Hybrid modeling of the human cardiovascular system using NeuralFMUs

Tobias Thummerer, Johannes Tintenherr, Lars Mikelsons

Angaben zur Veröffentlichung / Publication details:

Thummerer, Tobias, Johannes Tintenherr, and Lars Mikelsons. 2021. "Hybrid modeling of the human cardiovascular system using NeuralFMUs." *Journal of Physics: Conference Series* 2090 (1): 012155. https://doi.org/10.1088/1742-6596/2090/1/012155.

PAPER • OPEN ACCESS

Hybrid modeling of the human cardiovascular system using NeuralFMUs

To cite this article: Tobias Thummerer *et al* 2021 *J. Phys.: Conf. Ser.* **2090** 012155

Hybrid modeling of the human cardiovascular system using NeuralFMUs

Tobias Thummerer, Johannes Tintenherr and Lars Mikelsons
Chair of Mechatronics, Augsburg University, Germany
E-mail: {`tobias.thummerer@informatik, johannes.tintenherr@student, lars.mikelsons@informatik`}`.uni-augsburg.de`

Abstract. Hybrid modeling, the combination of first principle and machine learning models, is an emerging research field that gathers more and more attention. Even if hybrid models produce formidable results for academic examples, there are still different technical challenges that hinder the use of hybrid modeling in real-world applications. By presenting *NeuralFMUs*, the fusion of a Functional Mock-up Unit (FMU), a numerical ODE solver and an artifical neural network, we are paving the way for the use of a variety of first principle models from different modeling tools as parts of hybrid models. This contribution handles the hybrid modeling of a complex, real-world example: Starting with a simplified 1D-fluid model of the human cardiovascular system (arterial side), the aim is to learn neglected physical effects like arterial elasticity from data. We will show that the hybrid modeling process is more comfortable, needs less system knowledge and is therefore less error-prone compared to modeling solely based on first principle. Further, the resulting hybrid model has improved in computation performance, compared to a pure first principle white-box model, while still fulfilling the requirements regarding accuracy of the considered hemodynamic quantities. The use of the presented techniques is explained in a general manner and the considered use-case can serve as example for other modeling and simulation applications in and beyond the medical domain.

1. Introduction

The structural integration of physical white-box models into artifical neural networks (ANNs) to retrieve a hybrid model is a growing research field, see [1] or [2]. One milestone was the use of state-of-the-art numerical solvers for ordinary differential equations (ODEs) inside of ANNs instead of residual net structures to reproduce numerical integration and learn dynamic system behavior, like in [3]. The resulting combination of an ANN and an ODE solver - so called *NeuralODEs* - lead to improvements in model accuracy while also enhancing computation and memory cost.

Based on the idea of NeuralODEs we presented *NeuralFMUs* in [4], the structural integration of a first-principle model in form of a FMU, a modern numerical solver like *Tsit5* [5] or *CVODE* [6] and a feed-forward ANN. Further, we provided the open-source frameworks *FMI.jl*[1] and *FMIFlux.jl*[2] to allow for the setup and training of NeuralFMUs just like a convenient ANN in the Julia programming language. In this contribution, the considered technologies shall be tested with a model of the human cardiovascular system (s. Sec. 2.1).

[1] `https://github.com/ThummeTo/FMI.jl`
[2] `https://github.com/ThummeTo/FMIFlux.jl`

One of the core challenges in hybrid modeling, the fusion of data driven and first principle models, is to provide an efficient training process for the resulting heterogeneous structure. For training of ANNs in general, the gradient of the loss function according to the net parameters is needed. This belongs also to hybrid models: The gradient must be determined along the ANN, the numerical solver and the model of the physical system. The different jacobians must be determined by different methods, because of the availability (or non-availability) of information and interfaces between the NeuralFMU components. For example, the mathematical operations inside of ANNs are known, therefore many methods to retrieve partial derivatives, like automatic differentiation, are possible to use. Models exported from modeling tools on the other hand need different techniques, because they do not necessarily provide the symbolic structure of the model or often actively hide this information because it may contain sensible company knowledge. For FMUs, we suggest the use of the optional built-in function `fmi2GetDirectionalDerivative` or - if not available - sampling of additional simulation points and partial derivative approximation via finite (central) differences. Both approaches are implemented as part of *FMIFlux.jl*. Finally, if all jacobians over every component are retrieved, the needed differentiation chains must be deployed to the machine learning framework. For an overview and more detailed insight into the necessary technical steps, see [4].

In the following, some short style explanations about involved tools and standards are given.

1.1. Julia programming language
The Julia Programming Language (hereinafter: *Julia*) is a dynamic typing language, being developed at the *Massachusetts Institute of Technology* since 2009 and first published in 2012 [7]. Julia provides the ability to run code platform-independent and with performance benchmarks similar to native C-implementations. Regardless of the fact that Julia is relatively young (in terms of programming languages), the community is growing rapidly and many other research facilities joined the development of the language and language extensions.

1.2. Modelica
The physical modeling was performed in the object-orientated modeling language *Modelica* (`https://modelica.org/`). Modelica allows acausal modeling, meaning causalization of the system of equations is handled automatically by the compiler at compile time. This allows for the building of large models, while keeping them human understandable by using an intuitive topology and sub-model enclosures. The two most common tools for graphical supported programming with Modelica are Dymola (by Dassault Systèmes ®) and OMEdit (open-source), both tools allow for model export as FMU (s. next subsection).

1.3. Functional Mock-up Interface
Functional Mock-up Interface (FMI) allows for the simulation and parameterization of models outside of the original modeling environment in a standardized and platform-independent format. It is possible to generate standard-compliant models, so called FMUs, in two different modes: model exchange (ME) and co-simulation (CS). ME-FMUs offer an interface for the system dynamics, meaning the FMU computes a system state derivative for a given system state. In a subsequent step outside of the FMU, the next system state can be derived by numerically integrating the state derivative. CS-FMUs already include a numerical ODE solver, which allows for an even easier simulation. On the other hand, this inclusion prevents manipulation of the system dynamics before the numerical integration and consequently makes this mode unattractive for the aim of this contribution.

This paper further divides into three sections: The presentation of the cardiovascular system model, modeling assumptions and the extension to a hybrid model (s. Sec. 2), followed by training and validation of the hybrid model (s. Sec. 3) and finally a short conclusion, including current and future work (s. Sec. 4).

2. Modeling

2.1. Reference model

In [8], a model to simulate arterial pulse waves and their changes over aging of healthy patients is introduced. Together with the model itself, simulation results for arterial blood pressure, volume flow, cross section (luminal area) and photoplethysmography (PPG) for 4,374 model patients of ages between 25 and 75 years are published. The model and simulation data was validated with in vivo data from different sources. The fluid simulation is implemented as laminar, incompressible, newtonian 1D-flow, extended by a viscoelastic term to model diameter changes in arterial cross sections. The simulation setup and parameterization was done in *Matlab* (by MathWorks ®), the 1D-fluid simulation was performed with *Nektar1D* (`http://haemod.uk/nektar`) using the hemodynamic model from [9]. The arterial system is closed by boundary conditions: The heart is abstracted as predetermined mass flow into the Aorta, the vascular beds (the remaining boundary conditions of the fluid simulation) where modeled as three-element windkessels. Please note, that only the *arterial side* of the cardiovascular system is modeled: Starting at the heart, the blood flow passes multiple arteries and finally reaches the vascular beds.

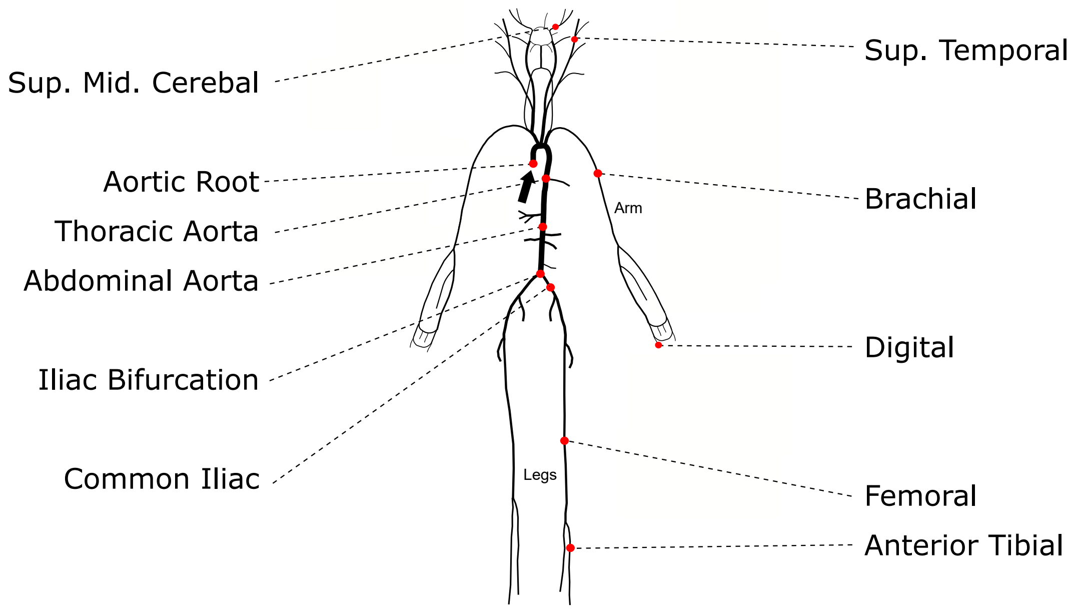


Figure 1: The simplified structure of the arterial side of the human cardiovascular system (figure adapted from [8]). The hemodynamics at the 10 selected artery segments (red) will be learned from data.

The simulation database of the introduced contribution, further referenced as Pulse Wave Database (PWDB) (`https://zenodo.org/record/3275625`), serves as an ideal starting point for the presented machine learning task. The PWDB includes ready-to-use data for 12 arterial segments, the following 10 segments of this set will be used for the later training process:

- *Thoracic Aorta*, end of segment 18
- *Abdominal Aorta*, start of segment 39
- *Iliac Bifurcation*, end of segment 41
- left *Superficial Temporal Artery*, end of segment 87
- left *Superior Middle Cerebral Artery*, end of segment 72
- left *Brachial Artery*, three quarters along segment 21
- left *Digital Arteries*, end of segment 112
- left *Iliac Artery*, half way of segment 44
- left *Femoral Artery*, half way of segment 46
- left *Tibial Artery*, end of segment 49

2.2. First principle model (Modelica)
We start by building up a first principle model in form of an intuitive, object-orientated Modelica model, using default components from the Modelica Standard Library (MSL) as far as possible. However, the presented procedure is not limited to Modelica models, of course.

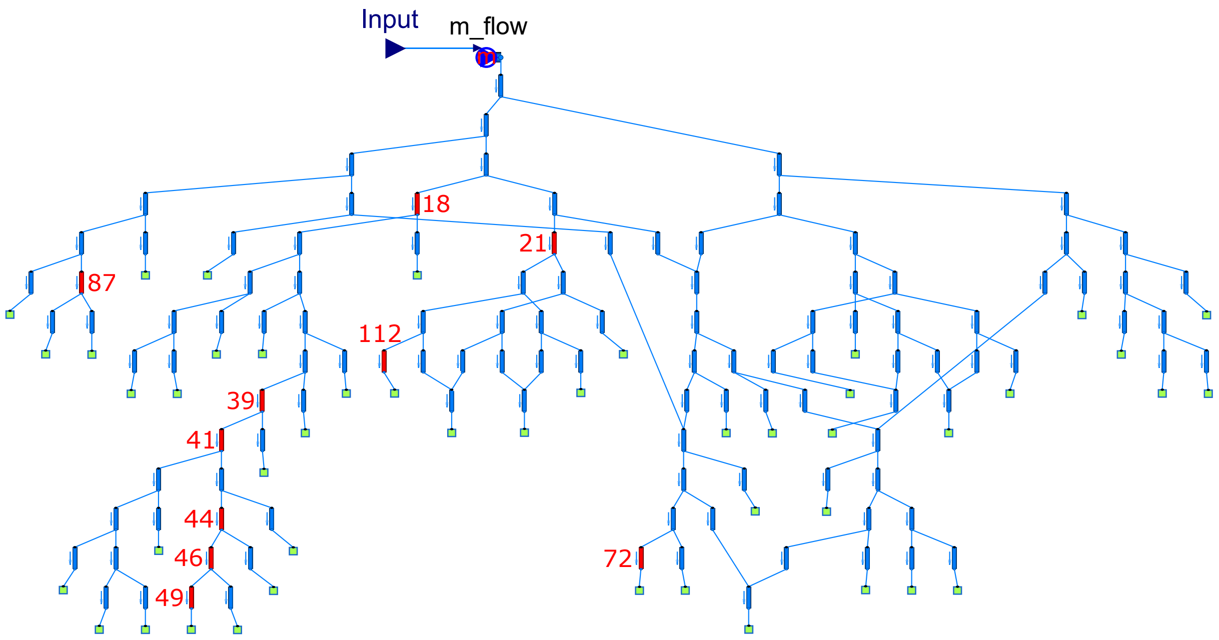
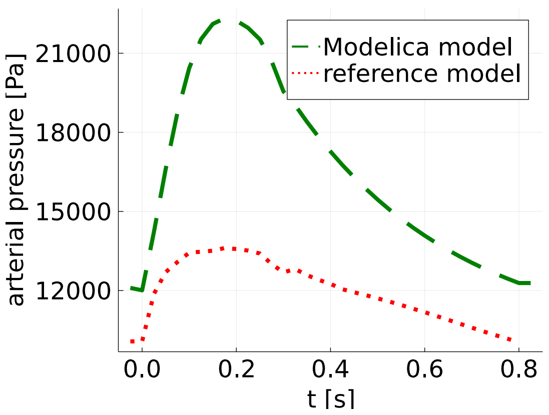


Figure 2: The physical model of the human cardiovascular system (arterial side), consisting of 116 static pipes (arterial segments, red/blue) and 46 three-element windkessels (vascular beds, green), in *Dymola*. From the set of pipes, 10 segments were picked for the machine learning task (red). The mass flow from heart into the Aorta *m_flow* is given as model input. The model was build with components of the MSL.

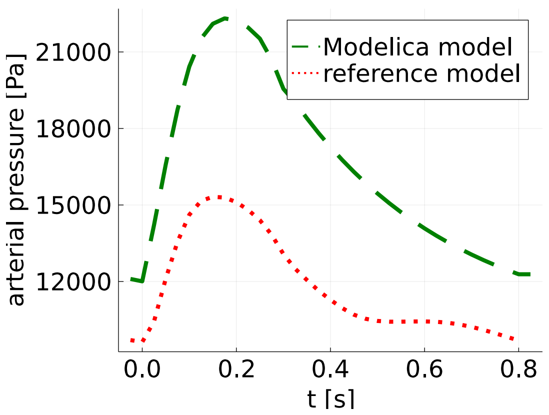
The first principle model of the cardiovascular system is a simplification of the reference model concerning the following aspects:

- The arteries are modeled as static pipes (MSL 3.2.3: Modelica.Fluid.Pipes.StaticPipe), meaning mass, momentum and energy balances are assumed steady-state. As a result, the pipe itself does not store any mass or energy, but physical conservation principles are satisfied.

- The diameters of the arterial segments are assumed constant, meaning the arterial cross section does not vary dependent on the system state (pressure).
- Finally, the arteries are assumed cylindrical, so the inlet and outlet radius are the same size. To parameterize the pipes with parameter data from the reference model, which assumes a pipe with constant diameter change over length, the mean value of inlet and outlet diameter was used.



(a) Thoracic Aorta (#18)



(b) Digital Arteries (#112)

Figure 3: Deviation between the Modelica model (green/dashed) and the reference model (red/dotted) for the simulation of a single pressure pulse wave of subject #1, observed at two different segments.

As to expect, these modeling simplifications lead to a significantly different simulation result compared to the reference model (s. Fig 3). However, the pulse wave characteristic is roughly visible despite the simplifications. The main reason for the difference between reference and simplified Modelica model is the lack of arterial dynamics: The system pressure inside the simplified model is the same at any location inside the arterial network, but varies over time with the heart pulse wave. On the other hand, the simplified model has a much better computational performance. The motivation for hybrid modeling is to retrieve model accuracy close to the reference model, but providing better computational performance at the same time.

To conclude, the only remaining system dynamics are the pressures over the 46 terminal windkessels, meaning the system state $\boldsymbol{x}_{wk}$ (***w**ind**k**essels*) can be uniquely described using a 46 entries state vector. To retrieve a more exact model that describes a dynamic pressure drop over the arterial segments dependent on arterial cross section change by elasticity, more states are needed. To setup the Modelica model for machine learning, placeholder states $\boldsymbol{x}_{art}$ (***art**eries*) were inserted into the system at the considered artery locations, resulting in the concatenated system state vector $\boldsymbol{x} = \boldsymbol{x}_{wk}|\boldsymbol{x}_{art}$. This state space expansion occurs by simply inserting components into the object-orientated model at the desired locations. Two placeholders types are examined: The insertion of fluid capacities with small capacitance (hereinafter: *C-placeholders*, one state) and the insertion of parallel circuits consisting of a capacitor and an inductance (hereinafter: *LC-placeholders*, two states). This reflects a straightforward model enhancement process: Adding dynamic placeholders to a model at the locations needed, while the correct physical term is learned from data. Note, that the use of a variety of placeholders is possible, like RLC-circuits for example. After this final step, both models (one with C- and one with LC-placeholders) were exported as FMUs.

2.3. Hybrid model
Hybrid modeling is a technique picked often, if higher modeling accuracy is needed, but conventional first principle modeling does not increase precision, is not economically practicable anymore or is simply not possible because of lack of system knowledge. On the other hand, almost any development process in medicine, mechanics or electronics generates data, that can be used to achieve better (more precise or faster computing) hybrid models.

During hybrid modeling, we will neglect knowing the reference system and view the problem from a different perspective: Starting with the simple Modelica model, we want to improve model accuracy without knowing the physical principles needed for this step. Therefore, the improvement process is done solely on basis of the reference model simulation data, not on physical system knowledge. This mirrors a typical use case: Model accuracy shall be improved, further first principle modeling is not an option, but measurement data of a more accurate (or the real) system is available and can be used.

To start with hybrid modeling, the FMU models exported from Dymola are imported into Julia using the library *FMI.jl*. With the library extension *FMIFlux.jl*, we can setup and train NeuralFMUs on basis of the Modelica model FMUs. For the considered use-case, two NeuralFMUs as in Fig. 4 (one with C- and one with LC-placeholders) with layer dimensions as in Tab. 1 were examined.

Table 1: Topologies of the considered NeuralFMUs.

Layer	Type	C-placeholders		LC-placeholders		Activation
		Inputs	Outputs	Inputs	Outputs	
#1	state vector separation	56	10 | 46	66	20 | 46	none
#2	data pre-processing	10	10	20	20	none
#3	bias	10	10	20	20	none
#4	data post-processing	10	10	20	20	none
#5	state vector merging	10 | 46	56	20 | 46	66	none
#6	FMU	56	56	66	66	none
#7	derivative vector separation	56	10 | 46	66	20 | 46	none
#8	data pre-processing	10	10	20	20	none
#9	dense	10	30	20	30	tanh
#10	dense	30	10	30	20	none
#11	data post-processing	10	10	20	20	none
#12	derivative vector merging	10 | 46	56	20 | 46	66	none

3. Training & Validation
During training, three pulse wave cycles for patient #1 are simulated ($2.466\,s$, varies between subjects because of different heart rates). The first pulse wave is ignored in the training process and allows the system to retrieve a stationary (periodically repeating) state. The training takes place on two cycles to promote learning a time-invariant system behavior, meaning a periodically repeating input (heart) should generate a periodically repeating output (arterial pressure curves) with same period length. Even if the PWDB includes data observed at $500\,Hz$, the training process is sampled down to $40\,Hz$ because of training performance optimization. To conclude, training takes place on only two pulse waves of one patient, resulting in 66 data points à 10

arterial pressure values. Note, that this is a very small training base in the field of machine learning.

The training results, comparing the original FMU pressure (from the Modelica model), the improved NeuralFMU pressures and the target pressure (reference system data) can be found in Fig. 5. As loss function, a simple mean-squared-error with increasing time horizon between the reference system and NeuralFMU arterial blood pressures was deployed. The loss function rates the deviation only for a random sub-set of three artery segments, while changing the sub-set every training cycle (*Batching*). The training itself was controlled via parameter freezing: Until reaching a significant small error (loss), parameters of the derivative ANN (s. Fig. 4: bottom) were locked to force compensation of state offsets by the state ANN (s. Fig. 4: top) and prevent early corruption of the system dynamics. After reaching a loss threshold, all net parameters were unlocked and trained in parallel.

For the C-placeholders, the static pressure wave from the FMU model is vertically scaled and shifted to fit the reference system data as good as possible. The LC-placeholders provide a better fit, because the LC-circuit additionally allows for phase-shifting the original pulse waves. The NeuralFMUs produce less data (only 10 dynamic locations instead of 116) and the output result does only approximate the reference model output, because of the relative small ANN layout and a lack of states to interfere with the system. On the other hand, there is a significant gain in performance, which motivates the use of NeuralFMUs for use-cases, that focus on less measurement locations and/or more on fast computations than maximum precision, like embedded hardware or wearable computing. The reference model has a simulation time[3] of $\approx 756\,s$ for 10 pulse waves, resulting in an average $\approx 75.6\,s$ per pulse waves. The NeuralFMUs on the other hand, have a simulation time[4] of only $\approx 0.20\,s$ per 10 pulse waves, meaning an average of $\approx 0.02\,s$ per pulse wave[5]. Even if both simulations were performed on different systems, a significant performance gain by a factor of ≈ 3750 is clearly visible. This scope can be used to improve accuracy through deeper and/or wider network topologies or the injection of more and/or more complex state placeholders if necessary.

Testing the NeuralFMUs with unobserved data from patient #2 leads to similar fitting pressure curves (s. Fig. 6), however the remaining average error is slightly bigger. Even if the derivative ANN (bottom) did learn a simplified abstraction of the pressure dynamics, the simplification itself was understood in a generalized manner. Please note at this point, that the heart rate differs between the two patients ($\approx 73\,bpm$ for #1 and $\approx 77\,bpm$ for #2). The state ANN (top) was only able to learn static offsets between the states of the reference system and the first principle model parameterized for patient #1, but the offsets in data differ for patient #2. It is visible in the plots, that a small vertical offset between training and testing results exists. These offsets could be reduced by training the state ANN on data of patient #2. A more detailed result validation is omitted at this point, but will be part of a pursuing contribution.

4. Conclusion & Future Work

We highlighted a workflow to improve a first principle model (at the example of a simple, object-orientated model) using a NeuralFMU without knowing the underlying physical equations necessary. The enhancements were learned solely based on a small set of data, in the presented example generated by a more accurate reference system. Starting with the FMI model export from the modeling tool, the resulting FMU was imported into the Julia programming language using the open-source library *FMI.jl*[6]. Inside Julia, a NeuralFMU was set up and trained, using

[3] AMD Ryzen$^{\text{TM}}$ 9 3900X on Ubuntu 20.04.2 LTS

[4] Intel® Core$^{\text{TM}}$ i7-8565U on Windows 10 Enterprise 20H2

[5] Even if the number of states and mathematical complexity of the FMUs differ between C- and LC-placeholders, the difference in simulation time of the resulting NeuralFMUs was marginal.

[6] `https://github.com/ThummeTo/FMI.jl`

the library extension *FMIFlux.jl*[7]. Finally, the simulation data was compared to the target values from the PWDB. The resulting NeuralFMU produces, dependent on the requirements of the underlying application, sufficiently accurate results. Model precision can be further improved by using more or other circuits for the state placeholders or wider and deeper ANN structures. Further, the use of more than one patient during training will improve the NeuralFMU prediction quality on unknown patients.

In terms of computational efficiency, a significant performance gain is visible for the considered example, making the presented approach interesting for performance critical applications. NeuralFMUs have further advantages, e.g. even though they might include black-box models, they are fully differentiable. This allows for the use of other machine learning techniques or efficient model examination methods, like gradient based algorithms.

NeuralFMUs open up to many new interesting use-cases, but bring up many new challenges, too. For example, most physical systems are stable by default, meaning they converge against a physical equilibrium if not disturbed. By manipulating the system dynamics via an ANN, this natural stability is not guaranteed anymore, which may result in a destabilized training process. Best practices for network initialization and stabilization methods during training must be considered and will be part of a pursuing contribution.

The corresponding sources for the presented example will be published soon as part of the *FMIFlux.jl* library repository.

Acknowledgments
This work has been partly supported by the ITEA 3 cluster programme for the project UPSIM - Unleash Potentials in Simulation.

References

[1] Willard J, Jia X, Xu S, Steinbach M and Kumar V 2020 Integrating physics-based modeling with machine learning: A survey
[2] Rai R and Sahu C K 2020 *Driven by Data or Derived Through Physics? A Review of Hybrid Physics Guided Machine Learning Techniques With Cyber-Physical System (CPS) Focus* vol 8 pp 71050–71073
[3] Chen T Q, Rubanova Y, Bettencourt J and Duvenaud D 2018 Neural ordinary differential equations (*Preprint* 1806.07366) URL http://arxiv.org/abs/1806.07366
[4] Thummerer T, Kircher J and Mikelsons L 2021 NeuralFMU: Towards structural integration of FMUs into neural networks (14th International Modelica Conference: Preprint, Accepted)
[5] Tsitouras C 2011 *Runge–Kutta pairs of order 5(4) satisfying only the first column simplifying assumption* vol 62 pp 770–775 URL https://www.sciencedirect.com/science/article/pii/S0898122111004706
[6] Hindmarsh A C, Serban R, Balos C J, Gardner D J, Reynolds D R and Woodward C S 2021 User documentation for cvode v5.7.0 (sundials v5.7.0) Tech. rep. URL https://computing.llnl.gov/sites/default/files/cv_guide-5.7.0.pdf
[7] Bezanson J, Karpinsky S, Shah V B and Edelman A 2012 Julia: A fast dynamic language for technical computing (*Preprint* 1209.5145) URL http://arxiv.org/abs/1209.5145
[8] Charlton P H, Mariscal Harana J, Vennin S, Li Y, Chowienczyk P and Alastruey J 2019 *Modeling arterial pulse waves in healthy aging: a database for in silico evaluation of hemodynamics and pulse wave indexes* vol 317 pp H1062–H1085 pMID: 31442381 (*Preprint* https://doi.org/10.1152/ajpheart.00218.2019) URL https://doi.org/10.1152/ajpheart.00218.2019
[9] Alastruey J, Parker K H and Sherwin S J 2012 *Arterial pulse wave haemodynamics* (Virtual PiE Led t/a BHR Group) pp 401–443 ISBN 9781855981331

[7] https://github.com/ThummeTo/FMIFlux.jl

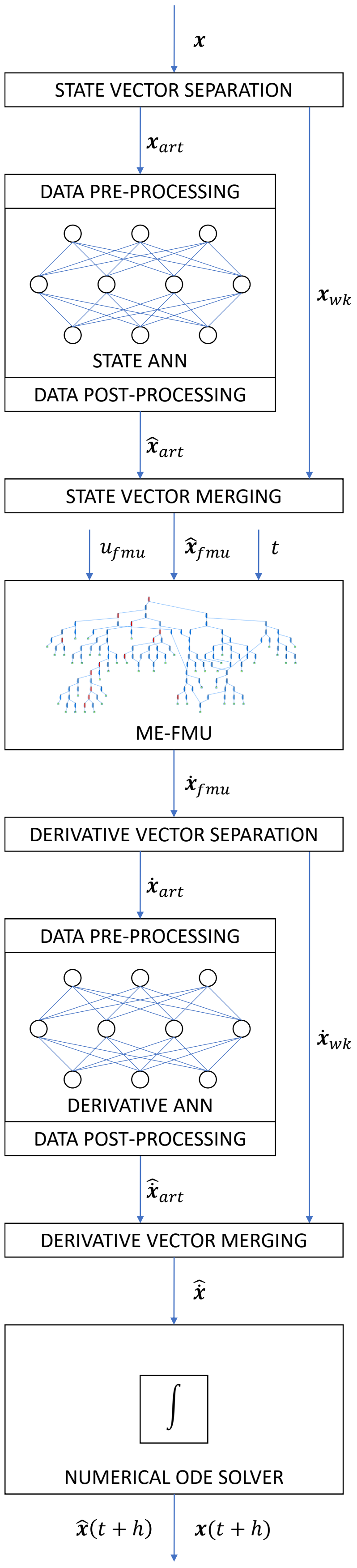
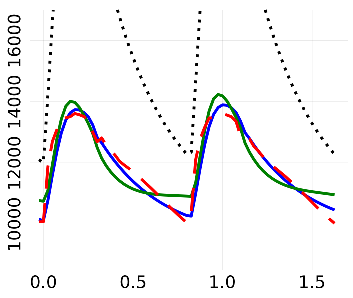
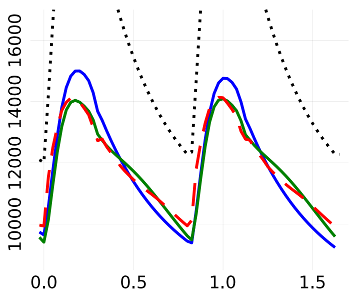

$\boldsymbol{x}$
STATE VECTOR SEPARATION
$\boldsymbol{x}_{art}$
DATA PRE-PROCESSING
STATE ANN
DATA POST-PROCESSING
$\boldsymbol{x}_{wk}$
$\hat{\boldsymbol{x}}_{art}$
STATE VECTOR MERGING
u_{fmu}
$\hat{\boldsymbol{x}}_{fmu}$
t
ME-FMU
$\dot{\boldsymbol{x}}_{fmu}$
DERIVATIVE VECTOR SEPARATION
$\dot{\boldsymbol{x}}_{art}$
DATA PRE-PROCESSING
DERIVATIVE ANN
DATA POST-PROCESSING
$\dot{\boldsymbol{x}}_{wk}$
$\hat{\dot{\boldsymbol{x}}}_{art}$
DERIVATIVE VECTOR MERGING
$\hat{\dot{\boldsymbol{x}}}$
$\int$
NUMERICAL ODE SOLVER
$\hat{\boldsymbol{x}}(t+h)$
$\boldsymbol{x}(t+h)$

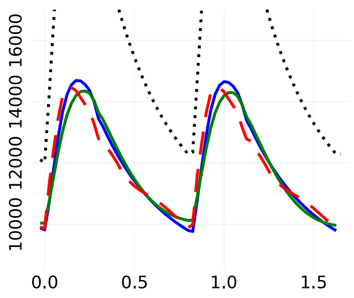
Figure 4: The hybrid model structure. From top to bottom: Because not all states of the NeuralFMU shall be part of the training process, the state vector $\boldsymbol{x}$ is separated into the two sub-vectors $\boldsymbol{x}_{wk}$ (the windkessels' states) and $\boldsymbol{x}_{art}$ (the arterial pressure states). Before being fed into the upper ANN, the arterial state values $\boldsymbol{x}_{art}$ are pre-processed to ensure being inside an ANN-compliant range and are transformed back after the ANN pass. The state ANN at the top of the topology is able to learn and compensate state dependent modeling failures, like static offsets in measurement data in this special case.
The new arterial state estimate $\hat{\boldsymbol{x}}_{art}$ is merged together with the original windkessel states $\boldsymbol{x}_{wk}$ into the modified system state vector $\hat{\boldsymbol{x}}_{fmu}$, that is further passed, together with the system input u_{fmu} and the current simulation time t, to the ME-FMU.
Inside the FMU, the current state derivatives $\dot{\boldsymbol{x}}_{fmu}$ are computed based on the given system state $\hat{\boldsymbol{x}}_{fmu}$.
As for the system state, the system state derivative vector is separated into the arterial $\dot{\boldsymbol{x}}_{art}$ and windkessel state derivative vector $\dot{\boldsymbol{x}}_{wk}$. The arterial pressure dynamics are pre-processed, fed to the ANN, post-processed and the resulting $\hat{\dot{\boldsymbol{x}}}_{art}$ is merged back together with $\dot{\boldsymbol{x}}_{wk}$ into the system state derivative estimate $\hat{\dot{\boldsymbol{x}}}$. The derivative ANN is therefore able to manipulate the system dynamics to retrieve a different system behavior. In the considered use case, the pressure drop over the arterial segments can be learned based on data. Because the ANN is learning on basis of state derivatives, generalized learning of the correct physical laws is promoted.
Finally, the system dynamics estimate $\hat{\dot{\boldsymbol{x}}}$ is numerically integrated with step size h into the next system state estimate $\hat{\boldsymbol{x}}(t+h)$ or the next system state $\boldsymbol{x}(t+h)$ respectively.



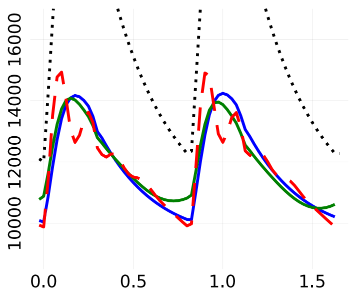
(a) Thoracic Aorta (#18)



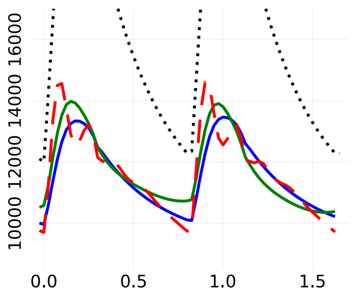
(b) Abdominal Aorta (#39)



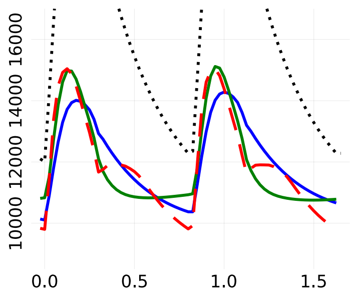
(c) Iliac Bifurcation (#41)



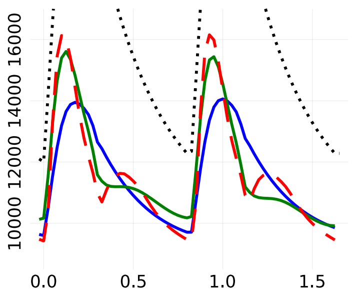
(d) Sup. Temp. Artery (#87)



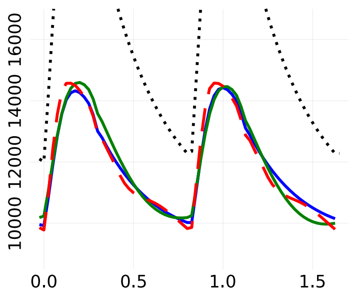
(e) Mid. Cerebral Artery (#72)



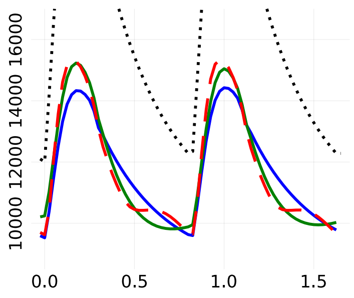
(f) Brachial Artery (#21)



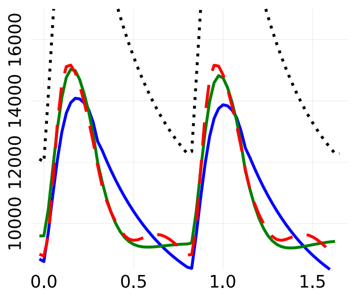
(g) Digital Arteries (#112)



(h) Com. Iliac Artery (#44)

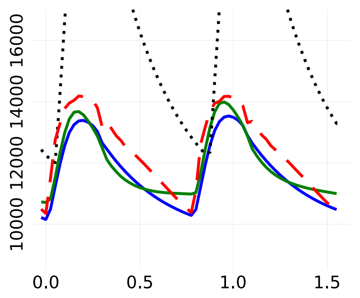


(i) Femoral Artery (#46)

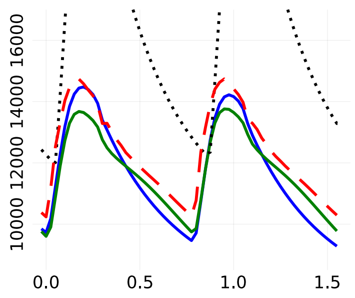


(j) Ant. Tibial Artery (#49)

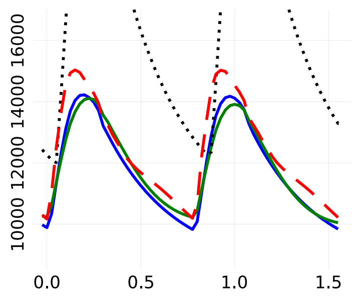
Figure 5: Training results of the 10 considered arterial segments, based on data of subject #1 of the PWDB from [8]. The horizontal axis labels the simulation time t in seconds, the vertical axis the arterial pressure in Pascals. The plots show the pressure of the Modelica model (black/dotted, clipped), the target reference system pressure (red/dashed), the pressure learned by the NeuralFMU with C-placeholders (blue/solid) and with LC-placeholders (green/solid).



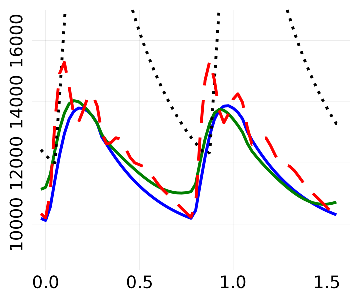
(a) Thoracic Aorta (#18)



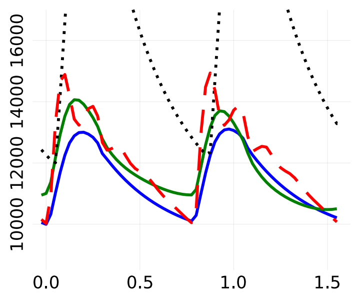
(b) Abdominal Aorta (#39)



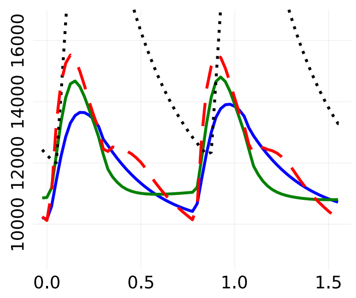
(c) Iliac Bifurcation (#41)



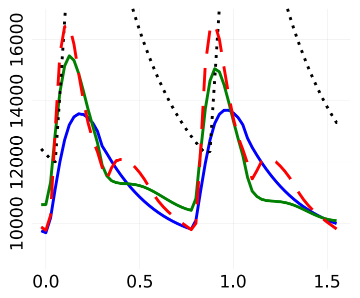
(d) Sup. Temp. Artery (#87)



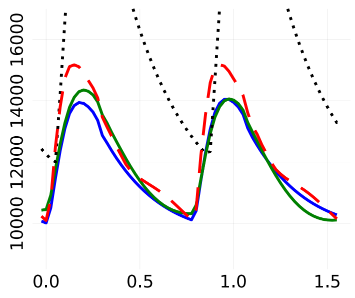
(e) Mid. Cerebral Artery (#72)



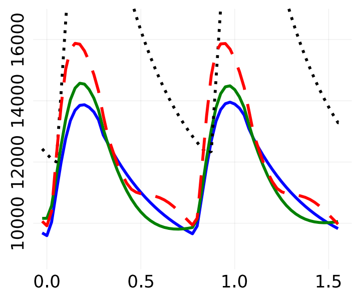
(f) Brachial Artery (#21)



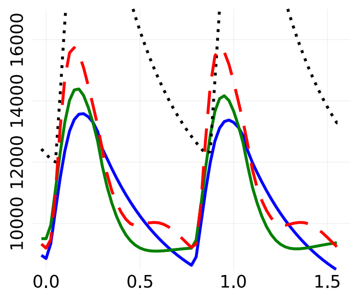
(g) Digital Arteries (#112)



(h) Com. Iliac Artery (#44)



(i) Femoral Artery (#46)



(j) Ant. Tibial Artery (#49)

Figure 6: Testing results on the 10 considered arterial segments. Tests where performed against data of (unknown) subject #2 of the PWDB from [8]. The horizontal axis labels the simulation time t in seconds, the vertical axis the arterial pressure in Pascals. The plots show the pressure of the Modelica model (black/dotted, clipped), the target reference system pressure (red/dashed), the pressure learned by the NeuralFMU with C-placeholders (blue/solid) and with LC-placeholders (green/solid).