FliPASED
WWW.FLIPASED.EU

D2.6 Validation of data science based methods for modelling and control

Balint Vanek, Bela Takarics, Bence Hadlaczky

GA number:	815058
Project acronym:	FLIPASED
Project title:	FLIGHT PHASE ADAPTIVE AEROSERVO-ELASTIC AIRCRAFT DESIGN METHODS
Funding Scheme: H2020	**ID:** MG-3-1-2018
Latest version of Annex I:	1.1 released on 12/04/2019
Start date of project: 01/09/2019	**Duration:** 40 Months

Lead Beneficiary for this deliverable:	SZTAKI
Last modified: 31/05/2023	**Status:** Delivered
Due date: 28/02/2023	

Project co-ordinator name and organisation:	Bálint Vanek, SZTAKI
Tel. and email: **+36 1 279 6113**	vanek@sztaki.hu
Project website:	www.flipased.eu

Dissemination Level		
PU	Public	X
CO	Confidential, only for members of the consortium (including the Commission Services)	

“This document is part of a project that has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 815058.”

Glossary

ADAM	Adaptive Moment Estimation
CNN	Convolutional Neural Network
DOF	Degree of Freedom
EKF	Extended Kalman Filter
FEM	Finite Element Method
GRU	Gated Recurrent Unit
IMU	Inertial Measurement Unit
LPV	Linear Parameter Varying
ML	Machine Learning
ReLU	Rectified Linear Unit
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network

Table of contents

List of Figures

1 Executive Summary

This report addresses the aspects of linear (parametrized) model approximation of dynamical systems, in view of control design. The model-free, or data-based approaches and their application to the flight data specific objectives will be described within the deliverable. In this work we are adopting big-data techniques to analyze the vast data provided by the complex sensing and control system. These methodologies are useful in mapping and revealing the underlying structure of the problem. Data science technologies for optimal usage of these data are developed in FliPASED, and recommendations for methods and useful sensor arrangements for future aerospace applications are described.

The machine learning based approach results are presented through a flexible state estimation of the wings of the T-Flex aircraft. The investigated methods are described along with the used state-space model of the aircraft. The obtained results are presented and evaluated. Finally, conclusions are drawn.

2 Motivation for flexible dynamics estimation

The dynamic behaviour, stability, and the effects of the aerodynamic drag of a large-wingspan aircraft are mainly influenced by the structural flexibility and shape of its wings during flight. Large commercial aircraft has large mass variation during flight, as fuel is consumed, hence optimal (minimum drag) configuration at one point of the mission might not be optimal in other parts of the flight. Aircraft design accounts for this change by simultaneously optimising the wing lift and drag for multiple points within the flight, but the typical optimization relies on passive means with the assumption that flaps have to be at zero deflection during the trimmed cruise phase of flight. On the other hand if a database (most likely derived by CFD tools) is available about the optimal wing shape and the corresponding flap deflections, leading to minimum drag at each point within the cruise flight envelope, significant reduction can be achieved in terms of fuel consumption. For each individual point in the flight envelope the optimal wingshape has to be achieved by an adequate wingshape controller, what might not only contain flap scheduling but also setpoint tracking of the optimal modal coordinates of the wing.

Therefore, utilizing a wing shape controller that minimizes the effects of drag can greatly improve the behaviour and fuel consumption of the aircraft. However, such a controller requires the measurement of the dynamics of the wing, more precisely, the modal coordinates which describe the structural and dynamic changes of the wing. For estimating the modal coordinates and reconstructing the wing shape a state observer is necessary because the direct and accurate measurement of these states is not feasible. Two approaches are investigated as possible solutions for the state estimation task. First, Extended Kalman Filtering (EKF) is presented, using a Linear Parameter Varying (LPV) system model. Second, a machine learning-based approach is introduced based on the new KalmanNet architecture with two different recurrent neural network configurations: one with linear layers and one with one-dimensional convolutional layers. The results are evaluated on the T-Flex aerial demonstrator aircraft and compared using the LPV-based EKF as a reference.

3 T-Flex dynamic model

This section will briefly present the dynamic model of the T-Flex used by the different state estimation methods.

3.1 Nonlinear model

The model of the T-Flex aircraft is given as a nonlinear state-space system. A reduced order model is used to decrease the computation burden. Details of the modelling and model order reduction are given in the articles [3], [7], [8]. The system has 48 states. The state vector $x \in \mathbb{R}^{48}$ consists of rigid body states, states related to the flexible dynamics of the wing and aerodynamic lag states (denoted as x_{flex}), and finally, states that represent the control surface inputs and their first derivatives. The rigid body motion is represented with a 6-DOF model with 12 states: states of translational (u, v, w) and angular (p, q, r) velocities, position (x, y, z), and orientation (ϕ, θ, ψ). In the used state-space model the states of the x and y positions were truncated because, under certain trim conditions, these can cause unstable poles to appear, which caused poor performance during state estimation.

The flexible dynamics of the wing are described with $w(x,t) = \sum_{i=1}^{\infty} \Phi_i(x) U_{fi}(t) \approx \sum_{i=1}^{6} \Phi_i(x) U_{fi}(t)$, where $\Phi_i(x)$ denotes the i^{th} mode shape and $U_{fi}(t)$ the i^{th} modal coordinate. So $x_{\text{flex}} \in \mathbb{R}^{14}$ contains the first six modal coordinates (U_{f1}, U_{f2}, U_{f3}, U_{f4}, U_{f5}, U_{f6}) their first derivatives ($\dot{U}_{f1}$, $\dot{U}_{f2}$, $\dot{U}_{f3}$, $\dot{U}_{f4}$, $\dot{U}_{f5}$, $\dot{U}_{f6}$) and two lag states (lag_1, lag_2). The reason for using only the first six modal coordinates in the model is that these have the largest contribution in the description of the flexible behaviour of the wings [3]. The objective of this research is the estimation of the x_{flex}.

The sensors and control surfaces of the system are depicted in Figure 1. The system has 13 inputs, $u \in \mathbb{R}^{13}$, which contains one turbofan engine input (Throttle) and 12 control surface inputs: four plus four ailerons (AileronR/L) on each wing and on the V-tail two plus two 'ruddervators' (TailR/L).

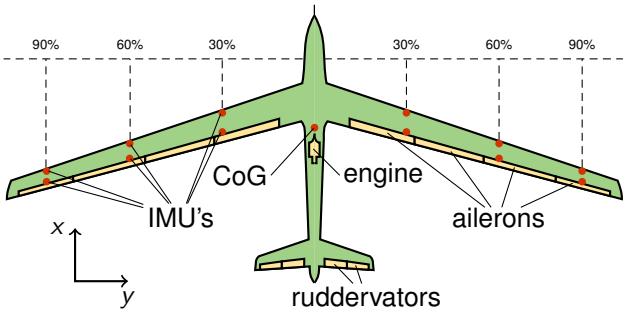

Figure 1: Demonstrator control surfaces and IMU locations

The output vector $y \in \mathbb{R}^{64}$ of the system has 23 rigid body-related sensors, which provide information about the position (X_E, Y_E, Z_E), orientation (ϕ, θ, ψ), translational (v_N, v_E, v_D) and angular velocity (p, q, r), and acceleration (a_{xB}, a_{yB}, a_{zB}) of the aircraft. Furthermore, the angle of attack (α), sideslip angle (β), static pressure (p_a) and total pressure (p_T), barometric altitude (h_{baro}), indicated (v_{IAS}) and the true airspeed (v_{TAS}) are measured as well. These sensors are located near the center of gravity (CoG) of the fuselage, as well as at the nose air data boom. Each wing of the demonstrator has six plus six additional inertial measurement units (IMUs). The IMUs of the leading-edge measure accelerations in the x, y, and z directions, while the IMUs of the trailing edge provide angular velocity data around the

x- and y-axis, and acceleration data in the z direction. The exact location of the IMUs can be seen in Figure 1 as well, the y axis of these sensors are aligned with the front and rear spar of the wing respectively and hence they are not parallel with the body axes. In addition, the wingtip coordinates are measured with a mono camera preventing acceleration-based estimation errors from diverging in time [2]. The coordinates of four wingtip points are measured in each direction.

3.2 LPV model

In our work, we created a Linear Parameter Varying (LPV) approximation of the nonlinear model of the T-Flex demonstrator [6]. It is essentially a point-wise linearization of the nonlinear state space system. Different trim points are defined by the – so called – scheduling parameters, thus creating a multidimensional grid. At each grid point the nonlinear system is linearized using the corresponding trim conditions. Then the resulting linear, state-space model is assigned to that grid point. So as the scheduling parameters change during simulation a linear model is selected from the LPV model according to the current values of the scheduling parameters, thus providing a linear approximation of the nonlinear system around that operating point. The higher the resolution of the multidimensional grid of the scheduling parameters, the more accurately can the LPV model approximate the behaviour of the nonlinear system. But as a drawback, a high resolution grid results in a large LPV model, which makes its usage more computation heavy. As scheduling parameters ($\rho \in \mathbb{R}^2$), the true airspeed (v_{TAS}) and the roll angle (ϕ) outputs were chosen. The grid for the LPV model consists of airspeed values from 35 m/s to 55 m/s with a 0.1 m/s resolution and the roll angles from $0°$ to $45°$ with $1°$ resolution. The state-space equations of the discrete-time LPV system are

$$\begin{aligned} x[k] &= A(\rho[k])x[k-1] + B(\rho[k])u[k], \\ y[k] &= C(\rho[k])x[k] + D(\rho[k])u[k], \end{aligned} \tag{1}$$

where $\rho[k]$ is the time-varying vector of the scheduling parameters at time step k, with a nominal sampling time of $1/200sec$. $A(\rho[k]) \in \mathbb{R}^{48\times48}$, $B(\rho[k]) \in \mathbb{R}^{48\times13}$, $C(\rho[k]) \in \mathbb{R}^{64\times64}$ and $D(\rho[k]) \in \mathbb{R}^{64\times13}$ denotes the parameter-dependent state-space matrices of the LPV system. The state vector is denoted as $x[k]$, the input vector as $u[k]$, while $y[k]$ is the output vector of the system at time step k. The main advantage of using an LPV model is that only linear algebraic operations are required to compute the state and output evolutions of the system, instead of the computationally more challenging nonlinear functions. Also by selecting the resolution of the grid of the scheduling parameters appropriately (with sufficient but not overly high resolution), the LPV model requires less storage space than the nonlinear model, while only providing minuscule differences in the state and observation trajectories compared to the nonlinear model.

4 Model-based wing shape estimation

Extended Kalman Filtering (EKF) is used as a model-based wing shape estimation approach. The EKF is the extension of the standard Kalman filter to be used with nonlinear systems for state estimation and sensor fusion. The EKF pipeline requires the full, nonlinear state-space description of the system and information about the model noise and observation noise in the form of noise covariance matrices (denoted as Q and R respectively). The discrete-time nonlinear state-space system is of the form:

$$\begin{aligned} x[k] &= \mathrm{f}(x[k-1], u[k]) + w[k], \\ y[k] &= \mathrm{h}(x[k], u[k]) + v[k]. \end{aligned} \tag{2}$$

Here, the nonlinear function $\mathrm{f}(.)$ is called state-transition function, while $\mathrm{h}(.)$ is called state-observation function. The $w[k] \in \mathbb{R}^{48}$ and $v[k] \in \mathbb{R}^{64}$ vectors are the model noise and observation noise vectors, which are described by their covariance matrices $Q \in \mathbb{R}^{48\times48}$ and $R \in \mathbb{R}^{64\times64}$ respectively. Both noise processes are assumed to have 0 mean, normal distributions, and the noise samples at each time step are mutually independent.

The general framework of the EKF consists of two main steps: *prediction* and *update*. In these steps, point-wise linearization is used to approximate the behaviour of the nonlinear system: the Jacobians of the nonlinear state-transition and state-observation functions are calculated to get the linear, state-space matrices $A[k]$, $B[k]$, $C[k]$ and $D[k]$ at each time step k. In the *prediction* step, the prior state estimation is calculated using the inputs of the current time step and the estimations from the previous time step with

$$\hat{x}[k|k-1] = \mathrm{f}(\hat{x}[k-1|k-1], u[k]). \tag{3}$$

The prior state estimation covariance $P \in \mathbb{R}^{48\times48}$ is

$$P[k|k-1] = A[k]P[k-1|k-1]A[k]^T + Q. \tag{4}$$

In the *update* step, first, the innovation

$$\tilde{y}[k] = y[k] - \mathrm{h}(\hat{x}[k|k-1], u[k]) \tag{5}$$

is calculated. Then the Kalman gain, $K_G \in \mathbb{R}^{64\times48}$

$$K_G[k] = P[k|k-1]C[k]^T(C[k]P[k|k-1]C[k]^T + R)^{-1}. \tag{6}$$

With the help of the Kalman gain, the posterior state vector

$$\hat{x}[k|k] = \hat{x}[k|k-1] + K_G[k]\tilde{y}[k], \tag{7}$$

and state prediction covariance

$$P[k|k] = (I - K_G[k]C[k])P[k|k-1] \tag{8}$$

is computed.

To obtain a point-wise linearization we use the LPV model. During simulation, the true airspeed and roll angle are used as the scheduling parameters in the LPV model and are measured at each time step k. The resulting $A[k]$, $B[k]$, $C[k]$ and $D[k]$ matrices are used in the calculations of the EKF. Then the EKF conducts the prediction and update steps. To determine Q, both the nonlinear and the LPV models are simulated with doublet inputs on the control surfaces and then the measured outputs and states are compared, and variances of the differences are calculated. The noise variances, R, of the onboard sensors of the T-Flex are specified based on the datasheets of the sensors and using data from previous flight tests [1].

5 Learning-based wing shape estimation

5.1 KalmanNet architecture

The other approach for estimating the flexible states of the T-Flex is to use machine learning. We employ the recently published KalmanNet architecture [5]. The algorithm (or *pipeline*) for the KalmanNet is presented in Figure 2. KalmanNet combines Kalman filtering with a neural network as it uses similar *prediction* and *update* steps, but without computing the state prediction covariance matrix (P). Consequently, the model noise covariance matrix (Q) is not involved. For providing the Kalman gain (Step 4 in Figure 2), a trained Recurrent Neural Network (RNN) is used, thus the observation noise covariance matrix (R) is not involved either. The neural network uses the innovation difference $\Delta y[k] = y[k] - \hat{y}[k|k-1]$, the forward update difference $\Delta \hat{x}[k] = \hat{x}[k-1|k-1] - \hat{x}[k-1|k-2]$, and the roll angle ϕ scheduling parameter as input features. The advantage of the KalmanNet compared to the EKF is that it does not require any information about the model of the noise processes and the promise of better generalization capabilities.

The standard Kalman gain predicting neural network [5] uses a Gated Recurrent Unit (GRU) as the recurrent layer and linear layers with Rectified Linear Units (ReLU) as the activation function. The neural network has a linear layer as the input layer with ReLU activation, followed by the GRU. After the GRU layer, there is another linear layer with ReLU activation, then the linear output layer. As the aircraft model we use is high-dimensional (48 states, 64 outputs), we slightly decreased the dimensions of each layer compared to the original architecture to reduce the computation burden.

Apart from the *linear RNN* architecture, we implement a different neural network that still uses a GRU cell, but instead of linear layers, it uses three convolutional blocks at the beginning of the network [9]. A convolutional block consists of a 1D convolutional layer followed by a ReLU activation function. After the ReLU a Batch Normalization layer is used, followed by a Dropout layer with 0.25 dropout probability. The output layer is a linear layer, which provides the Kalman gain matrix. The kernel size for each 1D convolution layer is seven. As the 1D convolutional layer requires a trajectory, or time-window of input features, simply using the forward update difference ($\Delta y[k]$), innovation difference ($\Delta \hat{x}[k]$) and roll angle (ϕ) input features of the current time step is not adequate. Therefore, we use the input features of the current time step and the input features from the previous 19 time steps in the time-window buffer. In Figure 2, the architecture with the convolutional layers represents the neural network. The number of features is shown below the convolutional blocks and the pool size below the max pooling layer. The number of units is indicated underneath the GRU and the linear layer. The dropout rate is shown below the dropout layer.

From here on – for the sake of brevity – the original KalmanNet architecture is referred to as the *linear* RNN architecture, while the second, new architecture is referred to as the *convolutional* RNN architecture after their defining layer types.

For initializing the layer weights, a standard normal distribution is used. Since the architecture incorporates a discrete-time system, it has a high sensitivity to the initial weight values. Therefore, the standard deviation of the normal distribution for the initialization has to be chosen very small ($5 \cdot 10^{-6}$) to avoid the otherwise highly diverging training process.

5.2 Training, validation and test data

For training a neural network, generally three different datasets are required: training, validation, and test datasets. The training dataset – as its name suggests – solely used for optimizing the weights

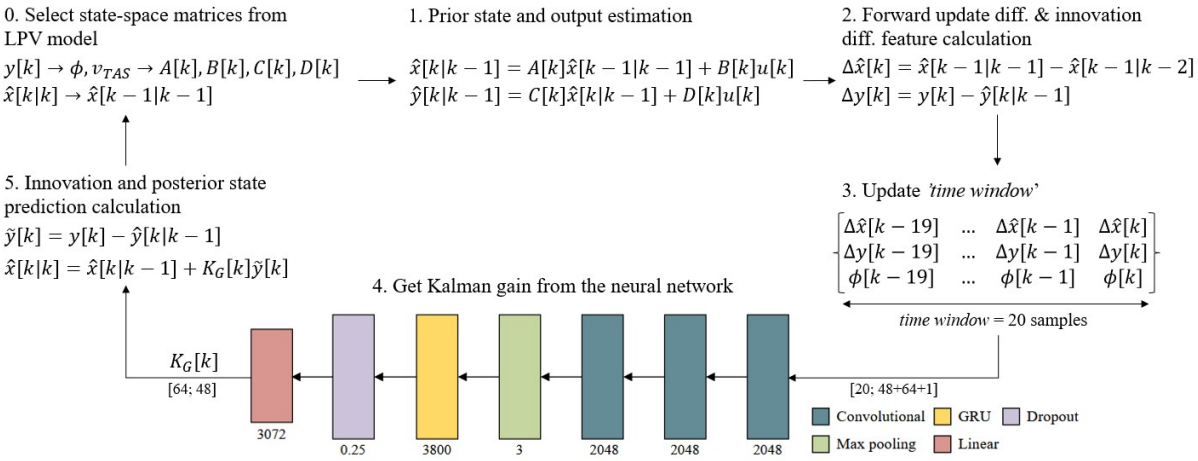

Figure 2: KalmanNet pipeline

and biases of the neural network. The validation set is used for testing the performance of the network during training on new data samples. The purpose of this is to monitor the stability of the training, to detect overfitting, and to fine tune hyperparameters. (Overfitting is the phenomenon when during training the network is no longer capable of getting lower loss values while maintaining its generalization capabilities and starts to memorize the training data, thus reaching smaller loss values on the training set, but greater and greater losses on previously unseen datapoints.) The test dataset is only used after the network is fully trained to obtain final performance metrics. The training, validation, and test datasets are generated using the high-fidelity nonlinear Simulink model of the T-Flex.

The training dataset has four different trajectories that are generated with the help of the baseline controller [4]. These four trajectories are the following:

- the oval-shaped 'horserace' track,
- an '8-shaped' track,
- a trajectory where the controller only receives roll angle (ϕ_{ref}) reference signals,
- a trajectory where the controller receives altitude (h_{ref}) and velocity (V_{ref}) reference signals.

These four trajectories are created with the intention to cover as many possible real-life flight conditions as we can in order to enhance the generalization capabilites of the neural networks. Also, to create rich datasets, while having realistic flight conditions, randomized wind gust and turbulence disturbances are used, together with Gaussian sensor noise, based on the flight test results of the T-Flex [1] for each dataset. The other purpose of applying wind loads is to have disturbances that cannot be incorporated into any covariance matrix. The trajectories of the training dataset are split into eight, 96-second long batches. The sampling time is set to 5 ms, which results in 19200-sample long training batches. For training, a single batch is randomly selected from the eight in each epoch. Validation and testing are conducted using only a trajectory where the aircraft follows the '8-shaped' track. The initial velocity is set to 42 m/s in all cases. The possible range of airspeed changes is between 39 m/s and 51 m/s, for the roll angle between 0° and 45°. The barometric altitude can change between 780 m and 820 m.

Architecture	Linear RNN	Convolutional RNN
Learning rate	$3.2 \cdot 10^{-6}$	$7.5 \cdot 10^{-5}$
Weight decay	$1.5 \cdot 10^{-7}$	$9.5 \cdot 10^{-5}$

Table 1: Hyperparameters

5.3 Training details

The quality of training and the performance of the neural network is influenced to a great extent by hyperparameters (e.g. learning rate, weight decay/L2 regularization factor). The hyperparameters of the neural network, contrary to the weights and biases (simply called parameters) of the network, can not be learned during training using the gradient-based optimization algorithm. The values of the hyperparameters have to be specified before the start of the training procedure via hyperparameter tuning algorithms. (During training, it is possible though to fine tune the hyperparameters using the validation metrics.) For the two neural network architectures, these are set with a custom-made hyperparameter optimization algorithm based on *RayTune*. The hyperparameter optimization has 20 runs, each lasting for 25 epochs. The hyperparameter optimization uses the same pipeline and same training and validation datasets as standard training runs. The optimized hyperparameters are presented in Table 1.

The gradient-based optimizer algorithm is ADAM for both architectures. To avoid overfitting, weight decay is used. The prediction error is calculated with Root Mean Squared Error (RMSE) function. However, although the linearized aircraft model is a stable system, the poles of the system are relatively close to the unstable region. So, a stability criterion is added to the loss function. It is possible to describe the complex system of the aircraft model joined with the Kalman filter with an error system $e[k+1] = (A[k] - K_G[k]C[k])e[k]$, where $K_G[k]$ is the Kalman gain, $e[k] = x[k] - \hat{x}[k|k]$ is the state prediction difference at time step k. If the state transition matrix of the error system $(A[k] - K_G[k]C[k])$ has any unstable poles, then the whole system is unstable. Hence, the RMSE loss is extended with the distance of the error system poles from the boundary of stability if it is greater than zero, thus making the loss value larger if the computed Kalman gain results in an unstable error system. This is especially useful for the convergence of the training.

An error metric is defined in decibels as $\text{RMSE}_{\text{dB}} = 10 \lg(\text{RMSE})$, for the sake of convenience during plotting, because the freshly initialized network tends to produce greater errors. This metric is solely used for evaluation and plotting. For optimizing the network weights, the standard RMSE loss value is used during backpropagation.

It is important to mention that the performance of the linear RNN architecture proved to be more stable than the convolutional RNN, which tends to get stuck in local optima. So, to overcome this issue, a reduction of the learning rate during training (called learning rate scheduling) is necessary in that case. The threshold is set at -21 dB – according to the decibel-based error metric – and the reduction factor is 0.05. The new learning rate is calculated as $\text{lr}^{\text{new}} = \text{factor} \cdot \text{lr}^{\text{old}}$.

6 Results

The performance of the different methods and architectures are evaluated on the 96-second long test dataset, where the aircraft follows the '8-shaped' track with wind and turbulence disturbances present. Since the main purpose of the state estimator design is to observe the states describing the flexible dynamics, only the results for these states are presented. The data with the nonlinear label is the ground truth. These show the real behaviour of the flexible states of the nonlinear model. The data having the EKF and the KNet labels are the results of the state estimation provided by the LPV-based EKF and the KalmanNet architectures respectively.

6.1 LPV-based EKF

The results of the LPV-EKF state predictions are shown in Figure 3. This method is capable of accurately predicting most of the states. However, it has a tendency to provide estimations with heavier noise in the case of the derivative states. This excess of noise is because the flight trajectories contain disturbances caused by wind and turbulence. These disturbances do not exert their effect in the form of additive noise like the observation noise or model noise. Also their statistical characteristics cannot be incorporated into neither the model nor the observation noise model, since wind conditions are not known accurately before flight and they tend to change as well. Apart from the effects of wind, minor errors occur during turning manoeuvres in state lag_1. The reason is that the LPV model is still just an approximation of the real, nonlinear system thus it is incapable of providing completely similar behaviour as the nonlinear model. However, these inaccuracies are inside the error tolerance for this problem.

6.2 KalmanNet

6.2.1 Neural network with linear layers

First, the slightly modified original KalmanNet architecture of [5] – which uses linear layers with the GRU – is trained and evaluated. The training run had 300 epochs. Using an Nvidia Tesla V100 GPU with 32GBs of RAM, the whole procedure took 25 hours. The summary of the training is presented in Figure 4 (*left*). The decibel-based metric is used for the plotting.

The trained model is evaluated on the same dataset as the LPV-based EKF. The results are shown in Figure 5. It can be seen that the performance of the linear RNN architecture is comparable to the EKF. In the case of the derivative states it even manages to proved more accurate predictions for $\dot{U}_f1$, $\dot{U}_f2$, and $\dot{U}_f4$. However, the estimation of $\dot{U}_f3$ and $\dot{U}_f5$ still proves to be challenging. Also in the prediction of lag_1 a similar error is present as in the EKF. Only here it is between 40 s – 60 s, where the aircraft conducts a climb from 782 m to 803 m with heavier acceleration.

6.2.2 Neural network with convolutional layers

Second, the proposed network architecture with convolutional layers is implemented, trained and evaluated. In this case, the training had 100 epochs that took 15 hours to complete using the V100 GPU. The 1D convolution expects a time series as an input, and the 20-sample long time window is used, which equals to 0.1 s trajectory with the 5 ms sampling time. Unfortunately, it was not possible to use a larger window size, as we ran out of GPU memory after a few training epochs (and training with CPU was not feasible, due to its slow execution speed). The training graph is shown in Figure 4 (*right*) with loss values in decibels.

Testing is done with the same dataset as in the previous approaches. The results are presented in Fig-

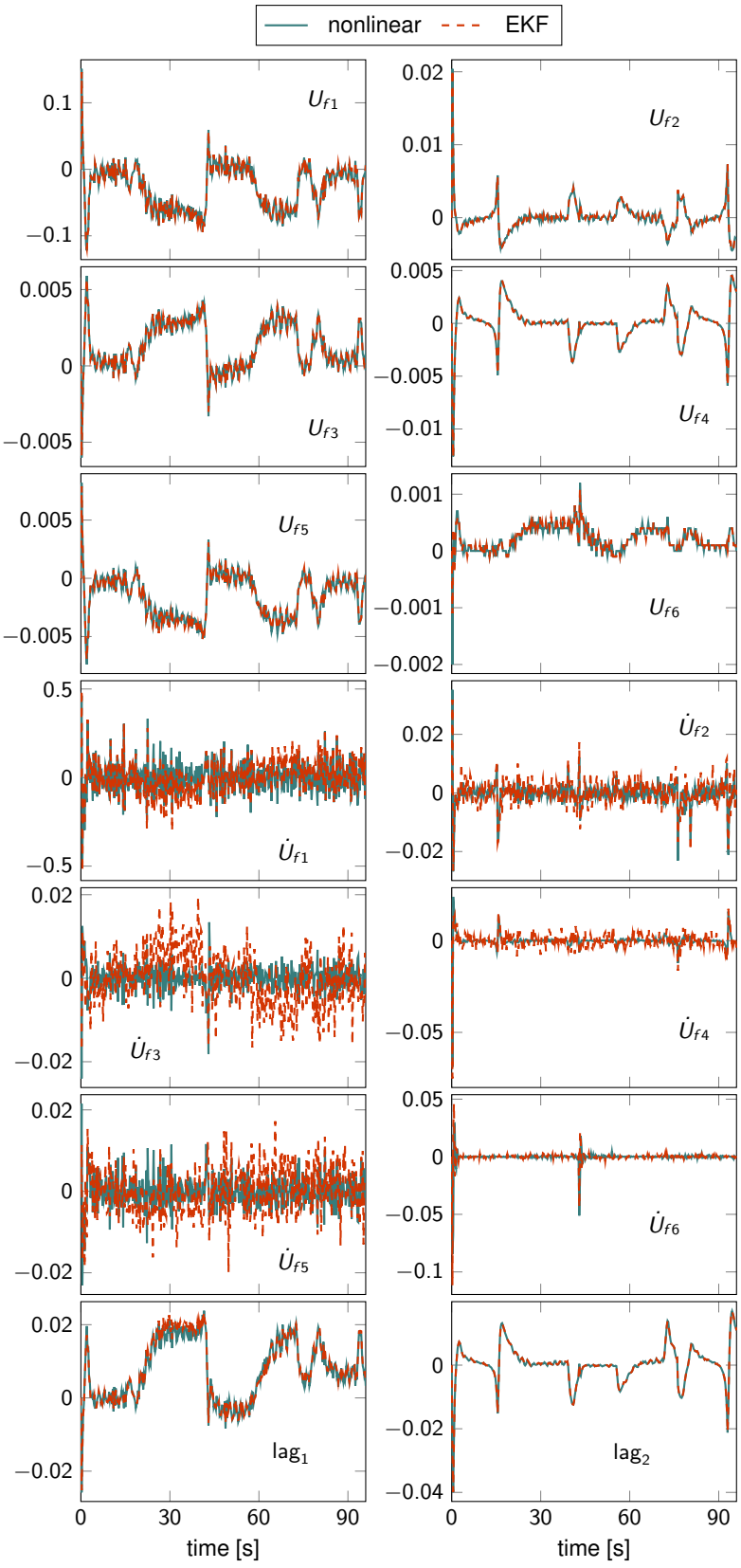

Figure 3: LPV-based EKF results

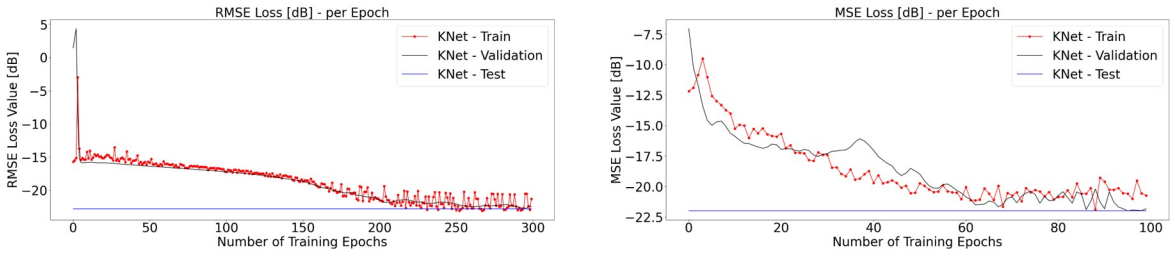

Figure 4: Linear (*left*) and convolutional (*right*) RNN architecture training graphs

Architecture	LPV-EKF	Linear RNN	Convolutional RNN
Total RMSE	0.0120	0.0053	0.0066
U_{fn} RMSE	$5.91 \cdot 10^{-4}$	$8.82 \cdot 10^{-4}$	$9.79 \cdot 10^{-4}$
$\dot{U}_{fn}$ RMSE	0.0183	0.0080	0.0100
lag_n RMSE	$6.95 \cdot 10^{-4}$	0.0014	0.0014

Table 2: Prediction errors

ure 6. The convolutional RNN architecture manages to give very similar predictions as the LPV-based filter and the linear RNN. In the case of lag_1 there are still larger errors present however, the prediction error in the 40 s – 60 s interval is smaller than for linear RNN architecture (Figure 5). Unfortunately, neither this architecture is capable of providing perfect estimations for the derivative states.

6.3 RMSE metric-based comparison

In Table 2 the prediction errors are presented in the RMSE metric used throughout the training of the neural networks. Based on this metric, the performance of the two learning-based approaches are better than the LPV-based EKF: having only half the total error value. When looking at the three different state groups (modal coordinates, derivatives of the modal coordinates, aerodynamic lag states) the followings can be observed. When estimating the modal coordinates, all three architectures provide similar performance. For the lag states the LPV-based EKF provides better performance than either architecture. In the case of the derivative states, where the accurate estimation of the states is a rather challenging task because of the significant level of disturbance, the learning-based methods fare much better than the LPV-based EKF. However, as discussed during the time domain analysis of results, this does not mean that any of the architectures can completely negate the effects of the disturbances. Comparing the two neural network architecture the linear RNN has slightly better performance as metrics concerned. However, it took significantly less time to train the convolutional RNN while it also used less GPU memory and the size of trained model is smaller than for the linear RNN.

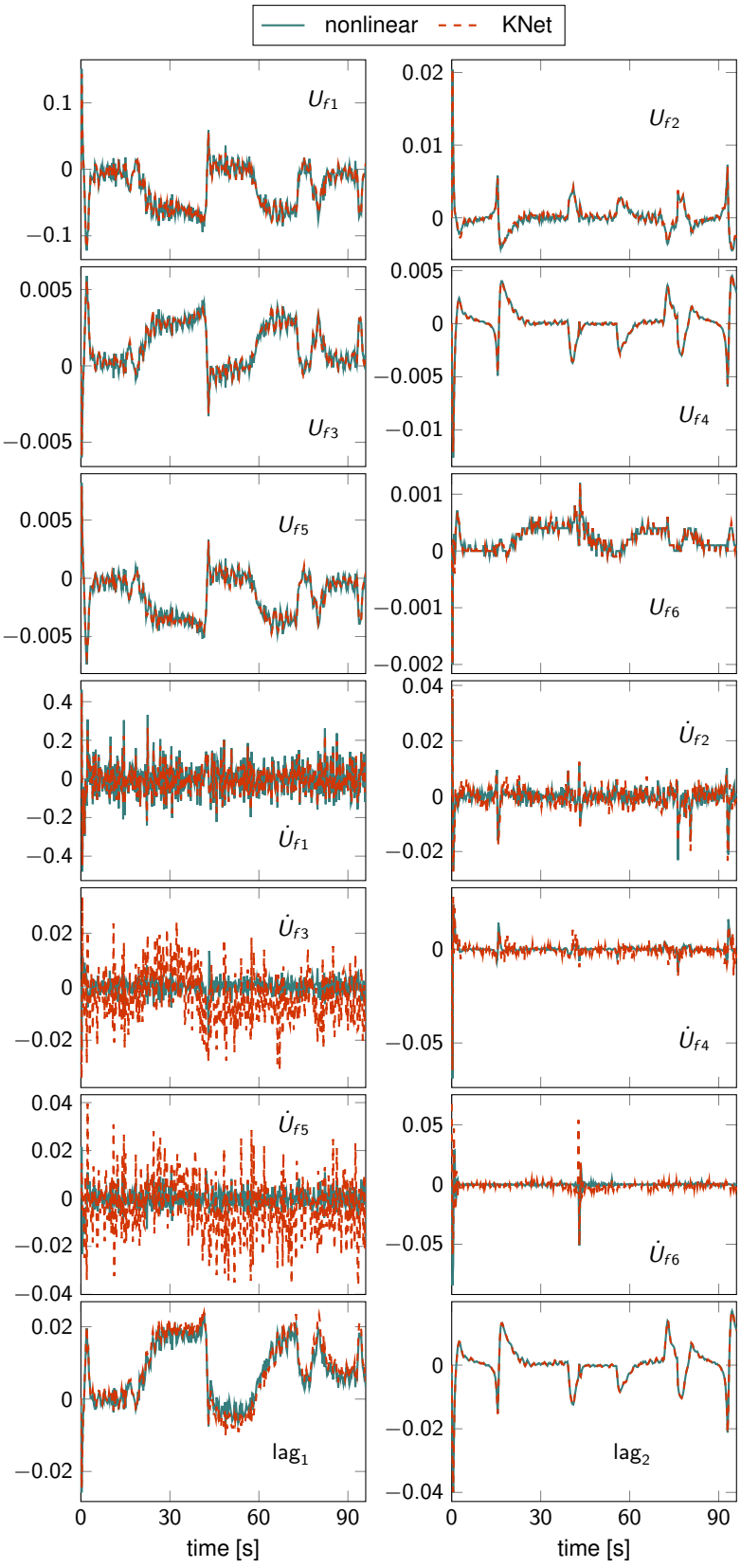

Figure 5: KalmanNet results with linear RNN architecture

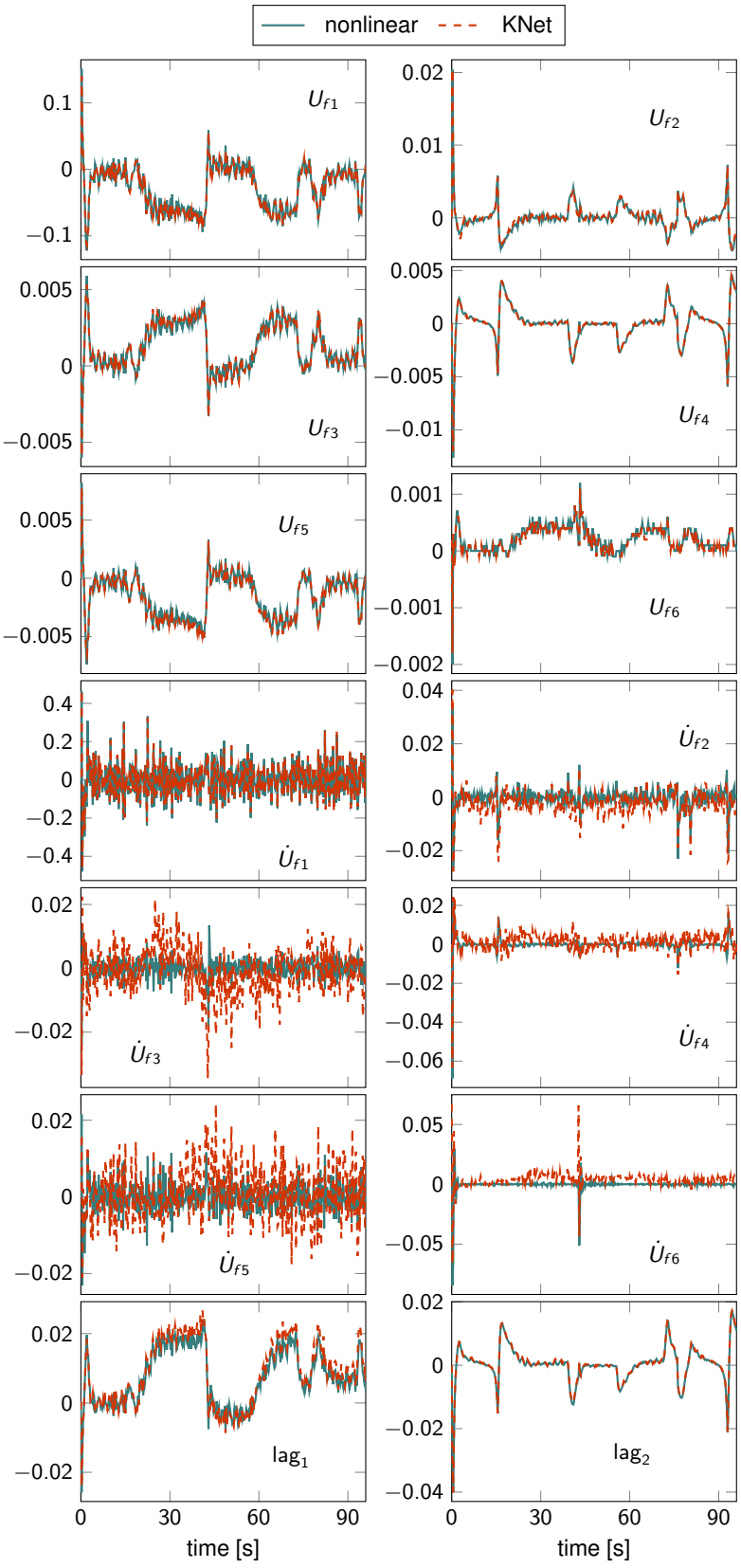

Figure 6: KalmanNet results with convolutional RNN architecture

7 Conclusion

For the flexible state estimation of the T-Flex a model-based approchs and a machine learning-based approach are proposed. The model-based approach uses an LPV-based EKF, while the machine learning-based solution utilizes the KalmanNet architecture with two different neural network setups. The behavior of the different architectures are explained. The data generation an the training procedure for the learning-based solutions are described. The results of the state estimation approaches using different techniques are presented and their performances are compared.

8 Bibliography

[1] Julius Bartasevicius, Sebastian J Koeberle, Daniel Teubl, Christian Roessler, and Mirko Hornung. Flight testing of 65kg t-flex subscale demonstrator. In *32nd Congress of the International Council of the Aeronautical Sciences*, pages 1–16. ICAS, 2021.

[2] Leandro R Lustosa, Ilya Kolmanovsky, Carlos ES Cesnik, and Fabio Vetrano. Aided inertial estimation of wing shape. *Journal of Guidance, Control, and Dynamics*, 44(2):210–219, 2021.

[3] Yasser M Meddaikar, Johannes Dillinger, Thomas Klimmek, Wolf Krueger, Matthias Wuestenhagen, Thiemo M Kier, Andreas Hermanutz, Mirko Hornung, Vladyslav Rozov, Christian Breitsamter, et al. Aircraft aeroservoelastic modelling of the flexop unmanned flying demonstrator. In *AIAA scitech 2019 forum*, page 1815, 2019.

[4] Daniel Ossmann, Tamas Luspay, and Balint Vanek. Baseline flight control system design for an unmanned flutter demonstrator. In *2019 IEEE Aerospace Conference*, pages 1–10. IEEE, 2019.

[5] Guy Revach, Nir Shlezinger, Xiaoyong Ni, Adria Lopez Escoriza, Ruud JG Van Sloun, and Yonina C Eldar. Kalmannet: Neural network aided kalman filtering for partially known dynamics. *IEEE Transactions on Signal Processing*, 70:1532–1547, 2022.

[6] Béla Takarics and Bálint Vanek. Robust control design for the flexop demonstrator aircraft via tensor product models. *Asian Journal of Control*, 23(3):1290–1300, 2021.

[7] Béla Takarics, Bálint Vanek, Aditya Kotikalpudi, and Peter Seiler. Flight control oriented bottom-up nonlinear modeling of aeroelastic vehicles. In *2018 IEEE aerospace conference*, pages 1–10. IEEE, 2018.

[8] Matthias Wüstenhagen, Thiemo Kier, Yasser M Meddaikar, Manuel Pusch, Daniel Ossmann, and Andreas Hermanutz. Aeroservoelastic modeling and analysis of a highly flexible flutter demonstrator. In *2018 atmospheric flight mechanics conference*, page 3150, 2018.

[9] Ming Zhang, Mingming Zhang, Yiming Chen, and Mingyang Li. Imu data processing for inertial aided navigation: A recurrent neural network based approach. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3992–3998. IEEE, 2021.