HYBRID SIMULATION MODELS –
A SOLUTION FOR CONCEPT TO REALISM CONTINUITY

DOCUMENT TYPE:	**DELIVERABLE**
DELIVERABLE N^0:	**D3.1A**
DISTRIBUTION LEVEL:	**PUBLIC**
DATE:	**02/06/2022**
VERSION:	**FINAL**

AUTHOR(S):	**LARS MIKELSONS (UNIVERSITY AUGSBURG)**
	KAY BIERZYNSKI (INFINEON)
	TOBIAS KAMP (DLR)
	CASPAR BIERI (DLR)
	OLAF VAN DER SLUIS (PHILIPS)

FORMAL REVIEWED:	**MEDINA ĆUSTIĆ (VIRTUAL VEHICLE)**
APPROVED:	**MARTIN BENEDIKT (VIRTUAL VEHICLE)**

PROJECT ACRONYM:	**UPSIM**
PROJECT TITLE:	**UNLEASH POTENTIALS IN SIMULATION**
ITEA PROJECT N^0:	**19006**
CHALLENGE:	**SMART ENGINEERING**
PROJECT DURATION:	**01/10/2020 - 30/09/2023**
PROJECT WEBSITE:	**WWW.UPSIM-PROJECT.EU**
COORDINATION:	**VIRTUAL VEHICEL RESEARCH GMBH**
PROJECT LEADER:	**DR. MARTIN BENEDIKT**

Contents

1 Introduction

UPSIM aims at providing tools, methods and processes for credible industrial simulation. One key capability for a company to generate credible simulations is to include newly gained information in an existing model. This results in the ability to reduce the gap between the model and the real product continuously in the advancing development process and during operation. However, this is not feasible relying on physical modelling only. Physical modelling starts with the determination of the purpose of a model and subsequently assumptions on the importance of physical effects are made. These assumptions represent the foundation of the model and are the major driver for both, the modelling effort, and the reachable accuracy of the model. Typically, low modelling effort comes along with limited model accuracy. Starting with extensive assumptions leading to low modelling effort often leads to a situation, where refinement of the model is not easily doable, and accuracy is at its limit. Consequently, re-modelling becomes necessary and continuity in reducing the gap between model and reality is broken. However, starting with lightweight assumptions leads to high modelling effort at the very beginning of a development cycle and hence to the risk of having the models available too late for first design decisions. Additionally, for such models, parameters are often not at hand in the beginning of a product development.

Recent approaches combine methods from machine learning with physical modelling. This allows the usage of data to refine and respectively improve physical models. The resulting models are called hybrid models in the following. Since nowadays data usually becomes available during development, and especially during operation, these hybrid modelling approaches pave the way to continuously reduce the gap between model and realism. Within UPSIM, different approaches for hybrid modelling are investigated. Subsequently, their ability to support concept to realism continuity is validated within the UPSIM use cases.

In this document, the approaches for hybrid modelling considered within UPSIM are described in Section 2. Moreover, the approaches are classified depending on how equations and data are combined using a taxonomy from [1]. In Section 3, usage of the approaches within the use cases is presented with a focus on how concept to realism continuity shall be achieved. Summarizing the approaches for concept to realism continuity, this deliverable is part of the milestone MS3 “Update in System Simulation Governance Processes and Methodologies and Tooling aligned”. Clearly, due to the point in time this document was prepared, Section 3 lacks a validation of the proposed approaches to reach concept to realism continuity. Therefore, the document will be updated with results and lessons learned from the realization of the use cases later during the project and released as Deliverable 3.1b.

2 Hybrid Modelling Approaches

Following the rapid progress in the field of machine learning in recent years, it is now time to apply the new methodologies to industrial applications. In the context of UPSIM, this applies especially to the data-based modelling of dynamic systems. For example, trained neural networks are already being used in control units. However, a great deal of effort to generate the necessary data and to select a suitable network architecture is required for even small subsystems. The modelling approach (data-based or physical) in industry is currently an either/or decision, so that the potentials of combined, i.e. hybrid models, remain unused.
In academia, methods that combine physical modelling and data-based modelling techniques became available recently. Figure 2-1 shows a taxonomy that can be used to order the approaches on a conceptual level.

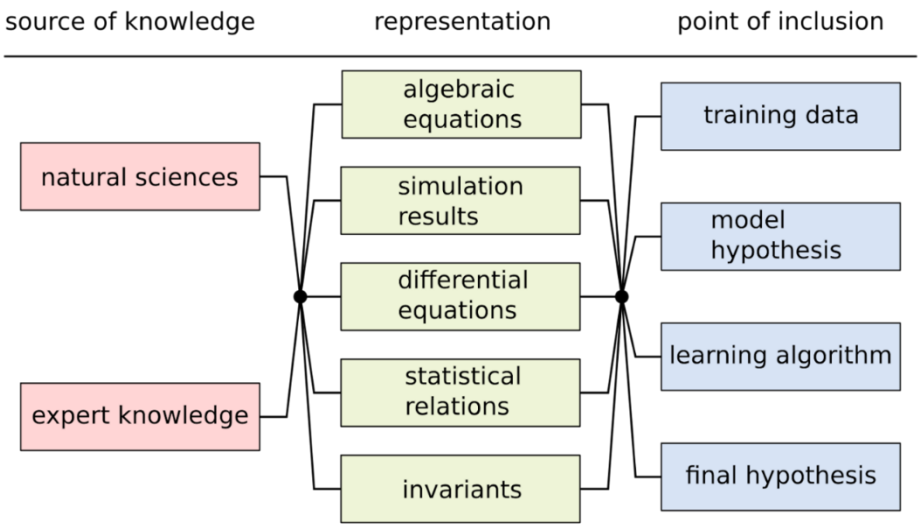


Figure 2-1 Taxonomy for the classification of hybrid models (based on [1])

The first relevant feature is the **source of knowledge** for the existing models.

Natural sciences provide knowledge that is theoretically founded. These include, for example, Newton's laws or the Navier-Stokes equations.

Expert knowledge is usually more application-specific and is available to rather small user groups. This knowledge does not necessarily have to be theoretically founded; for some applications, know-how rather than "know-why" counts. Model validity ranges are an example of this. Today, the question for which studies a simulation model can be used is mostly answered by experts based on their experience with this model, without being able to quantify the model quality.

Secondly, a distinction is made according to the type of **representation** of the models.

Algebraic equations or inequalities are formal statements about equality or inequality of different quantities. These can be, for example, kinematic relationships in mechanics, but also characteristic curves and maps.

Simulation results are obtained from the numerical evaluation of the models.

Differential equations, in contrast to algebraic equations, contain differentials or derivatives (temporal or spatial), which is particularly advantageous for the description of dynamic processes.

Statistical relations quantify uncertain quantities or describe their dispersion. State transition probabilities are an example of such relations. This category of representations is not limited to descriptions of the shape of multivariate distribution functions. It also encompasses e.g. requirements for special shapes of autocorrelation functions [3] of model deviations.

Invariants are properties which remain unchanged throughout mathematical transformations. As an example, consider the conservation of mass in hydraulic systems. In [28] such conservation laws are used to improve the training of neural networks.

Existing data can be **included** at different **points** within each of the above model representations to generate a hybrid model. The basis for this is a model hypothesis, i.e. the mathematical construct of the hybrid model. In the simplest case, this can be a neural network, but also a combination of a neural network and a model of a different structure. This model hypothesis is trained using the learning algorithm in such a way that the training/testing data is reproduced as accurately as desired, if possible.

Training data can be extended or pre-processed. For example, in [5] simulation results are used to generate learning data for an ANN-model of the longitudinal dynamics of a truck. In [23], simulation results of an injection moulding machine are used to compare two transfer learning approaches.

In **model hypotheses**, prior knowledge can be introduced, for example, via the choice of an adequate mathematical structure of the model. For example, in [6], knowledge about the quadratic dependence of drag forces on velocity acting on a model helicopter is used. Another example of forming a model hypothesis can be found in [9]. There, differential operators are represented or approximated by convolutional neural networks. In [24], the architecture of state-space neural networks is designed based on model knowledge.

The **learning algorithm** consists of the cost function and a matching optimisation algorithm. Models can be incorporated into the cost function as well as used for the optimisation itself. In [2] it is demonstrated how models, represented as PDE, can be integrated into the cost function to regularize the parameters of a neural network. In [25], a regularisation term based on Lyapunov stability is added to the loss function.

The **final hypothesis** is the final result of machine learning: a model. The predictions of the model can be compared to prior knowledge, for example, in order to introduce a plausibility check. If, for example, a model of a rocket reaches a speed above the speed of light, it can be concluded that the model yields unreliable predictions.

In the following, the approaches followed in UPSIM are classified, shortly presented and references for further reading are given.

2.1 Physics informed neural networks

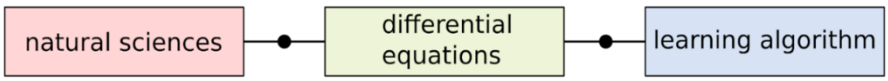


Figure 2-2 Classification of PINNs

Physics informed neural networks (PINNs) combine prior knowledge in the form of differential equations in the learning algorithm with measurement data to learn a function that best solves the differential equation while taking the measurement data into account [2].

Accordingly, the use of PINNs requires measurement data in addition to the model of the system. The goal is to train a feed-forward neural network (FFNN) that can represent the measurement data well. The model is inserted into the learning algorithm to reduce the solution space of the parameters of the FFNN.

For this purpose, the residuals of the differential equation are considered, which are generated when the network output and its corresponding derivatives, which can be efficiently computed via automatic differentiation, are inserted.

A PINN time-continuously models the solution to a differential equation as a FFNN. Assume that you have an approximate white-box system model in the shape of an implicit differential equation $F(\dot{x}, x) = 0$ which you want to improve. Given a set of training parameters Θ, a PINN models not the right-hand side $\dot{x}(x, t, \Theta)$ but the solution $x(t, \Theta)$ to a differential equation directly as a function of time$t$ as a neural network NN by

$$x(t, \Theta) = NN(t, \Theta)$$

Prior system knowledge is included into NN during the training phase: The usual data error terms in loss function $l(\Theta)$ are enriched by a physics-based regularization. Evaluations of e.g.

$F\left(\dot{NN}(t,\Theta),NN(t,\Theta)\right)^2$ at a number of arbitrarily scattered points in time t are added to the data error terms.

Thus, the physics informed loss function includes not only a measure of how well the FFNN can reproduce the measured data, but also an additional term that expresses how well the FFNN solves the differential equation. The additional term makes the system dynamics, which are only implicit in the measured data, explicitly available to the learning algorithm, resulting in a reduction of the required measured data. In [1], this approach is used to investigate its applicability in power systems by means of a simple example. In the conclusion, it is particularly pointed out that for more complex systems, despite considering models in training, large amounts of data are needed for successful training. However, a benchmark of PINN comparing it to conventional methods based on the example is left out.

PINNs learn the solution of a differential equation as a mathematical function of time. Consequently, the computation of the solution at a point in time corresponds to an evaluation of a neural network, which promises extremely performant models.

On the other hand, PINNs can only be used in application scenarios where a sufficient amount of measurement data is available. However, a measure for the amount of required data does not exist. Also, an application of the approaches published so far from this method field is only possible if the application scenario does not require variable initial values, since the solution of the differential equation is learned. Moreover, the integration of a model in the form of residuals is only possible if the model can be evaluated for arbitrary inputs and at arbitrary time points. Moreover, an application to discontinuous systems does not seem very promising since the model hypothesis itself is differentiable and thus can only poorly approximate discontinuous behaviour.

2.2 NeuralODEs

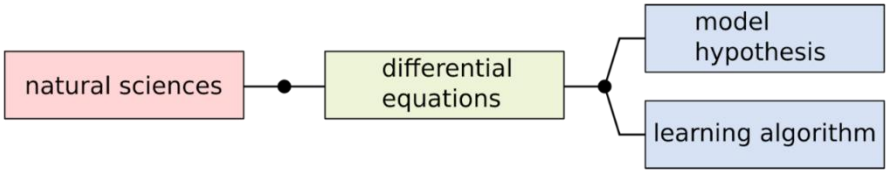


Figure 2-3 Classification of Neural ODEs

Computing the output of the layers in a deep neural network can also be interpreted, in the case of residual neural networks or normalizing flows, as the application of a solution procedure for ordinary differential equations to a nonlinear function [10], [11], [12], [15], [26]. For infinitely many layers, summands added per layer become infinitesimal. This is analogous to infinitesimal changes to system states as considered in differential equations [11]. At the same time, a layer of an FFNN can be interpreted as a nonlinear function depending on a set of parameters, which transforms an input vector into an output vector (potentially of different dimension) [12]. These ideas establish a bridge between deep FFNN and solutions to differential equations (see e.g. Figure 3-2). Consequently, methods of established techniques from one area can be used in the other. Moreover, model hypotheses can be constructed as a combination of conventional FFNN layers with ordinary differential equations. In this way, for example, previously constant parameters can be replaced by nonlinear mathematical expressions that can in principle depend on arbitrary other quantities. Considering a network as a differential equation in turn allows for the efficient computation of the sensitivities of the network with respect to its weights by means of the adjoint ordinary differential equation [13], a well-known method from the field of differential equations. Meanwhile, more advanced approaches attempt to learn systems which also contain discrete variables [20], [27].

NeuralODEs are a promising way to combine data-driven and physical model parts in one model and to use techniques from both fields, differential equations and Machine Learning, to solve these models efficiently.

However, to the authors' knowledge, no examples of industrial use of NeuralODEs exist yet. This is even more true for the approaches to learn models with continuous and discrete variables. Consequently, an assessment of the performance of the NeuralODEs approach in industrial settings

is lacking. Furthermore, no best practices exist for hyperparameter selection for the layers of artificial neural networks (ANNs). Also, experience regarding the model hypothesis does not exist. For example, it is unclear which parts of the system are better modelled as data-based and which should be included in the hybrid model as a physics-based equation to generate a computationally efficient hybrid model. In addition, to use the adjoint differential equation for the computation of sensitivities with respect to the system parameters, the differential equation must be continuous. Depending on the number of parameters, these require a representation of the model in a form that can be incorporated into corresponding frameworks [14]. Moreover, models that combine conventional networks and ordinary differential equations in the sense of neural ODEs are not per se faster to solve than conventional models. Again, additional methods are necessary to learn the system dynamics, which are not included in the existing models, in such a way that the differential equation solver needs as few evaluations of the proper model as possible [15].

2.3 U-Mesh

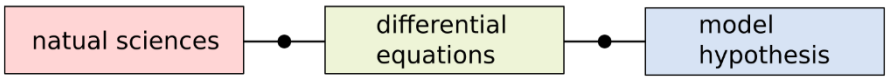


Figure 2-4 Classification of U-mesh

Mendizabal et al. [29] propose a framework by the name of 'U-mesh' that is based on the U-net architecture (Figure 2-5). The U-mesh framework is able to perform complex and complete volume deformation calculations of arbitrary shapes and extremely fast and accurate simulations are reported. The FFNN can learn a desired biomechanical model based on finite element analysis generated input data and predict deformations at haptic feedback rates with very good accuracy. U-mesh has an architecture that is similar to an auto-encoder. Auto-encoders typically transform the input space into a low-dimensional representation with an encoding path and expand it back to its original size through a decoding path. Constraining the latent feature vector space to be small ensures that the network will learn salient features from the high dimensional input data.

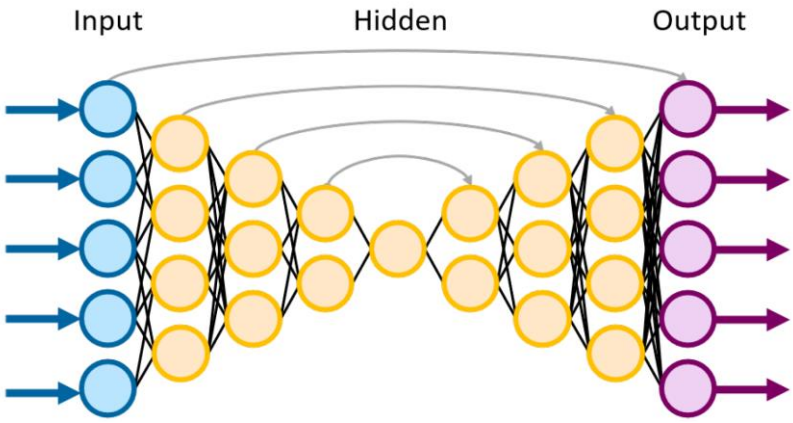


Figure 2-5 Simplified schematic representation of the U-mesh architecture

U-mesh uses a tensor of input constraints, consisting of traction forces on the surface boundary as input data. The domain ω is sampled using a 3-dimensional grid. Thus, the input data dimension is 3 times the grid size, i.e. a traction force for each point in the grid. The network generates an equally sized tensor of volume displacements as output. This architecture makes use of four types of operation layers: convolutional layers, pooling layers and up-sampling layers as depicted in Figure 2-6.

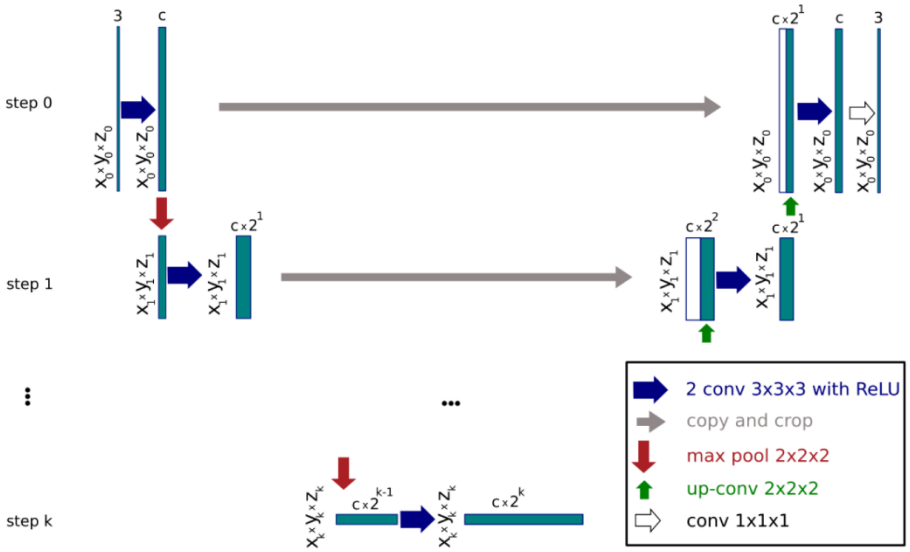


Figure 2-6 General U-mesh network architecture for an object with a resolution of $x \times y \times z$ nodes, c channels in the first layer and k steps

3 Hybrid modelling within the UPSIM use cases

3.1 Automotive Use Case

3.1.1 Brake System

From the first days of vehicle development, braking system played an essential role in vehicle design. While being pure mechanic in the past time, validation of such systems was straight-forward also due to less safety regulations: Every mechanical component must be designed so that the applied mechanical load and stress during operation can be handled over the lifetime of the system. Over time, the engines became more powerful and electrified, vehicles became more light-weight, top speed raised, and safety needs increased – as a consequence the braking systems needed to evolve, too. In consequence, functions like ABS, ESP and ADAS systems were integrated. Today the braking system is a complex mechatronic system, with electric and hydraulic subsystems, and "braking" itself a highly complex task. On the other hand, there is a tremendous trend towards Software-in-the-Loop (SiL) testing respectively validation via simulation. Thus, highly accurate multi-domain models of these systems are required. An example of a highly sophisticated braking system is the Integrated Power Brake (IBP) from Bosch that is considered in this use case. In order to investigate hybrid modelling for a brake system in a first step, the electric subsystem is considered (Figure 3-1). This subsystem includes the EC-motor of the linear actuator that applies the brake torque for this brake-by-wire system and its control.

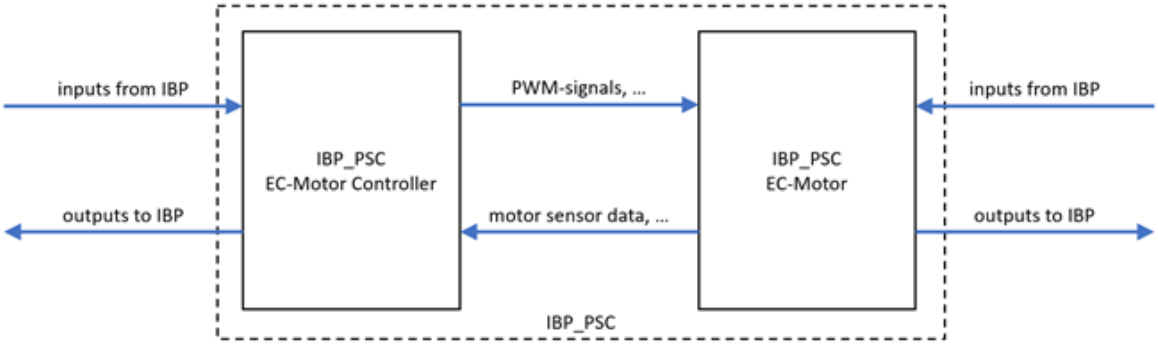


Figure 3-1 Topology of the subsystem PSC of the IBP of this use case

In particular, a model of the EC-Motor will be tested with real controller software in a SiL-Setup. Because the physical EC-motor model does not behave exactly like its real counterpart, in certain situations the safety features of the controller hardware detect an error with the EC-motor model limiting the number of software functions that can be tested in simulation. Hence, improving the model accuracy leads to more virtual testing capabilities. This improvement will be achieved using a hybrid modeling technique called NeuralFMU based on the concept of NeuralODEs.

NeuralODEs and NeuralFMUs
The field of research that deals with machine learning of dynamical physical systems is always confronted with the mathematical challenge of learning the state space trajectory of the considered system over time. Thereby, learning is performed with the help of system observations, i.e., measurements. The trained model shall reflect the real systems behavior, and thus mathematically learning means performing an optimization that minimizes the gap between data and model output. A breakthrough in learning dynamical systems was the idea of NeuralODEs (Section 2.2) that allows to combine physical modelling with ANNs.

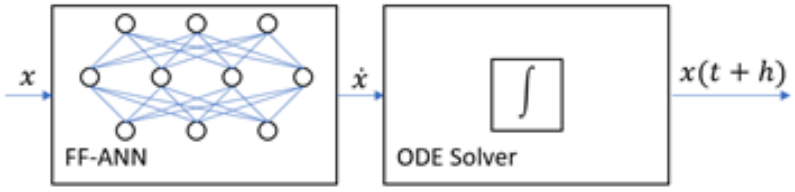


Figure 3-2 Topology of a NeuralODE

From a topologic view, a NeuralODE is the structural combination of an ANN and an ODE-solver as depicted in Figure 3-2. The main contribution of the original NeuralODE paper [11] was not only the idea of the topologic separation, but more a technical solution on how to train these structures. The challenge is that backpropagation of the gradient with respect to the ANN-parameters is needed, but when a numerical ODE solver is placed between the ANN and the problem solution, the gradient must be propagated through the solver itself. However, methods for sensitivity analysis combined with automatic differentiation offer an efficient solution.

Hybrid modeling with NeuralODEs is possible by extending the topology by physical equations and still to compute the required gradient via sensitivity analysis. One point of criticism is, that this approach requires white-box modelling, i.e. having the equations symbolically at hand. In real-world applications this is typically not the case since modelling is performed in commercial modelling tools that do not allow for export of the underlying equations.

To make the concept of NeuralODEs usable in such industrial applications, the concept of NeuralFMUs was introduced in [36]. Even if most modeling tools do not support the export of symbolic model equations, they support model export as a functional mock-up unit (FMU). FMI comes with two different flavors, model exchange (ME) and Co-Simulation (CS). In order to combine FMUs and ANNs resulting in a NeuralFMU according to Figure 3-3, a FMU capable of computing a state vector derivative and optional system outputs given a state vector, optional system inputs and the current point in time are required. Hence, within this use case ME FMUs are used.

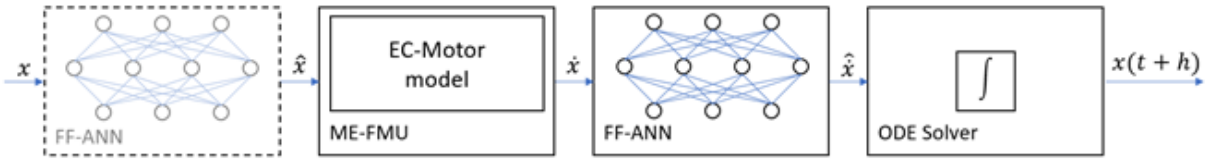


Figure 3-3 Topology of the considered NeuralFMU

The left ANN (semi-transparent) in Figure 3-3 is currently neglected in this use case, but in general it allows to transform states to compensate e.g., sensor drift or to make parameters state dependent. For the generation of such NeuralFMUs two Julia packages were developed. The packages are open-source and available under https://github.com/ThummeTo/FMI.jl (*FMI.jl*) and https://github.com/ThummeTo/FMIFlux.jl (*FMIFlux.jl*). Though the implementation was done with a focus on the requirements of the use case at hand, these packages allow to generate NeuralFMUs for a variety of use cases from different domains like e.g., the mechanic, electrical, hydraulic or pneumatic domain.

Training of the NeuralFMU

In contrast to conventional ANN training, initialization matters for NeuralFMUs (and also NeuralODEs). In particular, bad initialization may lead to slow simulation or even an unstable simulation and thus to slow training or even to abort of the training. This is because a randomly initialized ANN may generate an output that corresponds to a very stiff ODE.

Therefore, the training of the presented hybrid model is done in two steps: A collocation-based initialization and the final training for fine-tuning. The central idea of collocation is, that solving an ODE requires more than one function evaluation per timestep. Thus, instead of solving the NeuralFMU with an ODE solver, the ANN is isolated and trained without the other components. Therefore, on the one hand the training data is propagated through the FMU, to get the FMU state

derivatives corresponding to the measured trajectory. These serve as the input during training of the ANN. In order to obtain labels for the FMU state derivatives a collocation function is fitted to the input data, where the function is chosen such that, the derivative can be analytically computed. Evaluating this derivative at the timepoints of the measurement gives estimated state derivatives along the measured trajectory and thus the required labels for the training data.

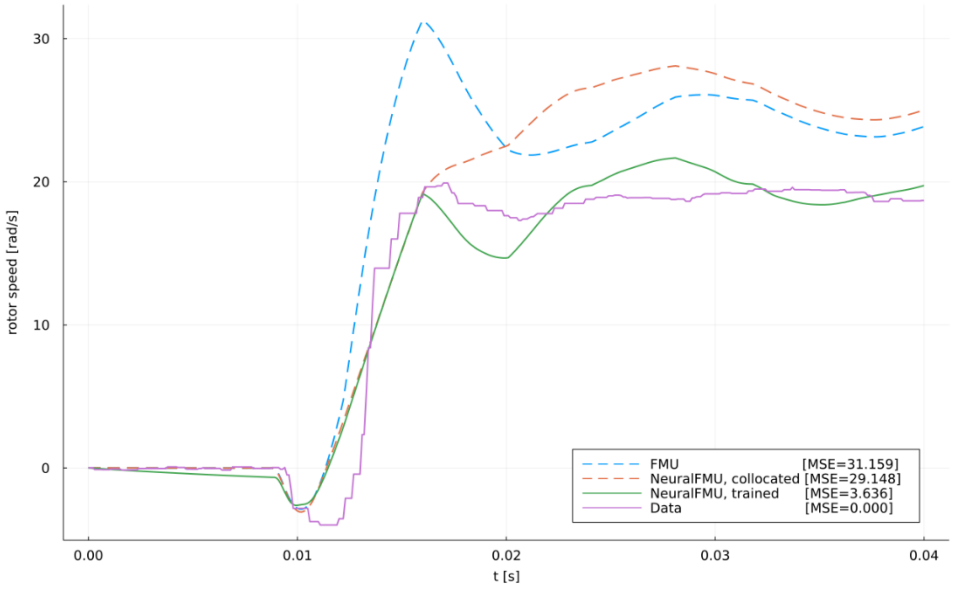


Figure 3-4 NeuralFMU after collocation (orange) and training (green)

For the final training, the entire hybrid model is simulated for a given time span (forward pass) and after simulation, the network weights are adapted to better fit the state trajectory from data (back-propagation), based on chained parameter-sensitivities through ANN, FMU and ODE solver. The results are shown in Figure 3-4: Even before full training, the simulation accuracy of the collocated NeuralFMU (orange) is already similar compared to the original FMU simulation model (blue). In particular it is visible, that collocation leads to an offset in the simulation results. This is due to the fact that collocation is performed only on state derivative level and thus deviations on state level have no influence.

Starting with the collocation result, after only 160 full training steps (60s per step), the trained NeuralFMU (green) clearly out-performs the first-principle simulation model in terms of total error in comparison with the real system trajectory (purple).

3.1.2 Vehicle Dynamics

The suspension of a vehicle does not only aim to maximize the comfort of passengers but critically influences the road-holding and cornering abilities. Thus, its design is safety-critical and defines the handling of the vehicle.

The design goals, i.e., maximizing the comfort by minimizing the (vertical) chassis acceleration and maximizing the road-holding ability by minimizing the tire deflection are inherently conflicting. This is because an increased damping eliminates lower frequency-accelerations of the chassis, preventing the wheel to follow the road in return. Very low damping on the other hand allows direct transmission of accelerations to the chassis with the risk of resonance effects. This is why the suspension design always implements a trade-off between comfort and roadholding, depending on the desired handling characteristics [30, 31]. Active and semi-active suspensions aim to unlock a degree of freedom, enabling a dynamic adaption of the damping characteristics. This can be achieved by the deployment of actuated dampers with variable damping characteristics [32].

In any case, defining the trade-off between the two properties is an important development effort and requires both, a requirements analysis for the considered vehicle type (e.g. sports car vs. limousine) and a system analysis. The latter can be realized by extensive measurement of single components, small assemblies and the whole vehicle. However, in the context of the general

ambitions to reduce the prototyping and experimentation expenses, it is inevitable to implement credible models of components, the suspension and the whole vehicle respectively. This becomes even more important, if the suspension contains actuated components and the design of a control-loop is required.

In the context of autonomous driving, high fidelity models of the vehicle dynamics are core assets of the development process. They are used within model-based filter-algorithms for state estimation, for optimization-based control approaches and for the offline training of artificial intelligence (AI)-based control designs, such as Reinforcement-Learning (RL). Such controllers are not likely to be applicable, if the underlying model is not able to represent the reality sufficiently well. Furthermore, the controllers of autonomous systems are safety-critical components, meaning the credibility of the underlying models is of high interest to the developers and the users.

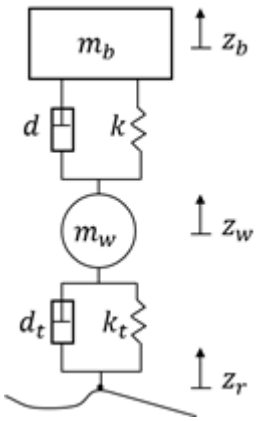


Figure 3-5 Linear quarter vehicle model

The quarter vehicle model

A widely used model to represent the vertical dynamics of a vehicle is the quarter vehicle model (QVM), which only considers one wheel and one quarter of the chassis [32, 33, 34]. As depicted in Figure 3-5, the wheel and the chassis are approximated by two masses that are linked by the suspension, consisting of a damper and a spring. The dynamics of the tire are incorporated as another spring-damper pair, which connects the wheel to the ground. The differential excitation imposed by the (dynamic) height of the ground z_r is considered as the input, while the accelerations of chassis and wheel respectively are possible outputs. Furthermore, the dynamic wheel load is of major interest as the maximal applicable lateral and longitudinal tire forces are determined by the wheel load. The model state space consists of five quantities, being the height and vertical velocities of body and wheel respectively and the height of the ground. Note that all states are aligned with the z-axis and movements in the horizontal directions are not considered.

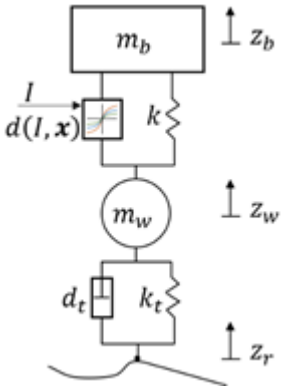


Figure 3-6 Non-linear QVM with semi-active suspension

Using the linear QVM as basic structure, further refinements can be added. Real dampers often exhibit non-linear characteristics, making the coefficient d dependent on the current differential displacement [32, 34], which leads to the non-linear differential equations:

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = m_b^{-1}[k(x_3 - x_1) + d(I, \boldsymbol{x})(x_4 - x_2)]$$
$$\dot{x}_3 = x_4$$
$$\dot{x}_4 = m_w^{-1}[k_t(x_5 - x_3) + d_t(u - x_4) - k(x_3 - x_1) - d(I, \boldsymbol{x})(x_4 - x_2)]$$
$$\dot{x}_5 = u$$

In our use case, a semi-active suspension is used, which contains a non-linear damper, actuated by a current I (see Figure 3-6). In general, the non-linear damper characteristics are often available through the datasheet of the manufacturer and can be integrated by approximate functions or look-up tables.

However, many effects are hard or impossible to incorporate into the model. Hard to consider are e.g., non-linearities introduced by the transmission of the wheel excitation to the spring-damper excitation, resulting from constructions that do not align the suspension directly with the wheels z-axis. The inference of the transmission ratios is a source of ambiguity when detailed blueprints of the suspension are not accessible. Another source of model-errors is the inexact measurement of the mass of the wheel, which is not trivial, since the mass of the suspension, i.e., of the spring, damper, wishbones, mountings etc. must be partly ascribed to the wheels mass. The coefficients of the tire model are dependent on the air pressure inside the tire and might follow a nonlinear characteristic. Furthermore, the model neglects effects of the rubber mounts of the articulated parts of the suspension and any kind of friction. Finally, the coupling of the four wheels through the chassis and influences by other mounted masses, e.g., the engine, cannot be considered. At this point, it becomes apparent that increasing the precision of the QVM by refining the model equations and incorporating the reviewed influences soon becomes infeasible.

Restoring the Concept to Realism Continuity

In order to provide a credible model of vehicle vertical dynamics, one soon reaches the limits of analytical inference. Therefore, we aim to use the QVM to the degree it provides feasible predictions and augment it with data-driven components. The analytical part of the obtained hybrid model represents the fundamental behavior and incorporates available knowledge of high reliability, as for example the well-known spring and damper characteristics. However, effects which are hard to isolate, like the influences of rubber mounts for example, are not explicitly modelled. The data-driven parts of the model are then entrusted to close the gap between the model and real measurement data by learning the unmodelled phenomena. By augmenting the reviewed differential equations with black-box components, we aim to implement a set of neural ODEs (see Section 2.2) to obtain a well fitted, hybrid QVM. Choosing this approach has the advantage that the outputs of the neural networks can be interpreted as physical quantities such as forces or coefficients. This yields the opportunity to directly enforce physical laws and thus enhance the scalability of the model to unseen inputs [35]. Before applying the hybrid approach to learn a model from real measurements, we used simulated data to validate the methodology. This is necessary, since many effects influence the measurements, making it hard to interpret *what is learned* by the black-box components. For a proof of concept, we deployed the linear and non-linear QVM to generate reference data by simulating the system for a defined input. The implementation in Julia is straightforward, using the *OrdinaryDiffEq.jl* package which allows defining an ODE problem and solve it over time. The generated trajectories of the state and the system-output serve as training data for the experiments. In a first step, we conducted a gradient-based parameter-fitting for the model parameters, starting from an artificially disturbed parameter-set and using the methods from *DiffEqFlux.jl* to conduct the training. While convergence could be achieved, it showed that the

gradient-based fitting finds the optimum much slower than a basic evolutionary optimization and is naturally prone to local optima. We therefore propose to use non-gradient-based optimization methods for parameter-identification, before optimizing additional parameters of AI-components.

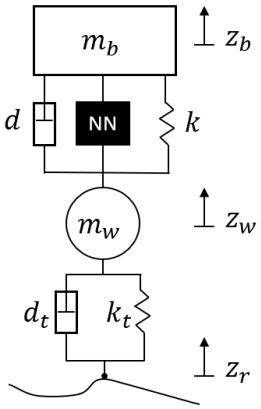


Figure 3-7 Hybrid QVM

Next, we augmented the linear QVM with a single single-input-single-output neural network (see Figure 3-7), which was trained to learn a non-linear friction force (see Figure 3-8). The friction-term has been incorporated in the generated reference data but was not modelled in the hybrid QVM. The obtained ODEs of the hybrid QVM now contain the output of the neural network, which is treated as a force F_{NN} and depends only on the relative velocity of the two masses:

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = m_b^{-1}[k(x_3 - x_1) + d(x_4 - x_2) + F_{NN}(x_4 - x_2)]$$
$$\dot{x}_3 = x_4$$
$$\dot{x}_4 = m_w^{-1}[k_t(x_5 - x_3) + d_t(u - x_4) - k(x_3 - x_1) - d(x_4 - x_2) - F_{NN}(x_4 - x_2)]$$
$$\dot{x}_5 = u$$

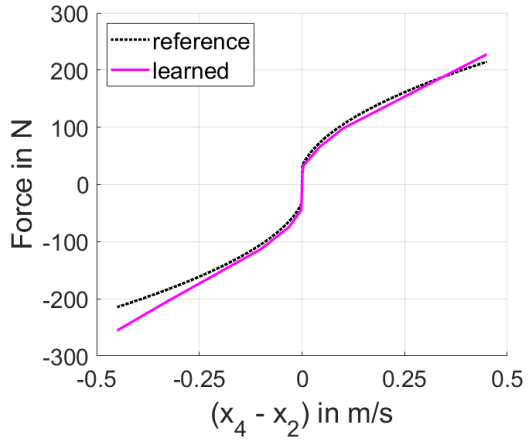


Figure 3-8 Learned and reference friction curve

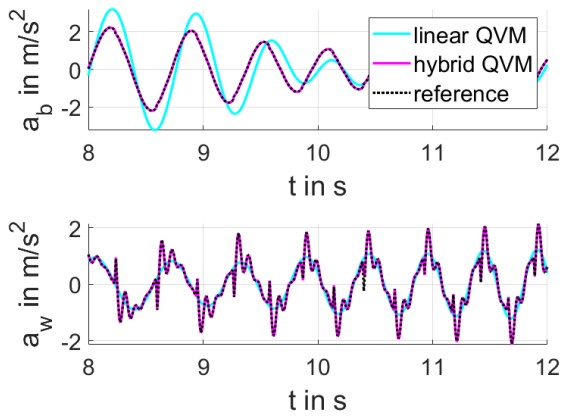


Figure 3-9 Hybrid QVM in comparison with reference data

We found, that through the training, the underlying friction force is well approximated by the neural network (see Figure 3-8). Note that the friction force is not explicitly learned, but the training solely relies on minimizing the deviation of the model-output trajectories to the reference data. A

detail from these trajectories for the body and wheel accelerations is depicted in Figure 3-9. It is evident, that the hybrid model (after training) fits the reference well, while the linear model (i.e. the model before training) naturally shows a significant deviation.

To sum up, up to this point we showed that neural networks can be used to learn underlying physical effects using the solution of the differential equations and refer them to training data. However, it is yet to show that this approach scales to real measurements and that models with more complex and multiple black-box components yield comparable results.

Providing measurement data

For a successful model fitting, it is crucial to provide representative data which contains the considered model outputs as a ground truth. The common way to characterize the vertical dynamics of a vehicle is to simulate the road excitations with hydraulic posts underneath all four wheels and measure the deflections and accelerations of the wheels and the chassis. The excitations can either simulate a real road or evolve sinusoidally with increasing frequency (sinus-sweep) in order to characterize the vertical dynamics in the frequency domain. In addition to the deflections and accelerations of the vehicle, the post (vertical) positions, velocities and accelerations as well as the individual dynamic wheel-loads are available and can be used for training and validation purposes. In extensive experiments, we have collected a large dataset with our experimental platform AI for Mobility (AFM), a modified series-production vehicle. After preprocessing, this data will be used for the upcoming modelling of the vehicle vertical dynamics using a hybrid QVM.

3.1.3 Driver Monitoring

The Driver Monitoring use case aims at monitoring the vital parameters of a driver by means of an AI-enhanced radar system. The overall system consists of the radar sensors, a microcontroller platform, AI models and other software components for tasks like data transfer and pre-processing. The AI algorithm will be trained on data gathered with real world setups as well as simulations. These simulations include a model of the sensor and of the interior of a car with a driver. The use of simulated data enables system verification and testing in diverse scenarios as well as testing over long-time spans. Furthermore, data generated during the simulation runs can be used to increase the robustness of the AI models, in particular by better generalization over more variants of known tasks and sensing environments. To improve the quality and robustness of the simulated data, information collected with real world setups can be exploited to model and predict environmental effects such as noise. In a real-world application, the sensor can also lead to measurement failures. Such failures can be caused by chip failure, deterioration due to external factors and damages, and wear and tear due to long usage. Therefore, for a more realistic modeling of the sensor in a given context, the influence of equations that take into account the main reasons for failure of operation must be considered together to the noise statistics extracted from the real data.

For the simulation, all the 3D modeling and scenery generation will be done using Blender [37], which is a free, cross-platform modeling, rendering and animation software. The electromagnetic simulations of the radar device will be performed on Ansys HFSS [38] (high-frequency electromagnetic simulation environment) using Ansys SBR+ [39] (Shooting and Bouncing Rays).

The simulation scene for this use case mainly consists of three parts: **radar sensor node model, driving model**, and **car model.** The scene components and their sub-parts are described in the following subsections. A general block diagram of the various modules for the driver monitoring use case is shown in Figure 3-10.

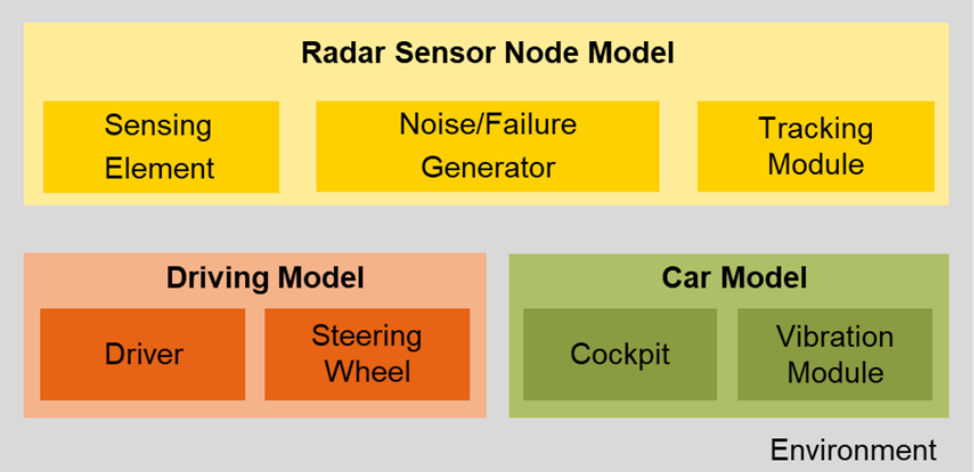


Figure 3-10 For every simulation, the scene is composed from 3 main models: the sensor node model, the driving model and the car model. The environment accounts for the scene and model configuration.

The scenario setting influences all three main model nodes. The selected scenarios are described after the scene element's description.

Radar sensor node model

The radar sensor node model includes all steps from data collection, scene-dependent sensor configurations, signal processing with real data augmentation and preprocessing, to motion and vital parameter tracking.

Raw information gathered from radar devices is not easily interpretable and often requires suitable preprocessing to be exploitable. In addition, the radar signal contains scattered information from the multiple constituent elements of the scene. Consequently, using mathematical models for the generation of environmental noise to be included in the simulation in relation to the particular scenario can be expensive and difficult to interpret during data processing. The use of equations that model sensor failures during use can also enable the observation of effects on measurements over long spans of time or chip damage, which are difficult to measure in the real world.

Through a data-driven approach, however, it is possible to transfer information from the real world to the simulated data, depending on the conditions in which the sensor is employed. This consequently allows to decrease the gap between simulation and the real world without directly impacting the complexity of the models.

The radar sensor node includes multiple sub-components, is the main component of the system and the component that is modelled using a hybrid approach. The entire block diagram of the radar sensor node is shown in Figure 3-11. The main sub-components are:

1. The **sensing element**: with the purpose of emulating the physical behavior of the radar sensor
2. The **noise/failure generator** to allow for more realistic simulations through the hybrid approach
3. The **tracking module** for the estimation of movements and vital parameters of the driver.

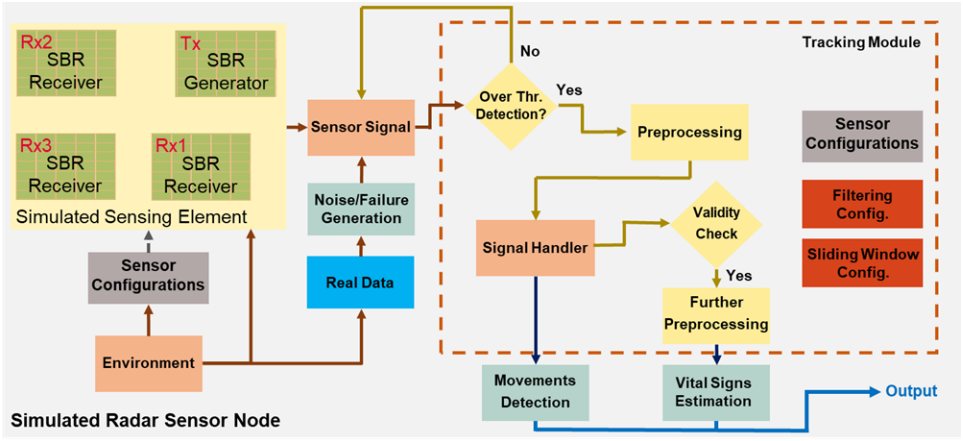


Figure 3-11 Representative block diagram of the simulated radar model. The blocks in orange represent the state of the device, i.e. its configuration in the scene (Environment), the generated signal (Sensor Signal) and the management of the signal once preprocessed

Sensing Element

Among the various radar technologies, the FMCW (frequency modulated continuous wave) is particularly suitable for the detection of small displacements, thanks to its ability of accurately estimating the position and velocity of targets within its range. This is possible thanks to the continuous generation of frequency modulated signals, which contrast the ambiguity of detecting the position of targets compared to other modulation technologies. In our simulation, we will use as sensing element a black-box model of the Infineon BGT60TR13C - XENSIV™ 60GHz radar [40]. This model is generated in Ansys HFSS [38] through Ansys SBR+ [39], which is an asymptotic simulator for modeling EM interaction in electrically large environments. The measurements of the sensing element are directly influenced by the environment, i.e. the scene in which it is used and the specific measurement requirements (vital signs or driver movements).

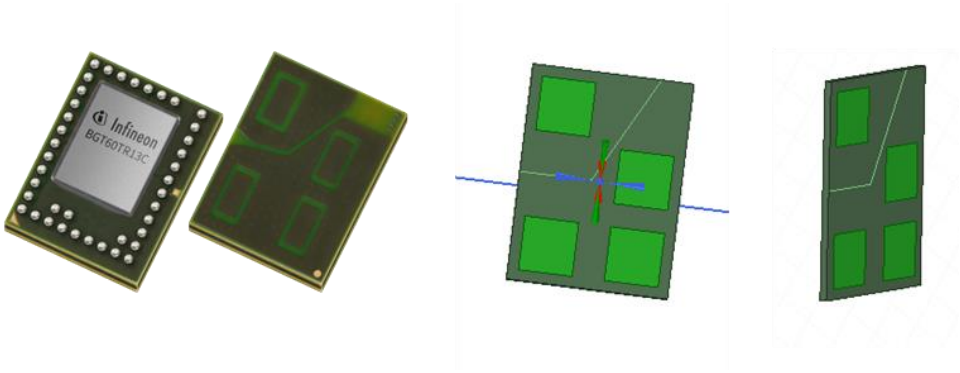


Figure 3-12 Real Infineon BGT60TR13C (left) vs Simulated Black Box model in Ansys HFSS SBR+ (right). The separation line is between the transmission channel (Tx) and the three reception channels (Rx).

Noise/Failure Generator

In this module, information collected in real experiments is exploited, along with equations that account for device failures to increase the robustness and degree of realism of the data generated by the simulation. Actual data is collected in the context of in-car driving, in a manner consistent with the constraints placed on the defined scenarios to be simulated. For each real driving context, a statistical analysis is performed on the raw sensor data in order to obtain a noise distribution over the entire measurement range. To extend the amount of available real-world data even further open-source data sets are also considered and analyzed to improve input for the noise modeling.

Depending on the type of scenario chosen, noise statistics generated from the real data are added to the simulated data upstream of the tracking module, so that the information given by the noise is also pre-processed and analyzed. The noise statistics for each simulation frame is estimated

using a regression neural network (AI model), trained on the real data statistics. Depending on the specific scenarios, failure information can be added to the simulated data by failure equations over long- or short-time spans. The training of the regression neural network can also be influenced by the failure equations used, in the form of training regularization terms. A regularization term can be added to the cost function to be optimized to avoid overfitting the model on corrupted signals. An example of a failure equation is the presence of noise such as Gaussian white, caused by wear and tear in the sensor over time. Given an ideal radar signal $S(t)$, an example of failure term $K(t)$ can be as follows:

$$S_n(t) = S(t) + K(t) = S(t) + \aleph(\mu, \sigma)$$

$$Loss = Error(y, \hat{y}) + \lambda_K \, Regularization(w)$$

Where λ_K is the regularization parameter, dependent by the employed K term.

The outcome expected from the failure equations is also exploited to inform the model of potential failures in evaluation and testing phases. This hybrid approach, which consider real data and support equations, allows theoretically to obtain a more accurate estimation of the parameters than a classical deep learning approach. A descriptive diagram of the Noise/Failure Module is shown in Figure 3-13.

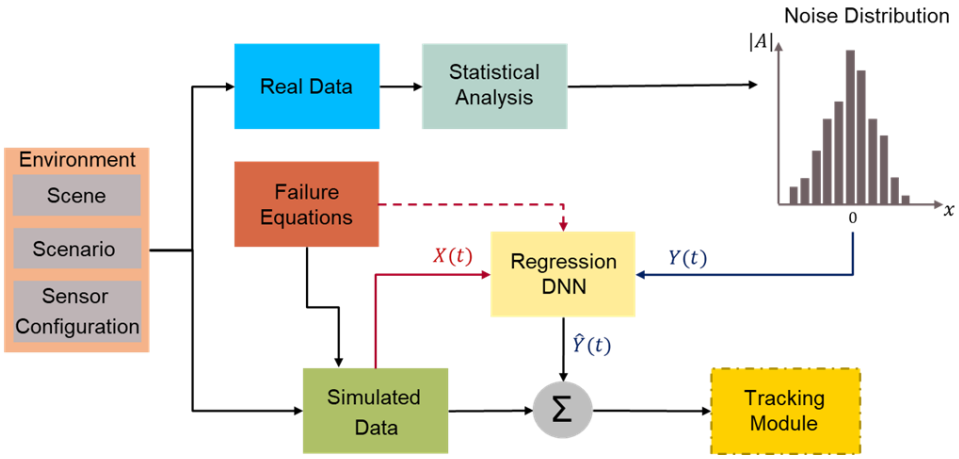


Figure 3-13 Block diagram of the Noise/Failure module

The environment defines the context information (Scene, Scenario and sensor configuration). The real data is used to calculate the noise profile. The simulated data at each frame, biased by the failure equations are used as input to the regression neural network. The regression network, which employs a regularization term for the cost function depending by the defined failure term, exploits the real noise distribution as output and predicts the noise for the new simulated data. The sum of the estimated noise and the simulated data is then given as input to the tracking module.

Tracking Module

This module is responsible for performing all preprocessing and classification steps on the sensor signal, in order to extract for each simulation frame, the movements, and potentially the vital parameters of the driver. In most radar applications, estimation of a person's vital parameters is possible exclusively by analysis of the phase information between transmitted and reflected signals. This is because the phase is highly sensitive to small displacements such as heart rate and respiratory rate, which are hardly detectable by amplitude or frequency analysis. The phase information, however, can be easily corrupted by other movements and or vibrations. In this case, vehicle vibrations and driver movements can alter the information content. A correct estimation of vital parameters can therefore be made only in the approximation of minimum car and driver movements. The tracking module has therefore the objective not only to preprocess the signal but also to estimate the movements of the driver and understand when there are the optimal

conditions to calculate the vital parameters. The tracking module will make use of deep learning models trained on real data and previously simulated together with signal processing methods to estimate movements and vital signs of the driver.

Driving model

The driving model consists of two main sub-components:

1. **The driver**: This is the model of a human being to the guide of the vehicle. For the development of the driver model, we used the mannequin and the animation of sporty driving taken from the Mixamo platform [41] as a base. These models have been adapted to the guide of a utility car and modified of proportion in order to match the average shoulder width of an adult man (Biacromial breadth - link [42]). The software used for the modification and scaling of the models is Blender [37].
2. **The Steering wheel**: i.e. steering movements in relation to driving actions. Although the steering is part of the car's cockpit, it is directly influenced by the vehicle's driver, scenarios and driving style. Consequently, it is considered as an integrative part of the driving model. In the case of our simulations, the steering wheel has been completely modeled in Blender [37].

Car model

The car model is mainly composed of two sub-components:

1. **The cockpit**: i.e., the internal elements of the car that do not particularly affect driving but can affect the signals reflected by the sensing element of the radar sensor node model. Since the configuration of the radar sensor, both in real applications and in the sensing-module is tuned to focus only on the arms and chest of the driver, modelling the driver seat is sufficient for a good approximation. The seat model has been chosen with Royalty Free License from CgTrader [43].
2. **The vibration module**: simulates the car vibrations according to the road condition. Although in real conditions, in most scenarios, most of the vibrations are attenuated by shock absorbers, the residual movements still cause noise added to the information sensed by the radar module. Such added noise can be especially critical for the computation of phase information, which is necessary for the extraction of vital parameters.

However, an implementation in the simulation environment of this module is for now planned as an optional goal, since it depends on the state of other sub use cases of the automotive use case. The idea would be for example to connect this module with the brake system that is based partly on real world data. By simulating at the same time, the brake system and driver monitoring, the vibrations generated by braking can be transmitted to the cockpit and driver model. The described approach might not be the most efficient way to implement the simulation of vibration, but it shows the possibilities for a cooperation. The specific details of a cooperative simulation and its implementation details such as hybrid modelling will be evaluated and discussed with the corresponding partners.

The basic components of the driving model along with the driver seat that is part of the car model, are depicted in Figure 3-14.

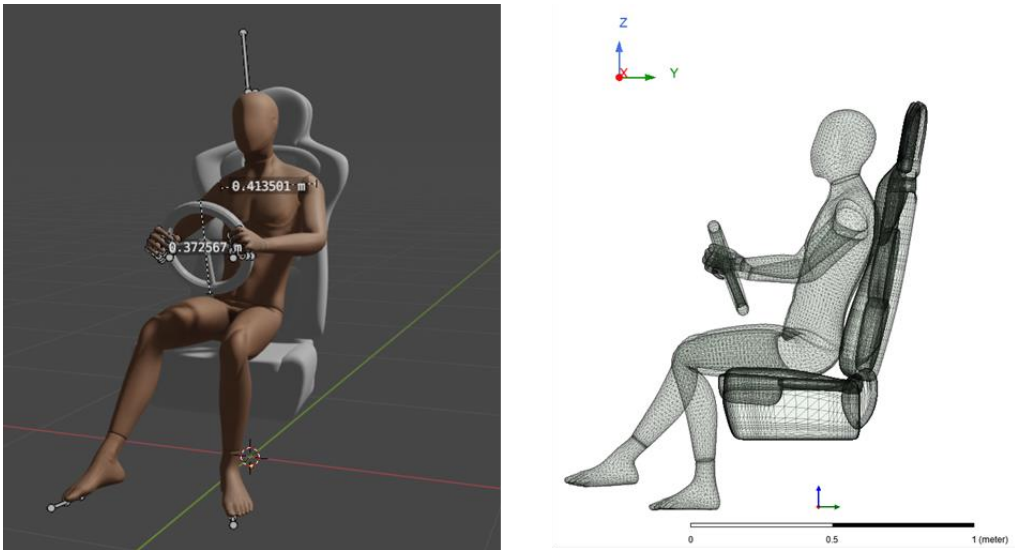


Figure 3-14 Driving Model and seat in two different projections and software. In the left image, 3D models are important in Blender software for generating the driving scenarios. The dimensions of the driver's steering and shoulders are depicted in the figure. On the right, the same models imported into Ansys HFSS for SBR+ simulation.

Next Steps

After the basic setup of the simulation models and environment is completed, IFAG will investigate more aspects that improve the quality of the simulations. For example, reality continuation concepts will be explored and implemented. Specifically, the modelling of the drifts in sensor data are hard to model without real world data and information since it has many causes such as hardware degradation or changes in the user behavior. Furthermore, many aspects of drift and its causes are still researched. Hence, a continuous update and extension of the noise generator is essential to improve the performance, robustness and safety of AI models trained with this simulation environment.

Another aspect is to introduce more randomness into the simulations. For example, under current conditions, all simulations are performed on a single driver model representing the average adult human. However, stature and other physiological characteristics of people can also influence the estimation of vital parameters and movements. Consequently, a set of various types of driving subjects should be considered in the future and for every time that the simulation starts with a driver entering the car, a subject will be randomly picked from the set of available models.

A further topic is to consider sources that impact the driver. Hence, besides vibrations that were mentioned in the car model section IFAG considers exploring the influences of gear shifting on the simulation. The gear in fact would not influence only the model of the car and the consequent conditions of vibration of the vehicle, braking and acceleration, but also the driving model. In the cars with the manual gear shift in fact, the action of change of gear, which involves the movement of the arms represents a common action in great part of the driving conditions and can make more difficult the vital signs estimation.

3.2 Medical Use Case

This use case focusses on decreasing the time to market of next generation medical devices, such as minimally invasive imaging catheters. To this end, product development should be strongly supported by virtual design and testing by means of validated high-fidelity digital twins of these devices and human tissue (e.g., arteries). Our improved understanding of the device-human tissue interaction from these interacting digital models will lead to quantified loading conditions within the device and the surrounding tissue. Consequently, relevant failure mechanisms at device and tissue level can be predicted by the computational models resulting in improved performance, reliability, and patient safety. Figure 3-15 illustrates the technological building blocks for the healthcare use case. The left part of the figure corresponds to the development phase of the new devices, while the right part of the figure corresponds to the usage or operational phase of the devices. Hybrid modelling could impact both phases: during the development phase, the

availability of data (e.g., materials, manufacturing, earlier generations) and resulting hybrid approaches can significantly reduce the development phase. With real data available the gap between the model and reality will decrease over time during the operation phase. Within this use case, a hybrid modelling concept will be applied and further developed to support the virtual testing demo in terms of near real-time simulation capability.

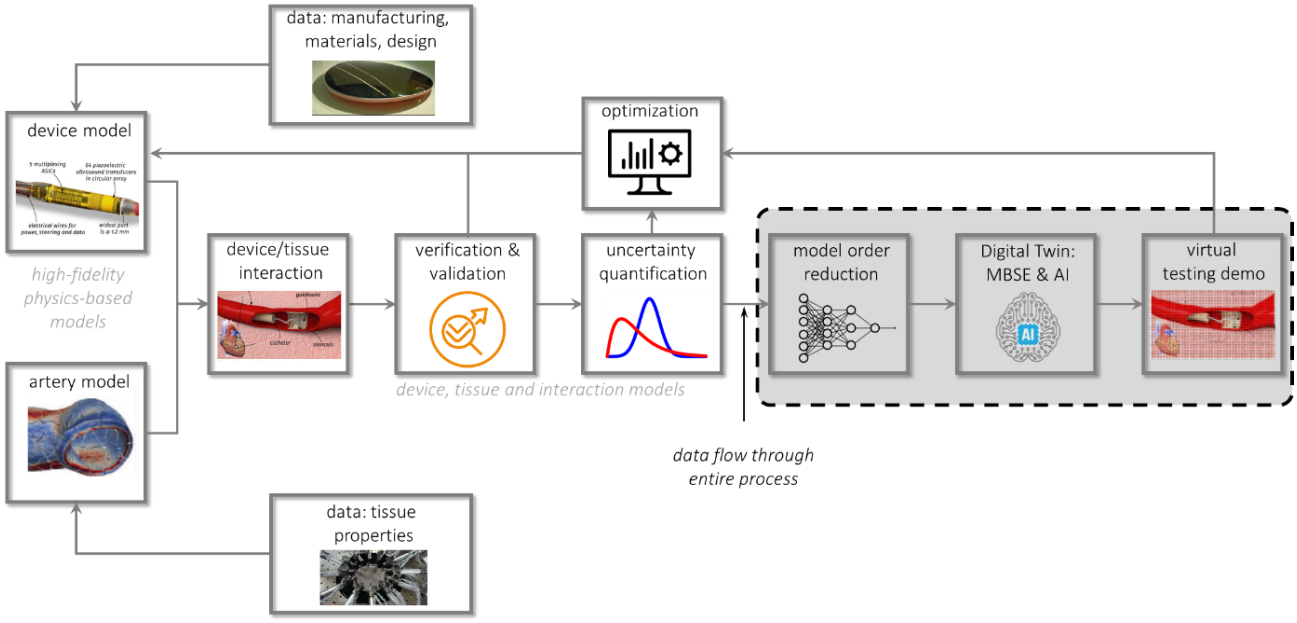


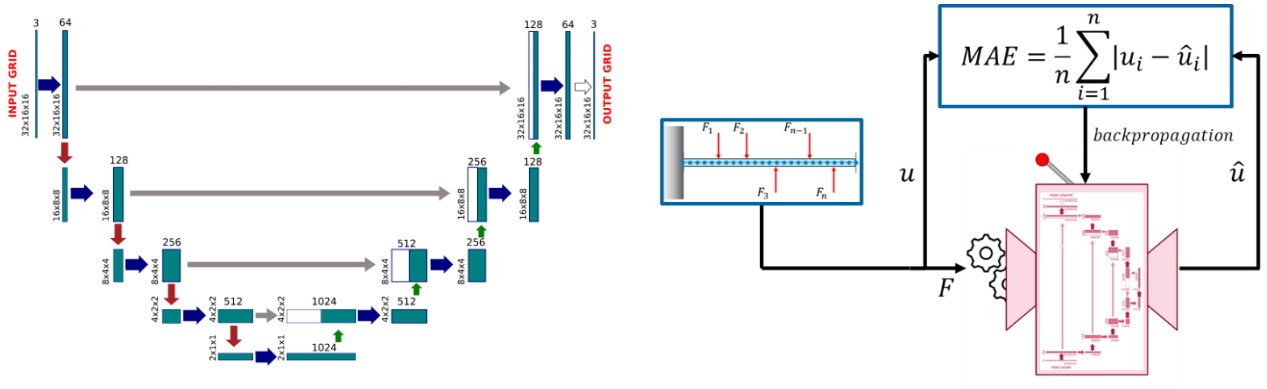
Figure 3-15 Schematic of the technological building blocks of the healthcare use case; the hybrid modelling activity is part of the virtual testing demo (grey-dashed region)

Near real-time simulations with sufficient accuracy can be achieved by reduced order modelling and hybrid modelling methods. Our use case will study both methods. Here, we will report on the selected hybrid modelling approach: physics-informed neural networks. One of the key ingredients in the healthcare use case is the ability to simulate the deformation behavior of the medical device during a medical procedure in near real-time. Results of experimentally validated high-fidelity physics-based models (left part of Figure 3-15) could be considered as ground-truth for these near real-time simulations.

In this use case, hybrid modelling, next to reduced order modelling, is considered as approach to realize real-time simulation capabilities in the virtual testing demo. To this end, physics-informed neural networks will be used to describe the deformation of a medical device.

3.2.1 Neural networks to describe the deformation of a medical device

As explained in Section 2.3, this work is based on U-Mesh, a data-driven method based on a U-Net architecture which is able to approximate a (nonlinear) relation between mechanical forces and displacements, proposed by Mendizabal et al. [29]. Figure 3-16a illustrates an example of a network architecture.



(a) (b)

Figure 3-16 (a) Example of network architecture for a simple bending beam [29]; (b) illustration of the neural network training scheme (MAE: mean absolute error)

As a proof-of-principle, a simple bending beam is considered which is loaded at multiple locations with prescribed forces. This simple test case is motivated by the fact that bending is one of the principal deformation modes of a medical device as a result of device-tissue interaction during a medical procedure. Upon realizing accurate results within this first proof-of-principle, the complexity will be increased towards a real medical device. First, the neural network is trained using a mean absolute error of the displacement field as evaluation criterion, as illustrated in Figure 3-16b. For this simple case, an analytical equation is used to generate the synthetic data to train the network as depicted in Figure 3-17. Evidently, for more realistic situations, analytical equations are not available. In these cases, the synthetic data will be generated by the high-fidelity physics models, possibly complemented with real-time sensor information.

$$y_{A,i} = \frac{-F_i}{6EI}(2l^3 - 3l^2 a_i + a_i^3)$$

$$\theta_{A,i} = \frac{F_i(l - a_i)^2}{2EI}$$

$$y(x) = \sum_{i=1}^{n} y_{A,i} + \theta_{A,i} x - \frac{W}{6EI}\langle x - a_i \rangle^3$$

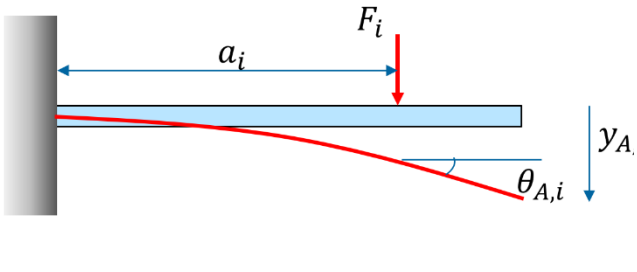
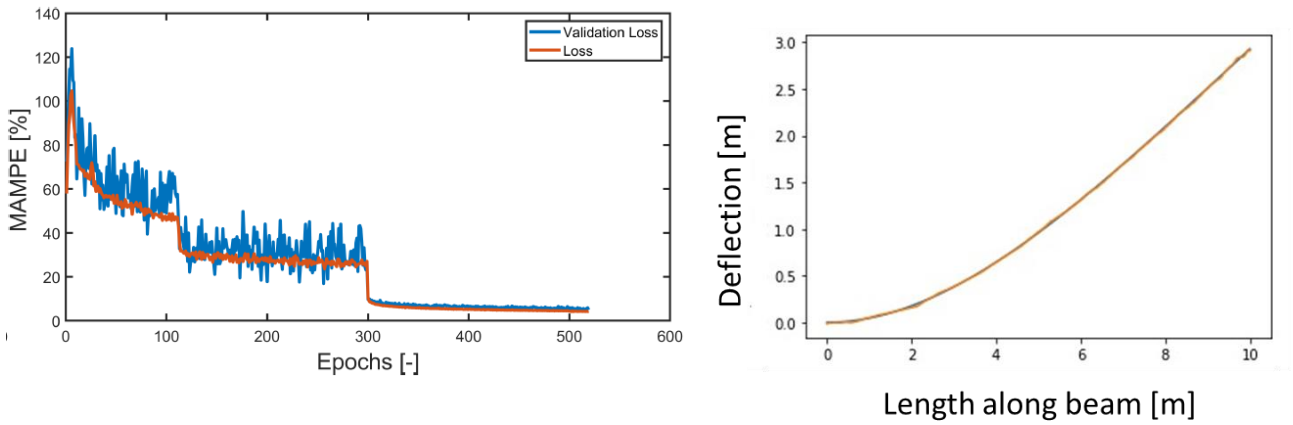


Figure 3-17 Analytical equations for the bending beam with arbitrary forces used to generate synthetic training data with the x-coordinate along the beam

Several network depths were tested in order to select the most appropriate values for the use cases in a tradeoff for the desired accuracy and computational cost.

Figure 3-18 shows the first results. The left picture illustrates the Mean Absolute Maximum Percentage Error (MAMPE), given in Figure 3-18a, as function of the training iterations. Figure 3-18b illustrates the trained (converged) result of the simple bending beam which is validated with the analytical equation.



(a) (b)

Figure 3-18 (a) Evolution of loss function during training; (b) calculated deflection along the beam from the trained network (overlay: analytical equation)

3.2.2 Physics-informed neural networks to describe the deformation of a medical device

To this end, a regularization function will be added to the MAE function, as depicted in Figure 3-1. Here, the regularization function is the classical bending beam equation. In addition, at the left bottom, an additional source of synthetic data is included: finite element simulations. Clearly, this is not required for the simple bending beam case but will become relevant for the more realistic cases.

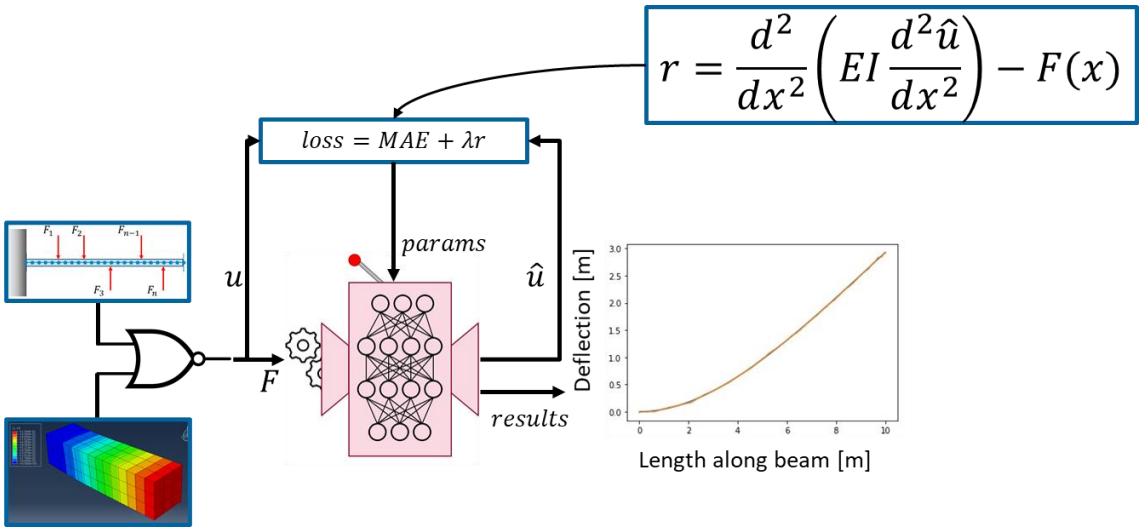


Figure 3-19 Schematic of a physics-informed neural network

This PINNs method is currently being implemented and verified.

4 Abbreviations

ACRONYM	DESCRIPTION
ANN	Artificial Neural Network
CS	Co-Simulation
FFNN	Feedforward Neural Network
FMI	Functional Mockup Interface
FMU	Functional Mockup Unit
IPB	Integrated Power Brake
MAE	Mean Absolute Error
MAMPE	Mean Absolute Maximum Percentage Error
ME	Model Exchange
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
PINN	Physically Informed Neural Network
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SiL	Software-in-the-Loop
QVM	Quarter Vehicle Model

5 Literature

[1] von Rueden, Laura, et al. "Informed Machine Learning--A Taxonomy and Survey of Integrating Knowledge into Learning Systems." arXiv preprint arXiv:1903.12394 (2019).

[2] Raissi, Maziar, Paris Perdikaris, and George E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations." Journal of Computational Physics 378 (2019): 686-707.

[3] Zhu, Quan Min, Li Feng Zhang, and Ashley Longden. "Development of omni-directional correlation functions for nonlinear model validation." Automatica 43.9 (2007): 1519-1531.

[5] Albeaik, Saleh, et al. "Deep Truck: A deep neural network model for longitudinal dynamics of heavy duty trucks." 2019 IEEE Intelligent Transportation Systems Conference (ITSC). IEEE, 2019.

[6] Punjani, Ali, and Pieter Abbeel. "Deep learning helicopter dynamics models." 2015 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2015.

[9] Long, Zichao, et al. "Pde-net: Learning pdes from data." International Conference on Machine Learning. 2018.

[10] Chang, Bo, et al. "Multi-level residual networks from dynamical systems view." arXiv preprint arXiv:1710.10348 (2017).

[11] Chen, Ricky TQ, et al. "Neural ordinary differential equations." Advances in neural information processing systems. 2018.

[12] Rackauckas, Chris, et al. "Diffeqflux. jl-A julia library for neural differential equations." arXiv preprint arXiv:1902.02376 (2019).

[13] Cao, Yang, et al. "Adjoint sensitivity analysis for differential-algebraic equations: The adjoint DAE system and its numerical solution." SIAM journal on scientific computing 24.3 (2003): 1076-1089.

[14] Innes, Michael. "Don't unroll adjoint: differentiating SSA-Form programs." arXiv preprint arXiv:1810.07951 (2018).

[15] Kelly, Jacob, et al. "Learning Differential Equations that are Easy to Solve." arXiv preprint arXiv:2007.04504 (2020).

[20] Jia, Junteng, and Austin R. Benson. "Neural jump stochastic differential equations." Advances in Neural Information Processing Systems. 2019.

[23] Tercan, Hasan, et al. "Transfer-learning: Bridging the gap between real and simulation data for machine learning in injection molding." Procedia CIRP 72 (2018): 185-190.

[24] Pulido, Belarmino, Jesús M. Zamarreño, Alejandro Merino, and Anibal Bregon. "State Space Neural Networks and Model-Decomposition Methods for Fault Diagnosis of Complex Industrial Systems." Engineering Applications of Artificial Intelligence 79 (March): 67–86. doi:10.1016/j.engappai.2018.12.007, 2019.

[25] Fu, Yiwei, et al. "A Dynamically Controlled Recurrent Neural Network for Modeling Dynamical Systems." arXiv preprint arXiv:1911.00089 (2019).

[26] Qin, Tong, Kailiang Wu, and Dongbin Xiu. "Data driven governing equations approximation using deep neural net-works." Journal of Computational Physics 395 (2019): 620-635.

[27] Fält, Mattias, and Pontus Giselsson. "System Identification for Hybrid Systems using Neural Networks." arXiv pre-print arXiv:1911.12663 (2019).

[28] Greydanus, Samuel, et al. "Hamiltonian Neural Networks." Neural Information Processing Systems. 2019.

[29] Mendizabal, A., Márquez-Neila, P., and Cotin, S. "Simulation of hyperelastic materials in real-time using deep learning". Medical Image Analysis 59: 101569, 2020.

[30] M. Mitschke and H. Wallentowitz, Dynamik der Kraftfahrzeuge, Springer, 2004.

[31] G. Rill, Road vehicle dynamics: Fundamentals and modeling. Ground vehicle engineering series, CRC Press, 2012.

[32] S. M. Savaresi, C. Poussot-Vassal, C. Spelta, O. Sename and L. Dugard, Semi-Active Suspension Control Design for Vehicles, Elsevier Ltd., 2010.

[33] M. Fleps-Dezasse, Linear Parameter-Varying Control of Full-Vehicle Vertical Dynamics using Semi-Active Dampers, 2018.

[34] R. Dessort and C. Chucholowski, Explicit model predictive control of semi-active suspension systems using Artificial Neural Networks ({ANN}), Springer Fachmedien Wiesbaden, 2017.

[35] E. Heiden, D. Millard, E. Coumans, Y. Sheng and G. S. Sukhatme, NeuralSim: Augmenting Differentiable Simulators with Neural Networks, IEEE International Conference on Robotics and Automation, 2020.

[36] T. Thummerer, L. Mikelsons, and J. Kircher, NeuralFMU: Towards Structural Integration of FMUs into Neural Networks, *Modelica Conference,* 2021.

[37] Blender Foundation. https://www.blender.org/. Accessed: 12.05.2022

[38] Ansys HFSS. https://www.ansys.com/products/electronics/ansys-hfss. Accessed: 12.05.2022

[39] Ansys SBR+. https://www.ansys.com/content/dam/resource-center/application-brief/ansys-sbr-plus.pdf . Accessed: 12.05.2022

[40] Infineon Technologies BGT60TR13C https://www.infineon.com/cms/en/product/sensor/radar-sensors/radar-sensors-for-iot/60ghz-radar/bgt60tr13c/. Accessed: 12.05.2022[41] Adobe. https://www.mixamo.com/ . Accessed: 12.05.2022

[42] Healthline. https://www.healthline.com/. Accessed: 12.05.2022

[43] Cgtrader. https://www.cgtrader.com/. Accessed: 12.05.2022

6 Acknowledgment

PROJECT PARTICIPANTS:

Virtual Vehicle Research GmbH (AT)
Virtual Vehicle Research GmbH (DE)

Aarhus University – DK
Agro Intelligence ApS – DK
3D Mapping Solutions GmbH – DE
Audi AG – DE
Automotive Solution Center for Simulation e.V. – DE
Infineon Technologies AG – DE
Deutsches Zentrum für Luft- und Raumfahrt (DLR) – DE
iCONDU GmbH – DE
LTX Simulation GmbH – DE
Robert Bosch GmbH – DE
Technische Universität Berlin – DE
University of Augsburg – DE
Virtual Vehicle Research GmbH – DE
Volkswagen A.G. – DE
softwarehelden GmbH & Co. KG
Eindhoven University of Technology – NL
In Summa Innovation b.v. – NL
KE-works BV – NL
LifeTec Group BV – NL
NLR - Royal Netherlands Aerospace Centre – NL
Philips Electronics Nederland BV – NL
Philips Consumer Lifestyle B.V. – NL
Reden BV – NL
Sioux LIME BV – NL
Unit040 Ontwerp B.V. – NL
University of Groningen – NL
BEIA Consult International – RO
Lucian Blaga University of Sibiu – RO
The Manufacturing Research Centre (MTC) – UK
SETLabs Research GmbH – DE

DISCLAIMER

This ITEA3 Project has been made possible by a financial contribution by the German Federal Ministry of Education and Research (BMBF), by the Austrian Research Promotion Agency (FFG), by the Rijksdienst voor Ondernemend Nederland (Rvo) and by the Innovation Fund Denmark (IFD). The Publication as provided reflects only the authors' view.