\partial \mathbf{x}_t\left(\mathbf{x}_{t-1}, \mathbf{u}\right)}{\partial \mathbf{x}_{t-1}} = \begin{pmatrix} 1 & 0 & -v \cdot \sin\left(\theta_{t-1}\right) \cdot \Delta t \\ 0 & 1 & v \cdot \cos\left(\theta_{t-1}\right) \cdot \Delta t \\ 0 & 0 & 1 \end{pmatrix} \tag{3.10}$$

The Jacobian matrix V is the derivative of the pose x_t with respect to the motion command u and equals the change of the motion, see Equation 3.11.

$$V = \frac{\partial \mathbf{x}_t\left(\mathbf{x}_{t-1}, \mathbf{u}\right)}{\partial \mathbf{u}} = \begin{pmatrix} \cos\left(\theta_{t-1}\right) \cdot \Delta t & 0 \\ \sin\left(\theta_{t-1}\right) \cdot \Delta t & 0 \\ \frac{\tan(\theta_{t-1}) \cdot \Delta t}{w_{wb}} & \frac{v \cdot \Delta t}{w_{wb} \cdot \cos^2(\theta_{t-1})} \end{pmatrix} \tag{3.11}$$

For the matrices G and V, it is important to consider the case of velocities being smaller than zero which equals driving backwards. This case can be treated as turning the vehicle

around and assuming to drive forward. Mathematically, this amounts to shifting the parameter θ by $\frac{\pi}{2}$.
The error matrix M considers the effect of motion errors, while the four parameters α_1 to α_4 weight the motion noise, see Equation 3.12.

$$M = \begin{pmatrix} \alpha_1 v^2 + \alpha_2 \varphi^2 & 0 \\ 0 & \alpha_3 v^2 + \alpha_4 \varphi^2 \end{pmatrix} \tag{3.12}$$

The introduced matrices provide enough information to calculate the covariance matrix for every time step.

$$P_t = G \cdot P_{t-1} \cdot G^T + V \cdot M \cdot V^T \tag{3.13}$$

The first term of Equation 3.13 represents the prediction step and the second term the uncertainty in accuracy of the motion. The calculation made with Equation 3.10 to Equation 3.13 equals the prediction step of a Kalman filter used for mobile robotics [9, 12]. The normalized eigenvectors of the x and y dependent submatrix of P_t, weighted by their eigenvalues define the orientation and size of an ellipse. This ellipse represents the area where the robot might be corresponding to the used α values. Without any correction, the covariance ellipse will only grow whenever the robot receives motion commands. For any kind of pose correction, sensor input is needed, but this is beyond the content of this work.
The velocity motion model discussed thus far is used for the simulation and the real robot. Its accordance to the simulation of the real robot will be discussed in Section 5.2. One issue in the velocity motion model is that only motions caused by a motion command are considered.

3.4.2 Odometry Motion Model

The velocity motion model is a tracking system which regards all motions carried out by wheels and steering. Therefor, the odometry motion model [9] commonly uses wheel encoders and in the case of Ackermann drive robots also a steering encoder to measure the velocity and the steering angle. A relevant alternative to wheel encoders is a Hall sensor in the BLDC motor, which is built into the prototype. The calculation for the odometry motion model is the same as for the velocity motion model, see Subsection 3.4.1, but they differ in the motion $\mathbf{u}$ used for this calculation. Thus, the odometry motion model can regard rolling caused by the kinetic energy of the robot but drifting and slippage are not considered. Nevertheless, it does not provide any kind of pose correction.

3.4.3 Kalman Filter

The Kalman filter is a mathematical method to improve error afflicted predictions. In mobile robotics, the Kalman filter is used for pose estimation and self localization. Therefor, the Kalman filter splits into two parts. Firstly, the prediction step, which calculates the estimated pose of the robot based on the motion commands it receives. This equals the calculations presented in Subsection 3.4.1. Secondly, the correction step,

which improves the estimated pose based on sensor input. Thus, the robot needs sensors, which will be attached in further work for the usage of a Kalman filter.
It is useful to visualize the odometry data gained with any of the before mentioned motion models.

3.5 Odometry Visualization

Three programs are used to visualize odometry data. They will be introduced in the following.
Firstly, the ROS package RViz which is a useful visualization tool for different kinds of messages and URDF files, see Figure 3.4. RViz allows to display a number of poses, which enables the visualization of a driven trajectory. The disadvantage of RViz is that it is not possible to visualize the covariance ellipse. There is a package[12] which claims to provide this but when it was used, bugs caused the covariance ellipse to be drawn incorrectly. Odometry visualization with RViz is useful to test the accuracy of the simulation empirically.

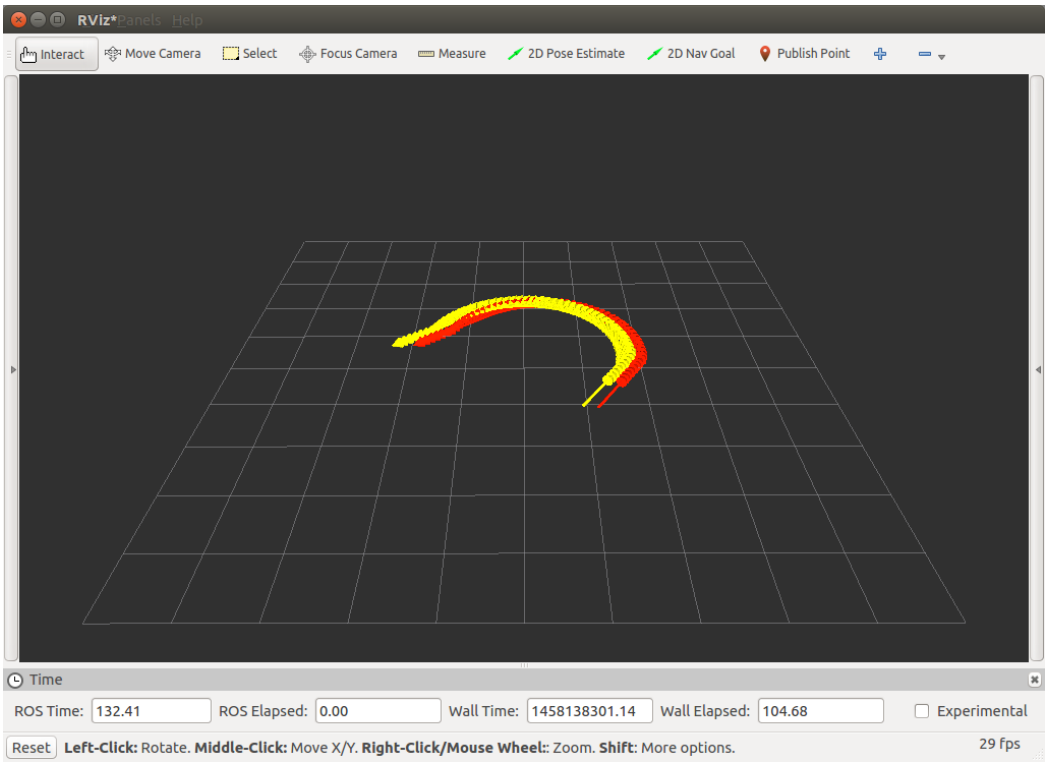


Figure 3.4: A screen-shot of RViz where two trajectories are compared. The arrows represent the estimated position as well as the estimated orientation at different time steps. It is not possible to visualize the pose uncertainty with this tool yet.

[12] RViz Covariance Plug-in: http://wiki.ros.org/rviz_plugin_covariance (17.03.2016)

Secondly, a node able to visualize the estimated pose, its covariance ellipse and the ground truth pose is created, see Figure 3.5. This node is created to assume the task of displaying the covariance ellipse, therefor it will be mentioned as covariance node. It is based on the geometry library used for the 'Mobile Robotics' lecture[13] at the Technical University in Vienna. The geometry library in turn uses the OpenCV[14] library for the display window. The disadvantage of the covariance node is that it is unable to display the driven trajectory. The created node is used to display the covariance and check its plausibility.

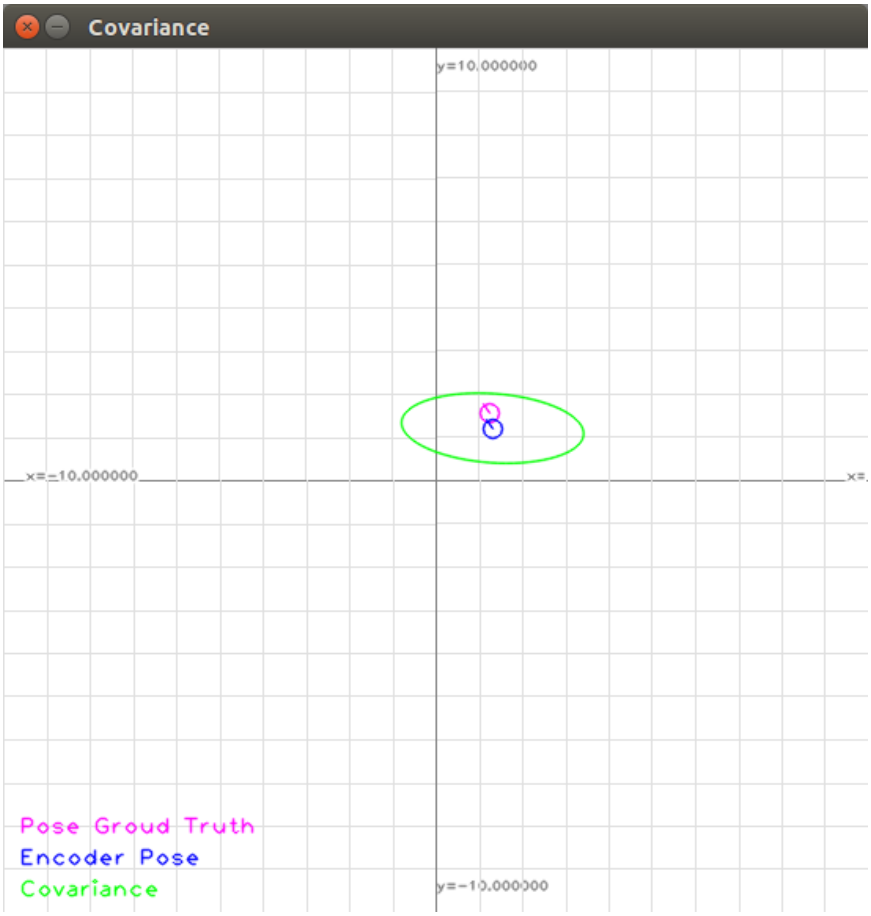


Figure 3.5: A Screen-shot of the covariance node where the position of the estimated pose and its uncertainty as well as the ground truth pose are visualized.

Finally, MATLAB[15], which is a common tool for multiple mathematical applications is mentioned. The tests presented in Chapter 5 are visualized with MATLAB, see Figure 5.1

[13]Mobile Robotics (LAV No.: 183.660) to find at: https://tiss.tuwien.ac.at/ (17.03.2016)
[14]OpenCV: http://opencv.org/ (17.03.2016)
[15]MATLAB: http://de.mathworks.com/ (17.03.2016)

and Figure 5.2. The advantage of MATLAB is its ability to display the trajectory as well as the covariance. Its disadvantage is that unlike RViz and the covariance node, it cannot display the trajectory or the covariance during the run time of the simulation.

CHAPTER 4

Applied Physics Engine

A core element of the simulation tool Gazebo is the physics engine. A physics engine is a means to apply physical properties to software. The goal is to enable the software to consider the relevant properties for a specific simulation.
Gazebo supports different physics engines for example Bullet Physics[1], DART[2] (Dynamic Animations and Robotic Toolkit) and ODE[3] (Open Dynamics Engine), which is its default physics engine. On one hand the physics simulation depends on the chosen physics engine on the other hand it depends on the used file format to define the robots structure. In the following the physical simulation carried out with ODE and URDF will be discussed.

4.1 Friction

Friction is a force that appears when two bodies share a common contact surface. Different types of friction are defined according to their source. For this simulation dry friction as well as rolling friction are regarded.
Dry friction can be subdivided into static friction and kinetic friction. Static friction force F_S appears between bodies with a common contact surface that are not moving relative to each other. Kinetic friction force F_N appears between two bodies which move relative to each other along their common contact surface. The size of the contact surface does not influence the friction force in either one of both cases. The dimensionless coefficients for friction are usually denoted as mu_S for static friction and mu_K for kinetic friction. These friction coefficients can differ for various directions of movement of the bodies against each other. The static friction and the kinetic friction forces which depend on

[1]Bullet Physics: http://bulletphysics.org/wordpress/ (22.08.2016)
[2]Physics Engine Dart: https://dartsim.github.io/ (22.08.2016)
[3]ODE Physics: http://www.ode.org/ (20.07.2016)

the normal force F_N are denoted in the Coulomb friction Law, see Equation 4.1.

$$F_S \geq \mu_S \cdot F_N \text{ and } F_K \geq \mu_K \cdot F_N \tag{4.1}$$

The static friction coefficient is always higher than the kinetic friction coefficient. That's because the contact surfaces interlock. A force greater than F_S is required to break the interlock. Once the interlock is broken a force greater than F_K is sufficient to maintain the movement of the bodies.
Rolling resistance is the force which counteracts the rolling of a circular body on a surface. It is caused by rolling friction and slippage between the circular body and the surface. This slippage can cause static or kinetic friction. Rolling friction is caused by attraction between atoms and deformation of at least one of the bodies. Analog to the static and kinetic friction coefficient the rolling friction coefficient is denoted as μ_R with the dimension of a length. The friction coefficient of rolling friction is usually much smaller than the friction coefficient of static and kinetic friction [6]. A torque D_R about the contact axis is required to move the circular body, shown in Equation 4.2.

$$D_R \geq \mu_R \cdot F_N \tag{4.2}$$

Rolling friction is treated as friction because of the resemblance of Equation 4.2 and Equation 4.1n even though it is not a friction per definition.

4.1.1 ODE Friction Approximation

The physics engine simulates friction at contact joints, which are joints without restrictions. Such contact joints are created and deleted in response to collision detection. Therefore contact joints usually have a lifespan of one time step.
For all contact joints the normal force F_N is calculated assuming frictionless contact. Two friction directions can be defined for a contact joint. The first friction direction can be defined, but does not have to be. If the first friction direction is not defined it is set to an arbitrary direction perpendicular to the normal of the contact surface. The second friction direction is perpendicular to both the normal of the contact surface and friction direction one. A contact joint of two bodies and its friction directions are shown in Figure 4.1.
For each friction direction a friction coefficient μ_x can be defined. One friction coefficient has to be set, the second coefficient is optional. In case of only one friction coefficient both friction directions will use this friction parameter. According to the normal force and the friction coefficient the maximum friction force F_M is calculated for each friction direction, see Equation 4.3.

$$F_{M_x} = \mu_x \cdot F_N \text{ with } x \in \{1, 2\} \tag{4.3}$$

The friction coefficient can range anywhere from zero to infinity. For finite values friction approximation will be calculated[4]. When a force lower than F_{M_x} is allied to a contact

[4]ODE Userguide: http://ode.org/ode-latest-userguide.html (22.08.2016)

joint, the joint is in 'sticking mode' which means that the friction force prevents the body from moving. When a force higher than F_{M_x} is applied, the joint is in 'sliding mode' and the body moves. In case of the friction coefficient being set to zero the contact will always slip, in case of the friction coefficient being set to infinity the contact will never slip. This implies that no distinction between kinetic and static friction is made by the physics engine. Furthermore rolling friction is not regarded within this friction approximation. This simplification can be justified with the difference between the friction coefficients of rolling and both static as well as kinetic friction. These simplifications help to minimize the computational effort.

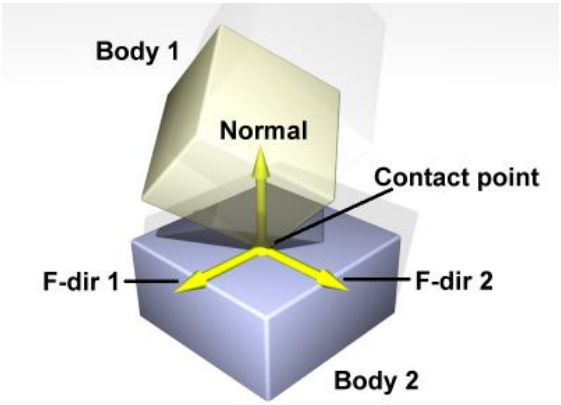


Figure 4.1: Visualization of a contact joint with friction direction one $F - dir1$ and friction direction two $F - dir2$ used for friction approximation[5].

When ODE is applied, Gazebo is able to simulate torsional friction[6], which appears when two bodies with a common surface rotate against each other on an axis normal to the contact surface. In the simulation this kind of friction occurs when the front wheels are rotating relative to the kingpins. Since this motion is small in comparison to the motion of the robot this friction is not regarded either.
The knowledge of the mass which is in correlation with the inertia of the bodies is essential for the friction approximation.

4.2 Inertia

Inertia is the property of bodies to remain in their state of motion unless an external force or torque is applied. Generally friction prevents a body from retaining a state of motion. Both the force and torque to change a bodies state of motion are depending on its mass. Therefore mass can be defined as cause of inertia.
Two different kinds of inertia are significant for the simulation.

[5]Contact Joint Illustration: http://ode.org/ode-latest-userguide.html (22.08.2016)
[6]Torsional Friction: http://gazebosim.org/tutorials at Physics Library - Torsional Friction(22.08.2016)

Firstly, the inertia of mass which is the property of every body to maintain their linear state of motion.
Secondly, rotational inertia which is the property of rotating bodies to maintain their state of motion. The inertia tensor, a symmetric tensor of degree two, of rigid bodies defines the rotational inertia in the center of mass system. Assuming homogeneous bodies the inertia tensor is shown in Equation 4.4.

$$I = m\sum_i \begin{pmatrix} y_i^2 + z_i^2 & -x_i y_i & -x_i z_i \\ -x_i y_i & x_i^2 + z_i^2 & -y_i z_i \\ -x_i z_i & -y_i z_i & x_i^2 + y_i^2 \end{pmatrix} \tag{4.4}$$

The vector $\mathbf{r_i}\,(x_i, y_i, z_i)$ in Equation 4.4 represents the position vector of an infinitesimal volume [13]. The sum of the inertia of these infinitesimal volumes is the rotational inertia. The diagonal elements represent the rotational inertia along the axes of the coordinate system. With an arbitrary rotation axis all elements of the inertia tensor contribute to the inertia.
Mass Inertia as well as Rotational Inertia are nessesary for the calculation of the kinetic energy E_{KIN} and rotational energy E_{ROT}, see Equation 4.5 [13].

$$E_{KIN} = \frac{1}{2}mv^2 \;\text{ and }\; E_{ROT} = \frac{1}{2}\omega^{\mathrm{T}} I \omega \tag{4.5}$$

The kinetic and rotational energy is calculated by the physics engine and regarded during simulation. This effects in rolling of the robot even if no more force is applied. This rolling motion is stopped by friction.
How the implementation of both friction parameters as well as inertia parameters onto the physics engine is achieved is presented in the following.

4.3 Physics Tags

The physics aspects of the physics engine introduced in the Section 4.1 and Section 4.2 are implemented into the simulation with multiple physics tags. These tags define certain physics parameters with basic SI units.
By example of the robots front wheels the usage of these physics parameters will be shown.
The front wheels' collisions, which represent all possible points of contact, and inertia are modeled as homogeneous cylinders. The front wheels are also shown as such cylinders to manage the computational effort. This model contains simplifications, but the empirically selected friction parameter compensates them, to ensure the expected behavior of the simulation.
The code presented in Figure 4.2 is a snippet of the URDF file which defines the robot model and will be discussed in the following. The code is reduced to the physically relevant parts.
The `<link>` tag names and defines a link, which requires a collision tag as well as a visual tag. These tags are a must for every link in a URDF file. The `<origin>` tag can

be used to shift and rotate the origin away from the center of the link. The <geometry> tag defines the shape of the link, which is a cylinder with the radius r and the height respectively length h.
The <inertial> tag defines the mass as well as the inertia tensor of the link. This tag is optional for a URDF file, but is required for physical simulation. The <mass> tag must be set different from zero, otherwise Gazebo would ignore the link. The diagonal elements of the inertial tensor, defined in the <inertia> tag, should be different from zero to prevent nonphysical behavior.
Additional tags for links and joints need to be defined in external <gazebo> tags and referenced to the according links and joints, since the URDF file format is not specifically designed for physics simulation.

```
<link name="front_wheel">
  <collision>
    <origin xyz="0.0 wheelwidth/2 0.0" rpy="pi/2 0.0 0.0"/>
    <geometry>
      <cylinder radius="r" length="h" />
    </geometry>
  </collision>
  <inertial>
    <mass value="0.005"" />
      <origin xyz="0.0 wheelwidth/2 0.0" rpy="pi/2 0.0 0.0"/>
        <inertia
          ixx="1/12 * mass * (3 * r * r + h * h)" ixy="0.0" ixz="0.0"
          iyy="1/12 * mass * (3 * r * r + h * h)" iyz="0.0"
          izz="1/12 * mass * r * r" />
  </inertial>
</link>
<gazebo reference="front_wheel">
  <mu1>10.0</mu1>
  <mu2>10.0</mu2>
</gazebo>
```

Figure 4.2: Code snippet from the URDF file defining the Ackermann robot.

The <gazebo> tag can be used to define visual aspects and physics parameters as well as simulation parameters.
The <material> tag can be used to set colors or materials for links. The friction coefficients μ_1 and μ_2 can be set with the tags <mu1> and <mu2>. The friction parameters for tires on asphalt are $\mu_S = 1.2$ and $\mu_K = 1.05$ [6] if both the tires as well as the asphalt are dry.
The friction parameter in the code shown in Figure 4.2 is found empirically. Therefore friction parameters used for this robot simulation differ from the references. Since no friction of any parts of the robots motor and steering is regarded the wheel friction also includes these friction aspects. In combination with the friction calculation approximation and the simplifications in the wheel model the influence of the friction parameter offset

can not be defined.
Simulation parameters such as ERP (Error Reduction Parameter) and the CFM (Constraint Force Mixing) can also be specified with in the <ERP> and <CFM> tags. Additionally the contact stiffness and damping which are mapped to ERP and the CFM can be set. These are the tags used for the simulation but many more are supported by URDF[7].

[7]URDF Tags: http://gazebosim.org/tutorials at Connect to ROS - URDF in Gazebo (22.08.2016)

CHAPTER 5

Results

The simulation and visualization is the primary goal of this thesis, whereby the focus is on the simulation. Thus, the results shown in this chapter focus on their reliability with respect to the used motion model. Both tests were also applied to the real vehicle, but a detailed discussion of the motion is contained in another thesis [5].

5.1 Simulation

The simulation, performed by `gzserver` and visualized by `gzclient`, as well as the plug-in, see Section 3.3.2, are launched together. Additionally, the URDF file is imported to an empty Gazebo world, see Subsection 3.1.2, and the plug-in to control the robot, see Subsection 3.3.2, is started. This is achieved with a `.launch` file[1] which can be used to launch multiple ROS nodes at the same time.
The previously mentioned URDF file has two weak points which will be introduced in the following:

- The joints responsible for the kingpins, which are meant to act like hinge joints, are acting like ball and socket joints in case of exceptionally strong forces. This problem, known as 'joint error', can happen, during simulation, when error creep and links are drifting off their pose. The 'joint error' problem of ODE[2] is well known along with ways to prevent it. A possible solution is introduced in the following.
  - For this simulation, the best way to prevent this error is to change the Gazebo real time update from its default value of 1000 Hz to 2000 Hz. To ensure real time simulation, the product of the Gazebo real time update and Gazebos

[1] `.launch` file: http://wiki.ros.org/roslaunch/XML (17.03.2016)
[2] ODE 'joint error': http://www.ode.org/ode-latest-userguide.html (17.03.2016)

maximal step size should be one, therefore the maximum step size should be changed from its default value 0.001 s to 0.0005 s.

- The simulation does not regard the vehicles damping which results in unsteady contact between the wheels and the surface. The impact of this simplification is not known at the moment, so two ways to avoid it are introduced.
  - Firstly, the implementation of a damping. This could be realized by using a different kind of joints[3]. The continuous joints which allow rotation around one axis have to be replaced by floating joints which allow motions for all six degrees of freedom. With the `<gazebo>` tag[4] the joints can be extended with a damping parameter. This is only one of multiple ways to implement damping into the simulation.
  - Secondly, the usage of Gazebo world which provides a 'muddy' surface, known as mud world can be used to ensure more stable contact of the wheels and the surface. The mud world surface would deform and cause stable contact joints which cause friction, see Section 4.1. This could possibly replace the implementation of damping to the vehicle. The impact of the use of the mud world to the vehicles motion was not tested.

To ensure the simulations accuracy, the simulated vehicles motions and the real vehicles motions are compared with the velocity motion model.

5.2 Simulation Accuracy

The accuracy of the simulated motions is tested in comparison to the velocity motion model. Since the real vehicle is also using this motion model, it can be used as reference for the real and the simulated vehicle. Two tests are carried out to quantify the accuracy of the simulation with respect to the motion model and the real car. To avoid wheel slipping and drifting during these tests, a low velocity was chosen. For both tests, the error parameters $\alpha_1 = 0.1$ to α_4 are set to 0.1.
For the first test, a straight line with the motion command shown in Equation 5.1 was driven with both the real and the simulated vehicle.

$$\mathbf{u} = \begin{pmatrix} v = 0.1\frac{\mathrm{m}}{\mathrm{s}} \\ \varphi_C = 0 \end{pmatrix} \tag{5.1}$$

The real and the simulated vehicle received this motion command until their velocity motion model noticed a driven way of two meters. The resulting trajectory and pose uncertainty are shown in Figure 5.1. The real car stops 4.5 cm before the reference because of inaccuracies in the measurement of the wheel size. The simulated car stops

[3] URDF joints: http://wiki.ros.org/urdf/XML/joint (17.03.2016)
[4] URDF `<gazebo>` tag: http://wiki.ros.org/urdf/XML/Gazebo (17.03.2016)

1.8 cm behind the reference. The reason for this deviation is that unlike Gazebo, the motion model does not regard the kinetic energy of the vehicle. This deviation could be suppressed by using the odometry motion model instead of the velocity motion model. Since the real and the simulated vehicle use the same motion model to keep compatible, the real vehicle should use the odometry motion model if the simulated one does. This would require wheel encoders and a steering encoder in the real vehicle.

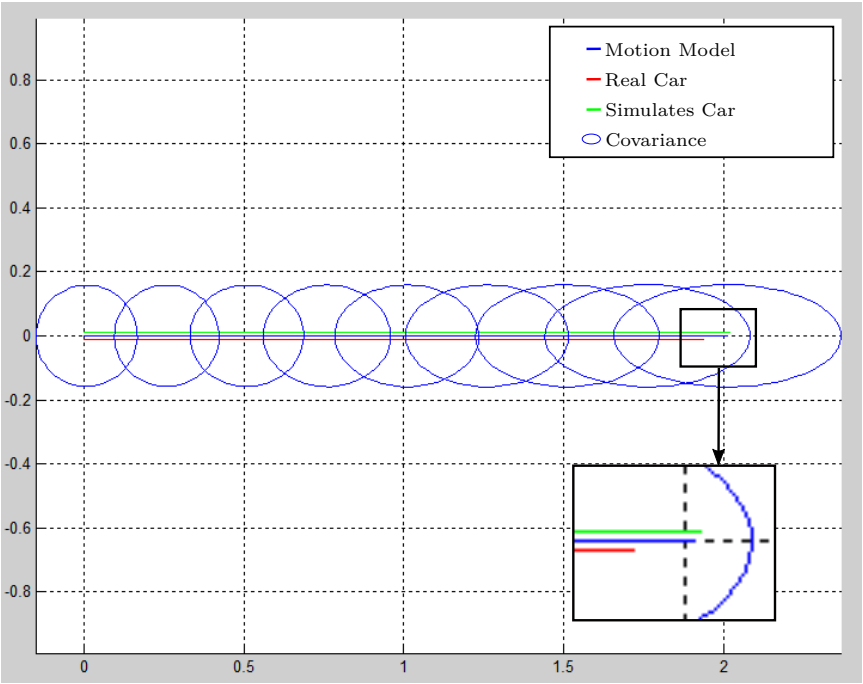


Figure 5.1: Visualization of the trajectory driven with the motion command $v = 0.1\,\frac{\mathrm{m}}{\mathrm{s}}$ and $\varphi_C = 0$ sent to both the real and the simulated vehicle and compared to the calculated motion according to this motion command. The zoomed-in area presents a detailed account of final part of the motion. The results discussed in Subsection 5.2.

For the second test, a semicircle with the motion command shown in Equation 5.2 was driven with the real and the simulated vehicle.

$$\mathbf{u} = \begin{pmatrix} v = 0.1 frac\mathrm{ms} \\ \varphi_C = \varphi_{MAX} \end{pmatrix} \tag{5.2}$$

The real and the simulated vehicle received this motion command until their orientation was shifted by $\frac{\pi}{2}$. The resulting trajectory and pose uncertainty are shown in Figure 5.2. Since a specific number of poses of is visualized the visualized end pose of the real and the simulated vehicles in Figure 5.2 is not shifted by $\frac{\pi}{2}$. The radius of the semicircle driven by the real car is 5 cm bigger than the reference. This is because of the unsteady steering of the RC-race car. The simulated vehicle drives a trajectory differing from a

semicircle. The reason therefore is not yet clear. It might be due to the unsteady surface contact or the 'joint error' discussed in Subsection 5.1.
While driving a straight line, the covariance ellipse only grows in the driving direction. While driving a curve, the covariance ellipse is also growing in both directions and rotating in the curve direction.
During both tests, the pose of the real and the simulated vehicle is inside the covariance ellipse. This is very important for the use of the robot and its simulation with a Kalman filter, which is an overall goal of the robot and the simulation.
The constituted tests are made with simple motions. The reason is a problem with the used velocity motion model. Driving more complex trajectories sometimes causes a shrinking covariance ellipse. The reason for this undesirable behavior is unknown and requires more detailed research. In combination with a Kalman filter, this does not influence the usability of this simulation, since the correction step rectifies this error.

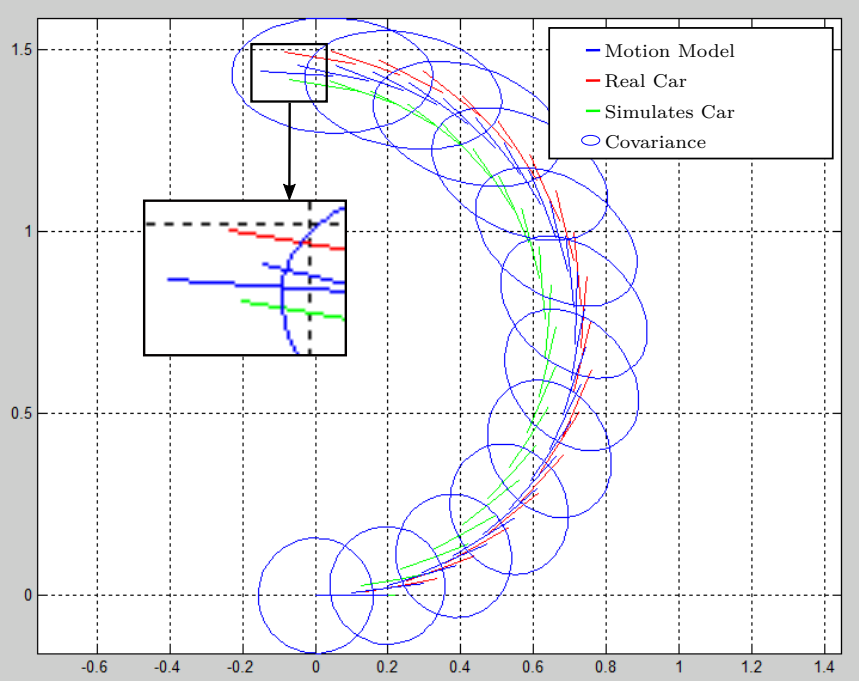


Figure 5.2: Visualization of the trajectory driven with the motion command $v = 0.1\frac{\mathrm{m}}{\mathrm{s}}$ and $\varphi_C = \varphi_{MAX}$ sent to the real and the simulated vehicle and compared to the calculated motion according to this motion command. The zoomed-in area presents a detailed account of final part of the motion. The results discussed in Subsection 5.2.

CHAPTER 6

Conclusion

In the field of robotics, simulations are a commonly used development tool. They allow analyzing the systems and algorithms even without access to the robotic hardware.
This thesis presents an introduction into the field of robotic simulation, using the open source software ROS and Gazebo by means of a robot with an Ackermann steering. Therefore knowledge of the Ackermann steering and the required facilities is provided. The focus is on the creation and testing of the simulation.
The simulation and visualization within this work is designed for motion research. Therefore, a URDF file holding the parts of the chassis which are vital for the robots motion is imported onto Gazebo. Since it not possible to simulate a closed loop of four links with a URDF file, a workaround is introduced.
A node functioning as a plug-in calculating the steering angle of the front wheels according to the Ackermann steering geometry is created to be used for that purpose. Since it is able to communicate with Gazebo, this plug-in is thereby controlling the simulated robot. Furthermore this node is responsible for the calculation of the introduced velocity motion model and for publishing the odometry data as well as the ground truth odometry data. The velocity motion model, which calculates a predicted pose as well as pose uncertainty based on the motion commands, serves as a base for the odometry data. This allows for easy comparison between the motion error of the real and the simulated vehicle.
Two tests were carried out to show the accordance of the simulated motions, based on the comparison between the different odometry data. These tests cover the minimum and the maximum value of the curve radius. The simulation of straight movements results in a small deviation of one percent. In turn, the simulation of winding movements is not as accurate. The reason for this might be the unsteady surface contact of the wheels or the 'joint error', see Subsection 5.2. The deviation of the simulated and real motion from the calculated motion is verified within the tests. During the whole test, the before mentioned deviation was covered by the covariance ellipse . This is essential for the simulation, especially with respect to the overall goal which is to use the real and the

simulated car for self localization and autonomous driving.
Even though the mandatory requirements are met as proven by the test results, the simulations deviation from the velocity motion model can be further reduced by some improvements such as:

- Measuring and implementing a damping to the URDF file and the plug-in. This will enhance the surface contact of the wheels.
- Measuring the wheel friction and replacing the approximated friction parameter.
- Equipping the front and the rear wheels with a differential. This will help to avoid wheels slipping while at the same time increasing steering accuracy.
- Powering the front wheels. This would only improve the accuracy if the last two points are realized.
- Creating a more detailed model with more complex link and inertia structure. This would allow for a more detailed simulation of the robot.

These pending improvements present an unavoidable necessity to further increase the accuracy of the simulation and are vital due to the simplifications made within this project.
The visualization of the vehicle could be improved by adding mesh files of the parts of the prototype such as the chassis or the computation units.
Since an overall goal is to extend the robot with sensors for self localization it is important to mention that the simulation can also be extended with sensors. The simulation tool Gazebo allows for implementation of such sensors and sensor input simulation. Especially the simulation of sensor noise, which is achievable with Gazebo as well, allows analyzing self localization systems with realistic conditions. Tutorials on how to implement sensors onto Gazebo can be found on its website[1].
To render this simulation compatible with other robots of the fleet, a dynamic creation of a URDF file applicable for every single robot of the fleet would be useful.
Thus the actual state of the simulation can evolve in two different ways. Firstly, into a more specific simulation of the allready existing prototype created by a student [5]. Secondly, into a flexible simulation for different Ackermann robots based on model cars. The simulation of a real car does not seem useful since the allocation of the weight should be regarded in this case.
In the near future it is planned to provide the robotics community with this simulation as well as a tutorial on assembling the corresponding robot. The Git[2] repository hosting service Github[3] will be used as a means of communication and further development of the simulation with the community.

[1] Gazebo Sensor Simulation: http://gazebosim.org/tutorials at Sensors (14.05.2016)
[2] Git: https://git-scm.com/ (14.05.2016)
[3] Github: https://github.com/ (14.05.2016)

Bibliography

[1] E. Ackeman. DARPA Awards Simulation Software Contract to Open Source Robotics Foundation. *IEEE Spectrum*, 2012.

[2] C.E. Aguero. Inside the Virtual Robotics Challenge: Simulating Real-Time Robotic Disaster Response. *Automation Science and Engineering*, 2015.

[3] E. Kaltenegger B. Binder M. Bader. Controlling and Tracking an Unmanned Ground Vehicle with Ackermanndrive. In *Vision Meets Robotics*, pages 19–25. OAGM-ARW, May 2016.

[4] S. A. Beiker. Einführungsszenarien für höhergradig automatisierte Straßenfahrzeuge. In *Autonomes Fahren*, 2015.

[5] B. Binder. Autonomous Race-Car, 2016. Bachelorthesis of B. Binder at Vienna University of Technology.

[6] W. Demtröder. *Experimentalphysik 1.* 2008.

[7] M. Quigley et al. ROS: an open-source Robot Operating System. In *IRCA Workshop on Open Source Software*, 2009.

[8] S. Thrun et al. Stanley: The robot that won the DARPA Grand Challenge. *Journal of Field Robotics*, 2006.

[9] S. Thrun W. Burgard D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents).* The MIT Press, 2005.

[10] N. Koenig A. Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems*, 2004.

[11] P. Ross. Robot, you can drive my car. *IEEE Spectrum*, 2014.

[12] R. Siegwart I. R. Nourbakhsh D. Scaramuzza. *Introduction to Autonomous Mobile Robots.* MIT Press, 2011.

[13] M. Bartelmann B. Feuerbach T. Krüger D. Lüst A. Rebhan A. Wipf. *Theoretische Physik.* 2015.