Proceedings of the OAGM&ARW Joint Workshop 2016 DOI: 10.3217/978-3-85125-528-7-27

Controlling and Tracking an Unmanned Ground Vehicle with Ackermanndrive

Eugen Kaltenegger[1], Benjamin Binder[1], Markus Bader[2]

Institute of Computer Aided Automation
Vienna University of Technology, Austria

Abstract
This work presents a tracking and control mechanism for an UGV (Unmanned Ground Vehicle) and its integration into ROS (Robot Operating System). The overall goal of which this work is part, is the creation of a fleet of ackermann robots to conduct studies in the field of autonomous driving. In order to achieve this goal a 1:10 RC-race car model is equipped with an Arduino board to control the vehicles actuators and a Raspberry Pi to host the ROS server. In addition, a physics simulation is used to model this car for testing. The shown results support the used velocity motion model and the applicability of the developed interface to control both platforms.

1. Introduction

During the last years, many studies have been conducted in the field of autonomous driving [6, 1] and the automotive industries as well as companies like Google are showing great interest in this market. Up to 2007, competitions like the DARPA Grand and Urban Challenge pushed research towards autonomous cars with great success [4]. Nowadays, events like the Freescale Cup[1], the Carolo-Cup[2] and others are created to target young students by using RC-race car models which in terms of costs are very attractive. With this in mind, the Institute of Computer Aided Automation at the Technical University of Vienna is planning to create a fleet of autonomous ackermann robots to attract students and to at one point take part in such a competition.
This paper describes the creation of the first of these vehicles and its simulation while also introducing a common interface and a tracking system supporting them.
The robot is based on a RC-race car with an ackermann steering. The computation and controlling of the robot is achieved through a Raspberry Pi and an Arduino Uno equipped with a motor-shield. In addition the vehicle is simulated with Gazebo [8], an open source software for physical simulation based on ODE (Open Dynamics Engine). To keep the vehicle and the simulation compatible, the same interface is used, which is based upon the open source software ROS (Robot Operating System) [2]. The vehicle and the simulation are both using the same velocity motion model [4] which equals the prediction step of a Kalman filter for motion tracking [4, 7]. The velocity motion model introduced by Thrun is defined for differential drive robots, but given several changes, which will be further explained, it can also be used for ackermann robots. Since they are commonly used in robotics and provide enough information for an ackermann robots motion, differential drive commands are chosen as input. A ROS node transforms the differential drive commands to ackermann commands, which include a velocity and a steering angle. The robot and its simulation publish their estimated pose and its uncertainty into ROS topics. This allows for easy comparison of the trajectory driven by the

[1]Freescale Cup: https://community.freescale.com/docs/DOC-1284 (25.04.2016)
[2]Carolo-Cup: https://wiki.ifr.ing.tu-bs.de/carolocup/ (25.04.2016)

RC-race car, its simulation and the motion model.
This paper is structured as follows. At first, related research is introduced and the interface as well as the adapted velocity motion model are described. Based on this knowledge, the robot and its simulation are annotated and their basic structure is discussed. Additionally, the trajectories of the two vehicles are compared and the reasons why the trajectories and the motion model deviate from each other are explained. Finally, further improvements for the adapted motion model in use with the robot and the simulation are introduced.

2. Related Work

Autonomous driving is currently a research topic of both major automobile manufacturers like Volvo, Ford or Nissan and newcomers to the topic of automobiles like Google [6]. At the DARPA urban challenge, universities like the Massachusetts Institute of Technology and the Stanford University present their research accomplishments [3]. An example for research on autonomous vehicles with ackermann drive using ROS is Marvin, the autonomous car by the University of Texas at Austin. Members of the Marvin-Team ported the software of the autonomous car to ROS and shared it this way. The ackermann group represents a community developing open source ROS packages for such vehicles. For the project discussed within this paper, ROS is used because is allows to combine and enhance such packages for navigation and odometry. Twist messages[3] and ackermann messages[4] are used to control the robot, and odometry messages[5] are used for tracking. The structure of ROS allows to combine all these different messages contained in different packages into one interface.

3. Interface

The interface is created to ensure compatibility between the robots of the fleet and the simulation. For that reason, the interface converts twist messages to ackermann messages. It also converts these ROS messages into serial commands and vice versa for those vehicles unable to run ROS. The converting structure of the interface is shown in Figure 1.
Twist messages are commonly used as motion commands because the six parameters they hold provide enough information to define motions in a three dimensional space. In the further, twist messages holding only one linear velocity and one angular velocity are assumed, since they provide enough information for motions in a two dimensional space. Ackermann messages contain a velocity, a steering angle and information about the acceleration and the jerk. The last two are not used for this project.
The velocity of the ackermann messages equals the linear velocity of the twist messages. The steering angle of the ackermann messages can be calculated with the knowledge of the cars geometry. A curve radius of an imaginary third front wheel is calculated by dividing the rotational velocity of the twist message by the its linear velocity. With this radius, the knowledge of the wheelbase and the usage of trigonometric functions, the steering angle φ can be calculated. Based on the motion commands the car and its simulation receive, they return odometry messages containing the estimated pose and its uncertainty. This information is calculated based on the motion model.

[3]Twist Messages: http://wiki.ros.org/geometry_msgs (25.04.2016)
[4]Ackermann Messages: http://wiki.ros.org/ackermann_msgs (25.04.2016)
[5]Odometry Messages: http://wiki.ros.org/nav_msgs (25.04.2016)

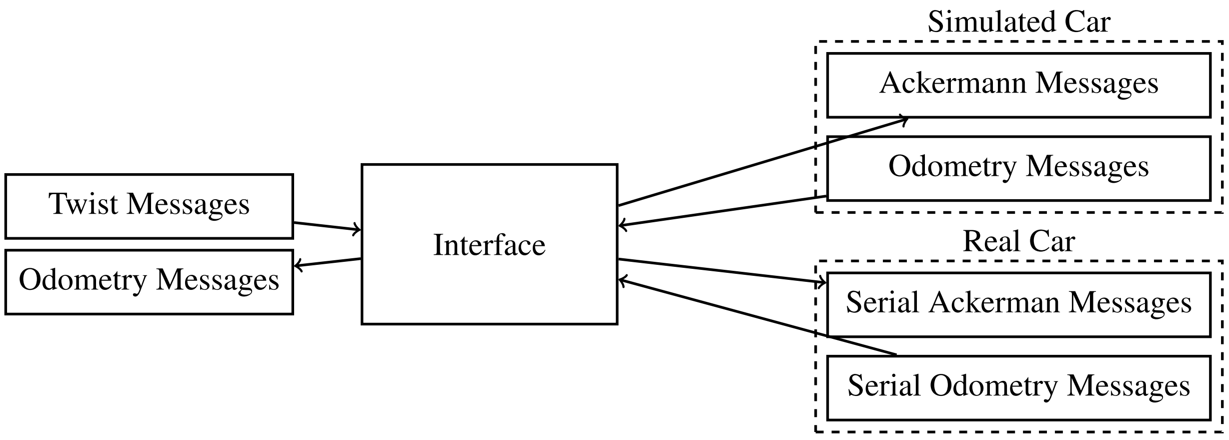

Figure 1: The interaction of the different messages of the interface.

4. Motion Model

Simple problems like wheel slips, bumps, and inaccuracies within the robot effect the robots motion [4, 7]. The tracking system for this project is based on the velocity motion model which considers these errors. Although the velocity motion model introduced by Thrun [4] is conceived for robots able to turn around their own axis, its simple structure allows its customization for ackermann drive robots. Based on the motion commands, the velocity motion model calculates the robots estimated pose and a matrix in which its uncertainty is contained. This is equal to the prediction step of a Kalman filter [4, 7].

Further, the motion model is applied to a flat space represented by x and y and the parameter θ which stands for the robots orientation as shown in Figure 2.

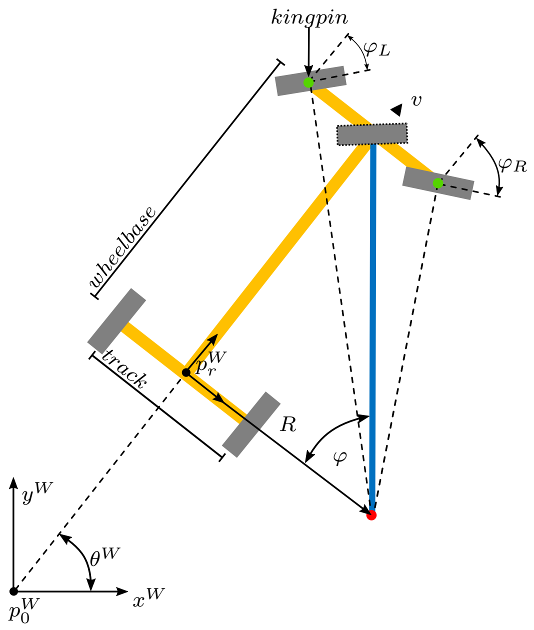

Figure 2: The geometry of the ackermann robot in the two dimensional space.

By adding the change of x, y and θ in one time step to the previous pose, the robots pose at any given time can be calculated recursively.

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

The change of the pose is represented by the jacobian matrix $G\left(\mathbf{x}_{t-1}, u\right)$, which is the derivative of the pose $\mathbf{x}_{t-1}$ with respect to the pose x_{t-1}.

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

The jacobian matrix $V\left(\mathbf{x}_t, u\right)$ is the derivative of the pose $\mathbf{x}_{t-1}$ with respect to the motion command u. This equals the change of the motion.

$$V = \frac{\partial \mathbf{x}_t\left(\mathbf{x}_{t-1},\mathbf{u}\right)}{\partial \mathbf{u}} = \begin{pmatrix} \cos\left(\theta_{t-1}\right) \cdot \Delta t & 0 \\ \sin\left(\theta_{t-1}\right) \cdot \Delta t & 0 \\ \frac{\tan(\theta_{t-1}) \cdot \Delta t}{w_{wheelbase}} & \frac{v \cdot \Delta t}{w_{wheelbase} \cdot \cos^2(\theta_{t-1})} \end{pmatrix} \tag{3}$$

The error matrix $M\left(u, \alpha\right)$ considers the effect of motion errors, while the parameter α considers their severity.

$$M = \begin{pmatrix} \alpha_1 v^2 + \alpha_2 \varphi^2 & 0 \\ 0 & \alpha_3 v^2 + \alpha_4 \varphi^2 \end{pmatrix} \tag{4}$$

The covariance matrix P_t contains the uncertainty of the pose.

$$P_t = G \cdot P_{t-1} \cdot G^T + V \cdot M \cdot V^T \tag{5}$$

The first term of calculation 5 represents the pose prediction and the second term the uncertainty in the accuracy of the motion. The covariance can be visualized by plotting the ellipse defined by the eigen-vectors and the eigen-values of this matrix. Without any correction, the covariance ellipse will grow whenever the robot moves. The growth rate of this ellipse is defined by α which depends on the robot and its environment.

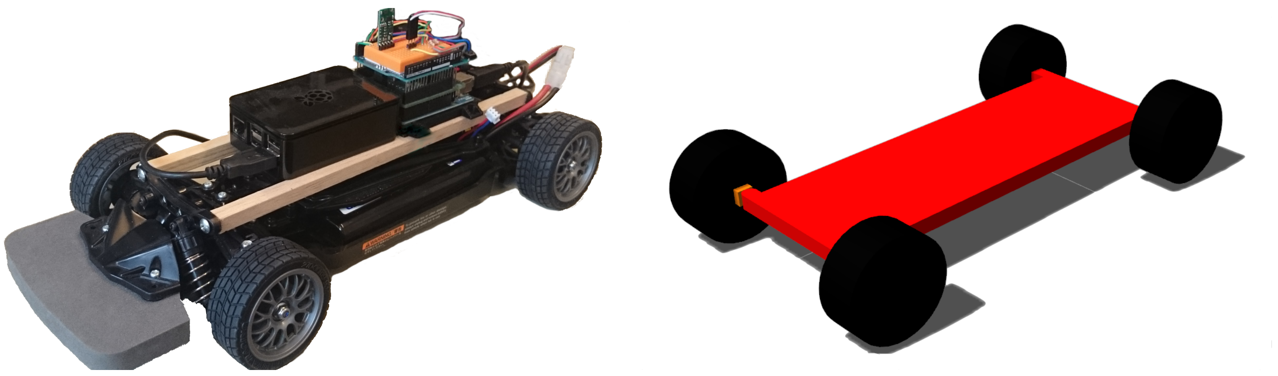
(a) (b)

Figure 3: The (a) real robot and its (b) simulation.

5. Real Car

A Tamiya RC-race car in the scale 1:10 is used as base frame for the ackermann robot, see Figure 3a. The vehicle is powered by a BLDC (Brushless Direct Current) motor, and a servo motor is used for steering. Since the position can be derived from internal hall sensors within the BLDC motor there is no need for additional encoders.
An Arduino Uno microcontroller is used because of its real time capability and its special hardware for such low level actors and sensors. Serial messages from the interface are the means of communication between the Arduino Uno and the Raspberry Pi. In this project, Raspbian is the operating system for the Raspberry Pi because it is based on Debian, which supports ROS. A W-LAN stick is installed on the Raspberry Pi to grant access from other workstations.
The Sensor Level CPU which is represented by the Arduino Uno is responsible for controlling the car, reading sensors and presenting the data in a useful way. A motion controller [5] is implemented for the BLDC motor. Three signals similar to sinus waves generated with pulse width modulation on the Arduino Uno are applied to the motor. The calculation of the pose and its covariance also takes place on the Arduino Uno based on the velocity motion model mentioned before. The calculation frequency is about $100Hz$ which results in an update rate of $0.01s$. The controlling structure of the vehicle is shown in Figure 4.

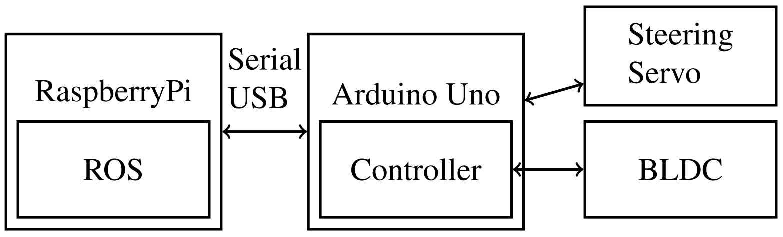

Figure 4: The control hierarchy from ROS to the cars actuators.

Since the steering appears to be the primary source of uncertainty, two improvements are considered. The first is to replace the unsteady steering with a more stable one. The second is to upgrade the car with an encoder for the steering.

6. Simulated Car

Validation of systems and algorithms is an important task in mobile robotics. Thus, Gazebo is used for visualisation and physical simulation of the robot. The simulation contains the parts which are vital for the robots motion. They are imported to Gazebo with a URDF (Unified Robot Description Format) file, see Figure 3b. In the first attempt to simulate the ackermann drive robot, a link was created for each part of the steering and they were connected with joints. The parent-child structure of joints in URDF makes it impossible to create such a closed loop, so a workaround was needed. To get an ackermann steering like behavior, a ROS plug-in is used to control the kingpins, see Figure 2. The plug-in calculates the angles for both front wheels and adjusts the kingpins accordingly. For these calculations, the knowledge of the wheelbase and the track is required. The curve radius of the imaginary third front wheel has to be calculated. It has to be considered that the radii of the left and the right front wheel differ by a half track width from the previously calculated radius. Based on this, the steering angles φ_L and φ_R can be calculated, using the trigonometric functions.

To avoid unintended movements of the kingpin joints, the Gazebos real time update has to be $2000Hz$ and the maximum step size $0.0005s$.
The following three improvements would increase the accuracy of the simulation. Firstly, detailed measurements should be taken to replace the wheels approximated friction parameter. Secondly, damping should be added to the vehicle. Finally, the front wheels should be powered and equipped with a differential.

7. Results

Two tests are carried out to quantify the accuracy of the real and the simulated robots motion. For the first test, a semicircle with the maximum steering angle and a velocity of $0.1m/s$ was driven. The low speed used during the test allows for errors stemming from wheel slipping and centrifugal force to be ignored. The motion model represents the motion commands in this test, so it can be used as a reference. The radius of the semicircle driven by the real car is $5cm$ bigger than the reference. This is caused by the unsteady steering of the RC-race car. The simulated car drives a trajectory differing from a circle. During the whole test, the positions of the real and the simulated car are covered by the covariance ellipse. In Figure 5a the test results are shown.
For the second test, a straight line was driven with a velocity of $0.1m/s$, based on the motion models response. The real car stops $4.5cm$ before the reference, because of inaccuracies in the measurement of the wheel size. The simulated car stops $1.8cm$ behind the reference. The reason for this deviation is that unlike Gazebo, the motion model does not regard the kinetic energy of the vehicle. Again, the covariance ellipse covers the position of the real and the simulated vehicle. The test results are shown in Figure 5b.
To increase the accuracy of the motion model, two improvements can be made. Firstly, the number of updates can be increased to downsize the time steps. Secondly, the kinetic energy of the vehicle should be considered by the velocity motion model.

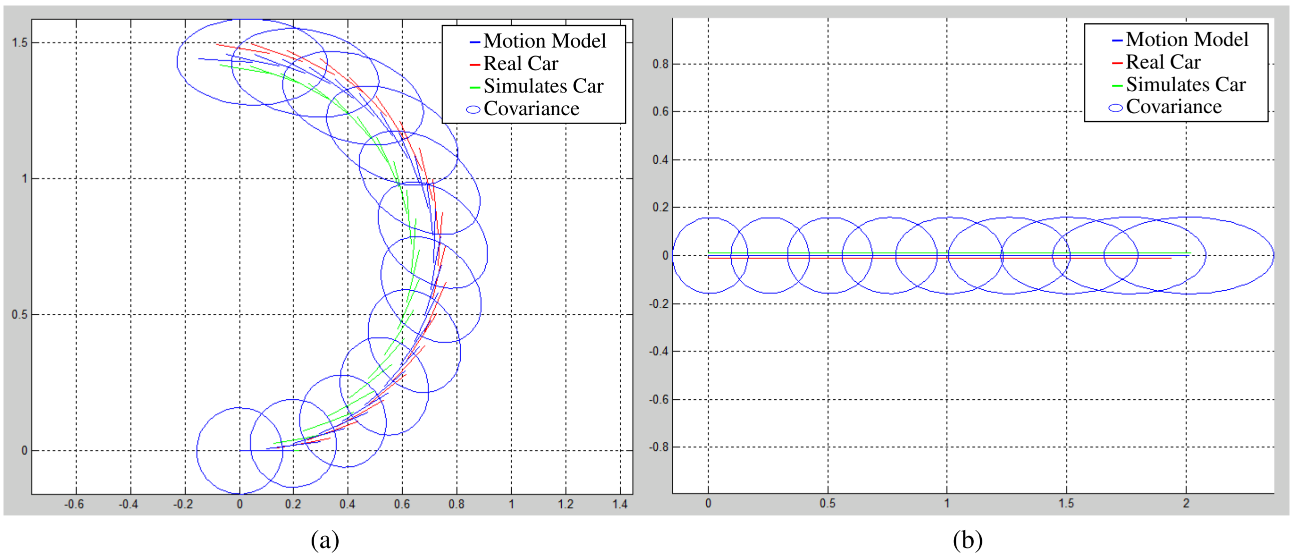

Figure 5: Comparison of trajectory and visualisation of the (a) rotational and (b) the straight behavior of the covariance ellipse.

8. Conclusion

This paper presents the creation of an ackermann robot, its simulation and their common interface. Furthermore, the implementation of a velocity motion model for these vehicles was explained. For future work, the created software will be allocated to the robotics community. Adding sensors like an IMU (Internal Measurement Unit) to the vehicle to increase the accuracy of the motion tracking is planned. Therefore, the interface needs to be extended to handle the new data input. This platform will be expanded by adding self localisation, thus sensor input is required. Based on the knowledge gained with the robot, further vehicles will be built.

References

[1] S. A. Beiker. Einführungsszenarien für höhergradig automatisierte Straßenfahrzeuge. In *Autonomes Fahren*, pages 197–217, 2015.

[2] M. Quigley et al. Ros: an open-source robot operating system. In *IRCA Workshop on Open Source Software*, 2009.

[3] S. Thrun et al. Stanley: The robot that won the darpa grand challenge. *Journal of Robotic Systems - Special Issue on the DARPA Grand Challenge*, 23(9):661–692, 2006.

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

[5] A. Kiruthika R. Agasthiya T. Ramesh. Speed control of a sensored brushless dc motor using flc. *International Journal of Engineering Research & Technology (IJERT)*, 3(4):1–4, 2014.

[6] P. Ross. Robot, you can drive my car. *IEEE Spectrum*, 51(6):60–90, 2014.

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

[8] P. Castillo-Pizarro T. V. Arredondo M. Torres-Torriti. Introductory survey to open-source mobile robot simulation software. In *Robotics Symposium and Intelligent Robotic Meeting (LARS), 2010 Latin American*, pages 150–155, 2010.