FREIE UNIVERSITÄT BERLIN

Controller Architecture for the Autonomous

Cars: MadeInGermany and e-Instein

Daniel Göhring

B-12-06
November 2012

FACHBEREICH MATHEMATIK UND INFORMATIK
SERIE B · INFORMATIK

Abstract. In this paper a realtime controller architecture for our autonomous cars, a highly equipped conventional car and an electric vehicle is presented. The key aspects for controlling a car are stability, accuracy and smoothness, which constrain design critereas of all kinds on controller components. This report presents solutions for a variety of controller components, and aspects of a controller implementation of an autonomous car. The algorithms described proved their applicability in dense urban Berlin traffic as well as on the Berlin Autobahn.

1 Car Introduction

In our project AutoNOMOS we use three different platforms. The Spirit Of Berlin, a Dodge Caravan marks the first autonomous car which was programmed at the Artificial Intelligence Group of Prof. Raúl Rojas at the Freie Universität Berlin since 2006. This car successfully participated in the Darpa Urban Challenge and qualified for the semi finals. Since 2010 we are working on a Volkswagen Passat, called "MadeInGermany" (MiG). MiG is a very modern car with modern sensors and a drive by wire architecture where we have access to throttle and brake actorics via CAN bus. Since 2011 we are also working on an electric vehicle, a Mitsubishi iMiev, called "e-Instein".

Fig. 1. Autonomous Cars (f.l.t.r.): Spirit of Berlin, e-Instein, MadeInGermany

2 Controller Specification

At the beginning of the development of a controller, one has to think about the environment the controller should work in and about the constraints the

controller has to satisfy. For our autonomous cars, the following specification aspects were defined:

- Safety: The controller must be fail safe on a high level with regards to software or hardware failures. Further, the controller must never perform maneuvers, which are not safe with respect to the physical limitations of the car or the environment. It must never operate beyond the mechanical limitations of the actoric. To give an example, a controller which operates on a flat street can to make other assumptions about grip in comparison to a controller which has to operate in off-road scenarios. Regarding actoric, a quickly responding and precisely adjustable actoric allows more aggressive maneuvers than the one reacting with huge time delays. As a rule of thumb translational and centrifugal forces are limited to 40 percent of physically possible values.
- Accuracy: Reaching desired control values quickly is most important for a safe driving of the car. Imprecise or slow controllers usually lead to oscillations within the upper controller level or behavior layer. Oscillations are one reason for an uncomfortable driving experience. For MadeInGermany, the goal was to have a lateral error to the planned trajectory of less than 10 cm at 100 km/h and a velocity error of less than 0.5 km/h.
- Comfort: Besides limiting the amount of lateral and longitudinal forces to a level which feels comfortable to a modest driver, another important aspect is to limit acceleration changes, i.e., the function of the acceleration vector over time must be continuous. Changes within the acceleration vector must remain small for a comfortable feeling of drive.

Some of these aspects contradict each other, e.g., safety and comfort. A safe controller might try to apply appropriate maneuvers as fast as possible but this can be uncomfortable, because humans prefer slow changes of accelerations. Further, comfortable maneuvers, e.g., braking late, can sometimes result in dangerous situations. The same holds for accuracy. A controller which tries to be too precise can result in an uncomfortable feeling of drive.

3 Controller and Planning Modules Overview

This section introduces the module chain and the corresponding module function description. Modules will be distinguished by their level of abstraction in terms of the platform on which they run. An overview of the given modules, their input parameters and their implementation name within the OROCOS-Framework are given in Fig. 2. The code within the AutoNOMOS project was designed to serve as a middle ware for a variety of autonomous platforms. Therefore, reusability of code was a very important aspect. Only the low controller modules are platform specific, high level behavior modules are independent of the platform they are executed on.

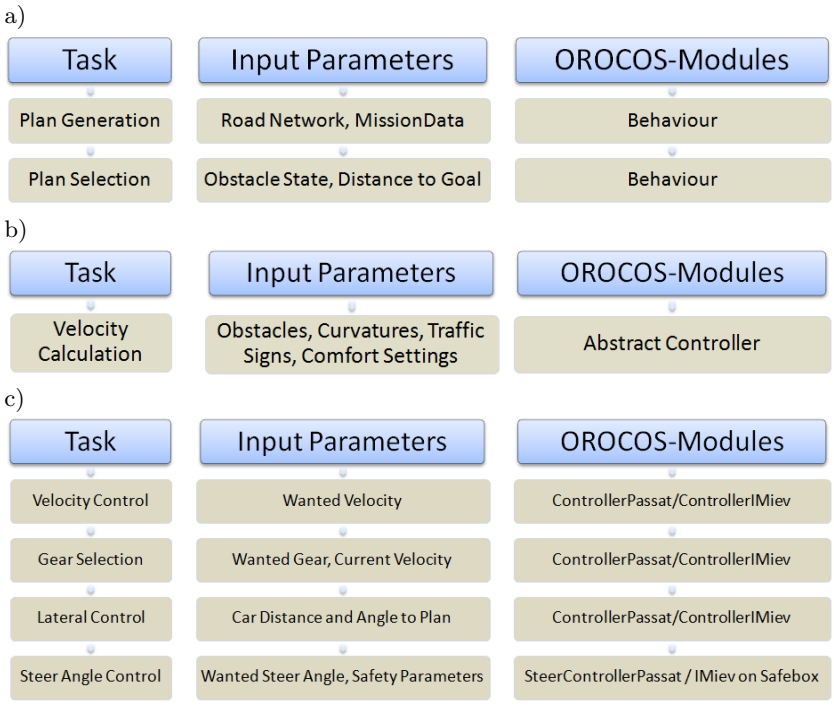

Fig. 2. Controller and behavior models. a) For these models, code and data are vehicle independent. b) Code is vehicle independent but not data independent. c) These modules are designed for a certain vehicle (VW Passat (MiG)).

4 Steer Control Chain (MiG)

The complete steer control chain includes the module for the behavior, which generates a planned trajectory. The behavior uses a road network definition file which includes the streets, and a mission file which defines the checkpoints to be visited while traveling to a certain destination. Just to mention, the behavior generates in each time frame a new, updated plan, mainly it does not remember old trajectories. Now, the generated planned trajectory is fed into a lateral controller. The task of the lateral controller is to compare the current position of the car with the alignment of the planned trajectory. Depending on other aspects as comfort and the current velocity, the lateral controller generates a wanted (desired) steering wheel angle. It was assumed there is a linear dependency between steering wheel angle and the angle of the front wheel, even though it is known that this assumption is just a rough approximation. However, for small angles this assumption provides a good estimate and bigger steering angles are executed at low velocities only, where the exact execution of the generated wanted steering angle is not as important as it is for high velocities. Future work remains to approximate the real function between the steering wheel angle and

front wheel angle, and also the function between velocity, steering wheel angle and wheel slip. Bringing back the focus to the control loop, the desired steering wheel angle is fed into the steer controller, who generates a desired momentum for the steer motor. The control loop is depicted in Fig. 3. In the next sections the steer angle controller and later the lateral controller are described "bottom up".

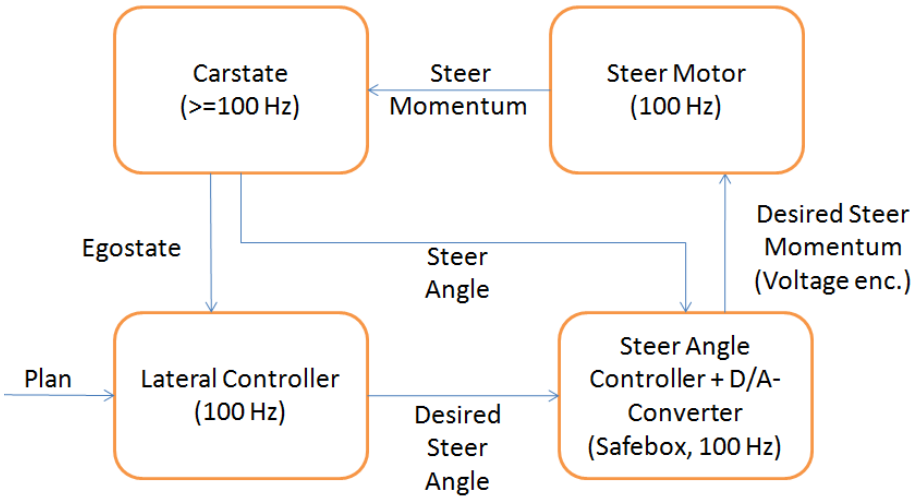

Fig. 3. Steer controller chain for MadeInGermany. The lateral controller generates a desired steer angle, feeds it to the steering wheel controller, which generates a desired momentum, encoded within a voltage.

4.1 Steer Angle Controller (MiG)

The steer angle controller gets a desired steering wheel angle as an input and generates a torque value. To avoid oscillations on the lateral controller, which is executed right before the steer angle controller within the control loop, the steer angle controller shall generate torque values to minimize the difference between desired steering wheel angle and current steering wheel angle as quick as possible. The steer motor, manufactured from Maccon, has no built in angle sensor. Thus, for feedback, we use an external angle sensor, which is built in stock in the MiG-Passat and sends its data via CAN bus with an update frequency of 100 Hz to the controller gateways. 100 Hz is also our control frequency. The sensory data are accumulated in the car state. This architecture is not optimal with regards to system feedback. Further, every system has a certain reaction time. To handle these imminent delays, a predictive controller was the solution of choice.

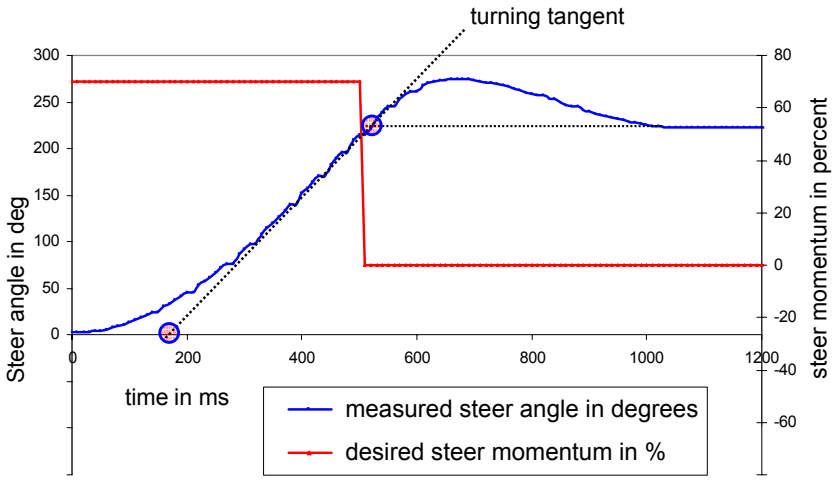

Fig. 4. Impulse response test for the steer motor.

Predictive control. To get a rough estimate about the delay within the control loop, we sent a torque impulse to the steer motor and measured the resulting angle change over time. The response curve was plotted in Fig. 4. The measured response time was about 150 ms. One can see the steering wheel accelerating, the maximum steer angle velocity was measured at $600^o/s$.

To overcome the system delay, a predictive PID-controller was used. The controller does not use the current steering wheel angle θ_{sw} as feedback, but a prediction value of what the steering wheel angle will be θ'_{sw}, assuming a constant change of the steering wheel angle within the next $t = 0.12s$ - which is the estimated system delay - while moving constantly with the currently measured steering wheel angular velocity ω_{sw}. We get:

$$\theta'_{sw} = \theta_{sw} + t \cdot \omega_{sw} \tag{1}$$

Now, where we calculated the values for the control loop, we feed them into the PID-rule. We use a classic PID-controller with a limited maximum integral for the integral term. This is important to avoid big overshoots.

Velocity dependent momentum adaptation. To accommodate fact that the necessary steer momentum varies with the car's velocity, a scale factor was introduced, which is multiplied with the controller output, c.f., Fig. 5. When the car is standing, one has to use a higher force to turn the front wheels, because the tire has a high friction with the concrete. When moving slow, the steering wheel can be turned quite easily. The higher the velocities are, the harder it gets to turn the steering wheel in. There are several causes for this: At first, the centrifugal

force gets bigger, pushing the front wheels back to their neutral position. Second, the rotational impulse gets bigger, to change the impulse vector, a higher torque is necessary. Third, car manufacturers limit the support of the steering servo motor to get the car more stable at higher velocities - our steer motor relies on the support of the steering servo motor. Experiments showed that the controller, in its precise mode, i.e., with maximum scale factors reaches a desired angle within one second with a precision of less than 0.5 degrees, after 1.5 seconds the accuracy is better than 0.15 degrees.

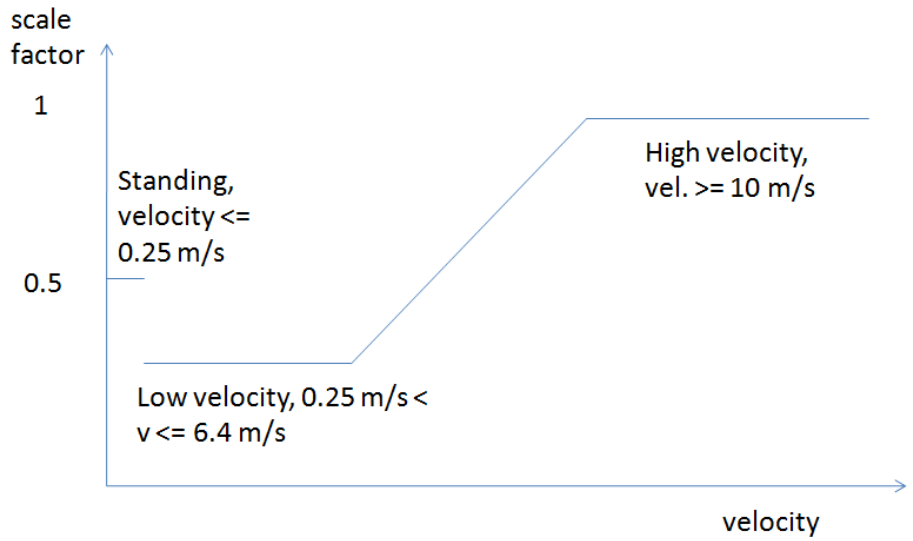

Fig. 5. Steer momentum scale factor function, modeled as a piecewise linear function. Medium momentum is necessary for a standing car, small momentum for low velocities and higher momentum for higher velocities - especially while steering away from the zero angle.

Steer Angle Limiter. To avoid the execution of wanted steering wheel angles, which might pose a threat to the car's safety and stability, a steer angle limiter was implemented, which calculates the maximum allowed steering angle for a given velocity under the assumption of a maximum allowed centrifugal force, see steer angle limiter code example. The steering wheel controller, together with the steer angle limiter runs on the safe box hardware, that's why it cannot be negatively affected by the other modules, running on the main computer.

```
/******* Steer Angle Limiter *******/
float getMaxSteeringWheelAngle(tolerance) {
mSinMaxFWAngle = fabs(WHEELBASE * MAXLATERALACCELERATION /
(mAvgFrontWheelSpeeds * mAvgFrontWheelSpeeds));
if ((mAvgFrontWheelSpeeds != 0.0) && (mSinMaxFWAngle < 1)) {
    maxWheelAngleAtGivenSpeed = asin(mSinMaxFWAngle) * 180 / Pi;
    return fabs((maxWheelAngleAtGivenSpeed /
   MAXWHEELANGLE * MAXSTEERINGWHEELANGLE) + tolerance);
}
return MAXSTEERINGWHEELANGLE; //in degrees
}
```

5 Steer Controller Chain (e-Instein)

The main difference of the e-Instein steer controller chain to the MiG controller chain is that the output voltage of the safe box encodes a steer angle for the Paravan module. The lateral controller generates a desired steer angle, feeds it to the steering wheel controller, which generates a calibrated angle value for the safe box. The safe box generates a voltage, encoding the steering wheel angle non-linear and feeds it to the Paravan system, see Fig. 7. The different modules are described in detail in the following sections.

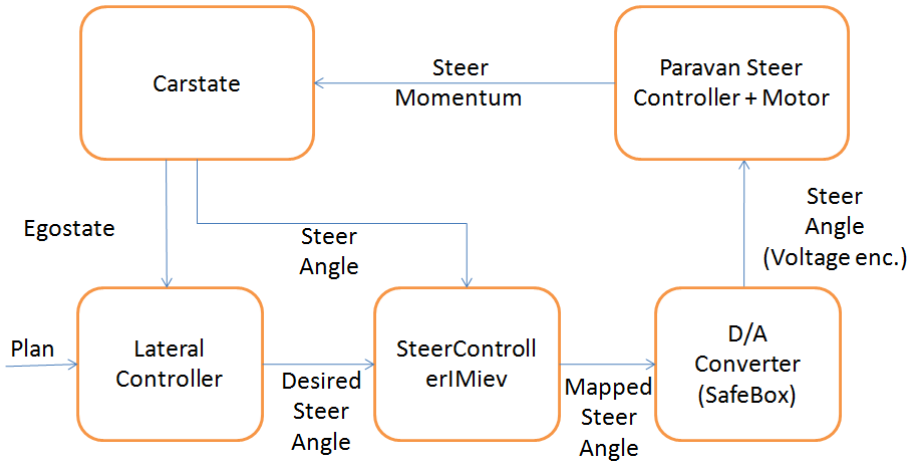

Fig. 6. Steer controller chain for e-Instein.

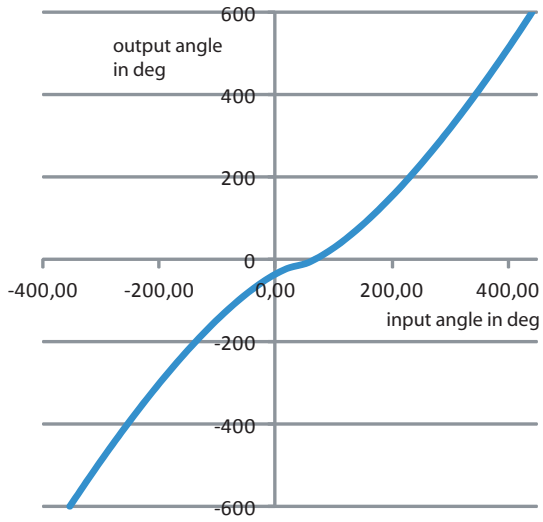

Fig. 7. An input angle is mapped to an output angle. The voltage at the paravan is not linearly dependend on the resulting steer angle. The function, of the form: $a \cdot (x-b)^c + d$ does not pass the origin of the coordinate system.

5.1 Steer Angle Controller (e-Instein)

Originally the steer controller for the e-Instein was assumed to be unnecessary, because the lateral controller generates a desired steering wheel angle and the Paravan module takes a voltage, encoding a steering wheel angle as an input. Unfortunately the mapping between both is not linear and we were missing a specification. Therefore, the steering wheel angle implements a linearization function, c.f., Fig. 7, to map the desired angle to a linearized angle, which feeds the Paravan system, c.f., Equation 2. Parameters a, b, c, d had to be chosen wisely to approximate the experimentally derived function.

$$linearizedAngle = \text{sgn}(desiredAngle - b) \cdot a \cdot (|desiredAngle - b|)^c + d \quad (2)$$

This function, unfortunately, gives only a rough estimate for the control voltage to reach a certain wanted angle. The Paravan system implements a dead zone, thus, it depends from which direction (left or right) the steering angle turns into a desired angle. This prediction function *pred* is sufficient to reach the steering wheel angle with an accuracy of +/- 5 degrees. To reduce the remaining difference between desired and current steering wheel angle below 5 degrees, a PID controller, somewhat similar to the one in MiG is used. The PID-controller can only compensate differences smaller than 10 degrees to avoid oscillations. The steering motor in the e-Instein is very slow (approx. 360 degrees per second), a non limited integral in the PID controller could result in harsh overshoots. Combining the prediction function and the PID-controller, we get the summarized control Equation 3, resulting in a linearized Angle, which can be linearly

encoded to a voltage, which again feeds the Paravan box.

$$linearizedAngle = \mathbf{pred}(desiredAngle) + \mathbf{pid}(desiredAngle, currentAngle) \tag{3}$$

The combined control form allow to reach a desired steering wheel angle with an accuracy below 1 degree, within one second - assumed the desired steering wheel angle is not too far away from the current one.

5.2 Lateral Controller (MiG and e-Instein)

The task of the lateral controller is to generate a desired steer angle to stabilize the car on the planned trajectory. Thereby the car must not swing left and right of the trajectory. To fulfill the control task a velocity adaptive PD controller was implemented.

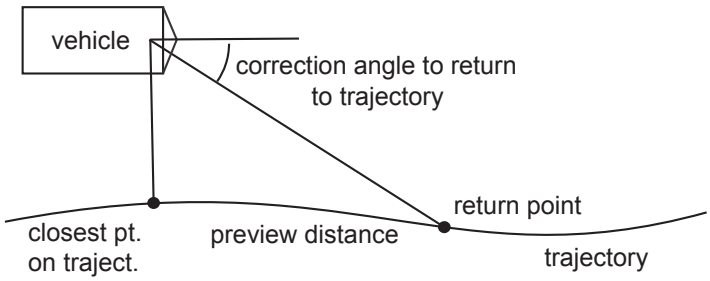

Fig. 8. Lateral controller sketch, showing closest point on trajectory, return point and correction angle.

Velocity adaptive PD-Controller (MiG and e-Instein) The velocity adaptive PD controller uses two weight sets, one weight set for low velocities and one set for high velocities. For medium velocities in between, the two weight sets are linearly interpolated. The resulting function is somewhat similar to the output scale function of the steering wheel controller of MadeInGermany, see Fig. 5. The control task is visualized in Fig. 8. Control input values are the current car orientation and the angle between the car's forward looking direction and the direction from the car's front axis' middle point to the return point of the trajectory, c.f., Fig. 8. The return point on the plan is calculated velocity dependent. The faster the car goes, the more distant the point is. This is useful to stabilize the car for higher velocities and to be able to perform sharp turns while going slow. Still, this approach tends to short cut curves by a small amount. The distance of the return point to the closest point on the trajectory (returnDist) is calculated as shown in Equation 4.

$$returnDist = \max(staticDist, velocity \cdot velocityScale) \tag{4}$$

The static distance value makes sure that the distance to the return point never gets smaller than a certain value, also when the car is standing. It was set to 2 meters in praxis, the scale factor was set to 1 s, which means that the return point is usually reached within one second, if the plan would be kept. In praxis, the car converges to the plan, only, because the desired angle is calculated each time again, converging to zero but never reaching zero.

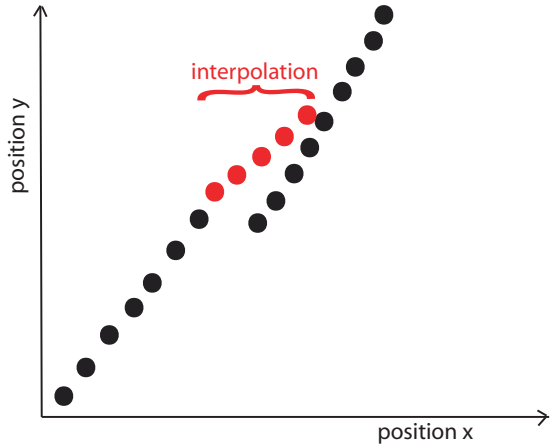

Fig. 9. Ego state over time from bottom left to top right. Suddenly an ego state jump occurred, was recognized by the controller. The jump in interpolated within a certain time, a useful value was $t = 2s$.

EgoState jump filtering (MiG). A second feature of the controller is to memorize the last egostate. Using a prediction, it can detect differences between the expected egostate in the next time step and the current egostate. If the projected difference to the lateral direction of the car is bigger than a certain amount, e.g., 0.1 meters, it is considered to be a critical ego state jump. The two dimensional egostate difference is substracted from the current egostate and interpolated to zero within two seconds. If within these two seconds another jump is detected, an accumulated difference is calculated and interpolation starts from new.

The ego state jumps are supposed to have different causes. Loss of UMTS connection result in loss of correction data and in ego state jumps. Multi path connections to localization satellites could also be a reason.

Car angle to IMU angle online calibration. A further feature is the online calibration of the supposed car orientation angle to the Inertial Measurement Unit (IMU) angle of the Applanix GPS. The Applanix calibrates its IMU at

every restart. Thus, the difference angle between IMU and car forward direction changes every time about 0.05 degrees, in worst case scenarios, whenever the calibration routine was ignored up to 0.5 degrees. At velocities of 100 km/h, 0.5 degrees wrongly calibrated IMU can result in a trajectory offset of 0.5 - 1 meter. Therefore a second calibration routine was implemented in the controller. When going almost straight and going faster than 60 km/h, the controller integrates the error to the trajectory over a certain time span, a useful value was $t = 4s$. If the integral was significantly below or beyond zero, an small positive or negative offset value is added to the car orientation angle. Thus, after ten iterations, i.e., within a minute, the IMU offset could be successfully neutralized.

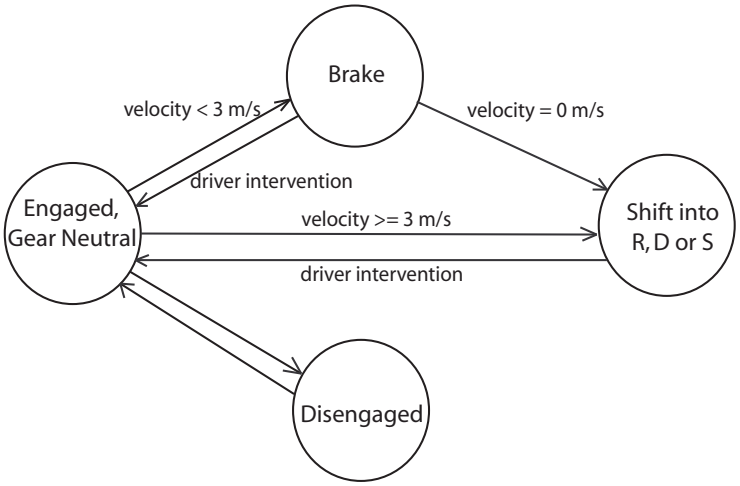

Fig. 10. Finite state machine of the gear shifting routine, enabling the driver to give the control back when going fast enough, or braking and shifting whenever going slow or standing. The disengage state can be reached from all states (left out in the figure).

6 Gear Selection (MiG and e-Instein)

The gear selection routine has to access the gear over CAN bus, and, if necessary send brake commands. The gear selection routine must fulfill at least two tasks: shift into drive whenever the car is engaged, i.e., made ready for autonomous drive. Initially in N, the brake must be pressed with 10 bar, then the gear can be set to drive, reverse or sport. Another task is to give back the control to the car, whenever a driver intervention occurred, which means, that the driver intervened by pressing throttle or brake (MiG only). After releasing brake or throttle, the computer should regain control over the car without stopping. Luckily, the gear allows shifts from neutral to drive, whenever the car is faster than 10 km/h or

approx. 3 meters per second. The finite state machine, depicted in Fig. 10 gives an idea about the functionality of the gear setting routine.

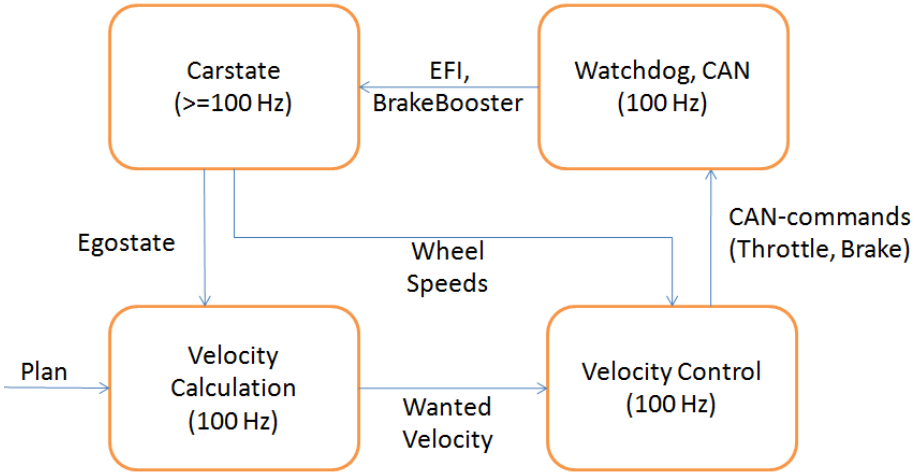

Fig. 11. Velocity control chain: Given a plan, a desired velocity is calculated by the velocity calculation module and fed into the velocity controller. Feedback is given by the ego state and the wheel speeds.

7 Velocity Control Chain (MiG)

The velocity control chain starts again with a planned trajectory from the behavior module. The velocity planner calculates a wanted/desired velocity on this trajectory. This wanted velocity is handed to the velocity controller, which generates throttle and brake commands, which are fed through the cargate and watchdog to the CAN bus of the MiG-Passat. The different modules are described in detail within the next sections.

7.1 Velocity Control (MiG and e-Instein)

Velocity control is executed by a PI-controller, running with a 100 Hz. Input values are a wanted velocity from the velocity calculator and a current velocity, coming via CAN bus and stored in the car state.

Brake and throttle controller. (MiG) The PI-controller compares the wanted velocity to the current velocity. Also here the integral is limited to a certain value. The output of the controller is mapped to throttle or brake. Negative outputs are mapped to brake commands and positive outputs are mapped to throttle. The

most important aspect here lies in the fact, that throttle and brake commands have different effects, i.e., 50 percent of brake results in much higher acceleration amounts than 50 percent of throttle. That is why positive outputs of the controller were weighted with a throttleConstant, which was set to 4.

Throttle and brake limitations. (MiG) For a comfortable ride, the brake commands were limited to 32 percent of their maximum value, which can result in up to $6ms^{-2}$ deceleration. This was especially important for testing the obstacle tracker. To give an example, wrongly classified obstacles could have a devastating effect when going with a 100 km/h on the highway, if they resulted in a full brake, endangering especially the following cars. Also the throttle commands were limited to 50 percent of their maximum during normal velocities. When starting to move, this value was limited to 35 percent and linearly interpolated to 50 percent at a velocity of 5 meters per second.

For a racetrack scenario, much higher thresholds were applied and can be activated through a flag. In this case, the steering angle limiter from Section 4.1 must be deactivated on the safe box as well.

Dynamic Handbrake. (MiG and e-Instein) Stopping a car needs special treatment. A car with an automatic gear tends to start rolling, even without any throttle or brake commands. The integral part of the PI-controller must have an integral weight together with a maximum integral value, big enough to stop the car. This alone is no difference to any other velocity to control. But if the desired velocity is set to zero, the controller tends to convergence to the wanted velocity or it swings around the wanted velocity, resulting in a stop and go. Thus, without special treatment, it can take several seconds to reach a full stop with the car. Further, if pressing the brake not hard enough, the Passat car (MiG) will not fully open the clutch of its direct shift gear, resulting in a semi closed clutch, which again causes severe damage to the clutch over time. A nice solution is to add an increasing amount of brake pressure over time, whenever the wanted velocity is set to zero, combined with the fact that the car is already slower than 4 meters per second. If the car is actually standing, a further fixed amount of brake pressure is added to the brake, to make sure, the clutch is fully open. This "dynamic handbrake" is also useful when stopping at a steep mountain to make sure the car will stop. In experiments the controller proved to be able to keep all velocities up to 100 km/h with an accuracy of 0.5 km/h.

7.2 Velocity Calculation (MiG and e-Instein)

The velocity calculation module is independent of the car it is running on. Important parameters for the velocity calculation are maximum allowed centrifugal accelerations and brake accelerations. These values are referred to as "comfort settings". The velocity calculation is generated instantly and recalculated each time step again. There are different aspects which have an effect on the currently desired velocity: static obstacles or traffic signs/lights on the planned trajectory,

dynamic obstacles on the trajectory, curves in front with a certain curvature and at a certain distance, , and of course the officially allowed velocity. All aspects on the trajectory result in a maximum velocity at which they can be approached currently, e.g., a curve in a distance of 5 meters results in a lower currently desired velocity than the same curve in 50 meters distance. For reasons of safety, from all desired velocities the smallest one is returned to the velocity controller.

The following sections will give a glimpse about the different calculations, which all use the same core function.

Velocity calculation for static obstacles. A first assumption when braking towards a static obstacle was, that the braking acceleration a shall be constant over the whole time. Let us assume, a static obstacle was detected in a certain distance Δs. To brake down with a constant acceleration a (brake accelerations are negative) and to stop right in front of the obstacle, the current desired velocity v_d is calculated as follows:

$$v_d = \sqrt{-2a\Delta s} \tag{5}$$

$$v_d = (-2a \cdot (\mathbf{obstaclePosition} - \mathbf{carPosition}))^{0.5} \tag{6}$$

$$v_d = (-2a \cdot (\mathbf{obstaclePosition} - \mathbf{carPosition}))^{b} \tag{7}$$

In Fig. 12 a desired velocity over distance to obstacle function is shown for a constant example brake acceleration. For a generalization, parameter b was introduced and is, for the case of constant accelerations, set to 0.5. Acceleration parameter a was set to -1.5.

In practice it showed that static accelerations do not fit the human perception of comfort. Humans tend to accept higher accelerations at high velocities, at low velocities the same accelerations feel uncomfortable. Therefore Equation 5 was changed to allow higher accelerations at higher velocities. This can be achieved through changing the values of parameter a and b, parameter b defines, how constant the acceleration is over time, $b = 0.5$ means constant acceleration. Through experiments was found, that the current function with $a = -0.65; b = 0.57$ leads to much better results regarding a higher acceleration at higher velocities and smaller accelerations while stopping:

$$v_d = (-2 \cdot (-0.65) \cdot (\underbrace{\mathbf{obstaclePosition} - \mathbf{carPosition}}_{\Delta s}))^{0.57} \tag{8}$$

The function is plotted in Fig. 12. Both functions result in the same desired velocity of approx. 100 km/h for a distance of 300 meters, or, with other words, the planned braking distance for a velocity of 100 km/h is the same, when approaching a stop sign. Not to increase the stopping distance for higher velocities was an important aspect within the design process, because the sensors in the car have a limited distance at which they can detect obstacles. In case of an unexpected obstacle, the planned velocity will instantaneously jump to a small value, resulting in a high braking pressure.

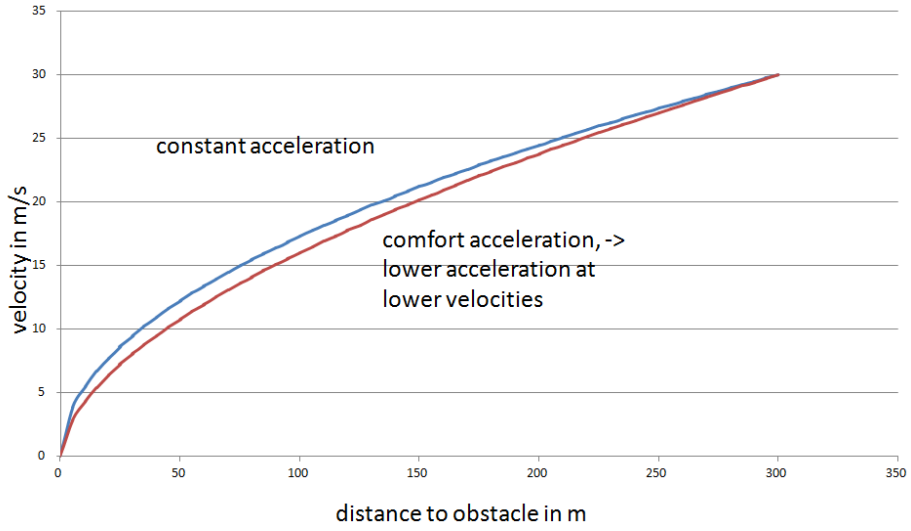

Fig. 12. Desired velocity function over distances to a static object. Blue curve depicts the function with a constant brake acceleration, red curve for a variable brake function.

Stopping for traffic lights. One exception of static obstacle are traffic lights. If turned red, they are treated as stop signs, when green, they can be just ignored. If a traffic light turns from green to yellow, it depends on the proximity of the car to the traffic light, if a braking maneuver is executed or not. If the car is closer than 42 percent of the necessary distance to brake down comfortably, which can be calculated by another form of Equation 8, where Δs is isolated, the car just passes the traffic light. This 42 percent point is called the "point of no return".

Velocity calculation for dynamic obstacles. The introduced function can be extended to braking down to dynamic obstacles easily. Therefore, an accurate estimate of the obstacle's velocity is necessary. Positive velocities mark an object which is moving away from the car. Lateral velocities are not taken into account. The desired velocity Equation 8 can be extended as follows, when v_o marks the velocity of the obstacle, s_c and s_o stand for the position of the car and of the obstacle, respectively:

$$v_d = (-2 \cdot a \cdot (\mathbf{s_o} - \mathbf{s_c} + (\mathbf{v_o}^{(1/b)})))^b \tag{9}$$

$$v_d = (-2 \cdot a \cdot (\underbrace{\mathbf{s_o} - \mathbf{s_c}}_{\Delta s} - \mathbf{v_o} + (\mathbf{v_o}^{(1/b)})))^b \tag{10}$$

The first equation assumes that the car will accelerate/decelerate to the obstacles velocity and approach the obstacle with zero distance. The velocity over distance to the obstacle function is depicted in Fig. 13. The second Equation keeps a safety distance depending on the obstacle's velocity. With the second equation, a very

comfortable car following behavior could be implemented, which worked both at high autobahn velocities as on inner city traffic.

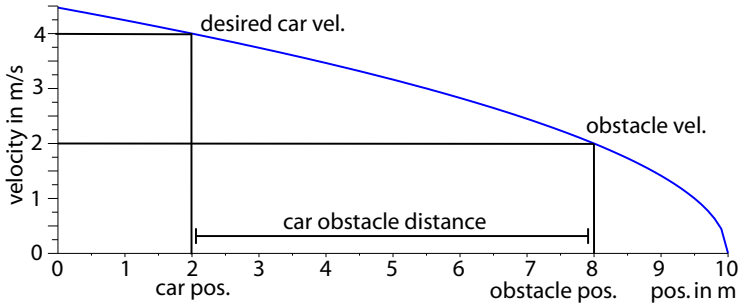

Fig. 13. The velocity over distance to an obstacle function, as shown in Eq. 9.

Velocity calculation in curves. Velocity calculation in front of curves is somewhat similar to braking in front of multiple moving obstacles. At first the planned trajectory is sampled in different distance steps, as Fig. 14 shows. For each point of the sample set and on the trajectory, the curvature c is calculated. For each given curvature $c \neq 0$, which is the reciprocal of the curve radius $c = 1/r$, given a maximum allowed centrifugal force F_z, we get a maximum allowed velocity v:

$$F_z = v^2/r \tag{11}$$

$$F_z = v^2 \cdot c \tag{12}$$

$$v = \sqrt{F_z/c} \tag{13}$$

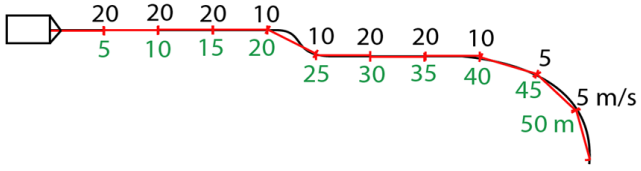

Fig. 14. Above the trajectory the maximum allowed velocity are shown; below, the distance of the car to that point is shown.

Now, with a maximum velocity v_p and a distance s_p for each of the sampled points on the trajectory, the first variant of Equation 9 can be applied, assuming

that v_p is the velocity of the obstacle (s_o) and s_p the distance of of the obstacle (Δs). After doing this for all points on the sampled trajectory, we take the smallest velocity as the wanted velocity for curvatures.

7.3 Combination of forces using Kamm's circle (MiG and e-Instein)

To avoid that the maximum lateral and longitudinal forces occur together, a second variant for the velocity calculation in front of curves can be applied. Here we take advantage of the shape of the "Kamm's Circle". In our case, both maximum forces are modeled as an ellipse, different shapes are also possible. For the most distant sample point on the trajectory the curvature and the maximum allowed velocity are calculated. Now, this velocity is propagated stepwise back to the car, but a new constraint here is, that in case of maximum lateral acceleration, no longitudinal acceleration must occur in the time step before. While going step by step back to the car, the propagated velocity it compared to the maximum allowed velocity in the next sample point and the minimum of the current and the propagated velocity is taken. The advantage of this method is, that both forces, lateral and longitudinal must not occur in arbitrary combinations, making the driving experience more comfortable.

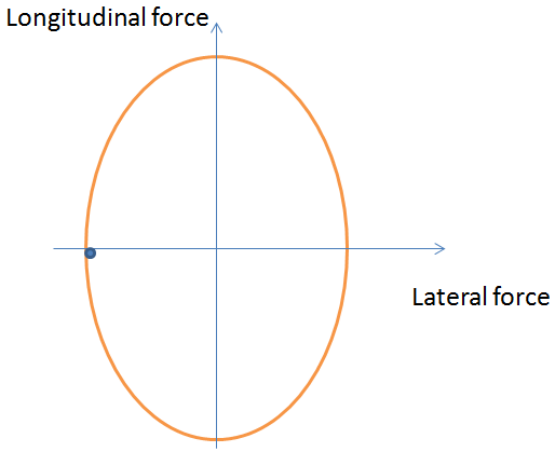

Fig. 15. Example form of a Kamm's Circle.

8 Derivation of a Car Model (MiG)

A precise car model is important for a predictive control and for a good controller simulation. In this section, a short overview of the steps taken to acquire an accurate model for MadeInGermany are presented.

8.1 Derivation of a Car Model

A car model is crucial for a strong simulation of the car's dynamics and to establish an accurate controller. As a model for position and orientation prediction over time an Ackermann drive was assumed for our car as long as centrifugal forces and accelerations were below $5ms^{-2}$. As a further abstraction a bicycle model for the car's dynamic was used, which consists of only one front and one left wheel in the middle of each axis. Given an shaft distance L, a velocity v at the front middle axis point and a steering angle α of the front wheel, the change of the car's orientation angle ω can be easily calculated:

$$\omega = \frac{v(t)\sin(\alpha)}{L} \tag{14}$$

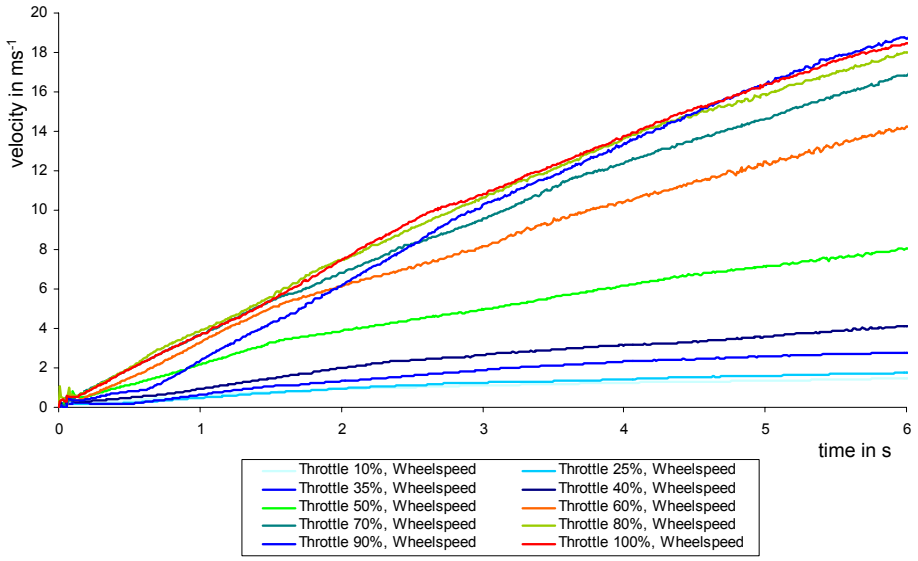

Fig. 16. Velocity of MiG over time for different throttle values.

8.2 Modeling Acceleration and Braking

For an accurate control of a wanted speed it is useful to have a model of the acceleration and braking behavior of the car, given the gear is set automatically from the gearbox. The desired function has as arguments a current velocity v_{t-1}, a time duration t, a current throttle or brake command u_0 and returns a new velocity v_t.

Fig. 17. Acceleration of MiG over time for different throttle values.

$$v_t = f(v_{t-1}, u, t) \tag{15}$$

For reasons of simplicity, the acceleration, including braking experiments were started with a standing car, then the car accelerated for various seconds, but not faster than 80 km/h. The braking experiments were performed at 30 and 50 km/h.

The velocity was provided by the four velocity sensors in the wheel.

Throttle model First, we were interested in the car's acceleration properties when using the throttle command. We recorded different velocity over time functions for different, but constant throttle commands. From the velocity over time functions we can derive the acceleration over time functions, as well.

Now a velocity over time function was be approximated. The velocity over time functions can be approximated with good accuracy by a polynomial function of the form:

$$v_t = at^b \tag{16}$$

The following parameters were derived through gradient descent, where b is constant 0.8, and where a depends on the chosen throttle strength u. Now a function for a, depending on the given throttle command u can be derived. Therefore a sigmoidal function of the form showed to be useful:

$$a = \frac{4.1}{1 + e^{15(0.55-u)}} + 0.4 \tag{17}$$

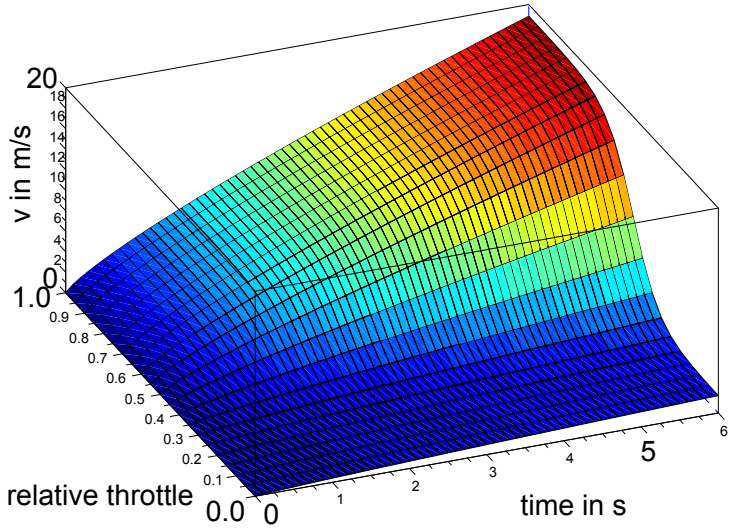

Fig. 18. Approximated velocity function over time and over relative throttle values.

The complete function for the car velocity v over time t, given throttle command u was assumed as:

$$v_t = f(u,t) = (\frac{4.1}{1+e^{15(0.55-u)}} + 0.4)t^{0.8} \tag{18}$$

A plot of the function is given in 18. Therefrom, the acceleration over time function can be derived easily:

$$a_t = f'(u,t) = 0.8(\frac{4.1}{1+e^{15(0.55-u)}} + 0.4)t^{-0.2} \tag{19}$$

Brake model Second, we wanted to analyze the car's braking behavior. Therefore we drove the car with 30 km/h and 50 km/h and executed constant brake commands, the resulting brake acceleration over time function is depicted in Fig. 19 (b). We measured the velocity over time until the car stopped.

As before we derived the acceleration over time functions for the different brake commands, the function for braking the car, e.g., from a speed of $v_0 = 8.3\frac{m}{s}$ was approximated by:

$$v_t = \max(0, f^{-1}(u,t) = v_0 - (13u^{0.8} + 0.2)t) \tag{20}$$

The two-dimensional function plot for a braking car is shown in Fig. 19 (c). The control delay of the throttle and of the brake was about 0.1 s each.

8.3 Car Model Conclusion

The acceleration and brake functions for a given current velocity and given throttle/brake command were successfully applied to our simulator. The lower

level controllers still had to be tuned in the real car because of the complex car dynamics, especially tire slip, mass distribution, system delays and slippage within all kinds of gears. However, this simple car model helped significantly to design the behavior and the higher level controllers within the simulator.

9 Summary

This technical report focussed on the main components of the autonomous cars "MadeInGermany" and "e-Instein". The specification described at the beginning was successfully met. MiG was able to safely drive through inner city traffic and drove also on the Berlin Autobahn with up to 100 km/h, where its mean lateral error to the trajectory was less than 10 cm. The velocity error was about 0.5 km/h. For e-Instein these values are still higher, but work is in progress. Future work will focus on energy efficient control routines to increase energy efficient control.

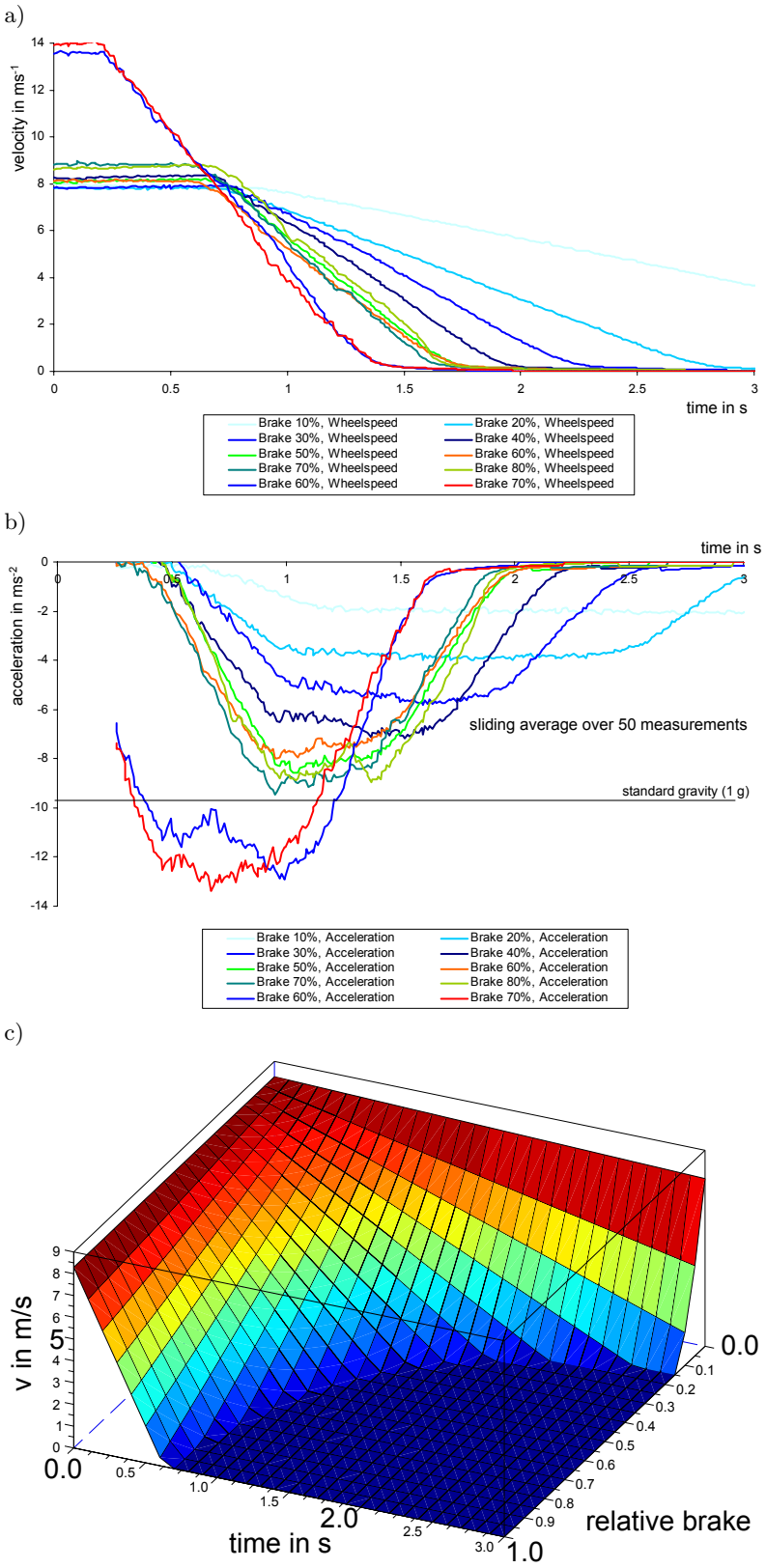

Fig. 19. Brake acceleration over time. (a) Ground truth for different relative brake pressure values (MiG). Velocity over time function. (b) Ground truth, acceleration over time function. (c) Approximated velocity over time function and over relative brake pressure values.