A Robust State Estimation For An Electric Race Car

Student Research Project / T3100

for the study program
Computer Science

at the
Baden-Wuerttemberg Cooperative State University Stuttgart

by
Dominik Stiller

Submission Date	June 8, 2020
Thesis Supervisor	B.Sc. Marco Busch
University Supervisor	Prof. Dr. Zoltán Ádam Zomotor
Matriculation Number, Course	4369179, TINF17A

Declaration of Authorship

I hereby declare that the thesis submitted with the title *A Robust State Estimation For An Electric Race Car* is my own unaided work. All direct or indirect sources used are acknowledged as references.

Neither this nor a similar work has been presented to an examination committee or published.

Sindelfingen	June 8, 2020	
Place	Date	Dominik Stiller

Confidentiality Clause

This thesis contains confidential data of *DHBW Engineering Stuttgart e.V.* This work may only be made available to the university supervisor. Any publication and duplication of this thesis–even in part–is prohibited.

An inspection of this work by third parties requires the expressed permission of the author and *DHBW Engineering Stuttgart e.V.*

Abstract

Modern electric race cars use sophisticated vehicle dynamics control software to exploit the maximum physical potential of the vehicle. For the best results, an accurate and robust state estimate even in the face of measurement failures and missing sensors is required. In this thesis, we describe the design and Simulink-based implementation of a three-stage state estimation. We propose methods for fusion of different sensors and a unified failure and sensor setup detection. We show our design's merit using a simulation with real sensor data.

Contents

Acronyms

4WD four-wheel drive

CAN controller area network

CG center of gravity

DV driverless vehicle

ECU electronic control unit

EKF extended Kalman filter

EV electric vehicle

GNSS global navigation satellite system

GPS Global Positioning System

IMU inertial measurement unit

TC traction control

TV torque vectoring

UKF unscented Kalman filter

UT unscented transform

VDC vehicle dynamics control

List of Figures

List of Tables

List of Symbols

α	Angular accleration	$\mathrm{rad\,s^{-2}}$
β	Vehicle sideslip angle	rad
$\mathbb{0}$	Zero vector	1
ω	Angular velocity	$\mathrm{rad\,s^{-1}}$
ω_{motor}	Motor speed	$\mathrm{rad\,s^{-1}}$
ω_{wheel}	Wheel speed	$\mathrm{rad\,s^{-1}}$
ϕ	Angular displacement around x-axis/roll angle	rad
ψ	Angular displacement around z-axis/yaw angle	rad
τ	Threshold for outlier detection methods	$*$
θ	Angular displacement around y-axis/pitch angle	rad
φ	Angular orientation	rad
a	Linear accleration	$\mathrm{m\,s^{-2}}$
i_{gear}	Gear ratio from motor to wheels	1
p	Linear displacement/position in earth-fixed coordinates	m
R	Corner radius	m
r	Linear displacement/position in vehicle coordinates	m
r_{dyn}	Dynamic tire radius	m
v	Linear velocity	$\mathrm{m\,s^{-1}}$

x	x-axis component of linear position in earth-fixed coordinates	m
y	y-axis component of linear position in earth-fixed coordinates	m
z	z-axis component of linear position in earth-fixed coordinates	m

1 Introduction

It was not even ten years after the invention of the automobile that people started to compete with them in races. Ever since, participants and spectators alike are fascinated by the combination of high-tech vehicles and human drivers, with the most popular racing series attracting over 450 million unique viewers every year [1]. Other than consumer cars, comfort and convenience is traded off for maximum performance, pushing technical limits ever further. Lately, electrical cars are gaining traction, and with them new possibilities for race car engineers.

1.1 Problem

Many electric cars are powered not by a central engine but by four wheel-individual motors. Additionally, these cars have the computing power to run complex vehicle dynamics control software, which helps to execute the driver's wishes while exploiting the maximum physical potential of the vehicle. Components like a traction control or torque vectoring limit and control each wheel's torque requests to optimize dynamic behavior and increase vehicle agility and stability. These components need a good estimate of the vehicle state as foundation for making optimal decisions and thereby increasing vehicle performance. Therefore, a state estimation that provides an accurate and robust vehicle state estimate even in the face of sensor failures and in unpredictable environments is required.

1.2 Scope

This thesis describes the design and implementation of a robust, accurate and flexible state estimation for two Formula Student-type race car with different sensor setups, detailing our thought process and considerations. We propose a three-stage architecture including an inertial measurement unit fusion, an extended Kalman filter and unified failure and sensor setup detection that we successfully apply in a simulation with recorded measurement data.

First, we provide background information on vehicle kinematics and present state-of-the-art estimation algorithms and failure detection methods (Chapter 2). Next, we analyze the vehicle platform and establish requirements for the state estimation architecture and components presented afterwards (Chapter 3). We describe the implementation as Simulink model (Chapter 4), which we use to evaluate the accuracy and robustness of the state estimation (Chapter 5). We conclude by proposing possible future improvements.

This project was conducted for the Formula Student team *DHBW Engineering Stuttgart e.V.*

2 Background

The state estimation of a vehicle sits at the junction of vehicle dynamics and control theory. Both knowledge of the dynamics of a race car and the algorithms to model their physics in equations and software is required to design a successful solution. Therefore, we explore the fundamentals of vehicle dynamics, estimation algorithms and sensor failure detection in this chapter.

Throughout this thesis, the conventions of ISO 8855 [2] will be used, which assumes a right-handed coordinate system. The vehicle coordinate system uses an upward z-axis with a forward x-axis and a leftward y-axis, while the earth-fixed coordinate system uses an upward z-axis with an eastward x-axis and a northward y-axis. We will use the vehicle's center of gravity (CG) as origin/reference point of the vehicle coordinate system. In case of mixed coordinate systems, left-superscript will be used to denote the reference frame (e.g., ${}^{V}x$ for vehicle coordinates, ${}^{E}x$ for earth-fixed coordinates).

2.1 Rigid Body Kinematics

The fundamental laws of mechanics apply to race cars as they do to any other body. These laws relate, among others, the body's linear position, angular orientation and their time derivatives, resulting in translational and rotational changes. Their three-dimensional vector definitions are shown in equations 1 and 2. To simplify the equations, a rigid body is assumed. This means, that deformations which occur in the vehicle during dynamic maneuvers are negligibly small or vanish. Therefore, points on the body maintain the same distance relative to each other at all times. Furthermore, a two-dimensional motion in the road plane can be assumed in many cases because the effects vertical dynamics are negligible. The simplified equations for that case will also be shown.

$$p = \begin{bmatrix} p_x, p_y, p_z \end{bmatrix}^T \tag{1a}$$
$$v = \begin{bmatrix} v_x, v_y, v_z \end{bmatrix}^T \tag{1b}$$
$$a = \begin{bmatrix} a_x, a_y, a_z \end{bmatrix}^T \tag{1c}$$
$$\beta = \arctan\left(\frac{v_x}{v_y}\right) \approx \frac{v_y}{v_x} \tag{1d}$$

$$\varphi = \begin{bmatrix} \phi, \theta, \psi \end{bmatrix}^T \tag{2a}$$
$$\omega = \begin{bmatrix} \dot{\phi}, \dot{\theta}, \dot{\psi} \end{bmatrix}^T \tag{2b}$$
$$\alpha = \begin{bmatrix} \ddot{\phi}, \ddot{\theta}, \ddot{\psi} \end{bmatrix}^T \tag{2c}$$

The linear *position* p of a body comprises a longitudinal component along the x-axis, a lateral component along the y-axis and a vertical component along the z-axis. Its first time derivative $v = \frac{dr}{dt}$ and second time derivative $a = \frac{d^2r}{dt^2}$ are the body's linear velocity and acceleration, respectively. The scalar magnitude $\|v\|$ of the velocity is called speed. In vehicles, the arctangent of the ratio of lateral and longitudinal velocities, often approximated as just the ratio, is furthermore denoted the *vehicle sideslip angle* β. We will use p for positions in earth-fixed coordinates, and r for positions in vehicle coordinates.

The angular *orientation* φ of a body can be described by the roll angle ϕ around the x-axis, the pitch angle θ around the y-axis and the yaw angle, or heading, ψ around the z-axis. The angular velocity ω and acceleration α describe the element-wise time derivatives of these angles.

2.1.1 Transformation of Linear Velocities

Every motion of a rigid body can be decomposed into linear and angular components. All points on that body experience the same angular velocity at all times, i.e. $\omega^A = \omega^B$ for any two points A, B on the body. Since we define the CG as reference point of the vehicle coordinate system and therefore as center of every rotation, the CG will never experience a linear velocity. However, every other point will experience different linear velocities depending on their position r relative to the reference point. If there is a non-zero angular velocity, equation 3 holds for any point P, assuming the velocity at the CG is known (the expanded form can be found in appendix A.1).

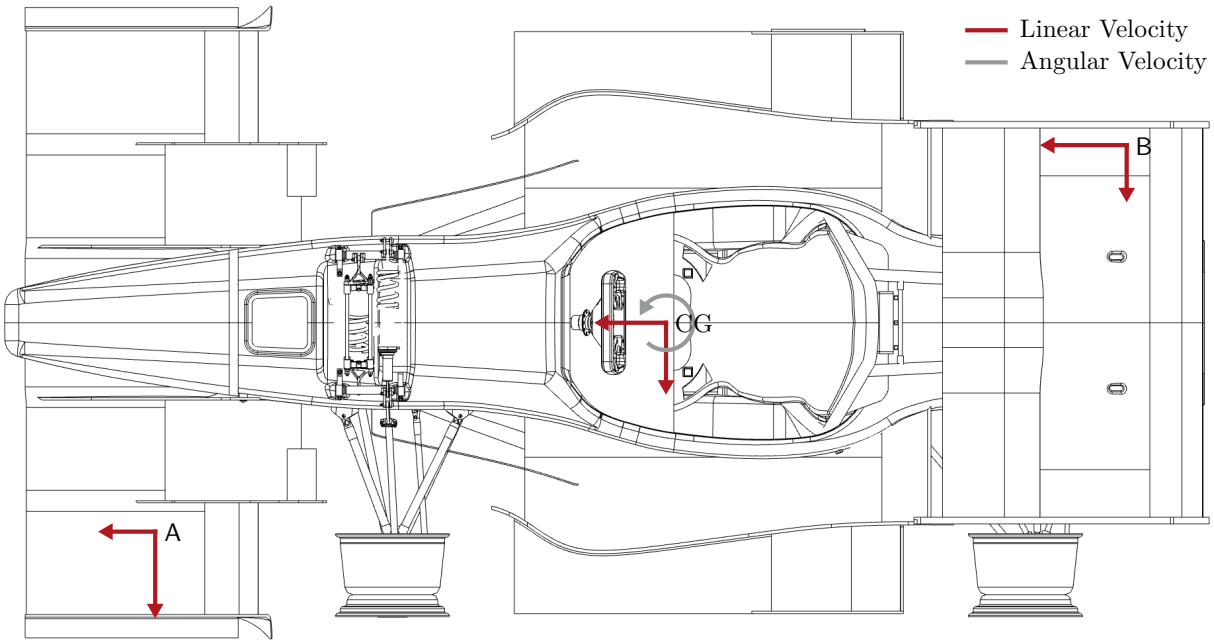


Figure 1: Experienced velocities at off-center points

$$v^P = v^{CG} + \omega \times r^P \tag{3}$$

This becomes easier to visualize when regarding the two-dimensional case, where v_z, $\dot{\phi}$ and $\dot{\theta}$ are disregarded, as shown in equation 4.

$$v^P = \begin{bmatrix} v_x^{CG} \\ v_y^{CG} \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \times \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} = \begin{bmatrix} v_x^{CG} \\ v_y^{CG} \\ 0 \end{bmatrix} + \begin{bmatrix} -\dot{\psi} \cdot r_y \\ \dot{\psi} \cdot r_x \\ 0 \end{bmatrix} \tag{4}$$

Let us regard the example scenario shown in figure 1, where the vehicle has a positive yaw rate and the CG is moving forward and to the left. Point A is at the front left ($r_x > 0$, $r_y > 0$), while point B is at the rear right ($r_x < 0$, $r_y < 0$). Since r is defined relative to the CG, its position is $\mathbb{0}$. Due to the positive yaw rate, A experiences a higher v_y but lower v_x than the CG. B, on the other hand, experiences a higher v_x but lower v_y than the CG. If the yaw rate were zero, all points would experience the same linear velocity.

2.1.2 Transformation of Linear Accelerations

Like the angular velocity, the angular acceleration is the same at every point on a rigid body, i.e. $\alpha^A = \alpha^B$ for any two points A, B on that body. However, these two points will generally not experience the same linear acceleration. Only if the rotational motion components ω and α are $\mathbb{0}$, the linear acceleration is equal for all points on the body, i.e.

$a^A = a^B$. In the general case, equation 5 [3, p. 140] holds for any point P at location r (the expanded form can be found in appendix A.2).

$$a^P = a^{CG} + \alpha \times r^P + \omega \times (\omega \times r^P) \tag{5}$$

We can identify two additional components which affect the experienced linear acceleration. The term $\alpha \times r$ is the tangential acceleration along the circular path around the center of rotation, which is $a_{tan} = \alpha \cdot r$ in scalar notation. More significant due to the squared angular velocity, however, is the introduction of the centripetal acceleration a_c in the term $\omega \times (\omega \times r)$. This term is the vector-equivalent of the acceleration resulting from the centripetal force F_c in equation 6.

$$F_c = \frac{mv^2}{r} \iff a_c = \frac{v^2}{r} = \omega^2 r \tag{6}$$

The two-dimensional form, shown in equation 7, is more intuitive than its three-dimensional counterpart. Here, a_z, $\dot{\phi}$, $\dot{\theta}$, $\ddot{\phi}$ and $\ddot{\theta}$ are assumed to be zero. The inward-direction of the centripetal effect can be seen by the negative signs of the angular velocity part.

$$a^P = \begin{bmatrix} a_x^{CG} \\ a_y^{CG} \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \ddot{\psi} \end{bmatrix} \times \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \times \left(\begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \times \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} \right) = \begin{bmatrix} a_x^{CG} \\ a_y^{CG} \\ 0 \end{bmatrix} + \begin{bmatrix} -\ddot{\psi} \cdot r_y \\ \ddot{\psi} \cdot r_x \\ 0 \end{bmatrix} + \begin{bmatrix} -\dot{\psi}^2 \cdot r_x \\ -\dot{\psi}^2 \cdot r_y \\ 0 \end{bmatrix} \tag{7}$$

We demonstrate these effects in the example scenario shown in figure 2, which is similar to the one in the previous subsection, but with an additional positive yaw acceleration. In point A, the tangential and centripetal acceleration cancel out and even surpass the linear acceleration experienced in the CG, resulting in a negative a_x and a much lower a_y. The opposite effect occurs in point B, where both a_x and a_y are amplified.

2.1.3 Calculate Angular Acceleration from Linear Acceleration

Direct measurement of the angular acceleration is often not possible. However, individual components of α can be calculated from the difference of two known linear acceleration vectors at different points A, B with known locations r^A, r^B using equation 8. It is derived from equation 5, with $\Delta a = a^A - a^B$, $\Delta r = r^A - r^B$, and a^{CG} being eliminated by the difference. The points must not be on the rotation axis of the calculated component of α, otherwise they experience no angular acceleration. Thus, at least three non-collinear

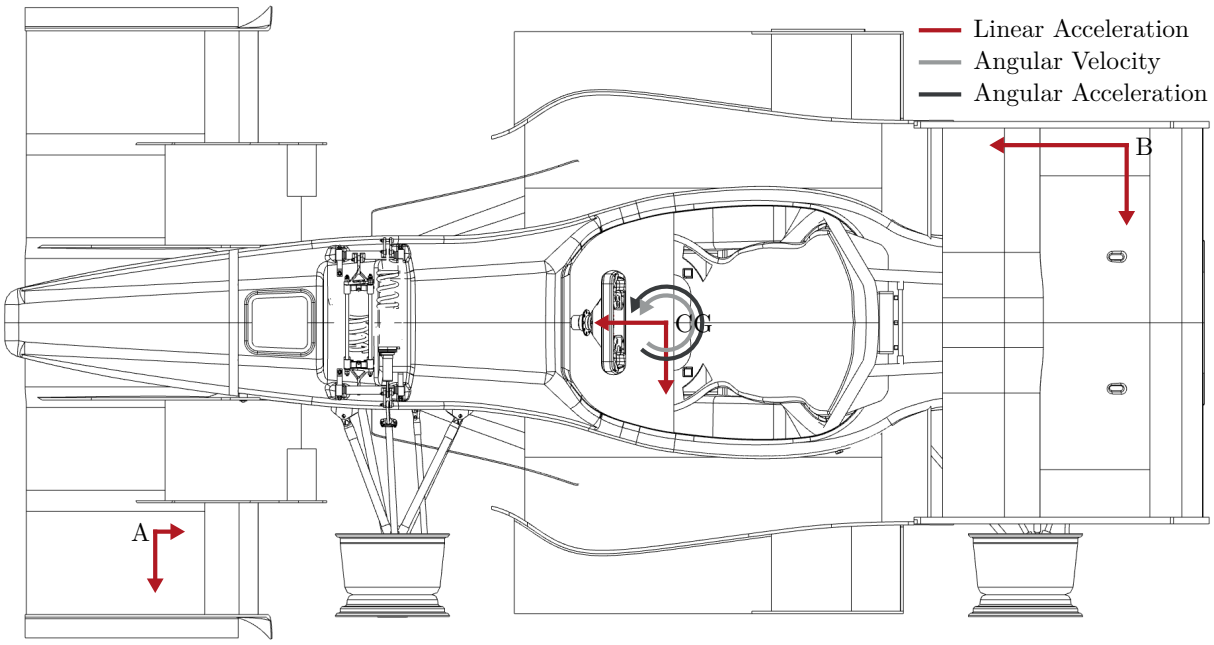


Figure 2: Experienced acceleration at off-center points

points are required to determine all components of α. While the inverse of a cross product is not uniquely determined, as can be seen by the scalar factor t, $t = 0$ works well in practice.

$$\Delta a = \alpha \times \Delta r + \omega \times (\omega \times \Delta r) \implies \alpha = \frac{\Delta r \times (\Delta a - \omega \times (\omega \times \Delta r))}{\|\Delta r\|^2} + t \cdot \Delta r, t \in \mathbb{R} \quad (8)$$

In two dimensions, this becomes easier to understand. Calculation of the yaw acceleration from the longitudinal and lateral accelerations of two points is shown in equation 9. We can see that the accelerations and distances from different axes are being correlated. The equation even shows how positioning both points on the z-axis would result in a zero division, which shows that points off the rotation axis are required. When A and B are directly in front of and behind the CG on the x-axis so $\Delta r_y = 0$, the equation simplifies to $\ddot{\psi} = \frac{\Delta a_y}{\Delta r_x}$. For practical applications, increasing the distance between the two points minimizes the effects of measurement uncertainty in the result.

$$\alpha = \frac{\begin{bmatrix} \Delta r_x \\ \Delta r_y \\ 0 \end{bmatrix} \times \left(\begin{bmatrix} \Delta a_x \\ \Delta a_y \\ 0 \end{bmatrix} - \begin{bmatrix} -\dot{\psi}^2 \cdot r_x \\ -\dot{\psi}^2 \cdot r_y \\ 0 \end{bmatrix} \right)}{\left\| \begin{bmatrix} \Delta r_x \\ \Delta r_y \\ 0 \end{bmatrix} \right\|^2} \implies \ddot{\psi} = \frac{\Delta r_x \Delta a_y - \Delta r_y \Delta a_x}{\Delta r_x^2 + \Delta r_y^2} \quad (9)$$

2.1.4 Transformation of Velocity Between Coordinate Systems

Most sensor measurements are done in the vehicle coordinate system. Others, such as position measurements from a global navigation satellite system (GNSS) (e.g., Global Positioning System (GPS), Galileo, GLONASS), will be made in earth-fixed coordinates, however. To relate these, we need to be able to transform the vehicle velocity from the vehicle coordinate system to the earth-fixed coordinate system, which has an inertial (i.e. non-accelerating) reference frame. A two-dimensional treatment is sufficient, since GNSS altitude information are not relevant here. Therefore, both coordinate systems can be thought of as parallel planes with the heading ψ as mutual rotation. Equation 10 shows how the velocity can be transformed using a simple rotation. The transformation effectively extracts the east-/northward components of the local velocities and adds them.

$$\begin{bmatrix} {}^{E}v_x \\ {}^{E}v_y \end{bmatrix} = \begin{bmatrix} cos(\psi) & -sin(\psi) \\ sin(\psi) & cos(\psi) \end{bmatrix} \begin{bmatrix} {}^{V}v_x \\ {}^{V}v_y \end{bmatrix} \tag{10}$$

2.1.5 Cornering Acceleration

The motion of a rigid body in the earth-fixed reference frame can be understood as a composition of a rotation and a translation, in our case with the CG as center of rotation. In vehicles, this rotation results from cornering. The acceleration as experienced in the CG is the result of the direct linear acceleration and the centripetal force resulting from the rotation, as shown in equation 11 [4, p. 146]. Other than in equation 5, the radius R is not the distance to the CG but the corner/curve radius. When driving straight, the radius approaches infinity and the centripetal force vanishes. Since R is usually not known, we can get rid of it using $R = v_\perp/\dot{\psi}$ with the tangential velocity $v_\perp$.

$$a_x = \dot{v}_x - \dot{\psi}^2 R = \dot{v}_x - \dot{\psi} v_y \tag{11a}$$
$$a_y = \dot{v}_y + \dot{\psi}^2 R = \dot{v}_y + \dot{\psi} v_x \tag{11b}$$

2.1.6 Velocity Estimation from Motor Speeds

While there is a plethora of methods for measuring vehicle velocity, such as optical cross-correlation, radar, 5^{th} wheel and GNSS, most of them are too expensive for widespread use. Therefore, velocity estimation in many automotive applications is based mainly on motor or wheel speed measurements, which are readily available, especially in electric cars.

Equation 12 shows arguably the simplest method to estimate the longitudinal velocity, involving only the dynamic tire radius r_{dyn}, the wheel speed ω_{wheel}, the motor speed ω_{motor} and the motor-to-wheel ratio i_{gear}.

$$v_x = \omega_{wheel} r_{dyn} = \frac{\omega_{motor}}{i_{gear}} r_{dyn} \tag{12}$$

This method can only be used for a crude approximation of the longitudinal velocity component. It falls short when the wheel slip, i.e. the relative motion of tire and road surface, increases or in the most severe case even locks up. While the previously mentioned methods are infeasible in most cases, they can be used as ground truth to evaluate estimation methods.

A simple improvement is averaging over all four wheels. Note that transforming each wheel's velocity into the CG using equation 3 is not necessary, since wheels can be assumed to be distributed symmetrically, therefore averaging effectively neutralizes the effect of off-center velocities. More advanced methods are presented in [5]. For transient maneuvers with high slip, wheel speed data can be augmented with acceleration measurements. The authors present three approaches. The first estimator uses equation 12 until a high-slip situation occurs, at which point acceleration measurements are integrated over time with the last wheel-based velocity as initial condition. The second estimator reduces noise through a Kalman filter which averages both measurements, weighted by a constant obtained with fuzzy logic. Finally, the third estimator reduces adverse effects of acceleration bias and an incorrect dynamic roll radius through regression.

2.2 Estimation Algorithms

Obtaining the true state of some physical system is next to impossible. Any sensor measurement contains noise and other errors, which distract from the underlying process. Therefore, the challenge is discerning errors from the true value. This is known as *filtering problem*, where we want to obtain the best estimate $\hat{x}_k$ of the system state x_k at time step k based on past observations $z_i, i \leq k$ [6, p. 67]. Usually, this involves the fusion of multiple sensors and physical models. In this section we explore common estimation methods to solve the filtering problem. We will regard their discrete-time instead of their continuous-time versions, since digital observations are based on sampling with a finite rate.

2.2.1 Extended Kalman Filter

The Kalman filter [7] is the most widely used estimation algorithm [8, p. 401]. With the trajectory estimation for the Apollo space program being one of its first notable applications [9], it can now be found not only in many vehicles, aircraft, spacecraft, but even in finance and econometrics. The Kalman filter assumes a linear system, which is often not the case, or only true in a small region of the full range. Therefore, a non-linear version based on the same ideas has been developed with the extended Kalman filter (EKF), which we will regard in this section.

The idea of the EKF is simple yet powerful: we observe the physical system through noisy measurements but predict the state with a mathematical model of the system. At every time step, we fuse the observations with the predictions based on our confidence in either, basically resulting in a weighted average. For example, a noisy measurement is rather unreliable, while models can never capture all nuances and uncertainties. The fusion yields a good estimate of the true state of the system, better than only measurements or predictions could alone.

The model describes the *state* x_k of the system at time k. For example, a train on a track can be described by its position and velocity which are related, so the state is two-dimensional. How the state evolves over time according to a process is shown in the *process equation* 13. The process function $f(x_{k-1}, u_k)$ propagates the previous state x_{k-1} and is controlled by external inputs u_k, e.g., the train engine accelerating the train, and disturbed process noise w_k, which represents unmodeled behavior, e.g., air resistance or friction. The process noise is assumed to be Gaussian with zero mean and a diagonal covariance matrix Q_k, i.e no noise terms are correlated.

$$x_k = f(x_{k-1}, u_k) + w_k \tag{13}$$

While the state is inherent to the system, it can not be directly observed. Rather, we have *observations* z_k. The elements of z_k and x_k must not necessarily coincide, i.e. the observation space and state space can differ; for example, we might have multiple observations of the train's velocity but none of the position. The measurement is described by the *measurement equation* 14. The measurement function $h(x_k)$ gives the observation vector which is disturbed by measurement noise, e.g., resulting from the sensor itself or transmission errors. The measurement noise is assumed to be Gaussian with zero mean and a diagonal covariance matrix R_k. We use the measurement equation to be able to fuse real measurements with predicted measurements from our model.

$$z_k = h(x_k) + v_k \tag{14}$$

The process and measurements equations acknowledge, that the state can only be modeled and observed with some degree of uncertainty. Therefore, in addition to the state estimate $\hat{x}_k$, we want to keep track of its covariance P_k, which is zero mean as well. Together with linearizations, this assumption allows the EKF to be much more computationally efficient than more generally applicable algorithms like the particle filter.

The EKF algorithm [10, p. 16 ff.] is recursive, with each iteration comprising a prediction step and a correction step, i.e. the previous state is first propagated to the current time step using the model and then corrected with the measurements from the real system. In the following description of the algorithm, we distinguish between the predicted state estimate $\hat{x}_k^-$ and covariance P_k^-, and the corrected state estimate $\hat{x}_k^+$ and covariance P_k^+.

Prediction At the beginning of every iteration, the previous corrected state is propagated using the process function to predict the current state (equation 15). Note that the prediction only depends on the directly previous state, not any other states in the history, revealing that the EKF assumes an underlying Markov process.

$$\hat{x}_k^- = f(\hat{x}_{k-1}^+, u_k) \tag{15}$$

The previous covariance also needs to be propagated, since the estimate uncertainty might change with every step (equation 16). The Jacobian matrix F_k of the process function, which can be though of as a vector derivative, needs to be evaluated for this step (equation 17). Additionally, the process noise needs to be accounted for model mismatches, since there are always aspects of the real-world which cannot be modeled. Therefore, leveraging the additive property of Gaussian noise, the process covariance Q_k is added to the propagated covariance.

$$P_k^- = F_k P_{k-1} F_k^T + Q_k \tag{16}$$

$$F_k = \left.\frac{\partial f}{\partial x}\right|_{x=\hat{x}_{k-1}^+} \tag{17}$$

The measurement function then predicts measurements from the estimated state (equation 18).

$$\hat{z}_k = h(\hat{x}_k^-) \tag{18}$$

Correction Once the prediction has been made, it can be corrected with observations obtained from the real system by taking a weighted average of both. The weighting is determined by the *Kalman gain* $K_k \in [0, 1]$ (equation 19), which also maps back from the observation space into the state space using the Jacobian matrix H_k of

the measurement function (equation 20). The Kalman gain is defined as the ratio of the state-innovation covariance P_{xy} to the innovation covariance P_{yy}, and so it captures the uncertainty in the measurements and the model prediction. When the measurement covariance R_k is small relative to the model covariance, the Kalman gain is high and so the measurement is weighted more. On the other hand, when the measurement noise is high, the Kalman gain approaches zero and the model prediction is weighted more.

$$K_k = \underbrace{P_k^- H_k^T}_{P_{xy}} \underbrace{(H_k P_k^- H_k^T + R_k)}_{P_{yy}}^{-1} \tag{19}$$

$$H_k = \left.\frac{\partial h}{\partial x}\right|_{x=\hat{x}_k^-} \tag{20}$$

In the best case, the model and the measurements coincide. However, this is usually not the case, so the real measurements z_k and the predicted measurements $\hat{z}_k$ differ, resulting in an *innovation* y_k, also called *residual*. The absolute innovation is higher when there is a larger mismatch between the model and the real system. The Kalman gain now determines, how much of the residual is used to correct the prediction. The posterior state estimate $\hat{x}_k^+$ is the final, best estimate for the iteration k and will be used in the next time step (equation 21).

$$\hat{x}_k^+ = \hat{x}_k^- + K(\underbrace{z_k - \hat{z}_k}_{y_k}) \tag{21}$$

Additionally, the covariance matrix needs to be updated to account for the correction (equation 22, I being the identity matrix). The covariance is reduced based on the Kalman gain, signifying how the certainty in our estimate has increased by the correction.

$$P_k^+ = (I - K H_k) P_k^- \tag{22}$$

2.2.2 Unscented Kalman Filter

While the EKF has been successfully applied to problems for a long time, it suffers a number of limitations [8, p. 402]:

- The error propagation in equation 16 is only reliably if the linearization in form of the Jacobian matrix F_k is sufficiently accurate. This might not be the case in highly non-linear systems or under the wrong initial estimate. In the worst case, the estimate might even diverge.

- Linearization can only be applied to differentiable functions, i.e. processes for which a Jacobian exists. Discontinuities and singularities might exist in some cases, making differentiation impossible.
- The derivation of Jacobian matrices can be very difficult and error-prone for complex systems, both the equations themselves and the conversion to code.

Therefore, alternatives to the EKF have been developed. A promising algorithm is the unscented Kalman filter (UKF) [11].

At its core, the UKF's improvement relies on the unscented transform (UT). This transformation is a method to approximate the properties of a statistical distribution undergoing a non-linear transformation. A small set of points ($2n + 1$ for an n-dimensional random variable) is sampled from the distribution using a specific, deterministic algorithm (not randomly as in Monte Carlo algorithms). These points contain second-order information about the distribution and can be transformed to capture properties such as mean and variance of the resulting distribution. Therefore, the mean is calculated to a higher order of accuracy than in an EKF, whereas the covariance is calculated to the same order of accuracy. Through the UT, linearization of the Jacobian is rendered unnecessary, which aids the filter's numerical stability in non-linear systems and enables application in discontinuous systems. Furthermore, no manual derivation of the Jacobian is necessary. Still, the efficiency and accuracy is equal or better than the EKF.

The UKF leverages the UT to adjust the estimate and covariance in both the prediction and correction step. While the general structure of the Kalman filter including prediction and correction remains, the sigma points are now used in every step.

The UKF has been shown to outperform the EKF in terms of accuracy in a large number of applications. While it too struggles with second-order moments, only the second-order EKF [12] can match its accuracy, but is more computationally complex [13]. Therefore, the EKF's popularity over the objectively better UKF can probably attributed to its long existence.

2.3 Failure Detection

The quality of the estimate produces by the estimation algorithm is limited by the quality of its input data, especially if the algorithm is not robust. In practice, sensor data can contain anomalous sensor samples. While these samples might actually reflect reality, they usually stem from errors and failures during measurement or transmission. In this section, the term *failure* will refer to any bad samples that should be discarded, not surprising but

correct samples which can occur in random processes. The challenge is to discern these two while minimizing the number of false positives (leads to discarding good measurements) and false negatives (degrades estimate quality). This improves estimation quality and can be crucial in safety-critical applications. This section presents common failure types and methods to detect them.

2.3.1 Failure Types

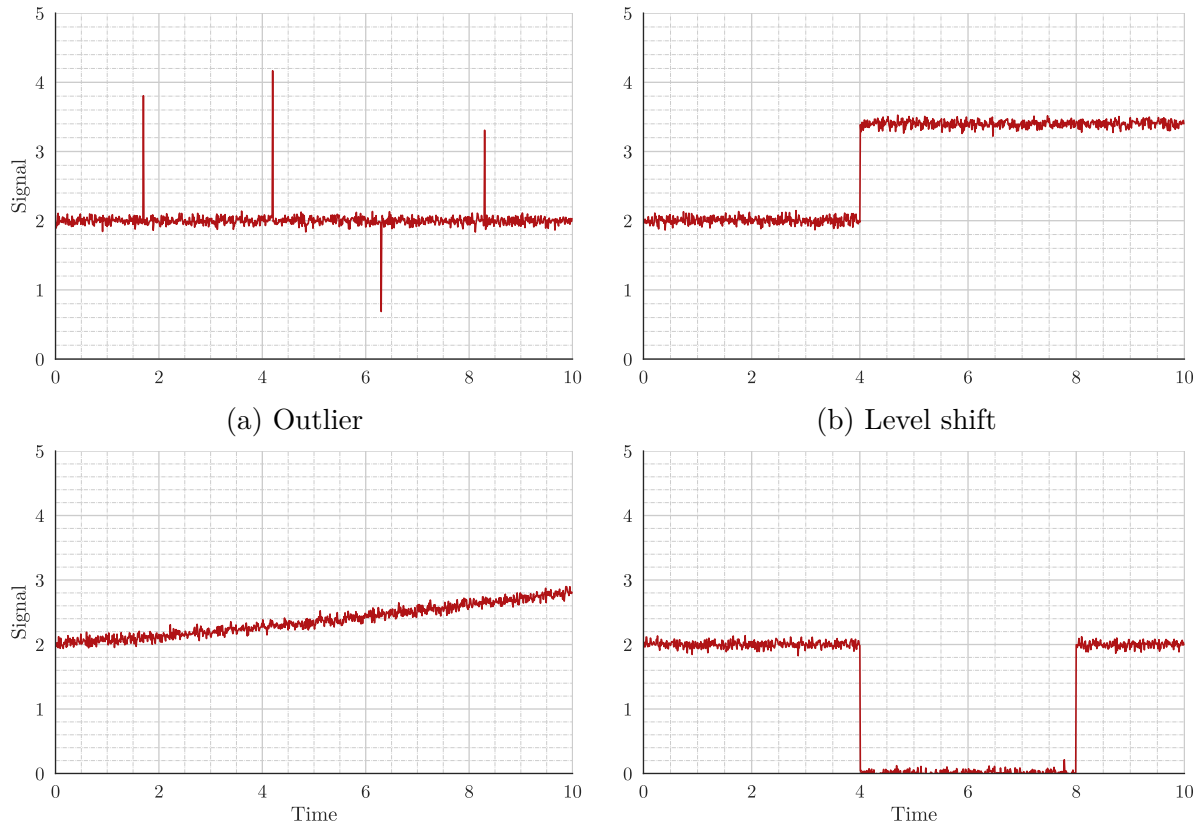
Failures can be the result of a variety of issues. For example, heavy vibration and shocks might cause acceleration spikes and an optical cross-correlation sensor might see a featureless surface, so the recognized velocity drops to zero. In cabled transmission, a physical connection might be defect, while in radio transmission, the signal might drop out temporarily or there might be electromagnetic interference. Possible causes can vary a lot between environments, but a highly dynamic environment with many mechanical and electric parts such as an electric race car is likely to produce failures at some point.

Failures can be classified in four broad categories [14, p. 19], [15, p. 165 ff.], shown in figure 3:

- Outlier/intermittent spike: a transient deviation from previous samples
- Level shift: a transient deviation of the mean value
- Drift/spurious trends: a slow variation of the mean value over time
- Null/signal dropout: not receiving input or zero

Drift often occurs as the result of integrating a biased signal, e.g., caused by low frequency noise or temperature changes. The true mean does not change, but the accumulating error results in an observed trending mean. While there are other cases of poor data quality such as clipping and excessive noise, they rather stem from calibration problems and hardware errors, so they are not regarded further.

Failures can be temporary or persistent. While outliers are inherently temporary, failures of the other three categories might or might not return to normal. For example, a GPS signal can drop out because the sensor has no satellite connectivity for a couple of seconds but then recovers (as in figure 3d), while another sensor can be permanently damaged and will not recover (as in figure 3b). Persistent but not transient failures are hard to detect, and especially spurious trends cannot easily be distinguished from real trends by their nature. This motivates the need for two separate methods, an approach that is also employed by [14]:



(a) Outlier

(b) Level shift

(c) Drift

(d) Null

Figure 3: Failure types

- A simple but strict method for transient failures
- A complex but lenient method for persistent failures

With this dual approach, false negatives for easy-to-detect transient failures can be minimized while false positives for hard-to-discern persistent failures can be avoided. Note that the choice of detection thresholds influences the strictness and therefore sensitivity to a high degree.

2.3.2 Rudimentary Methods

Arguably the simplest method is the range check, shown in equation 23. A plausible interval $[\tau_{min}, \tau_{max}]$ of values is defined in advance, and if the measurement z is outside of that range, it is marked as failure. For example, a speed of $200\,\mathrm{m\,s^{-1}}$ is highly unlikely for a race car, and also high negative speeds are implausible.

$$\tau_{min} < z < \tau_{max} \tag{23}$$

Another rudimentary method for detecting outlier is the difference of consecutive samples z_t, z_{t-1}, shown in equation 24. When the difference is higher than the maximum plausible change rate τ, the current measurement is marked as failure.

$$|z_t - z_{t-1}| < \tau \tag{24}$$

Performing a sanity check is a good first approach to detect gross failures. However, the thresholds should be set rather high to avoid false positives.

2.3.3 Statistical Methods

Methods based on the statistical properties of a signal can provide more granular checks than the previous rudimentary methods. One of the simplest robust statistics is the median, i.e. the center value when sorting a list of values, which is used in the method shown in equation 25 [16, p. 142]. An expected value $\tilde{z}_t$ is calculated using both the median of the last k samples and the median of the differences of the last $k+1$ samples. If the difference between the expected and the actual value exceeds the threshold, the current sample is marked as failure. The window size k influences the robustness but also the detection delay, and should therefore be chosen carefully. Note that, once the failure persists for more than k samples, following measurements will be regarded as valid.

$$|z_t - \underbrace{median(z_{t-k}, \ldots, z_{t-1}) + k \cdot median(z_{t-k} - z_{t-k-1}, \ldots, z_{t-1} - z_{t-2})}_{\tilde{z}_t}| < \tau \tag{25}$$

If an EKF is used, a chi-squared test can be used to detect anomalies based on the residuals, as shown in equation 26 [17, p. 4292], [18, p. 2050 f.]. Since the residual y is Gaussian distributed with the innovation covariance matrix P_{yy} (for nomenclature, refer to section 2.2.1), the normalized sum of squares of the residual values should be distributed according to a chi-squared distribution χ^2 with as many degrees of freedom as there are measurements. A failure is detected when the chi-squared test at a significance level of τ fails.

$$y = z - h(\hat{x}^-) \tag{26a}$$

$$P_{yy} = HP^-H^T + R \tag{26b}$$

$$r^T P_{yy}^{-1} r < \chi^2(\tau) \tag{26c}$$

Drift detection becomes easier when multiple sensors measure the same variable and can be compared. A simple variance-based approach based on this idea is shown in equation

27 [14, p. 20]. For each time step, the summed variance of n sensors is calculated, based on the mean μ_z of their measurements at that instant. Once the threshold $\tau \in (0, 1)$ is exceeded, the measurement with the highest contribution to the summed variance is marked as failure.

$$\sum_{i=1}^{n}(z_i - \mu_z)^2 < \tau \tag{27}$$

The last approach we will regard in this chapter can be used for outlier and drift detection, but requires $n > 2$ sensors for a variable like the previous method. Multiple instances of the same EKF, collectively called a bank of n EKFs, are executed in parallel. However, in the i-th filter, the measurement of the i-th sensor is disabled. In the event that a failure in the i-th measurement occurs, all filters except the one that does not incorporate the faulty measurement will show a very high residual.
Equation 28 [19, p. 3] shows how the sum of squared residuals[1] $NSSR^i$, normalized by the diagonal measurement covariance matrix R^i, is calculated from the residual y_i of the i-th filter. The NSSR combines all elements of the residual vector into a single metric, enabling easy comparison with a threshold τ_i. This threshold can be chosen empirically, or based on a desired statistical significance of the resulting χ^2 distribution [20, p. 3]. Note that a more sophisticated approach to fault isolation, i.e. locating the sensor producing failures, is required for the EKF bank to work with multiple simultaneous failures.

$$y^i = z^i - h^i(\hat{x}^-) \tag{28a}$$

$$\underbrace{(y^i)^T(R^i)^{-1}y^i}_{NSSR^i} < \tau_i \tag{28b}$$

[1] While the original paper talks about a *weighted* sum of squared residuals, we omitted the weight coefficient in favor of an adjustable threshold.

3 Design

In this chapter, we will look at the design of the state estimation and the vehicle platform it will be deployed to. The state estimation is built from the ground up, so there is a lot of flexibility regarding the design choices. The design is deliberately described in terms of generic components, enabling adoption on other platforms and software environments.

3.1 Vehicle Platform

The state estimation described in this thesis will be deployed to two electric four-wheel drive (4WD) race cars:

- a non-autonomous race car which is newly designed and manufactured
- an autonomous race car which is already existing but repurposed

These vehicles will in the following be referred to as EV and DV, respectively. Note that the DV is electric as well but has additional components required for autonomous driving.

3.1.1 Sensor Setup

The vehicles are equipped with a plethora of sensors, which can be used for state estimation. While some sensors like the ones for brake pressure are not relevant, most provide useful information. The relevant sensors are listed in table 1, with the last two columns denoting in which vehicle they are available. The sensors' locations in the vehicle shown in figure 4 In the ideal case, both vehicles would have the all sensors, since working with a homogeneous set of measurements is simpler, but financial and weight considerations mean that only the sensors necessary for the vehicle's use case are available.

Sensor type	Measured variables	EV	DV
Inertial measurement unit with gyrometer	a, ω	×	×
Optical cross-correlation velocity sensor	v, β	×	
Global navigation satellite system	$p, \|v\|$	×	×
Global navigation satellite system	ψ		×
Motor speed sensor	ω_{motor}	×	×

Table 1: Sensor setups for EV and DV

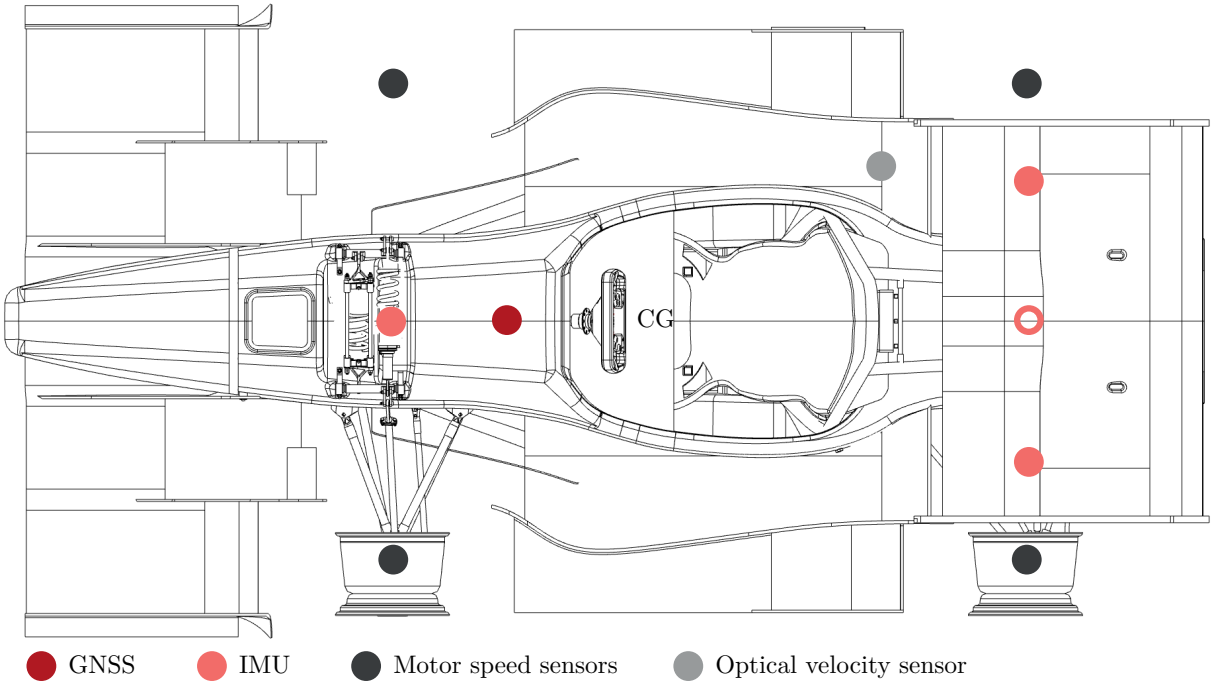


Figure 4: Locations of sensors in vehicle

IMU The EV features three inertial measurement units (IMUs) with gyrometers: one directly in front of the CG, and two situated on the rear left and right. This enables redundancy, since accelerations and rotations in the two-dimensional plane only require two IMUs. In the DV, the front one remains while only a single rear one, denoted by the circle with the missing center, is located directly behind the CG. While these sensors react very fast, the signals are rather noisy [21, p. 19 ff.].

Optical velocity sensor The optical cross-correlation velocity sensor, only available in the EV, provides longitudinal and lateral speed measurements, and therefore also measurements of the vehicle sideslip angle. It enables slip-free velocity measurements by correlating photosensor information of the surface over time, which uniquely determines the speed and direction of movement [22]. The fact that it is not influenced by slip, as velocity measurements from wheel speeds are, makes it a very valuable addition to the sensor setup. However, it is noisy and prone to temporary spikes and even failures on feature-less surfaces (see dark red line in figure 5 at 5.2 s).

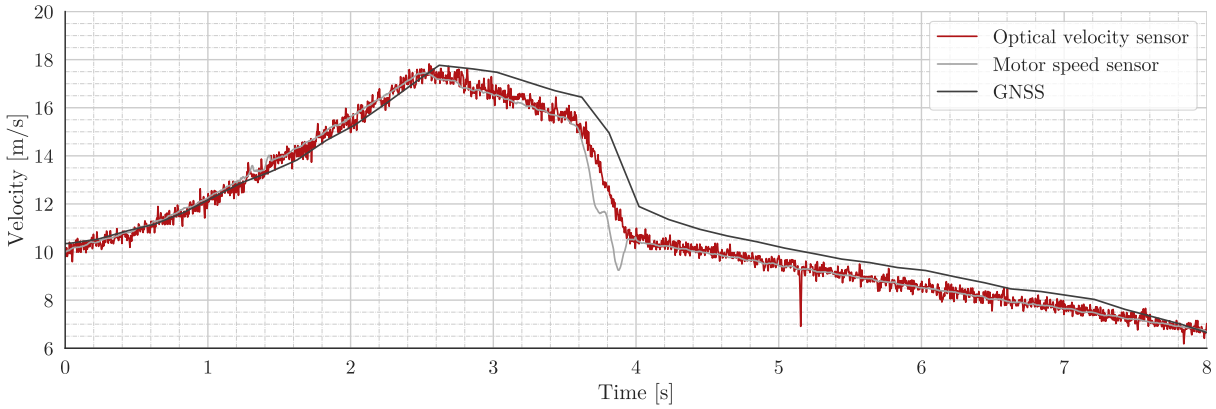


Figure 5: Comparison of velocity measurements from different sensors

GNSS A GNSS receiver is mounted in the front of both vehicles. Next to position measurements, they provide another source of speed information. The position accuracy is high in the range of a few centimeters. However, they are slow to react, with over 100 ms of delay in some situations [21, p. 27] (see dark grey line in figure 5). The receiver mounted in the DV additionally provides heading information, made possible by the relative position of two separated antennas mounted in the front and rear. This enables transformation of the speed into a velocity vector.

Motor speed sensor The rotary encoders in each of the four motors give individual rotation speed measurements for the four wheels. They are available in both vehicles and are assumed to be reliable, since the vehicle will not drive when they fail. However, when using them to calculate the vehicle velocity, deviations due to slip occur in highly dynamic situations (see light grey line in figure 5 at 3.8 s)

To summarize, the key differences between the EV and DV are as follows:

- Three IMUs in EV but two IMUs in DV
- No optical cross-correlation velocity sensor in DV
- No heading information in EV

The lack of the optical velocity sensor in the DV is inconvenient but not fatal, since the state estimation can compensate it. For track and obstacle recognition, the autonomous DV is furthermore equipped with stereo cameras and lidar sensors. These could be used for optical flow computations to gain more position, attitude and velocity information, but for this design, we rely solely on the sensors described in the previous paragraphs. The estimated state is, however, used to correct lidar measurements.

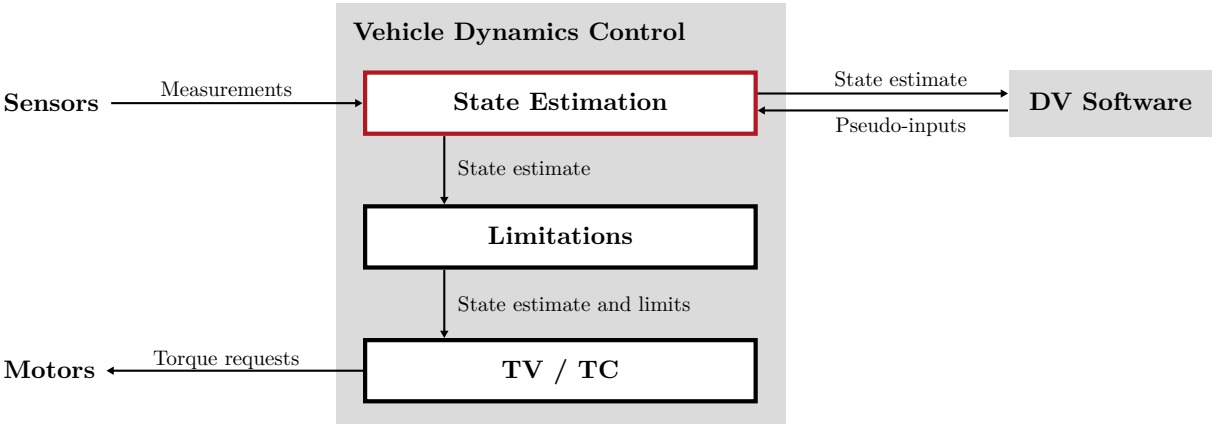


Figure 6: Integration of state estimation in software/hardware system

3.1.2 Computation System

All sensor and control signals converge at the central electronic control unit (ECU). The software running on that ECU is called vehicle dynamics control (VDC) and has several important tasks:

- Receive driver input from steering wheel and pedals
- Send torque requests to each of the four motors
- Limit speed and torque depending on situation
- Increase vehicle agility and stability through torque vectoring (TV)
- Optimize use of tire potential through traction control (TC)

Ultimately, the VDC's task is to help the driver to exploit the maximum physical potential of the vehicle through its TV and TC, collectively called performance components.

The state estimation is located between the sensor inputs and the aforementioned performance components, and is therefore part of the VDC as well. The integration with input and output signals is shown in figure 6. While the VDC is executed at a fixed rate, sensor inputs may arrive via a controller area network (CAN) bus system at different rates. For example, the VDC runs at 1000 Hz, i.e. it is scheduled to be executed every 1 ms, but optical velocity sensor measurements may arrive at 250 Hz and GNSS measurements even slower at 5 Hz. This may be due to a measurement overhead, such as satellite communication in case of the GNSS, which prohibits higher rates, but it may also be a conscious choice to reduce bus load and thus congestion. The state estimation must be able to fuse these measurements at different rates while providing continuous estimates at the highest rate. The time since the arrival of the last measurement is provided to aid this task. Note

Variable	Symbol	Unit	Coordinate system
Position	p_x, p_y	m	earth-fixed
Heading	ψ	rad	earth-fixed
Linear velocity	v_x, v_y	$\mathrm{m\,s^{-1}}$	vehicle
Linear acceleration	a_x, a_y	$\mathrm{m\,s^{-2}}$	vehicle
Yaw velocity	$\dot{\psi}$	$\mathrm{rad\,s^{-1}}$	–
Yaw acceleration	$\ddot{\psi}$	$\mathrm{rad\,s^{-2}}$	–

Table 2: State variables to be estimated

that this is different from incorporating *delayed* measurements, which is a much harder task.

The software to control autonomous driving in the DV runs on a separate, more powerful computer, which receives the state estimate from the ECU. The ECU then receives pseudo-driver inputs back from the motion controller and sends torque requests to the motors. However, the interface with the DV software is beyond the scope of this thesis.

3.2 Requirements

The goal of the state estimation is to provide a robust and accurate estimate of the vehicle state. This estimate will be used by the performance components of the VDC and the DV software, which therefore dictate the required state variables, which are listed in table 2. All quantities refer to the CG, which is regarded as center of the vehicle. However, all angular variables are valid for every point on the vehicle because it is assumed to be a rigid body. The position is defined relative to a reference point, but can easily be translated. Note that the vehicle state is only defined in the plane of two dimensions, since no three-dimensional state is required by subsequent components and vertical dynamics are negligible.

Given its goal and environment, several requirements must be fulfilled by our state estimation design:

1. Accurate estimation of vehicle state: the best possible estimate of the vehicle state given the current measurements and physical knowledge is required for maximal performance

2. Support for flexible sensors: both the EV and DV sensor setups must be supported, in the best case the state estimation should handle an arbitrary setup

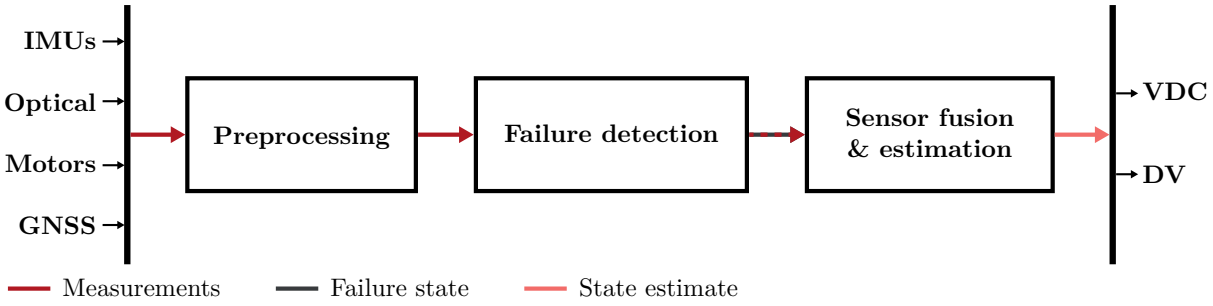


Figure 7: Architecture of state estimation

3. Robustness towards failures: outliers and other sensor failures may occur at any time and should not have an adverse effect on the estimation, which requires detection and appropriate handling

Additionally, our philosophy for the design and architecture is a preference for simplicity, according to Occam's razor ("the simplest solution is most likely the right one"). This is especially true if two methods work equally well but one is simpler. Only if simple methods do not yield the desired result, try more complex and complicated approaches. This philosophy has numerous benefit. A simple solution is one that makes less assumptions, and therefore generalizes better. It is also easier to understand, reason about and troubleshoot. This is of special significance for the application of the state estimation in a Formula Student team, where members and component maintainers change yearly and cannot dive deep into every topic. We apply this philosophy to small components but also to the high-level architecture, which facilitates maintenance and extension.

3.3 Architecture

Following our preference for simplicity, we propose a three-stage architecture, shown in figure 7. Measurements arriving from different sensors are first preprocessed, which, for example, includes unit conversions and coordinate transformations. In the next steps, failures are detected. The failure state informs the fusion of the measurements into an accurate state estimate (requirement 1) through an EKF, which is then used by the subsequent VDC's performance components and DV software.

A key feature is the unified mechanism for failure and sensor setup detection (requirements 2 and 3). Foundation is the realization that ultimately it does not matter if a measurement is unavailable due to an invalid signal or a missing sensor, therefore both cases can be treated the same. This allows a focused effort on a single component while reducing complexity. Still, the name *failure detection* will be used. The following sections describe each of the three component in detail.

3.3.1 Preprocessing

Goal of the preprocessing stage is to harmonize all measurements and pre-fuse them for the subsequent estimation stage. Mostly, this is only a simple conversion to SI units, e.g., velocities from $\mathrm{km\,h^{-1}}$ to $\mathrm{m\,s^{-1}}$, or angles from deg to rad. Doing these conversions as early as possible simplifies subsequent computations and reduces error potential due to wrong units. For other signals, more preprocessing is necessary.

IMU The most sophisticated preprocessing is required for the IMUs. The measurements are first rotated in three dimensions to correct any misalignment with the vehicle axes. This calibration must be performed every time the sensor orientation is changed. Then, we need a generic IMU fusion algorithm which can take an arbitrary number of sensors with known positions and fuse them into an estimate of the state variables a_x, a_y and $\ddot{\psi}$ ($\dot{\psi}$ will be estimated in the last stage). An accurate estimate of these variables is important, because they are extensively used in the sensor fusion and performance components. This helps with reducing noise, but also enables calculation of the angular acceleration, which cannot be measured directly with our sensors. We always calculate the linear acceleration and angular velocity in three dimensions, and, in case of more than two IMUs, the angular acceleration as well. We provide two approaches for IMU fusion with the same interface, enabling drop-in replacement. Since the available IMUs need to be known at this point, performing the failure detection here instead of in the actual failure detection component is necessary.

The first approach fuses measurements by averaging, and is therefore called *mean-based IMU fusion*. First, the angular velocity measurements from the gyrometers are averaged. Then, the angular accelerations are derived using equation 8 from all combinations of two IMUs and averaged. For example, in the EV, the combinations 1–2, 2–3 and 1–3 are regarded. In case only two IMUs are available, they are assumed to have a zero z-position and the angular acceleration is calculated using the simplified equation 9. In case only a single IMU is available, we assume $\alpha = \mathbb{0}$. Finally, the linear accelerations are transformed to the CG using equation 5 and averaged as well, which eliminates the effect of tangential and centripetal acceleration.

The second *maximum-likelihood-based IMU fusion* approach is more sophisticated and is presented in [23], with an extension proposed in [24]. The idea is to first find a maximum-likelihood estimate for ω given the linear acceleration and angular acceleration measurements of all sensors. This is done by solving a least-squares problem using the Gauss-Newton algorithm, with a fixed number of 10 iterations instead of checking for convergence. The maximum-likelihood estimate for a and α is then as simple as plugging the previously found estimate into a linear equation.

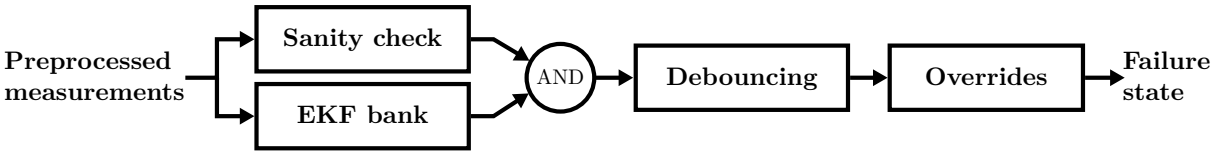


Figure 8: Architecture of failure detection

Optical velocity sensor The optical cross-correlation velocity sensor is located on the right side behind the CG, so its measurements need to be corrected to account for the component induced by a yaw rotation. Equation 4 is used to transform the measurements to the CG. Note that the yaw rate is required, so a direct signal from the IMU preprocessing is required.

GNSS Position measurements from the GNSS arrive in latitude-longitude-height coordinates, but need to be transformed to topocentric east-north coordinates, i.e. coordinates on a plane tangential to the earth at a certain reference point [25, p. 475 f.]. The transformation is based on the ellipsoid WGS 84 earth model used in GPS measurements. The reference point is the first known position, and all future positions will be represented in x-y-coordinates relative to the reference point.

Only speed measurements are available through GNSS, i.e. the absolute value of the velocity. However, since the vehicle sideslip angle β is usually small and less than 0.1 rad, the longitudinal velocity can be approximated to be the speed, i.e. $v_x \approx \|v\|$ and $v_y \approx 0$. This simplification works well in our design, since the GNSS speed will only be used for failure detection, not as part of the final state estimate. If separate measurements for the eastward and northward velocity become available, they can be transformed into vehicle coordinates using equation 10. Note that this requires heading information, which might not always be available.

Motor speed sensor The motor rotation speeds are used as another vehicle velocity source via equation 12, averaging over all four wheels to fuse them as described in section 2.1.6. Again, only the longitudinal velocity can be derived.

3.3.2 Failure Detection

Our approach towards failure detection is heavily based on the dual method presented in section 2.3.1, to help minimize both false-positives and false-negatives. The complete failure detection stage is shown in figure 8. It receives signals from the preprocessing stage and determines the binary sensor state (OK, NOT OK), i.e. whether the sensor is available and does not contain failures. The target is to support one sensor failure, since more are unlikely.

Every input for the estimation stage undergoes a sanity check, presented in equations 23 and 24. This is also the failure detection method performed to select the IMUs for the IMU fusion. The range of the signal and the difference to the last sample are checked for plausibility, and the time since the last sample is used to detect a sensor timeout by means of a state machine. This is also the mechanism to detect the sensor setup, because missing sensors will timeout permanently and therefore be regarded as failure. While we described many other methods in section 2.3.3, we chose the simplest method with the least parameters.

Accurate knowledge of the vehicle velocity is of special importance for the performance component. Since we have three velocity sources, we can apply the EKF bank approach from equation 28. We run three EKFs in parallel using the model presented in the next section. For each EKF, one sensor is disabled, so the other EKFs generate high residuals in case of a failure of that sensor. This case is checked by comparing the normalized sum of squared residuals with a threshold. The covariance of all velocity measurements is identical to ensure equal weighting. This is necessary to detect persistent failures, since residuals caused by failures would otherwise quickly return to zero even if failures persist because the low-covariance measurements pull the estimate towards the faulty value. Note that successful fault isolation requires at least three measurements, otherwise only fault detection is possible. Because velocities are also checked by the EKF bank, their plausibility check is configured to be rather lenient.

The results from the sanity check and the EKF bank are logically and-ed, with OK for the results of non-velocities from the EKF bank. Detected failures are then debounced to increase robustness in case failures appear and disappear quickly. Debouncing means that a signal is marked as NOT OK for a certain span of time after an failure has been detected, even if no failure is currently detected. Only if the time span passes without another failure, the signal is marked as OK once again. This logic is modeled using a state machine.

Last in the chain is a three-way override for each signal. Either the failure state can be directly forwarded, or forced to be OK or NOT OK. On the one hand, this allows to correct erroneous detection results, on the other hand it also provides a way to manually enable and disable sensors in case the sensor setup detection fails.

3.3.3 EKF

The final stage is the actual estimation of state variables. We use an EKF for this task, since it is the standard and has been successfully applied in a lot of vehicles. Also, the system is not extremely non-linear, minimizing the potential benefits of using an UKF.

The linear acceleration and yaw acceleration are already estimated in the IMU fusion, so it remains to estimate the position, heading, velocity and yaw rate. While the yaw rate is already part of the IMU fusion, it can further be improved through correlation with the yaw acceleration. That is exactly what the EKF is doing for all state variables as well: fusing available measurements which are corrected and correlated through a physical model.

To define the specific EKF algorithm, we first need to set up the vectors shown in equation 29. The state vector x contains our variables to be estimated, the input vector u contains variables that influence state propagation, and the measurement vector z contains our sensor measurements.

$$x = \begin{bmatrix} p_x\ , p_y,\ v_x,\ v_y,\ \psi,\ \dot{\psi} \end{bmatrix}^T \tag{29a}$$

$$u = \begin{bmatrix} a_x,\ a_y,\ \ddot{\psi} \end{bmatrix}^T \tag{29b}$$

$$z = \begin{bmatrix} p_{x,GNSS},\ p_{y,GNSS},\ v_{x,opt.},\ v_{y,opt.},\ v_{x,GNSS},\ v_{x,motor},\ \psi_{GNSS}, \dot{\psi}_{IMU} \end{bmatrix}^T \tag{29c}$$

Then we set up the process function to propagate the state vector and the measurement function to predict measurements. The process is defined by the system of differential equations shown in equation 30, which describes the derivative of the state vector with respect to time. Therefore they constitute our physical model of the vehicle. It is a basic kinematic model which often works better than more sophisticated models like the non-linear single track model [26]. For an explanation of the equations for position and velocity, see sections 2.1.4 and 2.1.5. It should be noted, that the lack of an initial heading, which is the case in the EV, makes accurate prediction of the position next to impossible and also impairs the estimates of other state variables.

$$\dot{x} = \frac{\partial x}{\partial t}(x, u) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{\psi} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} v_x \cdot cos(\psi) - v_y \cdot sin(\psi) \\ v_x \cdot sin(\psi) + v_y \cdot cos(\psi) \\ a_x + \dot{\psi} v_y \\ a_y - \dot{\psi} v_x \\ \dot{\psi} \\ \ddot{\psi} \end{bmatrix} \tag{30}$$

To predict the current state, we use a forward Euler integration as shown in equation 31, which evaluates the current derivative and adds it to the previous state. This method works well for small time steps Δt, but a higher-order Runge–Kutta method might be necessary to guarantee numerical stability for larger time steps.

$$\hat{x}_k^- = f(\hat{x}_{k-1}^+, u_k) = \hat{x}_{k-1}^+ + \Delta t \cdot \left.\frac{\partial x}{\partial t}\right|_{x=\hat{x}_{k-1}^+, u=u_k} \tag{31}$$

To propagate the covariance matrix, we need the Jacobian of the process function with respect to the state vector. Additionally, we need the process noise covariance, which is very hard to determine. Therefore we assume that process noise does not exist, but in exchange consider the uncertainty introduced by the measurements in the input vector, which are not considered in the standard EKF equations. This is shown in equation 32, which is adapted from equation 16. I_6 is the 6×6 identity matrix and Q_k is the input covariance matrix. F_k and B_k are the discretized Jacobians of the process function with respect to the state vector and input vector, respectively. See appendix B for the expanded forms of the process function's partial derivatives.

$$P_k^- = F_k P_{k-1} F_k^T + B_k Q_k B_k^T \tag{32a}$$

$$F_k = \frac{\partial f}{\partial x} = I_6 + \Delta t \cdot \left.\frac{\partial \dot{x}}{\partial x}\right|_{x=\hat{x}_{k-1}^+, u=u_k} \tag{32b}$$

$$B_k = \frac{\partial f}{\partial u} = \Delta t \cdot \left.\frac{\partial \dot{x}}{\partial u}\right|_{x=\hat{x}_{k-1}^+, u=u_k} \tag{32c}$$

The measurement function is just a linear function which maps state variables directly to their corresponding measurement prediction via the measurement matrix H (see appendix B.3). Individual measurements can be enabled and disabled using the binary measurement mask m, which is turned into a diagonal matrix. The mask is based on the sensor state from the failure detection. Additionally, for sensors which do not send data at the rate that the VDC is executed, measurements are only enabled when new data arrive . For this, we check if the time since the arrival of the last measurement is less than in the previous execution of the VDC. Furthermore, GNSS velocity measurements are only enabled if the other velocity sources are unavailable, since its delay in dynamic situations impairs the estimate.

$$\hat{z} = diag(m) H \hat{x} \tag{33}$$

The same measurement mask is also applied to the real measurements and the residual. Furthermore, we normalize the predicted and measured heading to be between 0 and 2π. Other than that, the correction stage of the EKF uses exactly the equations described in 2.2.1. Since the EKF works iteratively, the state vector and its covariance matrix need

to be stored between iterations. Before the first iteration, the state vector is is initialized with zeros and its covariance matrix is set to the identity matrix.

4 Implementation

The design and architecture described in the previous chapter is agnostic of the technology used for implementation. However, for evaluation and deployment of the state estimation to the race car, we are bound to the options offered by ECU our VDC software is running on, since the state estimation if part of that software, as described in section 3.1.2. We are using the ES910 produced by ETAS, which can connect to all CAN buses for signal transport and runs code on top of a real-time operating system. Being a rapid prototyping module, the ES910 has significantly higher computing power with a floating-point processor and diagnostic interfaces [27, p. 16]. Its software can be programmed with three technologies [27, p. 17], [28, p. 10]:

- ASCET: application development tool by ETAS, which generates safe C code from textual or visual models
- MATLAB/Simulink: a model-based design tool by MathWorks, which generates C code
- Custom code: write C code manually

While eventually the software will be transformed into C code, the model-based tools provide a convenient development experience, like visualization of signal flow, and safeguards against common errors. While both ASCET and MATLAB/Simulink are widely used, we decided on the latter option, since it is widely used for simulations and enables rapid testing of different approaches. More importantly, however, is the fact that the existing VDC software is already developed in Simulink. Therefore, integration and build pipelines are already set up and integration is simple.

4.1 Development

Our basic building blocks for creating Simulink models are blocks, subsystems, referenced models, signals and buses. Best practices are described in the official modeling guidelines[29], which we follow. Blocks are the basic functional unit, e.g., performing integration, boolean algebra, mathematical operations and input/output. They live in a

model, which defines their execution context and is a single file. A special block is the MATLAB function, which can execute any MATLAB code that can be compiled to C code. Subsystems are a visual or functional group of blocks inside a model which cannot be reused. Other models can be referenced and thereby included in a model, facilitating reuse and collaboration because they are separate files. Signals are connections between blocks in the same model or across model boundaries. They can be grouped into buses, which simplify interface design for high-level components with many input/output signals.

The VDC is a large Simulink model that includes the components shown in figure 6 as referenced models, which means the state estimation is stored in a separate model file. Previously, a rudimentary state estimation without proper failure detection receives measurements and provided a state estimate for the subsequent components. Effectively, we replace this component while keeping the same interface, which enables drop-in replacement without the need for adaption of any other components.

We model the components of the state estimation as subsystems, since they are never reused and issues arising from collaboration, e.g., merge conflicts, are not a problem. Only components which are reused, such as the sanity check, are created as separate models and then referenced. While any algorithm and mathematical operation can be modeled using Simulink blocks alone, we use MATLAB code for the implementation of longer algorithms like the EKF and IMU fusion but also many smaller operations. This approach proved to work well; Simulink acts as glue around the components which models signal flow, while MATLAB code is clearer when it comes to functionality and can easily be unit-tested easily in isolation. This separation of concerns makes the state estimation software clearer and more readable, another manifestation of our preference for simplicity.

We use strict bus definitions to describe the interface of components, including data types, dimensions and units. For example, there are buses for measurements from specific sensors, the total sensor state and the final estimate. While the definition of buses is development overhead, it helps to avoid logical errors like incorrect units that are rather hard to find and require manual troubleshooting. Furthermore, it helps to think of the components as black boxes with specified inputs and outputs, which then leads to the required functionality of the component. Single, short-lived signals inside of the components remain plain signals, however.

Previously, shared design data were stored in the MATLAB base workspace, which was set up by a parameter script. We migrated bus and parameter definitions as well as simulation configurations to data dictionaries, which are referenced from all models and submodels. This is the officially recommended way to store shared data, which makes it easy to track changes and control variable scope [30].

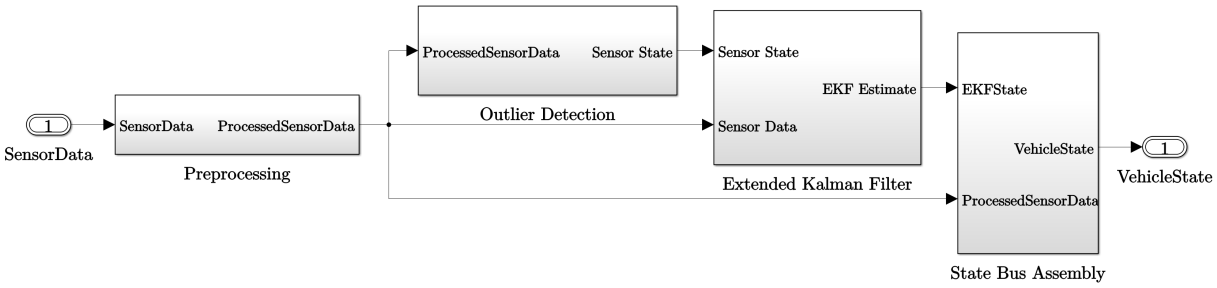


Figure 9: Implementation as Simulink model

Development comprised three major phases, following the three components shown in figure 7. First, the preprocessing stage was implemented, which can be directly connected to the state estimate output, making isolated development of this stage possible. Next, we developed the EKF responsible for sensor fusion, improving the measurements coming from the preprocessing stage. We also added a block to assemble the vehicle state bus by combining preprocessed data and the state estimate from the EKF. Only in the last step, we implemented the outlier detection, determining the measurement mask for the EKF. The final Simulink model for comparison with the designed architecture is shown in figure 9. During each phase, the components were tested with recorded measurement data from the same sensors to guide design decisions. While testing the state estimation model in the context of the whole VDC is possible, we created a separate model for development so we can test the state estimation in isolation. We were only able to perform model-in-the-loop testing, because the hardware required for hardware-in-the-loop testing was not available during development.

4.2 Deployment

During development, the state estimation model can easily be executed on the development computer. However, deploying the Simulink model to the ES910 platform requires a series of steps, shown in figure 10.

Code Generation The first step is to generate C code from the Simulink model and included MATLAB code using Simulink Coder with an specific target, adjusting code generation to the requirements of the ES910. The generated code includes a step function to execute one iteration of the model. Additionally to the code, a module and interface description is generated. Note that Simulink Coder does not support all language features, so some code might need to be restructured for code generation.

Figure 10: Build and deployment toolchain

Integration ETAS' INTECRIO is used to integrate the generated code module into the ES910 system based on the module description. This includes scheduling the VDC for execution every 1 ms and connecting CAN signals to their respective model inputs and outputs. INTECRIO builds all modules into an executable file and a file describing calibration and measurement variables.

Installation The executable file can be transferred to the ES910 and flashed onto its memory using ETAS' INCA. Once the ES910 is started, INCA can also be used to monitor signals, both internal in the software and external coming from the CAN, and control parameters.

Once the state estimation is deployed to the ES910 and connected to all sensors in the vehicle, it can be calibrated and tuned to the hardware. This includes adjustment of the failure detection thresholds, IMU rotation correction and EKF measurement covariance settings.

5 Evaluation

Fundamental: no ground truth, therefore residuals are a good indicator non-zero-mean residual means model mismatch [26, p. 158]

Results bad position in EV without heading measurements, because velocity cannot be used properly to estimate position

compare raw measurements with velocity estimate show results for each state variable effect of different debouncing times and thresholds show velocity without kistler to simulate dv

compare max change rate and mean based

max likelihood IMU fusion not better for omega and a, but less noise for alpha when more than 2 imus -> better EKF results does not work when 2 imus on same axis use mean based fusion because occams razor

show all wssrs for ekf bank compare with median outlier detection approaches

Discussion, impact of results on esleek prepare for DV initialization of position, maybe not include position when heading is unavailable

only testing can show if outlier detection works, if not maybe try chi-squared test and variance test

6 Conclusion

In this thesis, we first presented different approaches for state estimation and failure detection. Then we described the design and implementation of a robust and flexible state estimation for a race car. At the core is a simple three-stage architecture, comprising a preprocessing component for measurement harmonization and IMU fusion, a unified failure and sensor setup detection providing robustness and support for flexible sensor sets, and an EKF-based state estimation performing model-based sensor fusion. Our evaluation shows that the provided state estimate is accurate even in the presence of outliers and sensor failures.

While the estimation of the vehicle velocity is already good, it can likely be improved with a better algorithm for transformation of the motor speeds such as the ones presented in [5]. Automated calibration of the IMUs, i.e. determining the orientation so measurements can be aligned with the vehicle axes, possibly every time the state estimation is started, could be beneficial as well since many components rely on good IMU measurements. Lastly, a tighter integration with the DV software and camera/lidar information could improve state estimation and vehicle performance in the future.

Bibliography

[1] Formula One World Championship Limited, *Formula 1 viewing figures 2019: F1 broadcast to 1.9 billion total audience in 2019*, 2020. [Online]. Available: https://www.formula1.com/en/latest/article.f1-broadcast-to-1-9-billion-fans-in-2019.4IeYkWSoexxSIeJyuTrk22.html.

[2] ISO, *Road vehicles — Vehicle dynamics and road-holding ability — Vocabulary*, 2011.

[3] D. Gross, W. Hauger, J. Schröder, W. A. Wall, and S. Govindjee, *Engineering Mechanics 3*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, ISBN: 978-3-642-53711-0. DOI: 10.1007/978-3-642-53712-7.

[4] W. F. Milliken and D. L. Milliken, *Race Car Vehicle Dynamics*. Great Britain: Society of Automotive Engineers Inc, 1996.

[5] C. K. Song, M. Uchanski, and J. K. Hedrick, "Vehicle Speed Estimation Using Accelerometer and Wheel Speed Measurements," in *SAE Technical Paper Series*, ser. SAE Technical Paper Series, SAE International400 Commonwealth Drive, Warrendale, PA, United States, 2002. DOI: 10.4271/2002-01-2229.

[6] S. K. Mitter, "Filtering and stochastic control: a historical perspective," *IEEE Control Systems*, vol. 16, no. 3, pp. 67–76, 1996, ISSN: 1066-033X. DOI: 10.1109/37.506400.

[7] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960, ISSN: 0021-9223. DOI: 10.1115/1.3662552.

[8] S. J. Julier and J. K. Uhlmann, "Unscented Filtering and Nonlinear Estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004, ISSN: 0018-9219. DOI: 10.1109/JPROC.2003.823141. [Online]. Available: https://www.cs.ubc.ca/~murphyk/Papers/Julier_Uhlmann_mar04.pdf.

[9] M. S. Grewal and A. P. Andrews, "Applications of Kalman Filtering in Aerospace 1960 to the Present [Historical Perspectives]," *IEEE Control Systems*, vol. 30, no. 3, pp. 69–78, 2010, ISSN: 1066-033X. DOI: 10.1109/MCS.2010.936465.

[10] S. S. Haykin, Ed., *Kalman filtering and neural networks*, ser. Adaptive and learning systems for signal processing, communications, and control. New York: Wiley, 2001, ISBN: 9780471221548.

[11] Simon J. Julier and Jeffrey K. Uhlmann, "New extension of the Kalman filter to nonlinear systems," in *Signal Processing, Sensor Fusion, and Target Recognition VI*, Ivan Kadar, Ed., vol. 3068, SPIE, 1997, pp. 182–193. DOI: 10.1117/12.280797.

[12] M. Roth and F. Gustafsson, "An efficient implementation of the second order extended Kalman filter," in *14th International Conference on Information Fusion*, 2011, pp. 1–6.

[13] F. Gustafsson and G. Hendeby, "Some Relations Between Extended and Unscented Kalman Filters," *IEEE Transactions on Signal Processing*, vol. 60, no. 2, pp. 545–555, 2012, ISSN: 1053-587X. DOI: 10.1109/TSP.2011.2172431.

[14] J. Kabzan, M. d. I. La Valls, V. Reijgwart, H. F. C. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler, N. Gosala, M. Gupta, R. Sivanesan, A. Dhall, E. Chisari, N. Karnchanachari, S. Brits, M. Dangel, I. Sa, R. Dubé, A. Gawel, M. Pfeiffer, A. Liniger, J. Lygeros, and R. Siegwart, *AMZ Driverless: The Full Autonomous Racing System*, 2019. [Online]. Available: https://arxiv.org/pdf/1905.05150.pdf.

[15] H. Himmelblau, J. H. Wise, A. G. Piersol, and M. R. Grundvig, *Handbook for Dynamic Data Acquisition and Analysis - IES Recommended Practices 012.1*. 1994. [Online]. Available: https://trs.jpl.nasa.gov/bitstream/handle/2014/33780/94-0509.pdf.

[16] S. Basu and M. Meckesheimer, "Automatic outlier detection for time series: an application to sensor data," *Knowledge and Information Systems*, vol. 11, no. 2, pp. 137–154, 2007, ISSN: 0219-1377. DOI: 10.1007/s10115-006-0026-6.

[17] K. Hausman, S. Weiss, R. Brockers, L. Matthies, and G. S. Sukhatme, "Self-calibrating multi-sensor fusion with probabilistic measurement validation for seamless sensor switching on a UAV," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 4289–4296, ISBN: 978-1-4673-8026-3. DOI: 10.1109/ICRA.2016.7487626.

[18] M. I. Valls, H. F. Hendrikx, V. J. Reijgwart, F. V. Meier, I. Sa, R. Dube, A. Gawel, M. Burki, and R. Siegwart, "Design of an Autonomous Racecar: Perception, State Estimation and System Integration," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 2048–2055, ISBN: 978-1-5386-3081-5. DOI: 10.1109/ICRA.2018.8462829.

[19] T. Kobayashi and D. L. Simon, Eds., *Application of a Bank of Kalman Filters for Aircraft Engine Fault Diagnostics*, vol. Volume 1: Turbo Expo 2003, Turbo Expo: Power for Land, Sea, and Air, 2003. DOI: 10.1115/GT2003-38550.

[20] W. Xue, Y.-q. Guo, and X.-d. Zhang, "A Bank of Kalman Filters and a Robust Kalman Filter Applied in Fault Diagnosis of Aircraft Engine Sensor/Actuator," in *Second International Conference on Innovative Computing, Informatio and Control (ICICIC 2007)*, IEEE, 2007, p. 10, ISBN: 0-7695-2882-1. DOI: 10.1109/ICICIC.2007.3.

[21] C. Biel, "Konzeptionierung und automatisierte Parametrierung der Antriebsregelung eines Formula Student Rennwagens," 2019.

[22] A. Bellof, "Method and device for determining the direction and speed of an object," DE4313497 (A1), 1994. [Online]. Available: https://worldwide.espacenet.com/publicationDetails/biblio?FT=D&CC=DE&NR=4313497A1&KC=A1.

[23] I. Skog, J.-O. Nilsson, P. Handel, and A. Nehorai, "Inertial Sensor Arrays, Maximum Likelihood, and Cramér–Rao Bound," *IEEE Transactions on Signal Processing*, vol. 64, no. 16, pp. 4218–4227, 2016, ISSN: 1053-587X. DOI: 10.1109/TSP.2016.2560136.

[24] J. Wahlstrom, I. Skog, and P. Handel, "Inertial Sensor Array Processing with Motion Models," in *2018 21st International Conference on Information Fusion (FUSION)*, I. C. o. I. Fusion, Ed., Piscataway, NJ: IEEE, 2018, pp. 788–793, ISBN: 978-0-9964527-6-2. DOI: 10.23919/ICIF.2018.8455269. [Online]. Available: https://www.researchgate.net/profile/Johan_Wahlstroem3/publication/327483205_Inertial_Sensor_Array_Processing_with_Motion_Models/links/5cd575a092851c4eab924a23/Inertial-Sensor-Array-Processing-with-Motion-Models.pdf.

[25] M. S. Grewal, L. R. Weill, and A. P. Andrews, *Global Positioning Systems, Inertial Navigation, and Integration*, 2007. DOI: 10.1002/0470099720.

[26] Alexander Wischnewski, Tim Stahl, Johannes Betz, and Boris Lohmann, "Vehicle Dynamics State Estimation and Localization for High Performance Race Cars," *IFAC-PapersOnLine*, vol. 52, no. 8, pp. 154–161, 2019, ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2019.08.064. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2405896319303957.

[27] ETAS GmbH Stuttgart, "ES910.3-A Prototyping and Interface Module User's Guide," 2018. [Online]. Available: https://www.etas.com/download-center-files/products_ES900/ES910.3-A_UG_R09_EN.pdf.

[28] ——, "INTECRIO V4.7 User's Guide," 2019. [Online]. Available: https://www.etas.com/download-center-files/products_INTECRIO_Software_Products/INTECRIO_V4.7.2_Manual.pdf.

[29] The MathWorks, Inc., *Component-Based Modeling Guidelines - MATLAB & Simulink*, 2020. [Online]. Available: `https://www.mathworks.com/help/simulink/ug/component-based-modeling-guidelines.html`.

[30] ——, *Determine Where to Store Variables and Objects for Simulink Models - MATLAB & Simulink*, 2020. [Online]. Available: `https://www.mathworks.com/help/simulink/ug/determine-where-to-store-data-for-simulink-models.html`.

A Expanded Rigid Body Equations

A.1 Transformation of Linear Velocity

$$\begin{aligned}\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} &= \begin{bmatrix} v_x^{CG} \\ v_y^{CG} \\ v_z^{CG} \end{bmatrix} + \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \times \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} \\ &= \begin{bmatrix} v_x^{CG} \\ v_y^{CG} \\ v_z^{CG} \end{bmatrix} + \begin{bmatrix} \dot{\theta} r_z - \dot{\psi} r_y \\ \dot{\psi} r_x - \dot{\phi} r_z \\ \dot{\phi} r_y - \dot{\theta} r_x \end{bmatrix}\end{aligned}$$

A.2 Transformation of Linear Acceleration

$$\begin{aligned}\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} &= \begin{bmatrix} a_x^{CG} \\ a_y^{CG} \\ a_z^{CG} \end{bmatrix} + \begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} \times \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} + \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \times \left(\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \times \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} \right) \\ &= \begin{bmatrix} a_x^{CG} \\ a_y^{CG} \\ a_z^{CG} \end{bmatrix} + \begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} \times \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} + \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \times \begin{bmatrix} \dot{\theta} r_z - \dot{\psi} r_y \\ \dot{\psi} r_x - \dot{\phi} r_z \\ \dot{\phi} r_y - \dot{\theta} r_x \end{bmatrix} \\ &= \begin{bmatrix} a_x^{CG} \\ a_y^{CG} \\ a_z^{CG} \end{bmatrix} + \begin{bmatrix} \ddot{\theta} r_z - \ddot{\psi} r_y \\ \ddot{\psi} r_x - \ddot{\phi} r_z \\ \ddot{\phi} r_y - \ddot{\theta} r_x \end{bmatrix} + \begin{bmatrix} \dot{\theta}(\dot{\phi} r_y - \dot{\theta} r_x) - \dot{\psi}(\dot{\psi} r_x - \dot{\phi} r_z) \\ \dot{\psi}(\dot{\theta} r_z - \dot{\psi} r_y) - \dot{\phi}(\dot{\phi} r_y - \dot{\theta} r_x) \\ \dot{\phi}(\dot{\psi} r_x - \dot{\phi} r_z) - \dot{\theta}(\dot{\theta} r_z - \dot{\psi} r_y) \end{bmatrix}\end{aligned}$$

B Extended Kalman Filter Equations

B.1 Jacobian of Process Function With Respect to State Vector

$$\frac{\partial \dot{x}}{\partial x} = \begin{bmatrix} 0 & 0 & cos(\psi) & -sin(\psi) & -v_x \cdot sin(\psi) - v_y \cdot cos(\psi) & 0 \\ 0 & 0 & sin(\psi) & cos(\psi) & v_x \cdot cos(\psi) - v_y \cdot sin(\psi) & 0 \\ 0 & 0 & 0 & \dot{\psi} & 0 & v_y \\ 0 & 0 & -\dot{\psi} & 0 & 0 & -v_x \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

B.2 Jacobian of Process Function With Respect to Input Vector

$$\frac{\partial \dot{x}}{\partial u} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

B.3 Measurement Matrix

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$