Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master’s Thesis

Robust Distributed Pervasive Simulations in the Presence of Delays, Jitter and Losses

Benedikt Mehler

Course of Study:	M.Sc. Softwaretechnik
Examiner:	Prof. Dr. phil. nat. Christian Becker
Supervisor:	Johannes Kässinger, M.Sc., Dr. rer. nat. Frank Dürr
Commenced:	June 1, 2022
Completed:	December 29, 2022

Abstract

Pervasive simulation envisions to apply computationally expensive simulations in everyday scenarios on resource-constrained mobile devices. The PerSiVal project aims to realize a bio-mechanical simulation of the human arm as augmented reality application. To enable the execution of such computationally expensive simulations on resource-constrained devices surrogate models are applied. Nonetheless, the execution of the surrogate models can still be challenging. A solution to deal with the constraint resources is the offloading of the surrogate model to a server that is wirelessly connected to the mobile device. Challenges arise due to inevitable delays caused by processing and communication between devices. To counterweight the delays, previous work has applied a second, light-weight surrogate model with lower performance on the mobile device. The goal of this thesis is the design and evaluation of Kalman filter-based approaches for fusion in the presence of delays, jitter and losses. This work contributes an improved strategy for surrogate model derivation, improved and light-weight surrogate models for the muscle simulation, a distribution model for reproducible analysis and evaluation of distributed algorithms, an improved variant of the fusion algorithm from previous work and the design and evaluation of Kalman filter-based solutions to the fusion problem. While being computationally more demanding, the Kalman filter-based approaches show a significant advantage in dealing with delays, jitter and losses in the evaluated scenarios. Especially the constant velocity Kalman filter with input augmentation fusion works best in the tested scenarios. Conclusively, the Kalman filter is a powerful framework that has been successfully applied in the context of distributed pervasive simulations for continuous problems.

Contents

List of Figures

List of Tables

Acronyms

Anconeus *Musculus Anconeus*. 4

AR Augmented Reality. 1, 19, 77

Biceps *Musculus Biceps Brachii*. 4

Brachialis *Musculus Brachialis*. 4

Brachioradialis *Musculus Brachioradialis*. 4

BTTF Back to the Future. 63, 78

CPU Central Processing Unit. 18

CV Cross-Validation. 8

DoF Degree of Freedom. 5

FEM Finite Element Method. 1, 77

FIFO First-In-First-Out. 40

HMI Human Machine Interaction. 1

HPC High-Performance Computing. 3

HPS High-Performance Surrogate. 20, 77

IoT Internet of Things. 1

KF_{CA} Kalman filter with constant acceleration tracking. 67

KF_{CAF} Kalman filter with constant acceleration tracking and fusion. 67

KF_{CAU} Kalman filter with constant acceleration tracking and input augmentation fusion. 61

KF_{CP} Kalman filter with constant position tracking. 67

KF_{CPF} Kalman filter with constant position tracking and fusion. 67

KF_{CPU} Kalman filter with constant position tracking and input augmentation fusion. 59

KF_{CV} Kalman filter with constant velocity tracking. 67

KF_{CVF} Kalman filter with constant velocity tracking and fusion. 67

KF_{CVU} Kalman filter with constant velocity tracking and input augmentation fusion. 61

k-NN k-Nearest-Neighbor. 12

KF Kalman Filter. 1, 15, 20, 45, 65, 78

LPS	Low-Performance Surrogate. 20, 77
MA	Measurement Augmentation. 64, 78
MAE	Mean Absolute Error. 15
MAG	Mean Absolute Gradient. 24
MAXAG	Maximum Absolute Gradient. 24
MED	Mean Euclidean Distance. 23
ML	Machine Learning. 1, 6, 77
mocap	Motion Capture. 21
MSDE	Mean Squared Deformation Error. 26
MSE	Mean Squared Error. 14
MSG	Mean Squared Gradient. 24
NEES	Normalized Estimation Error Squared. 54
NIS	Normalized Innovation Squared. 54
NN	Neural Network. 1, 77
NPU	Neural Processing Unit. 18
OOSM	Out-of-Sequence Measurement. 63
PerSiVal	Pervasive Simulation and Visualization with Resource- and Time-Constraints. 1
PR	Polynomial Regression. 10
R^2	Coefficient of determination. 15
REL	Right Elbow Lateral. 22
ReLU	Rectified Linear Unit. 13
REM	Right Elbow Medial. 22
RMSE	Root Mean Squared Error. 15
RMSG	Root Mean Squared Gradient. 24
RMSS	Root Mean Squared Slope. 24
RSB	Right Shoulder Back. 22
RSF	Right Shoulder Front. 22
RSS	Residual Sum of Squares. 15
RTT	Round-Trip Time. 66
RWL	Right Wrist Lateral. 22
RWM	Right Wrist Medial. 22

SAD	Squared Activation Deviation. 26
SGD	Stochastic Gradient Descent. 11
SMAE	Squared Mean Activation Error. 26
SSA	Sum of Squared Activations. 25
SSMD	Sum of Squared Mean Deviations. 26
Triceps	*Musculus Triceps Brachii*. 4
TSS	Total Sum of Squares. 15
VR	Virtual Reality. 1

1 Introduction

Pervasive simulation aims to enable simulation technology in everyday scenarios [1]. The usage of novel Human Machine Interaction (HMI) technology, e.g., Augmented Reality (AR) or Virtual Reality (VR) devices, allows a nearly seamless integration in daily tasks and procedures. In the context of Industry 4.0 and Internet of Things (IoT) the business world is highly interested in the realization of smart applications. Especially scenarios where humans interact with virtual or automated systems require real-time capabilities, reliability and accuracy. To realize this vision, energy and time consuming computations need to be executed on resource-constrained mobile devices. Those mobile devices are usually battery-powered and restricted in their computational power and available memory.

The interdisciplinary project PN 7-1 Pervasive Simulation and Visualization with Resource- and Time-Constraints (PerSiVal) of the Cluster of Excellence SimTech [2] aims to implement the real-time simulation of a bio-mechanical model of a human arm in an AR environment. The demonstrator application realizes the following idea. A human is visually tracked. The pose is estimated from the video source and used as an input to simulate the muscular system. Finally, the simulation results are visualized as an overlay on the tracked human. The simulation is based on a biophysical continuum-mechanical Finite Element Method (FEM) model [3]. This computationally expensive approach cannot be deployed on today's mobile devices. In fact, even on modern supercomputers like Hawk most of the computational resources are occupied to simulate the electrical state of one skeletal muscle [1].

In order to realize such expensive simulations on resource-constrained mobile devices, surrogate simulation models are used. Those surrogate models focus on specific aspects of the simulation and therefore only represent a fraction of the original FEM model. The surrogate models are derived with the help of Machine Learning (ML) techniques, e.g., Neural Networks (NNs). Albeit the execution of a NN on a mobile device differs from the original FEM simulation by an order of magnitude, the computation can still be challenging. [1]

In previous work of the PerSiVal project, Hubatscheck [4] presents a distribution approach, where computations are offloaded from the mobile device to a server infrastructure over wireless communication. The mobile device executes a second, less accurate but faster, surrogate model. High quality remote updates are used to improve the local estimates. Issues arise due to the existence of inevitable delays caused by the simulation execution and communication between devices. To account for these delays, the remote values have been weighted depending on their age. Therefore, an older value receives less weight and fades out over time. This approach allows the usage of delayed updates.

This thesis is motivated by the question whether the results of the distributed surrogate models can be fused in a better way. The Kalman Filter (KF) is a prominent algorithm considered to be optimal in state estimation for linear systems in the presence of white Gaussian noise. It can also be applied in prediction, target tracking and sensor fusion. Thus, the goal of this thesis is the design and evaluation of advanced methods based on the KF to fuse local and remote simulation results with robustness against delays, jitter and losses. As a prerequisite, a continuous surrogate model for the pervasive simulation is required. Furthermore, a distribution model is needed to formulate the model assumptions of the underlying system. This allows a reproducible analysis and evaluation of the distributed simulation.

This thesis is structured in eight chapters. Chapter 2 introduces background and related work. The problem statement is given in Chapter 3. In Chapter 4 the derivation of an improved surrogate model is presented. Chapter 5 shows the developed distribution model for deterministic analysis and evaluation of the offloading strategies. An improved variant of [4] and the developed KF-based approaches are described in Chapter 6. Chapter 7 contains the analysis and evaluation of the developed methods. The thesis closes with a conclusion and future work in Chapter 8.

2 Background and Related Work

In this chapter the background and related work of this thesis are presented. Section 2.1 introduces the topic of the continuum-mechanical muscle simulation. The relevant foundations of ML for this thesis are shown in Section 2.2, followed by an introduction to the KF in Section 2.3. This chapter concludes with presentation and discussion of related work in Section 2.4.

2.1 Continuum-Mechanical Muscle Simulation

Usually, Hill-type muscle models are used for fast simulations. These models represent the muscle's mechanical response with simplifying assumptions, e.g., the muscles are one dimensional and reduced to a point mass [3]. However, advanced applications, e.g., computer aided surgery or augmented physiotherapy, require a muscle model with spacial deformations and forces. Röhrle [6] presents an approach based on a continuum mechanical FEM which allows the modeling of realistic muscular forces and geometries. Those kinds of simulations are computationally expensive, even on modern High-Performance Computing (HPC) systems [7, 8]. Obviously, such methods cannot be deployed on today's mobile devices for applications in real-time.

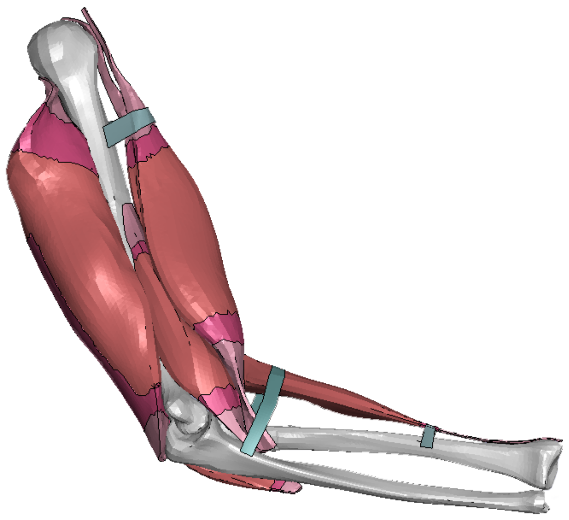


Figure 2.1: Rendering of the continuum-mechanical arm model. [5]

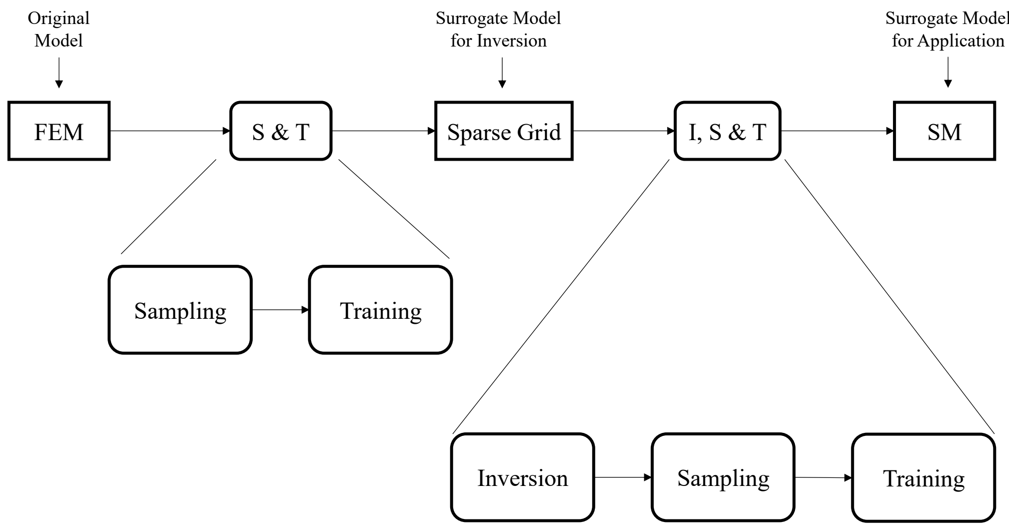


Figure 2.2: Model derivation process for the AR application.

In order to enable the simulation of specific aspects of the continuum-mechanical simulation, Valentin et al. [3] suggest an approach with a special type of surrogate model. In this work sparse grids with hierarchical B-Splines are used to represent a human arm model with two muscles, the Biceps and Triceps. This approach was extended in [5] and contains the five most important actuator muscles for flexion and extension around the elbow joint:

- *Musculus Biceps Brachii* (Biceps)
- *Musculus Brachialis* (Brachialis)
- *Musculus Brachioradialis* (Brachioradialis)
- *Musculus Anconeus* (Anconeus)
- *Musculus Triceps Brachii* (Triceps)

The continuum-mechanical arm model is rendered in Figure 2.1.

The PerSiVal use case is an AR application where a target human is visually tracked. The estimated motion is used as input to the simulated muscular system with activations and deformations which are finally visualized as an overlay in real-time. [5]

The FEM works in a forward way. The muscles are stimulated with so called activations. The conceptual idea of an activation is a kind of neurological control signal from the brain over the nerve system that triggers the muscular system to perform specific tasks, e.g., a coordinated movement. In a physiological sense, it is a bio-electrical signal that leads to specifically distributed contraction of muscle fibers. It is also known as neurological or cognitive drive. In the model the activation $a \in [0, 1]$ describes how much a muscle, as a whole, is stimulated. The simulation performs an iterative relaxation process that converges to an equilibrium. During this process the activations lead to changes in the forces that cause deformations in the elastic material which again alter the current forces and their impact. Those equilibrium states with activations and deformations in form of three-dimensional meshes are saved and used to derive the sparse grid surrogate models. The data points come from a setting with a static pose where the upper arm faces downwards to the ground. In the current approach, there is only one Degree of Freedom (DoF) in the elbow angle θ. [5]

For the PerSiVal AR application it is necessary to solve an inversion problem. Inputs and outputs of the simulation model are transposed. This inversion is not trivial since it has no unique solution. The elbow angles θ can be reached with multiple combinations of activations. The side constraint for the inverse optimization problem is based on the assumption that the human body tries to solve motion tasks as efficient as possible. In addition, it is assumed that the activations correlate with the energy needed to reach a respective elbow angle. The optimization hypothesis expects that minimum activation values yield the most efficient state to achieve a specific angle. In this thesis this approach is called minimum activation strategy. [5] The derivation process is depicted in Figure 2.2.

The sparse grid model for the elbow angle has the form $f_{\text{angle}} : \mathbb{R}^5 \to \mathbb{R}$. The respective sparse grid for the Biceps surface mesh has the form $f_{\text{biceps}} : \mathbb{R}^5 \to \mathbb{R}^{2809 \times 3}$, i.e., the mesh consists of 2809 three-dimensional points. Similar sparse grid models with different mesh size can be obtained by applying the same procedure. In this thesis, the focus is on the Biceps since not all muscle models have been available for this work. [5]

In the original work, a model of the form $f_{\text{move2acts}} : \mathbb{R}^4 \to \mathbb{R}^5$ was proposed for the AR use case. The input features are $\boldsymbol{x} = (\theta, \dot{\theta}, \ddot{\theta}, w)^\top$ with the elbow angle θ, the angular velocity $\dot{\theta}$, the angular acceleration $\ddot{\theta}$ and a weight w placed on the hand. The target features are the activation values in the interval $[0, 1]$ for the five muscles mentioned earlier. The dynamic aspects of the model have been achieved by augmentation methods in the inversion problem. The dynamics $(\dot{\theta}, \ddot{\theta})$ are added from generated elbow trajectories based on a polynomial trajectory model for human elbow motion [9]. The approach assumes that activations and deformations are reasonable for sufficiently slow movements. The weights are simulated with the help of counter-activation, i.e., the antagonistic muscles are activated to a certain degree to emulate additional weight. [5]

2.2 Machine Learning

In this thesis Machine Learning (ML) approaches are applied to create surrogate models for the pervasive simulation. The foundations of the methods used in this work are presented in this section.

Mitchell [10] defines ML as follows:

> "A computer program is said to learn from experience E with respect to some class of tasks T, and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

This abstract definition allows many different types of ML depending on the nature of what the system shall learn (T), which experience or training signal (E) it uses and how the system is evaluated (P) [11].

Prominent types of ML are supervised learning, unsupervised learning and reinforcement learning [12]. A ML task can be described by defining how the system shall process an example. This example is represented by a collection of features, which are typically notated as vector $\boldsymbol{x} \in \mathbb{R}^n$. Each entry x_i of $\boldsymbol{x}$ describes another feature [13].

Supervised learning is a class of approaches where a set of input and target vectors are given, e.g., classification and regression [12]. When the ML algorithm is asked to produce a function $f : \mathbb{R}^n \to \{1, \ldots, k\}$ that assigns one of a set of categories to the feature vector the task is called classification [13]. In contrast, regression defines the task where the desired function outputs a continuous value with the form $f : \mathbb{R}^n \to \mathbb{R}$ [13]. The input features are sometimes also called predictors or independent variables, the outputs can also be named responses or dependent variables [14].

Unsupervised learning aims to learn useful properties from a given dataset containing many features [13], e.g., clustering or dimensionality reduction [12].

The goal in reinforcement learning is to find suitable actions in a given situation by maximizing a reward [12]. Those systems typically interact with an environment, s.t. they receive feedback from the world they are interacting with [13].

In this thesis supervised learning approaches, namely regression, are applied to predict the continuous outputs of the pervasive simulation.

This section gives a brief summary of the most important aspects of ML that are relevant for this work. There exists a huge amount of other types, tasks, models, methods and applications which cannot be stated here. The interested reader might find further information in the books [11–16] or in the documentation of scikit-learn [17] or similar ML frameworks.

2.2.1 Model Fitting and Selection

In regression the result of a ML algorithm can be interpreted as a model that realizes the function $\hat{y} = f(\boldsymbol{x})$ with $f : \mathbb{R}^n \to \mathbb{R}$. This definition can also be extended to the multivariate case where $f : \mathbb{R}^n \to \mathbb{R}^m$. The learning procedure based on the given dataset is sometimes also called fitting. [13]

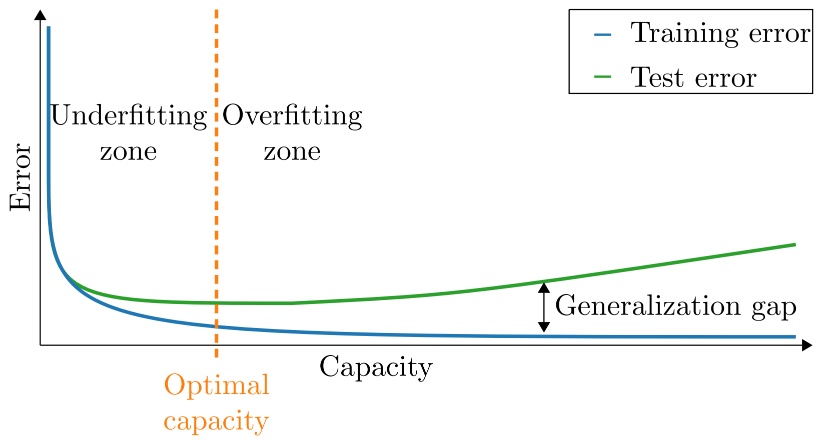


Figure 2.3: Typical relationship between capacity and error. Illustration based on [13].

A common way to represent a dataset is the design matrix where each row contains one example and the columns hold the features. As an example, the features $\boldsymbol{x} \in \mathbb{R}^n$ of a dataset can be represented as the design matrix $\boldsymbol{X} \in \mathbb{R}^{N \times n}$ with N examples. [13]

The central challenge of a ML model is the ability to produce correct results on new, unseen data during training, which is known as generalization. This motivates to split the given dataset in a training set and a test set. The model is solely fitted with data from the training set. Afterwards, its performance is evaluated with the held out test set. With the help of an error measure, the training performance can be evaluated. This is also called training error. Here lies a major difference between an optimization problem and machine learning. The training procedure can be understood as classical optimization problem, while ML also aims to minimize the generalization or test error. These two measures correspond to two major challenges in ML: underfitting and overfitting. While underfitting describes the issue that the model does not achieve sufficient performance on the training dataset, overfitting occurs when the model performs poorly on the test data, i.e., the difference between training and test error is too large. The capacity of a model describes “its ability to fit a wide variety of functions” [13]. On the one hand, a model with low capacity tends to underfit the training data. On the other hand, a model with high capacity can overfit if it memorizes details from the training set instead of learning the “true” relation. The relationship between capacity and error is visualized in Figure 2.3. The capacity of a model also correlates with its degrees of freedom. [12, 13]

The issue of under- and overfitting motivates the concept of regularization. According to Goodfellow et al. [13], regularization can be interpreted as any modification to a ML algorithm that intends to improve the generalization but not the training error. As an example, for parametric models a common form of regularization is the parameter norm penalty. The basic idea is to formulate an additional cost on high parameter values. This approach sanctions unnecessary high values and therefore restricts the degrees of freedom of a model. In practice, many regularization methods come along with a set of hyperparameters for tuning. [13]

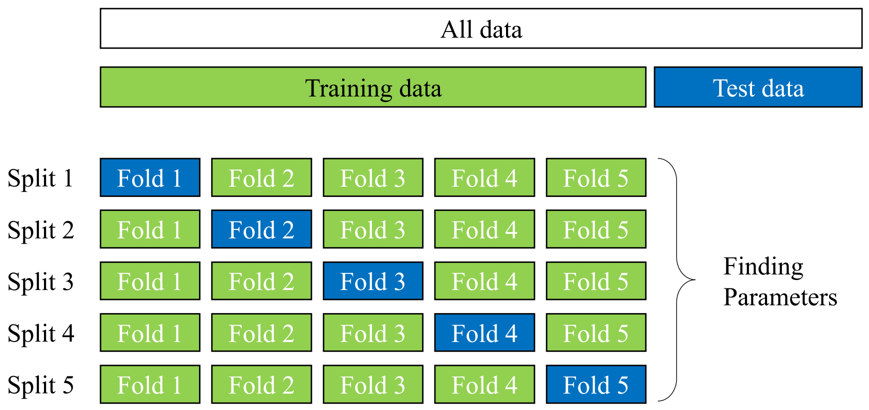


Figure 2.4: k-fold CV schematic. Green indicates a split of the dataset used for training, blue for testing. Illustration based on [18].

With all these concepts in mind, the question arises which model should be chosen. Not only the model but also the optimization method and regularization along with all their hyperparameters have to be selected and compared with respect to a chosen performance measure. The vast amount of degrees of freedom in this procedure are a challenge for real world applications. In addition, for many issues there are no better solutions available than trial and error. [13]

For the selection and tuning problem an extended approach is required. Iterative tuning of hyperparameters to find the best model can introduce overfitting on the test dataset. Therefore, a third, independent set, the validation dataset, is introduced. Finally, there are a training dataset for the model fitting, a validation dataset for the model selection and hyperparameter tuning and a test set for the final performance evaluation. [12]

The availability of data is an issue in many applications. For good model performance it is appreciated to use as much data for training as possible. Conversely, if the validation set is small, it might give an unreliable estimate of the generalization performance. To account for this target conflict, Cross-Validation (CV) can be used. A prominent representative for CV is the k-fold CV. The basic idea is to partition the data into k groups. In k runs, $k - 1$ groups are used as the training set, while the remaining group acts as validation set. Therefore, a proportion of $(k - 1)/k$ is used as training data in each run. The validation results are accumulated, e.g., by averaging over the runs. The approach is visualized in Figure 2.4. Here, the trade-off lies between data usage and the increased computation time by a factor of k. The special case $k = N$, with N total number of data points, is called leave-one-out method. [12]

5- or 10-fold CV is considered as good compromise between computational cost and estimation accuracy in the literature [14].

The entire CV based workflow is depicted in Figure 2.5. At first, the dataset is split in a training and a test set. The training data is applied with CV to find the best parameters and models. The most promising approaches are retrained on the whole dataset with the best hyperparameters found through the CV routine. Finally, the models are evaluated on the held out test set to estimate the generalization performance. [18]

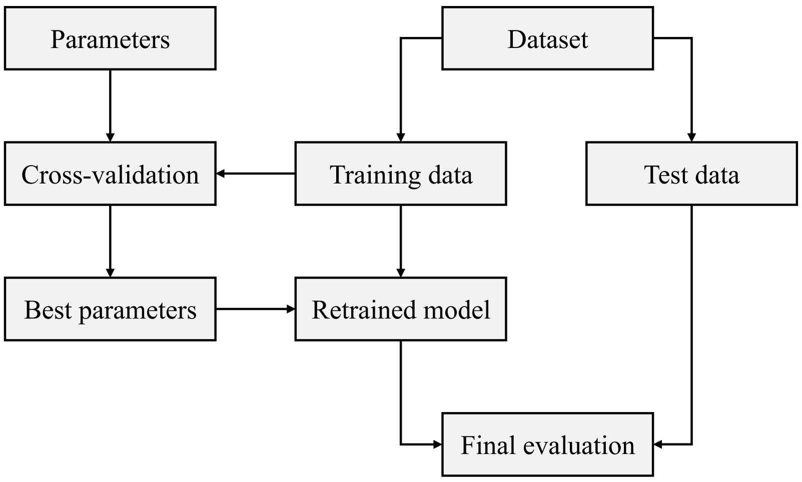


Figure 2.5: CV training workflow. Illustration based on [18].

Using CV to explore all combinations of hyperparameters is exponential in the number of parameters and therefore critical for practical application. There exist more advanced approaches to identify hyperparameters and select models, e.g., based on Baysian model comparison, which are out of scope for this thesis. [12]

2.2.2 Linear Regression

Linear regression is a parametric model that is linear regarding its parameters. Therefore, it has simple analytical properties. Nevertheless, nonlinear properties with respect to the input variables can be modeled with the help of basis functions. This section is mainly cited from Bishop [12] unless stated otherwise.

The most simple form, which is also linear regarding the features, is simply known as linear regression:

$$\hat{y} = f(\boldsymbol{x}, \boldsymbol{w}) = w_0 + w_1 x_1 + \cdots + w_n x_n \tag{2.1}$$

with the response $\hat{y}$, the input features $x_1, \ldots, x_n$ and the weight parameters $w_0, \ldots, w_n$.

A more useful class of models can be created as linear combination of a fixed set of nonlinear functions of the input variables, known as basis functions:

$$f(\boldsymbol{x}, \boldsymbol{w}) = w_0 + \sum_{i=1}^{M-1} w_i \phi_i(\boldsymbol{x}) \tag{2.2}$$

where $\boldsymbol{x} = (x_1, \ldots, x_n)^\top$ is the vector of input features, $\boldsymbol{w} = (w_0, \ldots, w_{M-1})^\top$ is the vector of weights, $\phi_i(\boldsymbol{x})$ are the basis functions, M is the total number of parameters in the model and w_0 is a fixed offset also called bias.

For convenience, a dummy basis function $\phi_0(\boldsymbol{x}) = 1$ can be defined, s.t.:

$$f(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{w}^\top \phi(\boldsymbol{x}) \tag{2.3}$$

with $\phi = (\phi_0, ..., \phi_{M-1})^\top$.

Polynomial Regression (PR) or linear regression with polynomial features of the form $\phi_j(x) = x^j$ can be used to account for nonlinear relations. The costs are in $O(n^p)$ with input dimension n and the order of the polynomial p. Even though this is a power law growth rather than exponential it is still expensive and often limits the usability. The interest lies in the powers of the features as well as their interaction terms. For example, given two features the transformed polynomial features of degree 2 have the form [19]:

$$\boldsymbol{x} = (x_1, x_2)^\top \tag{2.4}$$

$$\phi_{\text{poly2}}(\boldsymbol{x}) = \left(1, x_1, x_2, x_1^2, x_1 x_2, x_2^2\right)^\top \tag{2.5}$$

Different other choices as basis functions are possible, e.g., Gaussian, Sigmoid, tanh or Fourier.

To fit the linear model a cost or error function has to be chosen for minimization. A common choice is the least squares or sum-of-squared error approach with N examples in the dataset:

$$E_d(\boldsymbol{w}) = \frac{1}{2} \sum_{i=1}^{N} \left(y_i - \boldsymbol{w}^\top \phi(\boldsymbol{x}_i)\right)^2 \tag{2.6}$$

For a linear model this is equivalent to the maximum likelihood approach under conditional Gaussian noise. This formulation has an analytical solution, the so called normal equation for the least squares problem. It is acquired by setting the gradient of the error function to zero:

$$\boldsymbol{w} = \left(\boldsymbol{\Phi}^\top \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^\top \boldsymbol{y} \tag{2.7}$$

with $\boldsymbol{\Phi}$ as the $N \times M$ design matrix with the elements $\Phi_{ij} = \phi_j(\boldsymbol{x}_i)$, s.t.:

$$\boldsymbol{\Phi} = \begin{bmatrix} \phi_0(\boldsymbol{x}_1) & \phi_1(\boldsymbol{x}_1) & \dots & \phi_{M-1}(\boldsymbol{x}_1) \\ \phi_0(\boldsymbol{x}_2) & \phi_1(\boldsymbol{x}_2) & \dots & \phi_{M-1}(\boldsymbol{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\boldsymbol{x}_N) & \phi_1(\boldsymbol{x}_N) & \dots & \phi_{M-1}(\boldsymbol{x}_N) \end{bmatrix} \tag{2.8}$$

One method of regularization for linear regression is called parameter shrinkage or weight decay. Therefore, a weight-dependent regularization term E_w with the regularization coefficient λ is added to the data-dependent term E_d. One simple approach is the usage of the sum of squares of the weight vector:

$$E(\boldsymbol{w}) = E_d(\boldsymbol{w}) + \lambda E_w(\boldsymbol{w}) \tag{2.9}$$

$$= \frac{1}{2} \sum_{i=1}^{N} \left(y_i - \boldsymbol{w}^\top \phi(\boldsymbol{x}_i)\right)^2 + \frac{\lambda}{2} \boldsymbol{w}^\top \boldsymbol{w} \tag{2.10}$$

It has the benefit that the solution can be found in a closed form as before with:

$$w = \left(\lambda \boldsymbol{I} + \boldsymbol{\Phi}^\top \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^\top \boldsymbol{y} \tag{2.11}$$

A more general variant introduces the regularization term with a variable exponent q:

$$E_W(\boldsymbol{w}) = \frac{1}{2} \sum_{i=0}^{M-1} |w_i|^q \tag{2.12}$$

In that way q controls how strong high parameters are penalized in a nonlinear fashion. The special case $q = 1$ is called lasso regression, $q = 2$ is also known as quadratic penalizer or ridge regression. With this regularization approach, the challenge to determine the optimal model capacity is shifted from the selection of basis functions to finding optimal parameters for λ and q.

Without the closed form solution, the optimal parameters can be found with an optimization algorithm. A prominent method is the Stochastic Gradient Descent (SGD) that iteratively updates the model weights by following the steepest gradient of the error function:

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E \tag{2.13}$$

with the iteration number τ and learning rate η. Thus, it can also be used for sequential training in small batches of data. This is especially helpful if the dataset for training is huge.

An important property for this work is the estimation of the error variance σ^2, which can be estimated from residuals ϵ. With the assumption that the error ϵ is zero-mean Gaussian noise, i.e., normally distributed with mean 0 and variance σ^2 or $\epsilon \sim \mathcal{N}(0, \sigma^2)$, σ^2 can be calculated with:

$$y = f(\boldsymbol{x}, \boldsymbol{w}) + \epsilon \tag{2.14}$$

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - \boldsymbol{w}^\top \phi(\boldsymbol{x}_i)\right)^2 \tag{2.15}$$

In the multivariate case with the assumption of multivariate Gaussian noise with $\epsilon \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma})$ the error covariance matrix $\boldsymbol{\Sigma}$ can be computed from the residual vectors ϵ_n:

$$\epsilon_n = y_n - \boldsymbol{W}^\top \phi(\boldsymbol{x}_n) \tag{2.16}$$

$$\boldsymbol{\Sigma} = \frac{1}{N} \sum_{n=1}^{N} \epsilon_n \epsilon_n^\top \tag{2.17}$$

with weight matrix $\boldsymbol{W}$.

2.2.3 Nearest Neighbor Regression

The k-Nearest-Neighbor (k-NN) is a non-parametric regression approach that uses the examples from the training set closest to x in the input space by a defined distance metric [12, 14]:

$$\hat{\boldsymbol{y}} = \frac{1}{k} \sum_{\boldsymbol{x}_i \in N_k(\boldsymbol{x})} \boldsymbol{y}_i \tag{2.18}$$

where $N_k(\boldsymbol{x})$ models the neighborhood of the input vector $\boldsymbol{x}$ that returns the k closest points to $\boldsymbol{x}$ in the training set. $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ represent the input and target vectors of the k neighboring data points. A common choice for the distance metric is the Euclidean distance.

Instead of averaging, an alternative weighting approach can be applied. The KNeighborsRegressor from the scikit-learn package [20] allows the target vectors $\boldsymbol{y}_i$ to be weighted by the inverse of the distance, which yields interpolation behavior instead of bounded zones of averages. This approach can be formulated in a more general way:

$$\hat{\boldsymbol{y}} = \sum_{\boldsymbol{x}_i \in N_k(\boldsymbol{x})} w(\boldsymbol{x}_i, \boldsymbol{x}) \boldsymbol{y}_i \tag{2.19}$$

where the function $w(\boldsymbol{x}_i, \boldsymbol{x})$ gives the respective weight for $\boldsymbol{y}_i$ with $\sum_{\boldsymbol{x}_i \in N_k(\boldsymbol{x})} w(\boldsymbol{x}_i, \boldsymbol{x}) = 1$.

According to Hastie et al. [14], k-NN works well for low dimensional problems. Especially in regression performance degrades significantly with high-dimensional features due to the curse of dimensionality.

2.2.4 Neural Networks

Deep feedforward networks, feedforward neural networks or multilayer perceptrons, short NN, are mathematical models inspired by neuroscience. This section is mainly cited from Goodfellow et al. unless stated otherwise [13].

The approach is called network because it is composed of many different functions organized in layers. The size of those layers is called width and represents how many units, also known as neurons or nodes, are contained. The whole network consist of an input layer, an output layer and a number of hidden layers in between. In a dense network or fully-connected network each neuron is connected to all outputs of the previous layer. A typical NN architecture is depicted in Figure 2.6. Commonly, a node calculates a weighted linear combination with an additional bias followed by a nonlinear activation function ϕ:

$$h = \phi(\boldsymbol{w}^\top \boldsymbol{x} + b) \tag{2.20}$$

with h the output of the node, $\boldsymbol{x}$ the input vector, $\boldsymbol{w}$ the linear weight vector $\boldsymbol{w}$ and b the bias. For regression problems it is common to use a linear output layer. In this specific case, the activation functions is the identity $\phi(\boldsymbol{x}) = \boldsymbol{x}$. A NN without hidden layers and a linear output layer is equivalent to linear regression.

The universal approximation theorem is fundamental for NNs. According to Goodfellow et al. [13], any Borel measurable function from one finite-dimensional space to another can be approximated with an arbitrary non-zero error with the help of a feedworward network with linear output layer

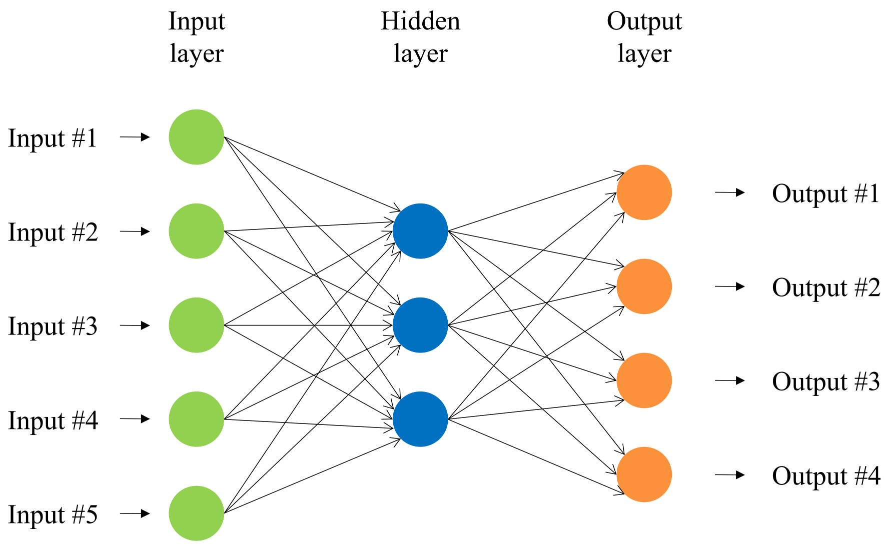


Figure 2.6: Exemplary NN architecture. The presented network has one input, hidden and output layer. It consists of 5 input features, 3 hidden nodes and 4 output nodes. The layers are densely connected, i.e., each node is connected to all nodes of the previous layer. Illustration based on [21].

which contains at least one hidden layer with a "squashing" activation function. Informally, this means that even a single hidden layer NN can approximate any function given enough nodes. Nevertheless, it gives no guarantee whether the training algorithm is capable of finding the parameters or whether it might overfit.

Goodfellow et al. [13] state, the Rectified Linear Unit (ReLU) of the form $\phi(x) = \max(0, x)$ is an excellent default choice as activation function for hidden units. Strictly speaking ReLU is not differentiable at $x = 0$, which should be problematic for a gradient based optimizer. In practice, the implementations just decide for one of the one-sided derivatives which can be heuristically justified since the numerical optimization is subject to numerical error anyway. There are a lot of other options and it is a very active field of research. Popular options are e.g., Sigmoid, tanh, leaky ReLU, parametric ReLU and many more. The universal approximation theorem is also applicable for NNs with ReLU as activation function, which is used in this thesis.

As stated in [13], early stopping is probably the most commonly used form of regularization due to its effectiveness and simplicity. Furthermore, it implicitly reduces the computational cost of the training. The parameter norm penalties, also known as weight decay, which have already been discussed in Section 2.2.2 for the linear regression, can also be applied for NNs. Among others, dataset augmentation, the addition of artificial noise or the usage of dropout layers are also popular regularization approaches.

NNs are trained with a method called back-propagation. The basic idea is to represent the whole NN as computational graph of functions. Beginning with the cost function to optimize, the gradient is propagated backwards through the network by recursive application of the chain rule. In this way, the error gradient flows backwards through the network where the respective parameters are adapted accordingly.

Goodfellow et al. [13] note, there is no consensus whether there is an optimal optimizer that should be chosen for NN training. The most popular choices are SGD, SGD with momentum, RMSProp, RMSProp with momentum, AdaDelta and Adam. The authors [13] suggest to use the algorithm the user is the most familiar with to ease hyperparameter tuning.

2.2.5 Feature Scaling

According to [19], many learning algorithms benefit from standardizing of the dataset. Some approaches even specify concrete requirements about the dataset, e.g., the data must be normally distributed with zero mean and unit variance.

Standardization is a preprocessing that removes the mean and scales the data s.t. the mean is zero and the variance one. This is achieved by:

$$z = \frac{x - \bar{x}}{\sigma} \tag{2.21}$$

with the transformed feature z, the raw feature x, the mean $\bar{x}$ of x and standard deviation σ of x.

Another choice of preprocessing is the linear mapping to a specific range. For example the min-max-scaling transforms the feature to the interval $[0, 1]$ with:

$$z = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{2.22}$$

The target range in the interval $[-1, 1]$ is also a common choice, which can be achieved by the max-abs-scaling. This scaling does not center the data but scales all values by the maximum absolute value of the dataset:

$$z = \frac{x}{\max(|x|)} \tag{2.23}$$

There exist various other approaches, e.g., non-linear transforms, transforms robust to outliers, normalization, that are out of scope of this thesis. [19]

2.2.6 Metrics

This section introduces the metrics used for evaluation and model comparison in this work. As introduced in Section 2.2.2, the Mean Squared Error (MSE) is a common cost function applied in regression problems. It can be motivated from the maximum likelihood for linear models in the presence of Gaussian noise [12]. Thus, it is an important metric for model performance:

$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{2.24}$$

with the ground truth y, the prediction $\hat{y}$ and dataset size N. A lower value indicates a better model.

For convenience, the Root Mean Squared Error (RMSE) can be used, which represents the MSE but on the same scale as the target variable [11, 12]:

$$\mathrm{RMSE}(y, \hat{y}) = \sqrt{\mathrm{MSE}(y, \hat{y})} \tag{2.25}$$

Another metric for regression model performance is the Mean Absolute Error (MAE). It represents the average absolute error, which can be intuitively interpreted since it is in the same scale as the target variable. It is defined as follows [22]:

$$\mathrm{MAE}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i| \tag{2.26}$$

A lower value indicates a better model.

The Coefficient of determination ($\mathbf{R^2}$) yields another well interpretable measure [11]:

$$\begin{aligned} \mathrm{R}^2(y, \hat{y}) &= 1 - \frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2} && (2.27) \\ &= 1 - \frac{\mathrm{RSS}}{\mathrm{TSS}} && (2.28) \end{aligned}$$

with the empirical mean of the response $\bar{y} = \sum_{i=1}^{N} y_i$, the Residual Sum of Squares (RSS) $\mathrm{RSS} = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$ and the Total Sum of Squares (TSS) $\mathrm{TSS} = \sum_{i=1}^{N} (y_i - \bar{y})^2$. The $\mathbf{R^2}$ represents the ratio of the variance in the predictions and the variance of the constant prediction $\hat{y}_i = \bar{y}$ [11]. This can be interpreted as proportion of explained variance. The maximum is 1 and a higher value indicates a better fit. Furthermore, a constant model that always outputs the expected value gets a score of 0. Typically the $\mathbf{R^2}$ score is in the interval $[0, 1]$ but can become negative for arbitrary worse models. [22]

2.3 Kalman Filter

Since its publication by Kalman in 1960 the Kalman Filter (KF) has become a popular algorithm for various tasks. The KF was originally developed to solve practical problems in communication and control of statistical nature, e.g., prediction of random signals, separation of random signals from noise or signal detection in the presence of noise. The approach defines and solves the so called Wiener problem from the "state" perspective. [23]

The key idea of the KF is to improve the estimate of a desired signal by exploiting model knowledge about the system under observation. Thus, a random signal is filtered by blending a model-based prediction and a measurement in order to acquire an optimal estimate, which is better than either the prediction or the measurement alone. Under the linear Gaussian assumption, the KF is an optimal estimator in the least square sense. [24]

Symbol	Description
$\boldsymbol{x}_k$	State vector at time k
$\hat{\boldsymbol{x}}_k$	Estimation of state vector $\boldsymbol{x}$ at time k
$\hat{\boldsymbol{x}}_{k\|i}$	Estimation of state vector $\boldsymbol{x}$ at time k based on time i
$\boldsymbol{P}_k$	State covariance matrix at time k
$\boldsymbol{P}_{k\|i}$	State covariance matrix at time k based on time i
$\boldsymbol{A}$	State transition function or state transition matrix
$\boldsymbol{u}_k$	Control input at time k
$\boldsymbol{B}$	Control input model or control function
z_k	Measurement vector at time k
$\boldsymbol{C}$	Measurement function or measurement matrix
y_k	Innovation at time k
$\boldsymbol{S}_k$	System uncertainty or innovation covariance matrix at time k
$\boldsymbol{K}_k$	Kalman gain at time k
$\mathbf{w}_k$	Process noise at time k
$\boldsymbol{Q}$	Process noise covariance matrix
$\mathbf{v}_k$	Measurement noise at time k
$\boldsymbol{R}$	Measurement covariance matrix

Table 2.1: Kalman filter symbols.

The Kalman Filter Algorithm The KF can be expressed as algorithm as follows. It is a two-step recursive estimator for linear discrete-time state space models in the presence of white Gaussian noise. The two steps are the *prediction* and the *measurement update* or *correction*. [24]

There exist many notation variants. The notation used in this thesis is derived from [24–27]. The symbols are described in Table 2.1.

Given a linear system in discrete-time state space form:

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}_k\boldsymbol{u}_k + \mathbf{w}_k \tag{2.29}$$

$$z_k = \boldsymbol{C}\boldsymbol{x}_k + \mathbf{v}_k \tag{2.30}$$

The KF algorithm consists of the following steps:
Initialization:

$\hat{\boldsymbol{x}}_0$ and $\boldsymbol{P}_0$ are initialized.

Prediction:

$$\hat{\boldsymbol{x}}_{k+1|k} = \boldsymbol{A}\hat{\boldsymbol{x}}_k + \boldsymbol{B}\boldsymbol{u}_k \tag{2.31}$$

$$\boldsymbol{P}_{k+1|k} = \boldsymbol{A}\boldsymbol{P}_k\boldsymbol{A}^\top + \boldsymbol{Q} \tag{2.32}$$

Correction:

$$\boldsymbol{y}_{k+1} = \boldsymbol{z}_{k+1} - \boldsymbol{C}\hat{\boldsymbol{x}}_{k+1|k} \tag{2.33}$$

$$\boldsymbol{S}_{k+1} = \boldsymbol{C}\boldsymbol{P}_{k+1|k}\boldsymbol{C}^\top + \boldsymbol{R} \tag{2.34}$$

$$\boldsymbol{K}_{k+1} = \boldsymbol{P}_{k+1|k}\boldsymbol{C}^\top\boldsymbol{S}_{k+1}^{-1} \tag{2.35}$$

$$\hat{\boldsymbol{x}}_{k+1} = \hat{\boldsymbol{x}}_{k+1|k} + \boldsymbol{K}_{k+1}\boldsymbol{y}_{k+1} \tag{2.36}$$

$$\boldsymbol{P}_{k+1} = (\boldsymbol{I} - \boldsymbol{K}_{k+1}\boldsymbol{C})\boldsymbol{P}_{k+1|k} \tag{2.37}$$

The process and measurement noise random variables $\mathbf{w}_k$ and $\mathbf{v}_k$ are uncorrelated and have zero mean with the covariance matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ [27]:

$$\mathbb{E}[\mathbf{w}_k\mathbf{w}_l^\top] = \begin{cases} \boldsymbol{Q} & k = l \\ 0 & \text{otherwise} \end{cases} \tag{2.38}$$

$$\mathbb{E}[\mathbf{v}_k\mathbf{v}_l^\top] = \begin{cases} \boldsymbol{R} & k = l \\ 0 & \text{otherwise} \end{cases} \tag{2.39}$$

$$\mathbb{E}[\mathbf{w}_k\mathbf{v}_l^\top] = \begin{cases} 0 & \forall k, l \end{cases} \tag{2.40}$$

with $\boldsymbol{Q}$, $\boldsymbol{R}$ symmetric and positive semi-definite.

In the one dimensional case, the Kalman gain describes a ratio in the interval $[0, 1]$ that controls how much the innovation $\boldsymbol{y}$ is considered in the update step. The ratio is calculated based on the relation between prediction and measurement uncertainty. The same principle is applied in the multivariate case, but here the Kalman gain is a matrix. [24]

The state covariance matrix $\boldsymbol{P}_k$ is a measure for the theoretical performance. The KF filter assumes that everything given to it is true. Thus, if the assumptions do not hold, $\boldsymbol{P}_k$ does not necessarily indicate the true estimation error. [24]

The system model can be derived from discretized differential equations [26].

One important concept is observability [24]: States in the filter can be observed, hidden or unobservable variables. An observed state can be directly measured in $\boldsymbol{z}$, i.e., the respective entry in the measurement matrix $\boldsymbol{C}$ is one. Hidden states are not contained in $\boldsymbol{z}$ but can be derived in the KF due to the defined dependencies in the state transition matrix $\boldsymbol{A}$. In contrast, unobservable variables cannot be estimated by the filter. The observability of a given system can be analyzed with the observability criterion from Kalman [26].

There exist extensions and alternative approaches for non-linear systems and non-Gaussian distributions, e.g., the Extended Kalman Filter, die Unscented Kalman Filter or the Particle Filter, which are beyond the scope of this thesis [24].

2.4 Related Work

The main scope of this thesis is the context of distributed pervasive simulations. The first class of related work, about continuum-mechanical muscle simulation, has already been introduced in Section 2.1. Other classes of literature, frequently borrowed from other domains, are introduced in their respective chapters. The basics for ML methods applied in this thesis are presented in Section 2.2. Related work of probabilistic modeling of communication technologies is introduced in Section 5.2. The foundations of the KF and its related work regarding prediction, tracking and delay handling are presented in Section 2.3 and Chapter 6.

A relevant class of related work is ML on mobile devices. There exist different sub-classes of approaches, among others, the reduction of the model complexity, the usage of hardware acceleration or the distributed deployment. Lane et al. [28] present the DeepX software accelerator, which can be applied to optimize the internal structure of a trained NN. An example for the usage of hardware acceleration in form of a Neural Processing Unit (NPU) is presented by Tan and Cao [29] where for each NN layer it is decided whether it shall be executed on the NPU or Central Processing Unit (CPU). Teerapittayanon et al. [30] present an approach in the context of image classification where a NN architecture is split s.t. parts can be executed on the server while others remain on the mobile device. Basically, if the desired accuracy level is not reached at an intermediate output, it is forwarded to the next higher level for further improvement. A survey of deployment methods for deep learning on mobile devices is presented in [31]. The named approaches are generic and do not target mobile simulations in real-time.

Regarding simulation on mobile devices, Dibak et al. [32] present the application of the reduced basis method for offloading computations in the context of heat simulation in materials. This work focuses on another type of surrogate model for a different use case.

Another class is the previous work in the PerSiVal project. Kässinger et al. [1] and Hornischer [33] apply a genetic algorithm to identify a reduced set of representative muscle coordinates which can be interpolated to the whole muscle surface mesh with ML models. This approach is orthogonal to this thesis and can be combined. Another work by Belz and Mehler [34] applies forecasting methods to temporarily suspend the execution of the simulation surrogate model on the mobile device. In this thesis the KF is explored as alternative forecasting model.

The closest work to this thesis is the work of Hubatscheck [4]. He describes an offloading architecture, where a lower quality surrogate model is used on the mobile device to improve the simulation accuracy in the presence of lower frequent and delayed high quality updates. In this thesis the proposed continued update strategy from [4] is improved, which allows a more efficient implementation. The performance is compared to the KF-based fusion strategies. This work presents a KF fusion strategy that is equivalent to the continued update strategy under specific parametrization. Additionally, the KF variant provides an error estimate. Furthermore, the KF variant can be extended to a higher order filter which outperforms the continued update strategy in the presence of delays, jitter and losses in the evaluated scenarios.

3 System Model and Problem Statement

This chapter introduces the system model for this thesis in Section 3.1 and presents the problem statement in Section 3.2.

3.1 System Model

In the context of the Augmented Reality (AR) continuum-mechanical muscle simulation the system can be modeled as depicted in Figure 3.1. A human target is tracked with a sensor by the mobile device. This mobile device performs the visualization and is wirelessly connected via an access point to a server infrastructure. A very small latency is required for the AR visualization. The server infrastructure can be an edge or cloud setup, while an edge deployment is more likely due to desired low latencies. The mobile device is battery-powered and limited in its computational

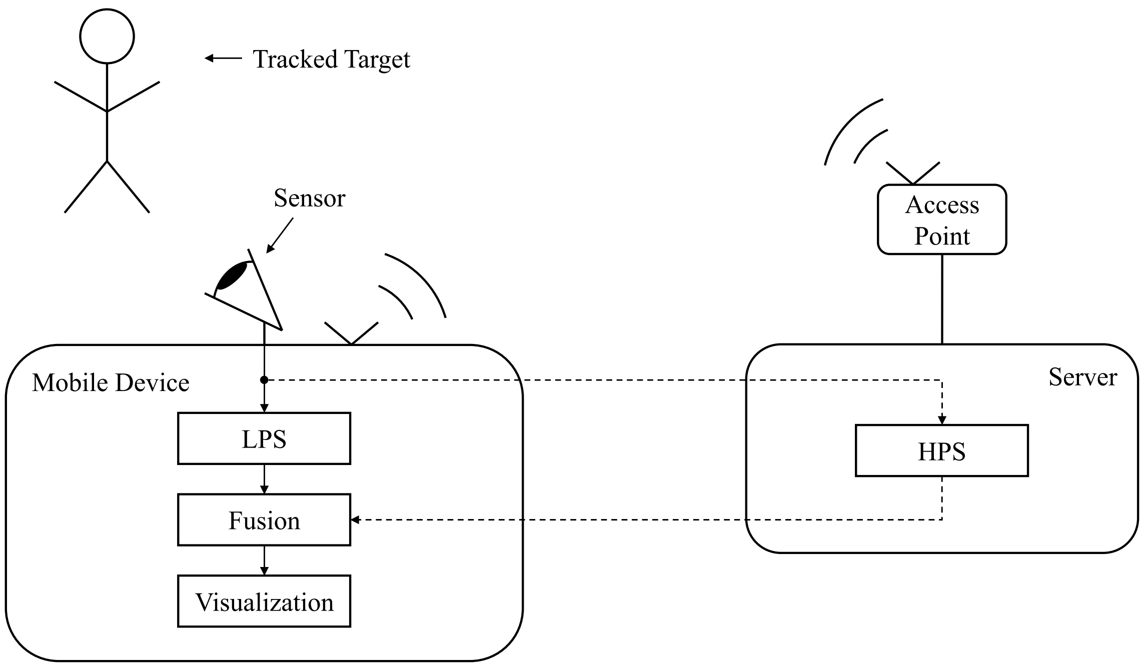


Figure 3.1: System model overview. The main components of the system are the mobile device and the simulation server. The mobile device senses the environment and runs the target application, in this example the bio-mechanical simulation of a human arm with AR visualization. The simulation is implemented with the distributed HPS and LPS models and a fusion strategy. The mobile device is wirelessly connected to an access point that establishes the connection to the simulation server.

resources and available memory. Thus, the goal is to achieve an acceptable simulation performance, with high accuracy and frame rate, while minimizing the energy consumption and utilization of the mobile device. Consequently, reducing the amount of communication is also of interest. When the cost of the simulation is particularly high, offloading techniques can make the difference whether the application can be realized at all. The simulation is implemented with the help of surrogate models. In previous work, Hubatscheck [4] has presented a distribution approach with two models, a Low-Performance Surrogate (LPS) and a High-Performance Surrogate (HPS). The HPS is deployed on the remote server, while the LPS is applied on the mobile device to counterweight inevitable delays caused by the processing and transmissions between devices. In order to combine the local and remote simulation results a fusion strategy is required. The wireless communication is assumed to be a high-bandwidth state-of-the-art technology, e.g., Wi-Fi according to IEEE 802.11. Nevertheless, delays with jitter and losses occur in practice, especially if the network load is high or high bit error rates are present, which need to be dealt with.

3.2 Problem Statement

This thesis aims to design and evaluate advanced fusion methods with robustness against delays, jitter and losses.

As a prerequisite, a continuous model for visualization is required. In a preceding research project, Belz and Mehler [34] found issues in the given surrogate model. The model suffers from discontinuities in the activations and deformations over generated smooth trajectories. Furthermore, the model is limited in the available elbow angles in the range of $\approx [60, 100]\,°$. Those shortcomings lead to the research for an improved model, which is addressed in Chapter 4. The original FEM, on which the simulation is based, works in the opposite direction of the AR surrogate model. Therefore, an improved solution for the inversion problem is required. In addition, to quantify the smoothness of the solution, a smoothness metric is needed. Furthermore, realistic movement data of a human arm is recorded and preprocessed as prerequisite for the evaluation.

A second problem is the analysis and evaluation of the distributed algorithms. In practice, concurrency, imperfect clocks and disturbances make real distributed systems difficult to analyze and evaluate. Therefore, a distribution model is developed in Chapter 5 that allows reproducible emulations of the distributed system under specified model assumptions. This enables the deterministic analysis and evaluation of the advanced fusion methods.

The main problem of this thesis is the design of advanced fusion approaches with increased robustness in the presence of delays, jitter and losses. The prominent Kalman Filter (KF) and its possible applications in the distributed pervasive simulation context are explored in Chapter 6. Furthermore, mechanisms to incorporate delayed simulation results are investigated to deal with delays, jitter and losses.

Finally, the developed approaches are analyzed and evaluated in Chapter 7 regarding their robustness against delays, jitter and losses.

4 Continuum-Mechanical Pervasive Simulation

In this chapter the acquisition and preprocessing of movement data is presented in Section 4.1. Section 4.2 introduces a smoothness metric for the model derivation in Section 4.3 which presents a new solution to the surrogate inversion problem. The new light-weight surrogate models are trained and tuned in Section 4.4 and evaluated in Section 4.5. In Section 4.6 the combined deployment of the activation and deformation surrogate models is analyzed.

4.1 Movement Data Acquisition and Preprocessing

In this section the data acquisition process and necessary preprocessing measures are presented. The evaluation of algorithms for the PerSiVal application requires representative human motion data. To this end, a Motion Capture (mocap) system from Vicon Motion Systems Ltd, UK, is utilized to record realistic arm movements. The setup works with a set of eight Vicon Vantage V8 infrared cameras and optical markers.

For the recording six markers are placed on the experimentees right arm. Each joint, the shoulder, elbow and wrist position is tracked with the help of two markers. The setup is shown in Figure 4.1 and the markers are described in Table 4.1.

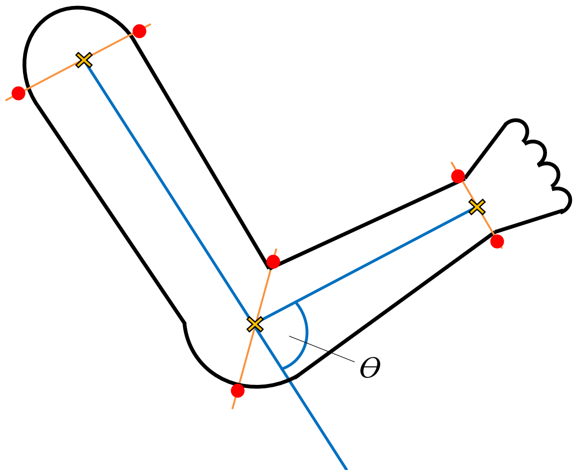


Figure 4.1: Motion capture setup. The red dots indicate the positions of the optical trackers. The estimated joint positions are marked with yellow crosses. The elbow angle θ is defined as exterior angle.

Shortname	Description
RSB	Right Shoulder Back
RSF	Right Shoulder Front
REM	Right Elbow Medial
REL	Right Elbow Lateral
RWM	Right Wrist Medial
RWL	Right Wrist Lateral

Table 4.1: Motion capturing markers.

To acquire realistic movement data two motion tasks are defined:

1. Periodic movement – The experimentee shall perform a repeated arm movement from full flexion to full extension. This task is performed multiple times with different speeds. For reproducibility, the movement is performed to a periodic acoustic signal with a defined frequency. Full flexion and extension shall be reached at the acoustic beats, i.e., the elbow angle frequency is effectively halved. Data with the following frequencies has been recorded: 30, 35, 50, 60, 70, 100 min^{-1}.
2. Arbitrary movement – The experimentee shall perform arbitrary motions.

The movement is recorded with the software Vicon Nexus 2.12 at a frame rate of 100 Hz. The measurement contains a list of three-dimensional coordinates of the tracked markers for each recorded frame. To approximate the position of the joints inside the experimentees body, the deformation of the arm is neglected. Thus, the center point between the markers yields the position of the joint:

$$\boldsymbol{p}_{(a,b)} = \boldsymbol{p}_a + \frac{1}{2}(\boldsymbol{p}_b - \boldsymbol{p}_a) \tag{4.1}$$

$$= \frac{1}{2}(\boldsymbol{p}_a + \boldsymbol{p}_b) \tag{4.2}$$

where $\boldsymbol{p}_i = (x_i, y_i, z_i)^\top$ is the vector with the x, y and z coordinate of a tracked marker.

The elbow angle θ can be estimated with the inner product of the bone vectors [35]:

$$\boldsymbol{b}_\mathrm{f} = \boldsymbol{p}_{(\mathrm{RWM,RWL})} - \boldsymbol{p}_{(\mathrm{REM,REL})} \tag{4.3}$$

$$\boldsymbol{b}_\mathrm{u} = \boldsymbol{p}_{(\mathrm{RSB,RSF})} - \boldsymbol{p}_{(\mathrm{REM,REL})} \tag{4.4}$$

$$\theta = 180\,^\circ - \arccos\left(\frac{\boldsymbol{b}_\mathrm{f}\boldsymbol{b}_\mathrm{u}}{||\boldsymbol{b}_\mathrm{f}||_2||\boldsymbol{b}_\mathrm{u}||_2}\right) \tag{4.5}$$

with the forearm bone vector $\boldsymbol{b}_\mathrm{f}$, the upper arm bone vector $\boldsymbol{b}_\mathrm{u}$. As a side note, these bone vectors do not model the physiological bones but link the estimated joint positions.

One practical problem is that the tracking of the markers is not flawless. Some markers are suffering from dropouts where the mocap system looses track of a point for one or more frames. In the offline processing setting, where future values are known, missing points can be estimated by linear interpolation. The process is exemplarily shown in Figure 4.2.

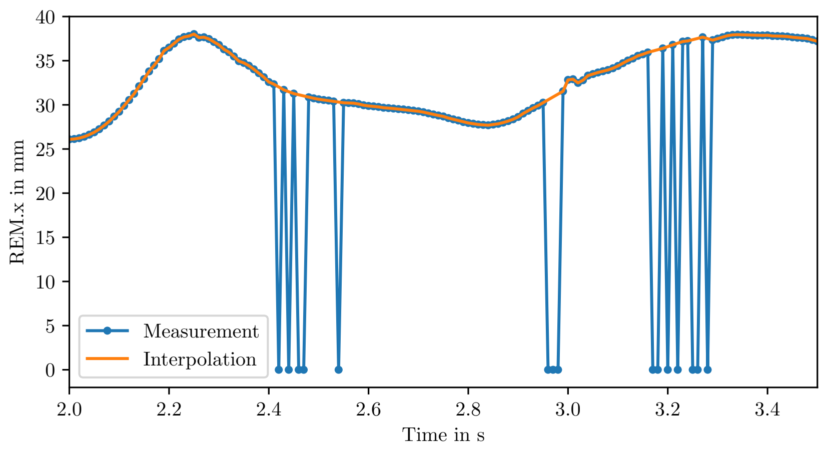


Figure 4.2: Tracking dropouts and interpolation of missing values. Invalid measurement values have the value 0. In the offline processing the missing values are estimated with linear interpolation.

Due to the shortcomings of the given surrogate model, another preprocessing step has to be added before the motion data can be used for evaluation. As mentioned in the problem statement in Section 3.2, the given surrogate model has a limited value range in the interval $\approx [60, 100]\,°$. As it will be presented in Section 4.3, this limitation is not an issue of the inversion process. Instead, its origin is either in the sparse grid surrogate or in the continuum-mechanical model itself. Either way, to account for this, the movement data is rescaled to the valid input range of the surrogate model derived in Section 4.3. A min-max-scaling is applied, i.e., the minimum and maximum value of the elbow angle θ in all measurements are identified and used to linearly map all measured angles θ to the interval $[55, 95]\,°$. In this way, states that the surrogate model cannot estimate are avoided. As an updated FEM or sparse grid model is not in scope of this thesis, this workaround is applied to make the best use of the available data.

4.2 Definition of a Smoothness Metric

As described in the problem statement in Section 3.2, the given surrogate model from previous work suffers from discontinuities. Section 4.3 targets to derive a new surrogate with improved smoothness. Therefore, a measure of smoothness is required to quantitatively evaluate the possible solutions. To the authors knowledge there exists no popular or commonly applied metric for this purpose. In the context of regression problems in ML usually the accuracy is targeted, for example with scores like the MSE, MAE or Mean Euclidean Distance (MED). The problem that will be elaborated in Section 4.3 is different because there is no ground truth available. The challenge is to find a solution that solves the given task where the smoothness is basically a side constraint. Standard deviation and variance are classical measures of dispersion, which capture the distribution of values around the mean. While those metrics model the smoothness to some degree, the activation values are not assumed to be normally distributed over the elbow angles. Thus, these metrics are not well suited for the given task.

Inspired by the method of Horn and Schunck [36], another smoothness metric is derived. The method of Horn and Schunck estimates optical flow with the help of assumed global smoothness in the flow. The smoothness constraint is formulated as cost term in an energy functional to be minimized [37]. There, the smoothness term is designed as integral over the squared gradient of the flow. The gradient of a function represents the local change. Therefore, in the method of Horn and Schunck minimization of that term ideally yields the smoothest solution. In this work, the term is applied as smoothness measure for one-dimensional functions. In the discrete case, the integral can be approximated with a sum. The square is kept as non-linear penalizer to prevent outliers. In order to ease interpretability, the metric is normalized by the number samples. As a result, the Mean Squared Gradient (MSG) is defined as:

$$\mathrm{MSG}(\boldsymbol{y}) = \frac{1}{N}\|\nabla\boldsymbol{y}\|_2^2 \tag{4.6}$$

with the vector $\nabla\boldsymbol{y}$ that represents the sampled gradient values of the function $f(x)$ and the number of samples N. Similar to the RMSE, the Root Mean Squared Gradient (RMSG) can be used for convenience to interpret the result in the scale of the feature:

$$\mathrm{RMSG}(\boldsymbol{y}) = \sqrt{\mathrm{MSG}(\boldsymbol{y})} \tag{4.7}$$

Furthermore, the linear variant, the Mean Absolute Gradient (MAG) can be defined, as it allows a more intuitive interpretation:

$$\mathrm{MAG}(\boldsymbol{y}) = \frac{1}{N}\|\nabla\boldsymbol{y}\|_1 \tag{4.8}$$

Finally, the Maximum Absolute Gradient (MAXAG) can be computed to detect the largest discontinuity in a given feature vector:

$$\mathrm{MAXAG}(\boldsymbol{y}) = \max(\|\nabla\boldsymbol{y}\|_1) \tag{4.9}$$

The MAG represents the average change in a feature vector. Informally, with the quadratic term the MSG also captures the consistency around that average change. For example, if some functions connect two values y_s and y_e monotonically increasing, the MAG has the same value for all functions, because the average change is the same. With the squared term the MSG penalizes outliers in the gradient, e.g., a jump, s.t. the line between both points is the optimal course in the MSG sense. This example is no formal proof but it shall give an intuition what both metrics capture.

In this work, given a vector of values the gradient is estimated with a second order accurate central differences approach using the gradient function from the python numpy package [38].

A follow up term research shows that there already exists a definition of the RMSG or Root Mean Squared Slope (RMSS) which is used as surface roughness measure [39–41]. The RMSS is formally introduced as:

$$\mathrm{RMSS}(f) = \sqrt{\frac{1}{l}\int_0^l \left(\frac{\mathrm{d}f(x)}{\mathrm{d}x}\right)^2 \mathrm{d}x} \tag{4.10}$$

with the sampling length l in the interval $[0, l]$. This supports the choice of the RMSG as reasonable smoothness metric.

4.3 Model Derivation

As described in the problem statement in Section 3.2, the given surrogate model suffers from discontinuities in the predicted activation and deformation values. Therefore, an alternative inversion approach is proposed in this section to improve continuity. The original inverted model has been augmented with additional information to create the input features angular velocity, acceleration and weight. This work focuses on the static relationship between activations, the elbow angle and Biceps coordinates.

The inversion problem is defined as follows: Given a set of sparse grids $\{f_{\text{angle}}, f_{\text{biceps}}\}$ a surrogate model for the desired function $f_{\text{angle2acts}}$ is searched for with $f_{\text{angle}} : \mathbb{R}^5 \rightarrow \mathbb{R}$ that describes the relationship from the five muscular activations to the elbow angle θ and $f_{\text{biceps}} : \mathbb{R}^5 \rightarrow \mathbb{R}^{2809 \times 3}$ that represents the mapping from the activations to the spacial coordinates of the biceps surface and $f_{\text{angle2acts}} : \mathbb{R} \rightarrow \mathbb{R}^5$ the inverse of f_{angle}. Since there is no unique solution to the inversion of $f_{\text{angle2acts}}$ because multiple combinations of activations can lead to the same elbow angle, an additional constraint is needed. The given surrogate models have been trained on a dataset derived from an optimization with subject to minimum activation values. The minimum activation values are motivated by the assumption that the human body tries to solve motion tasks as efficient as possible. Under the hypothesis that the activations are correlated to the energy consumption, the minimization of activations should yield good results.

The inversion problem is formulated as optimization problem with a cost function in analogy to [13]. Let the error function $E_{\text{angle}}(\boldsymbol{a})$ be the squared angle error:

$$\boldsymbol{a}^* = \arg\min_{\boldsymbol{a}} E(\boldsymbol{a}) \tag{4.11}$$

$$E_{\text{angle}}(\boldsymbol{a}) = (f_{\text{angle}}(\boldsymbol{a}) - \theta_t)^2 \tag{4.12}$$

with θ_t the desired target angle and the vector of activations $\boldsymbol{a} \in \mathbb{R}^5$. $\boldsymbol{a}^*$ denotes the vector of activations that minimizes the cost function.

Furthermore, constraints are formulated as separate cost term E_c and added with the help of a Lagrange multiplier λ_c [12]:

$$E(\boldsymbol{a}) = E_{\text{angle}}(\boldsymbol{a}) + \sum_{c=1}^{C} \lambda_c E_c \tag{4.13}$$

where C is the number of additional error terms. Various error terms have been designed during this thesis:

- Sum of Squared Activations (SSA) – The SSA cost term aims to minimize the activations and therefore represents the approach from already existing work. In [5] the sum of activations is used. The squared penalty is also a common choice for cost functions. The quadratic term penalizes activations non-linearly, i.e., it puts an over-proportional weight on higher activation values. Thus, the quadratic penalizer is chosen due to its stronger tendency to avoid high activation values. Similarly to linear regression regularization, other exponents can also be investigated, which is out of scope for this thesis.

$$E_{\text{SSA}}(\boldsymbol{a}) = \|\boldsymbol{a}\|_2^2 \tag{4.14}$$

- Squared Mean Activation Error (SMAE) – The core idea of the SMAE is to define a target value for the mean activation $\bar{a}_\mathrm{t}$ and penalize the squared deviation to the actual mean. This term models the effect that a specific state is not reached in the most efficient way but with a defined average level of activations. This approach could be used to model the tension and relaxation for a given static elbow angle.

$$E_\mathrm{SMAE}(\boldsymbol{a}) = \left(\left(\frac{1}{5} \sum_{i=1}^{5} a_i \right) - \bar{a}_\mathrm{t} \right)^2 \tag{4.15}$$

- Sum of Squared Mean Deviations (SSMD) – The SSMD is motivated by a physiological effect called co-activation. Simplified, it describes the pattern that a motion task is usually not executed by a single muscle. As an example, flexion is very unlikely solely the effect of a contracted Biceps but rather a joint activity with the other flexors, e.g., Brachialis and Brachioradialis. Mathematically, this can be described with the squared deviation to the activation mean. While the SSA aims for low activation values, the SSMD tries to achieve similar activations of all muscles. Unfortunately, this implementation is rather naïve since it is likely that not all muscles are equally important for specific tasks. Without further assumptions about the muscular system, this term is rather a proof of concept than a solution. The operation $\overset{*}{-}$ denotes an element-wise subtraction between a scalar s and a vector $\boldsymbol{x}$, s. t. $s - \boldsymbol{x} = (s - x_1, \ldots, s - x_n)^\top$.

$$E_\mathrm{SSMD}(\boldsymbol{a}) = \left\| \left(\left(\frac{1}{5} \sum_{i=1}^{5} a_i \right) \overset{*}{-} \boldsymbol{a} \right) \right\|_2^2 \tag{4.16}$$

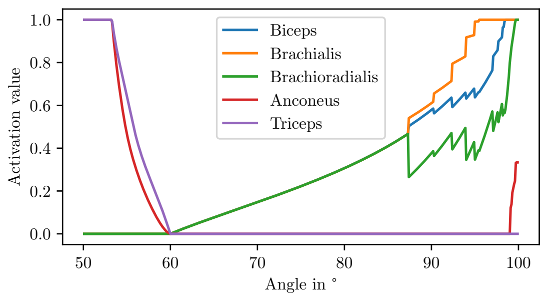
- Squared Activation Deviation (SAD) – The SAD can be used to specifically minimize the deviation to a vector of target activation values $\boldsymbol{a}_\mathrm{t}$.

$$E_\mathrm{SAD}(\boldsymbol{a}) = \|\boldsymbol{a} - \boldsymbol{a}_\mathrm{t}\|_2^2 \tag{4.17}$$

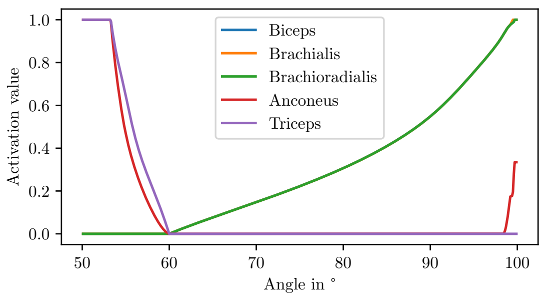
- Mean Squared Deformation Error (MSDE) – The MSDE targets to minimize the deviation to a target surface deformation $\boldsymbol{c}_\mathrm{t}$ of the Biceps. In this way, activation values are acquired that lead to a muscle deformation closest to a target muscle form. In this representation f_biceps maps to one flattened vector of coordinates $f_\mathrm{biceps} : \mathbb{R} \longrightarrow \mathbb{R}^N$ with $N = 2809 \cdot 3$.

$$E_\mathrm{MSDE}(\boldsymbol{a}) = \frac{1}{N} \|f_\mathrm{biceps}(\boldsymbol{a}) - \boldsymbol{c}_\mathrm{t}\|_2^2 \tag{4.18}$$

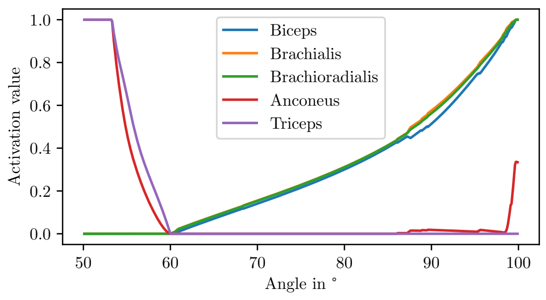
The training of surrogate models requires a dataset that represents a solution to the inversion problem. An iterative optimization approach is applied to generate this training dataset. The process starts with no muscular activation $\boldsymbol{a} = (0, 0, 0, 0, 0)^\top$. With the sparse grid f_angle the rest angle can be determined at $\theta \approx 60\,°$. Therefore, a greedy search for activations is performed that leads to a change in the elbow angle with a step size of $\Delta\theta = 0.1\,°$ in both directions. Each iteration is initialized with the activations from the previous cycle. This approach aims to find a trajectory through the elbow angles that minimizes discontinuities and avoids faraway local minima. To solve the minimization problem, the bound L-BFGS-B optimizer from the python scipy package is applied with numerical gradient estimation.



(a) Result of the minimum activation strategy.



(b) Result of the nearest activation strategy.



(c) Result of the minimum deformation strategy.

Figure 4.3: Comparison of strategies to solve the inversion problem.

The following cost function implements the minimum activation strategy, which was proposed in earlier work [5]:

$$E_{\text{minact}}(\boldsymbol{a}) = E_{\text{angle}}(\boldsymbol{a}) + \lambda_{\text{SSA}} E_{\text{SSA}}(\boldsymbol{a}) \tag{4.19}$$

with $\lambda_{\text{SSA}} = 1\text{e}{-3}$. The results are shown in Figure 4.3a. The plot shows strong discontinuities, especially for high angles θ.

Targeting to find a smoother solution yields the nearest activation strategy. It aims to find activations that are closest to the activations from the previous iteration $\boldsymbol{a}_{\text{t}} = \boldsymbol{a}^*_{\tau-1}$:

$$E_{\text{nearestact}}(\boldsymbol{a}) = E_{\text{angle}}(\boldsymbol{a}) + \lambda_{\text{SAD}} E_{\text{SAD}}(\boldsymbol{a}) + \lambda_{\text{SSA}} E_{\text{SSA}}(\boldsymbol{a}) \tag{4.20}$$

with $\lambda_{\text{SAD}} = 1\text{e}{-3}$ and $\lambda_{\text{SSA}} = 1\text{e}{-6}$. The SSA term is kept but with weaker weight to add an incentive for smaller activation values while the SAD term is dominant. The results are shown in Figure 4.3b. For small angles it does not make a significant difference compared to the minimum activation strategy. For large angles, on the contrary, the flexors are now activated nearly identically.

The results appear to be artificial. Nearly identical activation values are most likely a result of the mathematical formulation that cannot be backed with biological reasoning. Nevertheless, according to the model and inversion strategy it is a valid solution. Motivated by the question whether there might be a better approach that does not rely on assumptions about the activations, the minimum deformation strategy is developed. Similar to the minimum activation strategy, this strategy is based on the assumption that a human body tries to solve motion tasks as efficient as possible. But instead

	Activations			Coordinates		
Strategy	RMSG	MAG	MAXAG	RMSG	MAG	MAXAG
Minimum activation	0.077748	0.029176	0.995729	0.324711	0.218868	2.631073
Nearest activation	0.052141	0.022973	0.830327	0.304378	0.208463	1.127719
Minimum deformation	0.051803	0.023376	0.494052	0.304689	0.208879	1.207651

Table 4.2: Inversion strategy smoothness comparison.

of proclaiming that the activation itself is a good measure, the strategy relies on the assumption that the minimum deformation between two iterations will yield the most efficient transition. Each kind of movement has to correlate with some kind of energetic cost. As an additional benefit, the resulting surface mesh of the biceps is likely to be very smooth over a given trajectory. Ideally, it even is the smoothest. The error function is defined with the target coordinates acquired from the activations of the previous iteration $\boldsymbol{c}_t = f_{\text{biceps}}(\boldsymbol{a}^*_{\tau-1})$:

$$E_{\text{mindef}}(\boldsymbol{a}) = E_{\text{angle}}(\boldsymbol{a}) + \lambda_{\text{MSDE}} E_{\text{MSDE}}(\boldsymbol{a}) + \lambda_{\text{SSA}} E_{\text{SSA}}(\boldsymbol{a}) \tag{4.21}$$

with $\lambda_{\text{MSDE}} = 1\text{e}{-3}$ and $\lambda_{\text{SSA}} = 1\text{e}{-6}$. Again, the SSA remains in a weakened position to add a incentive for small activations in case the MSDE does not have a unique solution. The result is presented in Figure 4.3c. Again, the values for small angles look very similar to the previous approaches. For large angles there are similar activations but with subtle differences. Especially, the Anconeus plays a stronger role for $\theta \geq 86\,°$. The approach does not suffer from discontinuities compared to the minimum activation strategy. In addition, it does not overly insist on similar activation values in contrast to the nearest activation approach. The smootheness metric results for the three strategies is shown in Table 4.2. Both, the nearest activation and minimum deformation strategies, outperform the minimum activation approach in all scores. The nearest activation and the minimum deformation approach show very similar results. Surprisingly, compared to the minimum deformation strategy the nearest activation strategy yields a higher MAXAG regarding the activations, while the MAXAG of the coordinates is slightly lower. Intuitively, one would assume the opposite based on the respective optimization goals. In summary, both approaches perform similarly well regarding the smoothness.

The minimum deformation strategy is the most promising approach to deal with the inversion problem. While being computationally more demanding, it yields reasonable results without strong assumptions about the activations itself. Therefore, it is used for this thesis. The SMAE and the SSMD remain conceptually to be used in future work.

With none of the strategies, the solver was able to find solutions for angles $\theta < 53.3\,°$ and $\theta > 99.6\,°$ given an angular error threshold of $1\text{e}{-2}\,°$. All three strategies lead to the same interval of reachable elbow angles with this given threshold. For the remaining data points, the angular RMSE is smaller than $1\text{e}{-4}\,°$ and therefore negligible. Conclusively, the limited angular sensitivity of the surrogate model has to be caused either by the sparse grid surrogate or by the FEM model itself. Since those models are out of scope for this thesis, a workaround based on rescaling of the movement data is applied as presented in Section 4.1.

Finally, a training dataset for the surrogate models is created. A weighted k-NN with Euclidean distance is chosen for interpolation of the dataset and defined as the ground truth model. This choice is based on the facts that the sample width is quite narrow and the dimensionality of the model is low, i.e., $d = 1$. The k-NN model is sampled uniformly with $N = 1000$ angles in the interval $[53.3, 99.6]\,°$. Due to the existence of the sparse grid models, the best value for k can be determined. The model f_{angle} can be used to verify which angles actually should have been reached with the sampled and interpolated activations. This gives an estimate of the approximation error and helps choosing k. The minimum RMSE is reached with $k = 2$, RMSE = 1.81e−4 for $k \in [2, 10]$. Thus, the 2-NN is used as ground truth model and applied in the creation of the training dataset. As a side note, the distance-weighted 2-NN with the Euclidean distance for interpolation in one dimension is equivalent to classical linear interpolation. For this experiment the KNeighborsRegressor from the python scikit-learn package is applied. The training and selection of surrogate models based on this dataset is described in the following Section 4.4.

4.4 Model Selection

In Section 4.3 the process for sparse grid inversion and dataset creation has been presented. This section contains the training and selection of surrogate models for the AR application based on this dataset. It is preferable to avoid the deployment of the 2-NN ground truth model because it requires the complete original dataset. Furthermore, the search for the k nearest neighbors can be expensive for large datasets and high dimensions. In this one-dimensional scenario with evenly spaced sampling points it is trivial and efficient to compute the relevant neighbors, s.t. the model size is the main motivation to find a surrogate model.

As introduced in Section 2.2, the CV workflow is applied to evaluate the model candidates. The dataset from Section 4.3 with $N = 1000$ samples is randomly split in a training and test set with a 80/20 ratio. The training set is applied with 5-fold CV grid search approach to identify good hyperparameters and models. This exhaustive search of parameter values is very expensive. Therefore, the search space is rather restricted for this thesis. This trade-off is acceptable, since the goal of this thesis is not to find the very best model but rather a range of models with different levels of performance and efficiency. The models are the foundation for the distribution approaches presented in Chapter 6 and evaluated in Chapter 7.

The experiments in this section are based on the following scikit-learn implementations in python: Pipeline, MaxAbsScaler, MinMaxScaler, StandardScaler, TransformedTargetRegressor, KFold, GridSearchCV, MLPRegressor, PolynomialFeatures, LinearRegression, Ridge.

4.4.1 Activation Models

In this section a surrogate model for the function $f_{\text{angle2acts}} : \mathbb{R} \to \mathbb{R}^5$ is searched with the elbow angle θ as input feature and the muscle activations $\boldsymbol{a} = (a_1, \dots a_5)^\top$ as target features. As a reference, the 2-NN ground truth model has 2320 parameters.

Linear Regression As a first model, the linear regression is applied with polynomial features. The CV approach evaluates models over the space of the following design choices:

- Feature scaling $\in$ {Identity, MinMax, MaxAbs, Standard}
- Polynomial degree $d \in [1, 30]$

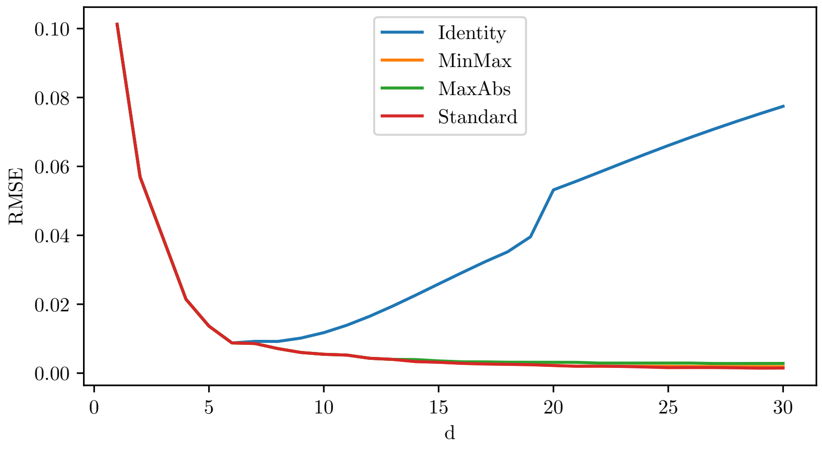


Figure 4.4: Activation: Polynomial regression CV results. The plot shows the RMSE over the polynomial degree d. The colors indicate the courses for the different scaling methods.

The activation values are already in the range $[0, 1]$. Therefore, no target scaling is applied. Figure 4.4 shows the mean RMSE of the CV folds over d. Until $d = 6$ the difference between the scaling approaches is negligible. For $d > 13$ the MaxAbs scaling performance begins to diverge. The same applies for the MinMax scaling for $d > 22$. The standardization yields the best performance for high values of d. Thus, the Standard scaler is chosen as preprocessing for the regression models. There is no sign of overfitting for MinMax, MaxAbs and standardization since the RMSE is further decreasing over d. Therefore, no explicit regularization is applied. In principle, the whole range of d up to 30 can be used. Practically, it is questionable whether it is worth, due to the increasing cost and the saturating course.

Neural Network As a second approach NNs are investigated in the search for a surrogate model. Due to the higher training cost and number of hyperparameters the search space is further limited. The following design space is explored in the CV process:

- Feature scaling: {Identity, MinMax}
- # hidden layer: $l \in [1, 3]$
- # hidden nodes per layer: $h \in \{x = 2^p \mid p \in [0, 10]\}$

The following design choices are fixed during evaluation:

- Activation function: ReLU
- Batch size: 32
- L^2 regularization: $\alpha = 1\text{e}{-4}$
- Optimizer: Adam with learning rate $\eta = 1\text{e}{-3}, \beta_1 = 0.9, \beta_2 = 0.999$
- Early stopping with 10 % data as validation set with tolerance tol = $1\text{e}{-4}$ and $n_{\text{no\_change}} = 10$
- Maximum number of epochs: 1000

Rank	Architecture	RMSE	Parameters
1	NN [1 [8, 128, 128] 5]	0.005979	18325
2	NN [1 [4, 128, 256] 5]	0.006277	34957
3	NN [1 [8, 256, 256] 5]	0.006424	69397
4	NN [1 [4, 256] 5]	0.006613	2573
5	NN [1 [4, 64, 512] 5]	0.006672	36173
6	NN [1 [32, 128, 128] 5]	0.006690	21445
7	NN [1 [4, 64, 64] 5]	0.006753	4813
8	NN [1 [4, 64, 128] 5]	0.006895	9293
9	NN [1 [4, 1024] 5]	0.006949	10253
10	NN [1 [8, 64, 256] 5]	0.006955	18517

Table 4.3: Activation: Top 10 NN architectures in CV.

The 10 best NN architectures of the CV search are shown in Table 4.3. The notation $[n[h_1, \dots, h_l]m]$ describes the NN architecture with the number of input features n, the number of target features m and the number of hidden nodes in each hidden layer $h_1, \dots, h_l$.

The best NN has a slightly worse RMSE than the polynomial regression model with $d = 10$ and a RMSE of 0.005474. Additionally, the NN is far more expensive. Compared to the ground truth model the NN needs ≈ 7.9 times more parameters, while the regression model only needs about 2.37 %. The only other NN with a nearly reasonable size is the architecture [1 [4, 256] 5]. Thus, a second CV run for fine tuning with models 1 and 4 is performed over the the following parameters:

- Feature scaling: {Identity, MinMax, Standard}
- L^2 regularization: $\alpha \in \{\alpha = 10^{-p} \mid p \in [1, 7]\}$

In this experiment, the MinMax scaler is confirmed to work best with the given architectures. Furthermore, for the [1 [8, 128, 128] 5] model $\alpha = 1\text{e}{-4}$ is the optimal parameter in the search range. The architecture [1 [4, 256] 5] performs slightly better with $\alpha = 1\text{e}{-5}$ which yields a RMSE of 0.006154. Nonetheless, none of the NNs of the analyzed design space can outperform the higher order polynomial models.

The performance on the test set is evaluated in Section 4.5.

4.4.2 Biceps Deformation Models

This section shows the search for a surrogate model of the function $f_{\text{angle2coords}} : \mathbb{R} \to \mathbb{R}^{2809\times 3}$ which represents the relation between the elbow angle θ and the 3D surface mesh of the Biceps. Previous work [5] shows that the prediction of the surface mesh with NNs works better if the relative deformation to the rest state at $\boldsymbol{a} = (0, 0, 0, 0, 0)^\top$ is used as target instead of the coordinates. The relative deformation is given by $\boldsymbol{c} = \boldsymbol{d} + \boldsymbol{o}$, with the coordinates $\boldsymbol{c}$, the deformations $\boldsymbol{d}$ and the rest point offset $\boldsymbol{o}$. Therefore, the deformations are also used as target features in this work. As a reference, the 2-NN ground truth model has $\approx 3.9\text{e}6$ parameters.

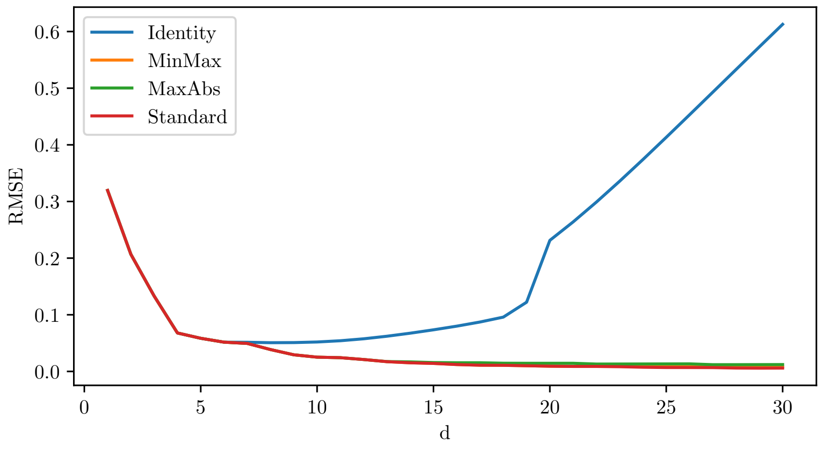


Figure 4.5: Deformation: Polynomial regression CV results. The plot shows the RMSE over the polynomial degree d. The colors indicate the courses for the different scaling methods.

In contrast to the activations, the deformations are not a priori defined on a fixed interval but share the same unit. In order to avoid a prioritization of some coordinates by individual feature scaling another method needs to be developed. The coordinates are scaled as a group, i.e., the relative scale between single coordinates is preserved.

Linear Regression The CV evaluates the polynomial regression models over the space of the following design choices:

- Feature scaling $\in$ {Identity, MinMax, MaxAbs, Standard}
- Target scaling $\in$ {Identity, MinMaxGroup, MaxAbsGroup, StandardGroup}
- Polynomial degree $d \in [1, 30]$

The CV results suggest that the target scaling makes a negligible difference. Therefore, no target scaling is assumed for further evaluations. Figure 4.5 shows the mean RMSE of the CV folds over d. The course of the RMSE is qualitatively identical to the activation case. Thus, the Standard scaler is also chosen as preprocessing for the regression models.

Neural Network The search space for the deformation case is further restricted due to the higher cost caused by the increased output dimension. The following design space is explored in the CV process:

- Feature scaling: {MinMax}
- Target scaling: {Identity, MinMaxGroup}
- # hidden layer: $l \in [2, 5]$
- Architectures: arch $\in \{f_{\text{arch}}(n, m, l+1) \mid l \in [2, 5]\}$

Rank	Architecture	RMSE	Parameters
1	NN [1 [10, 92, 880] 8427]	0.039564	7507059
2	NN [1 [7, 38, 227, 1382] 8427]	0.049848	11978808
3	NN [1 [21, 415] 8427]	0.052748	3514804
4	NN [1 [5, 21, 92, 415, 1868] 8427]	0.060459	16567906
5	NN [1 [92] 8427]	0.313271	783895

Table 4.4: Deformation: NN architectures CV results.

The function $f_{\text{arch}}(N, M, L)$ produces a NN architecture with exponential layer growth given the number of features N, the number of targets M and the number of layers L where only the hidden layers and output layer are counted. Thus, the number of hidden layers $l = L - 1$. This architecture generation function has been developed in [1]:

$$h_i = \lceil ae^{bi} + c \rceil \tag{4.22}$$

$$a = \frac{M}{M + L} \tag{4.23}$$

$$b = \frac{\ln\left(\frac{M-N+a}{a}\right)}{L} \tag{4.24}$$

$$c = N - a \tag{4.25}$$

The rest of the hyperparameters for NN training is chosen as in the activation scenario.

The CV results show that the MinMaxGroup scaler outperforms the Identity scaler for each architecture. The CV result is shown in Table 4.4. The best performing architecture is the [1 [10, 92, 880] 8427] with $l = 3$ and an RMSE of 0.039564 and $\approx$ 7.5e6 parameters. Therefore, it contains $\approx$ 1.9 times more parameters than the ground truth model. Another potential candidate is the [1 [21, 415] 8427] architecture with $\approx$ 89.9 % of the parameters.

The performance on the test set is evaluated in Section 4.5.

4.5 Model Evaluation

The surrogate model test set performance for the activation models is shown in Table 4.5. The polynomial regression models provide a wide range of performance over degree d. In this special case with one input feature, the number of parameters is linear in d. Therefore, it is possible to use high order polynomials without over-proportional costs. The R^2 score shows that performance increases strongly over the lower degrees and begins to saturate for high values of d. The degree is limited to 15 because the model performance is considered to be good enough for the planned use case. While the NNs also show good performance, their costs are many times higher. Therefore, the best option is to choose a polynomial regression model with the desired performance.

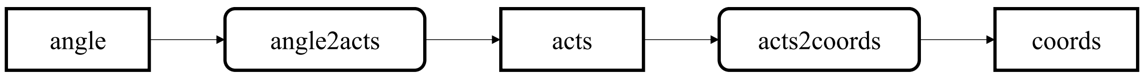
The test set performance for the deformation case is presented in Table 4.6. While the costs are higher compared to the activation case the results are qualitatively the same. One noticeable difference is the existence of large negative R^2 scores for the NNs. The coordinate dataset contains

Rank	Model	RMSE	MAE	R^2	Parameters
1	PR_{15}	0.003755	0.002097	0.999591	80
2	PR_{14}	0.004037	0.002107	0.999513	75
3	PR_{13}	0.004566	0.002615	0.999321	70
4	PR_{12}	0.004882	0.002953	0.999242	65
5	NN [1 [8, 128, 128] 5]	0.006297	0.003541	0.998546	18325
6	PR_{11}	0.006355	0.003606	0.998576	60
7	PR_{10}	0.006560	0.003567	0.998505	55
8	PR_{9}	0.006949	0.003744	0.998315	50
9	PR_{8}	0.008026	0.004689	0.997809	45
10	PR_{7}	0.009980	0.005886	0.996564	40
11	PR_{6}	0.010145	0.005726	0.996501	35
12	NN [1 [4, 256] 5]	0.013009	0.006570	0.993412	2573
13	PR_{5}	0.017945	0.010406	0.987222	30
14	PR_{4}	0.028953	0.017117	0.968828	25
15	PR_{3}	0.050625	0.030221	0.908523	20
16	PR_{2}	0.071321	0.041287	0.826938	15
17	PR_{1}	0.109163	0.074534	0.655474	10

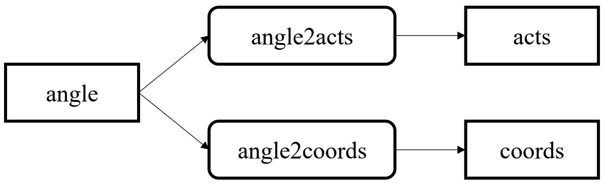
Table 4.5: Activation: Surrogate model test set performance. Ordered by RMSE.

Rank	Model	RMSE	MAE	R^2	Parameters
1	PR_{15}	0.016345	0.010192	0.976380	134832
2	PR_{14}	0.017925	0.010687	0.976225	126405
3	PR_{13}	0.019880	0.011820	0.975936	117978
4	PR_{12}	0.024380	0.016101	0.975521	109551
5	PR_{11}	0.028248	0.018790	0.975046	101124
6	PR_{10}	0.029411	0.018782	0.974786	92697
7	PR_{9}	0.033986	0.022051	0.973631	84270
8	NN [1 [10, 92, 880] 8427]	0.043808	0.030764	-7.2e19	7507059
9	PR_{8}	0.045595	0.031645	0.971059	75843
10	PR_{7}	0.057715	0.040136	0.969713	67416
11	PR_{6}	0.060629	0.041143	0.968462	58989
12	PR_{5}	0.066975	0.047086	0.965783	50562
13	NN [1 [21, 415] 8427]	0.071310	0.049670	-1.5e19	3514804
14	PR_{4}	0.078593	0.054800	0.962658	42135
15	PR_{3}	0.161229	0.111231	0.929121	33708
16	PR_{2}	0.245172	0.163248	0.880201	25281
17	PR_{1}	0.371736	0.267774	0.793868	16854

Table 4.6: Deformation: Surrogate model test set performance. Ordered by RMSE.



(a) Model chaining approach. The coordinates are computed based on the elbow angle by a chain of models with the activations as intermediate result.



(b) Model stacking approach. The coordinates and activations are computed independently based on the elbow angle by a stack of models.

Figure 4.6: Strategies for activation and coordinate estimation.

features that nearly do not change. Therefore, their variance is very small which can lead to huge R^2 values. In general, the reached R^2 scores are lower compared to the activation case. This indicates that the deformation variance is harder to capture in the models. As in the activation case, the NNs are far more expensive compared to the polynomial regression models. Thus, a polynomial regression model with desired accuracy-cost-trade-off can be chosen according to the application requirements.

4.6 Model Combination

In previous work [5] model chaining is applied to calculate the coordinate values. This approach is shown in Figure 4.6a. In a first step, the movement data is used as features for activation prediction. Then, the activations are applied as inputs to predict the deformation. The second model, which maps from activations to coordinates, is a surrogate for the sparse grid that represents the forward simulation like the original FEM model.

This approach is not optimal for the AR application. Firstly, errors in the first prediction propagate through the second model and lead to inconsistencies. Secondly, the $f_{\text{acts2coords}}$ function has more DoFs and therefore is a harder problem that requires more expensive surrogate models.

The $f_{\text{acts2coords}}$ NN model from [5] has $\approx$ 43.9e6 parameters. It is evaluated on the same test set as the $f_{\text{angle2coords}}$ models in Section 4.5. Its evaluation with ground truth activations yields an RMSE of 0.680851, an MAE of 0.424991 and an R^2 of -3.5e18. Thus, it produces an $\approx 1.8\times$ higher RMSE with $\approx 2600\times$ more parameters compared to simple linear regression.

This does not mean that the given NN is a weak model. It is designed to fulfill another purpose and therefore uses the available resources sub-optimally for the reduced task. As a result, focus on the necessary aspects of a surrogate model plays an important role in efficient deployment. The parallel model stacking approach, depicted in Figure 4.6b, is better suited for the given task. Even the linear regression yields better results by also avoiding inconsistent states by error propagation.

5 Distribution Model for System Emulation

This chapter presents the developed distribution model for analysis and evaluation of the distributed algorithms. Real distributed systems are difficult to analyze and evaluate due to e.g. concurrency, imperfect clocks or disturbances. Therefore, effects seen in a real system are hard to analyze and reproduce. Instead, a distribution model is developed and applied for deterministic emulation of the distributed system. The main motivation for the developed distribution model is to allow reproducible simulations for evaluation purposes. The model allows a single threaded emulation of concurrent and deterministic processes distributed on nodes that communicate with messages over communication channels. Section 5.1 introduces the model concept and architecture and Section 5.2 presents the probabilistic modeling of jitter and losses.

5.1 Cycle-based Emulation of the Distributed System

The main components of the distribution model are nodes, channels, processes and messages. A node represents a device with computational resources and network interfaces. Nodes are connected via unidirectional communication channels. Application functionality is implemented in processes which are deployed on nodes. Processes can exchange information with the help of messages. Messages are transferred through a network interface of a node over a channel to the respective receiver. The architecture of an exemplary system with two nodes is visualized in Figure 5.1. Each node contains one process. The nodes are connected with unidirectional channels. With this setup a classical request-response or client-server structure can be modeled.

Processes A process is an abstraction of a functional aspect of a system. The model can be split into a logical and a physical view. In the logical view, there exist only processes which exchange information in the form of messages. Processes can be interpreted as functions that produce a set of output messages based on a set of input messages: $f_P : \{m_{\mathrm{i}_1}, \ldots, m_{\mathrm{i}_k}\} \rightarrow \{m_{\mathrm{o}_1}, \ldots, m_{\mathrm{o}_l}\}$.

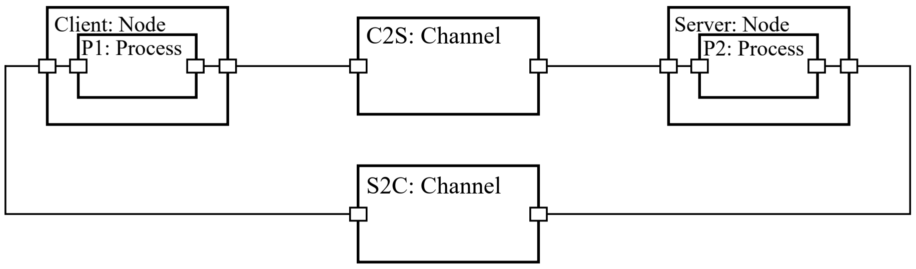


Figure 5.1: Distribution model architecture of an exemplary system with two nodes. Each node contains one process. The nodes are cyclically connected with unidirectional channels.

This logical view is not aware of the concept of time or delay in a physical sense. A process simply produces an output in the form of a message based on the input messages it was given. If it needs a notion of time, an artificial message can be passed to it that contains the timestamp of the beginning of its execution. The same applies for algorithms with pseudo-random behavior [42]. Those algorithms need to be transformed s.t. they are deterministic and implement randomness with a pseudo-random number generator that receives the seed via an input message. In this way, the processes behave deterministically, which is a necessary property to achieve reproducibility in the emulation. In consistency with those assumptions, processes create all messages during the execution but deliver them at once when the process terminates, i.e., processes do neither send nor receive messages during execution.

Nodes A node represents a device, e.g., mobile device, personal computer or server, on which processes can be deployed. Nodes can possess communication interfaces which can be connected via channels. For this thesis, two types of nodes are relevant. The first type of node is message invoked. This means, whenever a desired message is received, the deployed process is executed. The second type of node is a time-based node. This type does not require the arrival of a desired message but schedules its process on a cyclic basis. Strictly speaking, this could also be modeled with the help of an artificial message with the desired period. Thus, the time-based type is a convenience definition. The nodes are modeled with a single processing unit and one deployable process. This restriction is made for simplification and may be extended in future work.

Cycle-based Emulation The key idea of the distribution model is the realization of concurrency in a sequential manner. Thus, concurrency of processes deployed on nodes is achieved with a two-phase cycle-based emulation of the system with discrete time steps. The system is scheduled periodically with a predefined emulation cycle time T_E. Each emulation cycle contains two phases.

In Phase I, the node execution phase, all existing nodes are executed once. In this phase the processes can be executed, if the scheduling conditions of the process are fulfilled, i.e., the message of interest has arrived or the time trigger is active. While the processes implement the logical view of the system, the nodes realize the physical view. The processing delays T_P are emulated by the nodes. The logical process is fully executed in the cycle in which it is triggered. The delay is emulated by holding back the output messages for n further emulation cycles. Consequently, a subsequent execution of the process is prohibited until the delay time has elapsed. This behavior is implemented with a blocking counter.

In the following Phase II, the channel execution phase, all channels are executed. Consequently, the messages are passed after all processes have terminated in Phase I. In this way, concurrency is reduced to the granularity of one emulation cycle.

In combination with the presented assumptions and modeling decisions the discretization causes a minimum amount of time that is needed for any activity. A process on a node requires at least one cycle to produce an output message, since the termination check is performed at the beginning of the next node cycle in Phase I. The output messages are only delivered to the channel after successful termination. A transmission over a channel also requires at least one emulation cycle. The messages are exchanged in Phase II. Thus, the receiving process can consider the message at the earliest in the next Phase I. This effectively results in a minimum transmission time. Therefore, every action

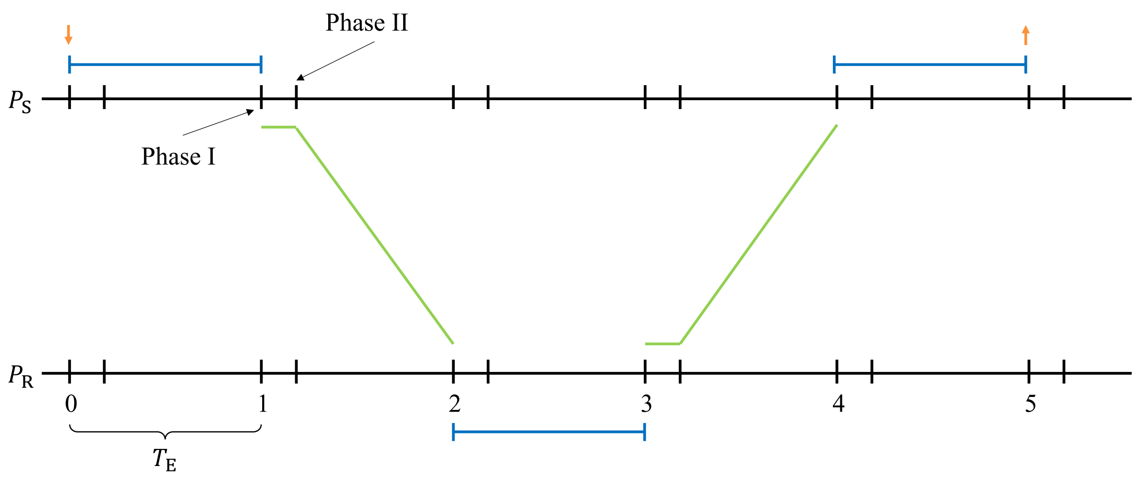


Figure 5.2: Shortest round-trip between two processes P_S and P_R on two nodes. The orange arrows indicate the information input and output of the sending process P_S. The blue lines illustrate the duration of the process execution. The green lines show the message flow between processes over time.

requires at minimum T_E. Consequently, the shortest round-trip between two processes on two nodes requires four emulation cycles. The processing of the second node's response is carried out in the fifth emulation cycle. This is exemplary visualized in Figure 5.2.

Channel Design In general it is assumed that the communication works in an asynchronous way. A sending process is not blocked except for the handover of the messages to the concurrent communication layer. Various models of a channel can be implemented for emulation of different channel behavior and situations.

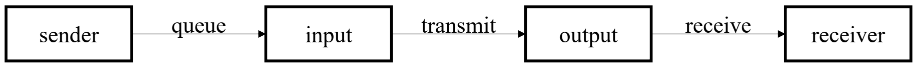


Figure 5.3: Architecture of the tail-drop channel.

The *tail-drop channel*, visualized in Figure 5.3, is modeled with two buffers, the input and the output buffer. If a node sends a message over the channel, it is placed in the input buffer. After n emulation cycles the message is transmitted to the output buffer where it can be fetched by the receiver. This holding back for n cycles emulates the channel delay T_C. The channel is rejecting further send requests while a message is in transit, i.e., tail-drop behavior. This behavior comes with a drawback. If a node cannot successfully queue a message due to dropping, it is lost without further notice. The model could be extended s.t. the sending process receives an artificial input message with a state feedback. Nonetheless, this results in further delays, since the earliest retry is possible after the next process termination. While this approach is a valid model, it is considered too restrictive for modeling real world systems with concurrent communication layers. This shortcoming is solved with the head-drop channel.

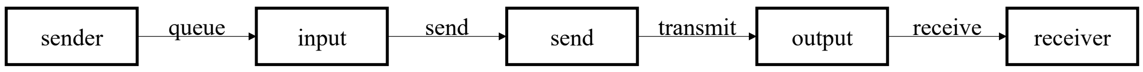
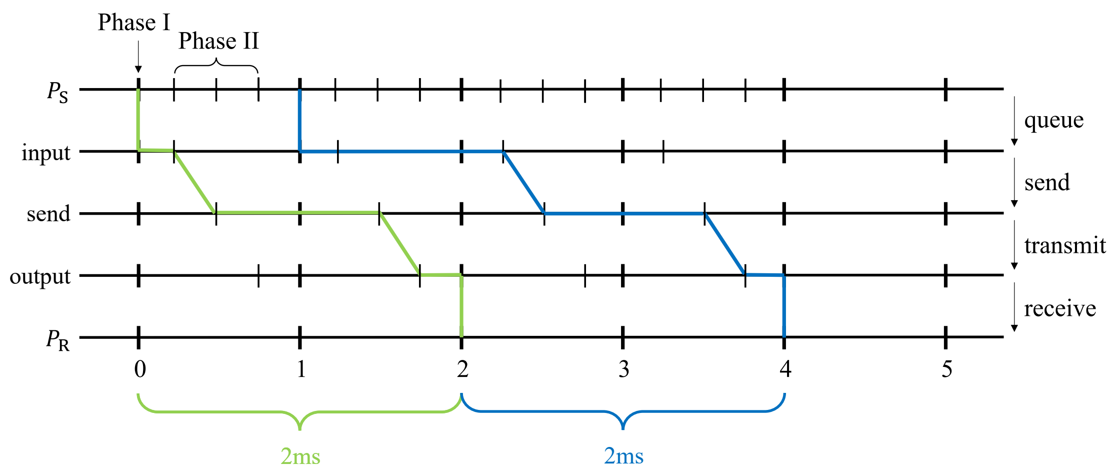


Figure 5.4: Architecture of the head-drop channel.

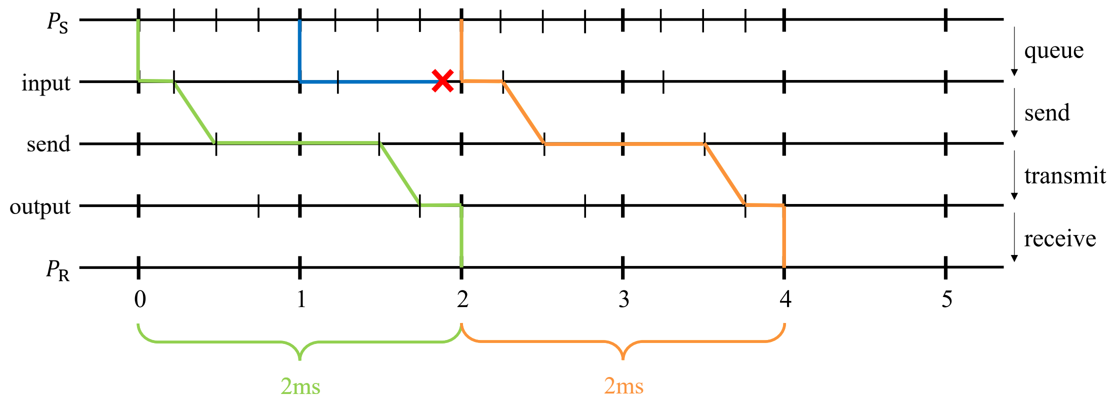
The *head-drop channel*, illustrated in Figure 5.4, is designed to emulate communication systems that run concurrently to the sending process on a lower level. In this channel variant, messages are not immediately dropped if the preceding message is still in transit. Instead, the new message is appended to the input queue. Whenever the channel is idle, i.e., the send queue is empty, the oldest message from the input queue is transferred to the send queue. To emulate the channel delay T_C, the message is kept there for n emulation cycles. As soon as the message is transmitted to the output queue it can be received by the message consumer. A message is passed along the sequence of queues in First-In-First-Out (FIFO) order. Thus, the order of messages is preserved. The send queue has size one, i.e., it can store only one message at a time. This represents a bottleneck in the communication, i.e., there are no parallel transmissions. The output queue also has size one. With the FIFO property this only hands over the most recent message to the receiver. Another interpretation of this behavior is a receiving process that discards all but the newest message. The input queue is modeled as circular buffer. This implements the head-drop behavior, i.e., if the queue is full, the oldest message is dropped in favor of the new one. With an input queue size of one, only the newest message of the sender is kept. This "latest-greatest" property is chosen to achieve a minimum delay for the AR application. Usually, messages cannot be unqueued once sent in state-of-the-art systems. Since queue sizes have an upper bound in practice a full queue typically results in some kind of backpressure. For example, the sending process can be suspended until the communication system can accept the send request which is called buffer-blocking. This effect increases the model complexity a lot and is therefore out of scope for this thesis. The suggested communication pattern can be implemented in practice using an application layer packet processing framework.

Figure 5.5 illustrates two examples of message flows through the head-drop channel over emulation cycles. For this example the channel delay T_C is 2 ms, the send period T_S is 1 ms and the emulation cycle time T_E is 1 ms. In Figure 5.5a the first message m_1 is queued by the sending process P_S at $t = 0$ ms. Since the send queue is free, the message is moved to the send queue. To account for the delay T_C, the message is kept there for one further emulation cycle. In the following emulation cycle, m_1 is passed over to the output queue. The receiving process P_R fetches the m_1 at $t = 2$ ms, which effectively results in the desired delay T_C. Although the second message m_2 is send by P_S at $t = 1$ ms, it has to wait in the input queue until m_1 leaves the send queue. Afterwards, it follows the same procedure as m_1. Consequently, for m_2 the delay T_C is extended by the wait time of 1 ms in this case, which results in a total transmission delay of 3 ms. Figure 5.5a shows the same scenario but with a third message m_3 sent at $t = 2$ ms. Due to the input queue size of one, m_2 is dropped and replaced by m_3. At this point m_2 is lost without further notice, which results in the consecutive transmission of m_1 and m_3.

Extensions of the head-drop channel model with message losses and randomized delays are presented in Section 5.2.



(a) Delay of subsequent messages.



(b) Dropped message at input queue after subsequent send attempt.

Figure 5.5: Head-drop channel examples. The graphics illustrate the message flow through the head-drop channel model over emulation time.

5.2 Probabilistic Modeling of Communication Channels

The model presented so far has a deterministic transmission behavior with a constant channel delay, i.e., the resulting message delay can vary dependent on the time the message has to wait in the input queue. This assumption does not hold for state-of-the-art communication technology, e.g., wireless channels with varying bit error rates leading to retransmissions or shared networks with varying cross-traffic leading to varying queuing delay in network elements. In this setting, the developed system has to deal with varying transmission delays, message losses, duplicates and out-of-sequence arrivals. This thesis focuses on varying transmission delays and message losses. In the following, these effects are modeled in a probabilistic way.

Mukherjee [43] presents an analysis of round-trip delays of Internet packets. The analysis is based on three settings, containing data for a regional, a backbone and a cross-country network segment. According to Mukherjee, the low frequency components of delay can be approximated with a shifted gamma distribution. The shape and scale parameters vary with network segment and load.

Furthermore, Mukherjee states that he cannot provide a precise answer why the distribution is gamma. Nonetheless he reasons that the gamma distribution is versatile, e.g., it contains exponential, Erlang and χ^2 distributions as special cases, which are known to model nature well. In addition, the delays are positive and the empirical histograms show non-symmetric shapes with long tails. These properties can be captured with the gamma distribution. Mukherjee assumes that possible reasons for the large tail can be systematic batching and FIFO processing.

Based on [43], Corlett et al. [44] provide an analysis of datasets about one-way packet delays on the Internet. They found that during quiet periods very long and thin tails occur which can be well modeled with a shifted exponential distribution. This is consistent to the results of Mukherjee [43] because the exponential distribution is a special case of the gamma distribution.

Sui et al. [45] present a characterization of Wi-Fi latencies in large-scale operational networks. Sui et al. argue that while the wired latencies are relatively stable and can be optimized, e.g., with content delivery networks, the last hop to a mobile device can be unpredictable. They consider the Wi-Fi latency to be the dominant factor for the end-to-end delays. Their results show that the Wi-Fi latencies follow a long tail distribution. As an example, their measurements show a low median of 3 ms, while the 90th percentile is around 20 ms and the 99th percentile is around 250 ms.

Chou and Miao [46] present an approach for optimized streaming where the network delays are also modeled as gamma distribution. In more detail, they model the network to be an independent time-invariant packet erasure channel with random delays. This means that the sender queues a message, which can be instantaneously lost with some probability p_l, independent of the send time t_s. If the message is not lost in this process, it reliably arrives at the receiver. The forward-trip time or transmission delay is randomly drawn from a probability distribution $p_d(\tau \mid \text{not lost})$. Each message loss and delay is independent of other messages. Chou and Miao [46] state that the independence and time-invariance is reasonable over short periods of time, i.e., a few seconds, under the assumption the sender messages are not self-congesting. In other words, each message of a sender has to leave the bottleneck queue before the next message of the same sender arrives. Otherwise, the messages cause a congestion. Therefore, loss and delay of a message are approximately i.i.d. based on the underlying state of the network. More sophisticated modeling can be realized with hidden Markov models. Chou and Miao [46] note, given the recent past it is sufficient to simply estimate p_l and $p_d(\tau \mid \text{not lost})$, which allows a change of the parameters over time.

Kalman and Girod [47] present an extension to the i.i.d. model, where successive delays over the channel are modeled as a first order discrete Markov process. The authors of [47] claim that their approach yields far better accuracy than the i.i.d. model. The application of this modeling approach is beyond the scope of this thesis.

In this work, an approach based on the work of Chou and Miao [46] is implemented. The channels are also modeled as independent time-invariant packet erasure channels with random delays. The loss probability p_l is considered in the transition of a message from the input to the send queue. This represents the point of time in the real system where the message is actually sent. If the message is not dropped, the delay T_C is drawn from the distribution $p_d(\tau \mid \text{not lost})$. To be more precise, since the probabilities model the state of the underlying network, the random variables are drawn from their distributions at the beginning of each emulation cycle. In case they are not required they are simply dropped. In this way, with the help of a seeded pseudo-random number generator, the pseudo-random properties of the channels are reproducible over time, s.t. multiple emulation runs share the same network behavior. As a side note, the probabilistic loss mechanism has not been

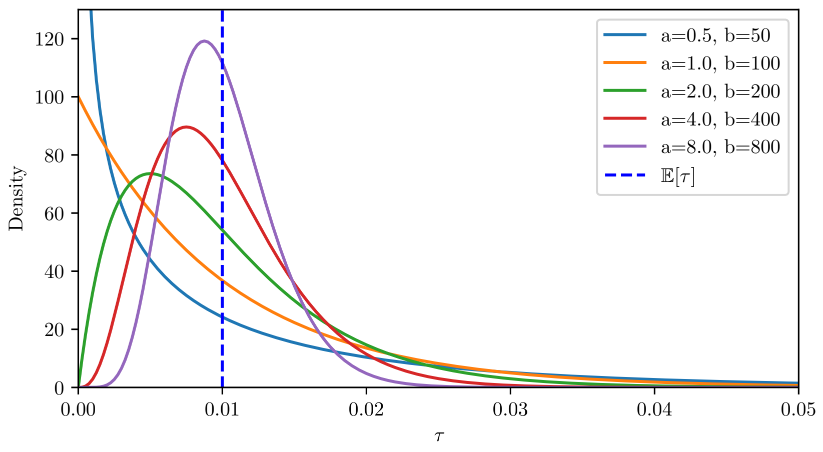


Figure 5.6: Example for gamma distributed delays τ. The plot shows the probability density functions for different combinations of a, b, s.t. the expected value $\mathbb{E}[\tau] = 10\text{e}{-3}$.

implemented during this thesis. This is acceptable because the focus is on delay and jitter. The resulting behavior is equivalent to $p_l = 0$. Note that messages can be lost anyway in case of sender caused congestion.

The analysis, precise modeling and characterization of a potential deployment system is a topic on its own and therefore out of scope for this thesis. Thus, the delays are assumed to be simply gamma distributed. The gamma distribution probability density function is given by [11, 12, 43]:

$$\text{Gamma}(\tau|a, b) = \frac{b^a}{\Gamma(a)} \tau^{a-1} e^{-b\tau} \tag{5.1}$$

$$\Gamma(x) = \int_0^\infty u^{x-1} e^{-u} du \tag{5.2}$$

with the random variable τ, shape parameter a, the inverse scale or rate parameter b and $\tau, a, b > 0$. With the following equations the distribution parameters can be calculated from the expected value $\mathbb{E}[\tau]$ and variance $\mathbb{V}[\tau]$:

$$a = \frac{\mathbb{E}[\tau]^2}{\mathbb{V}[\tau]} \tag{5.3}$$

$$b = \frac{\mathbb{E}[\tau]}{\mathbb{V}[\tau]} \tag{5.4}$$

The proof can be found in Appendix B. Therefore, the distribution can be conveniently estimated from a given or assumed set of expected value and variance. As an alternative, a shift parameter could be added as in [46]. This case requires further insights and assumptions about the split between constant and randomized delays and their dependencies, e.g., message sizes, load, disturbances, and is therefore neglected in this thesis. As an example, different courses of the gamma distribution with the same expected value $\mathbb{E}[\tau] = 10\text{e}{-3}$ are shown in Figure 5.6.

Furthermore, for bidirectional communication the forward and backward channel are assumed to be symmetric. Thus, the overall transmission delay of the round-trip τ_s is divided into the forward and backward channel by halving the expected value $\mathbb{E}[\tau_s]$ and variance $\mathbb{V}[\tau_s]$. The proof can be found in Appendix B:

$$\mathbb{E}[\tau_{1/2}] = \frac{1}{2}\mathbb{E}[\tau_s] \tag{5.5}$$

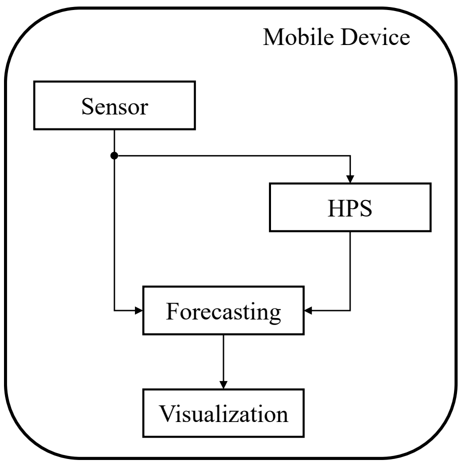
$$\mathbb{V}[\tau_{1/2}] = \frac{1}{2}\mathbb{V}[\tau_s] \tag{5.6}$$

In the modeled channel, successive messages have to wait for the preceding transmission to be completed. This approximation is only valid for connections with a dominant bottleneck, as for example with Wi-Fi as first/last hop. With the presented probabilistic model message order is preserved. The model can be easily extended to account for this scenario. Instead of the input and send queue, a transmission buffer can be used. As above, when a message is queued, it can be lost with some probability p_l. If it is not lost, it is placed in the transmission buffer for a random delay drawn from $p_d(\tau \mid \text{not lost})$. Consequently, the stay of the messages in the channel is independent from each other, s.t. messages can overtake. A crucial difference of this extension is that successive messages do not wait for the preceding transmission to be completed.

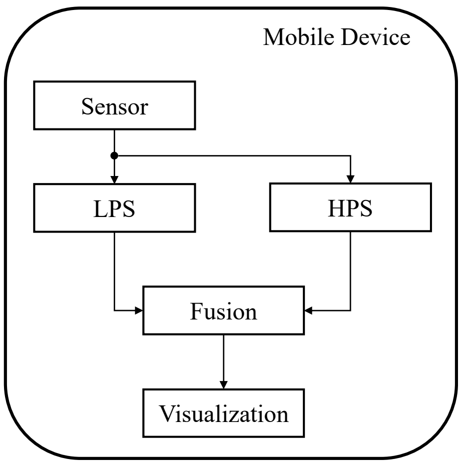
The probabilistic FIFO head-drop channel model with order preservation is applied in this work.

6 Distribution Approaches

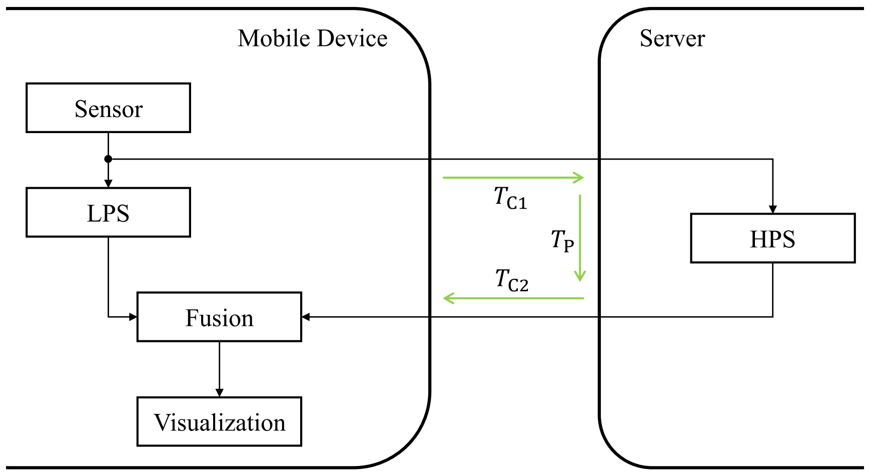
In this chapter possible applications of the Kalman Filter (KF) for the pervasive simulation problem, which is described in Chapter 3, are investigated. In the pervasive simulation case, the goal is to offload the expensive simulation to a server. The distribution architecture is visualized in Figure 6.1c. On the mobile device there is a sensor, which acquires input data for the pervasive simulation with a fixed frequency f_S. This data is transferred to the remote device, where the offloaded simulation is executed in the form of a HPS model. The simulation results are send back to the mobile device



(a) Scenario I.



(b) Scenario II.



(c) Scenario III. T_{C1} and T_{C2} show the transmission delays and T_P the processing delay of the HPS execution.

Figure 6.1: Offloading scenarios.

for visualization. In this scenario inevitable delays arise due to necessary communication and processing times. This architecture is based on the work of Hubatscheck [4]. There, the usage of a LPS model on the mobile device is proposed. The LPS can be used to improve the infrequent and delayed simulation results from the server and acts as fallback in case of server or communication issues. Hubatscheck [4] presents a fusion algorithm that merges the LPS results with continued HPS results.

In this work KF-based approaches are investigated and compared to the continued update strategy from [4]. The KF is an optimal estimator for linear systems in the presence of white Gaussian noise. If the assumption holds that the given system is linear and suffers from disturbances that behave like white Gaussian noise, the KF is an optimal estimator for the given problem. Consequently, under the given assumptions there is no better solution. Even though the assumption might only hold to a certain degree, for many problems like filtering and fusion of noisy sensor measurements, even under existence of delays, there exist well studied solutions and approaches. Thus, with reasonable analogies and assumptions, those approaches can be transferred to the pervasive simulation problem.

The following scenarios are investigated in this chapter:

I **One Simulation Model. No Transmission Delays. No Processing Delays. No Loss.**

The simulation is based on the heavyweight HPS. This scenario can be motivated in two ways. Motivation A: The delays are negligibly small but the communication costs shall be minimized. Motivation B: There exists no remote device. The HPS is deployed on the mobile device as depicted in Figure 6.1a. The goal is to minimize computational costs by temporary suspension of the HPS for some time steps. This idea has been already investigated in [34] where forecasting models are applied to predict intermediate outputs. For this scenario, time series models are applicable that solely rely on the history of simulation results. In this work, the KF is applied as an alternative forecasting model.

II **Two Simulation Models. No Transmission Delays. No Processing Delays. No Loss.**

Scenario II is similar to Scenario I but here a second surrogate model, the LPS, is added. The same motivations as in Scenario I apply. The architecture of this scenario is visualized in Figure 6.1b. Instead of pure time series forecasting methods, the LPS can improve the forecasting quality with its explanatory character, i.e., it adds information about the relation between simulation inputs and outputs.

III **Transmission Delays, Processing Delay and Message Loss.**

Scenario III confronts the idealized Scenarios I and II with the existence of delays and message loss and is visualized in Figure 6.1c. Thus, it represents a setting with the HPS on the server and an optional LPS on the mobile device with conditions of a real world system. A crucial difference to the previous scenarios is that due to the delay, updates from the server cannot be considered for the current estimate anymore.

Section 6.1 presents an improved variant of the continued update strategy from [4]. Relevant details for the application of the KF are shown in Section 6.2. The pervasive simulation problem targeted in this thesis is not a classical sensing and fusion problem. Nevertheless, the idea is to draw analogies between the two perspectives and to transfer solutions known from the control and signal theoretical point of view to the pervasive simulation case. The following analogies motivate the approaches presented in this chapter: The basic idea is to interpret the simulation result of the HPS as sensor measurement with very low noise, ideally no noise. Scenario I is addressed in Section 6.3 as a classical tracking problem. In Scenario II the LPS is added as second sensor with a higher frequency and lower accuracy. This is tackled in Section 6.4 with analogies to a sensor fusion problem. Since the LPS does not fulfill the white Gaussian noise property, due to a bias like error, alternative approaches are investigated in Section 6.5. Finally, Scenario III is addressed in Section 6.6 with the analogy of delayed measurements.

6.1 Improving the Continued Update Strategy

The continued update fusion strategy from Hubatscheck [4] is defined as follows:

$$\Delta l = l_{t+h} - l_t \tag{6.1}$$

$$c_{t+h} = u_t + \Delta l \tag{6.2}$$

$$m_{t+h} = \alpha \cdot c_{t+h} + (1 - \alpha) \cdot l_{t+h} \tag{6.3}$$

with the LPS result l_t, the HPS update u_t, the continued update c_{t+h}, the merged update m_{t+h} and the merge parameter $\alpha \in [0, 1]$. The HPS update from the server u_t is extrapolated with the change in the LPS result Δl. Finally, the continued update c_{t+h} is merged with the LPS result l_t as weighted average with the weight parameter α.

The contribution of this work is the following improvement:

$$\begin{aligned} m_{t+h} &= \alpha \cdot c_{t+h} + (1 - \alpha) \cdot l_{t+h} && (6.4)\\ &= \alpha \cdot (u_t + \Delta l) + (1 - \alpha) \cdot l_{t+h} && (6.5)\\ &= \alpha \cdot (u_t + l_{t+h} - l_t) + l_{t+h} - \alpha \cdot l_{t+h} && (6.6)\\ &= \alpha \cdot \underbrace{(u_t - l_t)}_{o_t} + l_{t+h} && (6.7)\\ &= \alpha \cdot o_t + l_{t+h} && (6.8) \end{aligned}$$

with the offset $o_t = u_t - l_t$ between the HPS and LPS at time t. This conversion has convenient properties. On the one hand, the approach can be described as offset or error correction at time t. Thus, α describes how strong the error correction shall be considered. If $\alpha = 1$ the error is fully compensated. On the other hand, this form can be implemented more efficiently. The fusion process only needs to remember the local estimate l_t of the time of the request to the server. When the HPS update u_t arrives delayed, the offset o_t is calculated once and used as correction term for the following time steps weighted with α.

6.2 Kalman Filter in Practice

This section introduces relevant basic concepts and helpful approximations for the practical application of the KF before the it is applied to the distributed pervasive simulation problem in the next sections.

6.2.1 Numerical Instability

According to Labbe [24], the KF can suffer under numerical instability in practice. The issue arises when the update equation leads to non-symmetric covariance matrices due to floating point errors in the subtraction:

$$\boldsymbol{P}_{k+1} = (\boldsymbol{I} - \boldsymbol{K}_{k+1}\boldsymbol{C})\boldsymbol{P}_{k+1|k} \tag{6.9}$$

A traditional workaround preserves symmetry by averaging the error between the symmetric values [24]:

$$\boldsymbol{P}_{k+1} = \frac{1}{2}(\boldsymbol{P}_{k+1} + \boldsymbol{P}_{k+1}^{\top}) \tag{6.10}$$

Instead, Labbe [24] suggests to use the so called Joseph equation which ensures symmetry even though the Kalman gain might not be optimal:

$$\boldsymbol{P}_{k+1} = (\boldsymbol{I} - \boldsymbol{K}_{k+1}\boldsymbol{C})\boldsymbol{P}_{k+1|k}(\boldsymbol{I} - \boldsymbol{K}_{k+1}\boldsymbol{C})^{\top} + \boldsymbol{K}_{k+1}\boldsymbol{R}\boldsymbol{K}_{k+1}^{\top} \tag{6.11}$$

Derivations for the equation can be found in [24, 26]. Nonetheless, the filter can still diverge if $\boldsymbol{P}_{k+1|k}$ becomes non-negative for some reason [24, 48]. For critical applications, e.g., where failure might lead to loss of equipment or life, Labbe [24] highlights the importance of avoiding those issues. As stated in [16, 48] other solutions are the use of the square root filter, the information filter or the combination, the square root information filter, which are out of scope for this thesis. In general, there is a trade-off between numerical stability and computational effort [48]. In this thesis the KF is implemented with Equation (6.11).

6.2.2 Filter Initialization

According to Labbe [24], many schemes for filter initialization exist. He proposes the following, where the filter remains uninitialized until the first measurement z_0 is received:

$$\boldsymbol{x}_0 = \boldsymbol{C}^{-1}z_0 \tag{6.12}$$

Since $\boldsymbol{C}$ is rarely square, the Moore-Penrose pseudo-inverse can be used instead. For the error covariance initialization one approach is to use the measurement uncertainty $\boldsymbol{R}$ for identical states and squared maximum values for hidden states. This method is also described by Bar-Shalom et al. [49] as one-point initialization. They suggest to use half the known maximum value squared. Further improvements can be made if domain knowledge is applied. For example, if the state contains a velocity $\dot{x}$ but only the position x is observed, its initial value can be estimated from the difference of the two first measurements [24]. Bar-Shalom et al. [49] call this method two-point differencing. In this case the state vector and state covariance matrix can be estimated with:

$$\boldsymbol{x}_0 = \begin{bmatrix} x \\ \dot{x} \end{bmatrix} = \begin{bmatrix} z_0 \\ \frac{z_0 - z_{-1}}{\Delta t} \end{bmatrix} \tag{6.13}$$

$$\boldsymbol{P}_0 = \begin{bmatrix} \boldsymbol{R} & \frac{\boldsymbol{R}}{\Delta t} \\ \frac{\boldsymbol{R}}{\Delta t} & \frac{2\boldsymbol{R}}{\Delta t^2} \end{bmatrix} \tag{6.14}$$

Here, z_{-1} is the first received measurement, z_0 is the second and Δt is the time passed between measurements. Thus, $k = 0$ indicates the time step at which the KF is properly initialized and yields the first valid estimate.

Bar-Shalom et al. [49] highlight that the initial state covariance has to be consistent with the error in the initial state. They recommend to choose a covariance s.t. the error is at most two times the respective standard deviation.

6.2.3 N-th Order Kalman Filter

The choice of the system model is crucial for the performance of a KF. One option to acquire the linear state space model is the derivation via differential equations. When there is no further knowledge about the system available, a kind of black box approach, one can still make use of the Newtonian equations. This is especially helpful for continuous processes, i.e., where constancy of a derivative of the tracked signal can be assumed. The order of the system is determined by the order of derivatives contained in the differential equation. The content of this section is based on [24] unless stated otherwise.

Zeroth Order Kalman Filter The zeroth order KF can also be described as constant position model. It assumes that there is no change in the state x, i.e., x has a constant value:

$$x = x_0 \tag{6.15}$$

with position x. This gives the following state space formulation:

$$\boldsymbol{x} = [x] \tag{6.16}$$
$$\boldsymbol{z} = [z] \tag{6.17}$$
$$\boldsymbol{A} = [1] \tag{6.18}$$
$$\boldsymbol{C} = [1] \tag{6.19}$$

First Order Kalman Filter The first order KF is also known as constant velocity model. As the name suggests, it assumes a motion with constant velocity of the form:

$$v = \frac{\mathrm{d}x}{\mathrm{d}t} \tag{6.20}$$
$$x = vt + x_0 \tag{6.21}$$
$$x_t = x_{t-1} + v_{t-1}\Delta t \tag{6.22}$$
$$v_t = v_{t-1} \tag{6.23}$$

with position x, velocity v, time t and the sampling period of the KF Δt.

This results in the following state space formulation:

$$\boldsymbol{x} = \begin{bmatrix} x \\ \dot{x} \end{bmatrix} \tag{6.24}$$

$$\boldsymbol{z} = [z] \tag{6.25}$$

$$\boldsymbol{A} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \tag{6.26}$$

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{6.27}$$

Second Order Kalman Filter Analogously, the second order KF or constant acceleration model can be defined with the following equations:

$$a = \frac{\mathrm{d}^2 x}{\mathrm{d}t^2} \tag{6.28}$$

$$x = \frac{1}{2}at^2 + v_0 t + x_0 \tag{6.29}$$

$$x_t = x_{t-1} + v_{t-1}\Delta t + \frac{1}{2}a_{t-1}\Delta t^2 \tag{6.30}$$

$$v_t = v_{t-1} + a_{t-1}\Delta t \tag{6.31}$$

$$a_t = a_{t-1} \tag{6.32}$$

with position x, velocity v, acceleration a, time t and the sampling period of the KF Δt. Which results in the following state space formulation:

$$\boldsymbol{x} = \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} \tag{6.33}$$

$$\boldsymbol{z} = [z] \tag{6.34}$$

$$\boldsymbol{A} = \begin{bmatrix} 1 & \Delta t & \Delta t^2 \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{bmatrix} \tag{6.35}$$

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \tag{6.36}$$

This derivation of kinematic systems using Newtonian equations can be arbitrarily extended to e.g. jerk, snap, crackle, pop and so on. Labbe [24] states, for the best performance the filter order needs to match the system's order. Usually, adding terms beyond the dynamic of the real system degrades the estimate. Furthermore, a lower order filter can track a higher order system. This requires an increased process noise but often provides good results. The author [24] also suggests to experiment with this approach before considering a more complex adaptive approach.

6.2.4 Covariance Estimation

The choice of covariance matrices are crucial for the KF performance. The state covariance matrix $\boldsymbol{P}$ does not necessarily represent the error covariance of the states. When the filter is overconfident of its estimate the filter is called "smug". In practice, the estimated state covariance $\boldsymbol{P}$ depends solely on the definitions of $\boldsymbol{Q}$ and $\boldsymbol{R}$. Therefore, $\boldsymbol{P}$ is the result of the model which highly depends on the assumptions and accuracy of $\boldsymbol{Q}$ and $\boldsymbol{R}$. Real systems and sensors typically do not have a Gaussian error with zero mean but kurtosis and skew. [24]

The matrices $\boldsymbol{R}$ and $\boldsymbol{Q}$ can be acquired from sensor specification, derived from measurements or applicated in a trial and error approach. In this way $\boldsymbol{R}$ and $\boldsymbol{Q}$ are design parameters of the KF [25]. Labbe [24] and Marchthaler and Dingler [26] describe the following approaches to approximate the process covariance matrix $\boldsymbol{Q}$:

Continuous White Noise Model The basic idea of the continuous model is the discretization of the continuous white noise $\boldsymbol{Q}_c$. The process covariance $\boldsymbol{Q}$ can be computed with:

$$\boldsymbol{Q} = \int_0^{\Delta t} \boldsymbol{A}(t)\boldsymbol{Q}_c\boldsymbol{A}(t)^\top \mathrm{d}t \tag{6.37}$$

where $\boldsymbol{A}(t)$ is the state transition matrix at time t. $\boldsymbol{Q}_c$ depends on the spectral density Φ_s of the white noise. It is placed at the position in the matrix where the expected disturbance occurs. For example, with the constant acceleration assumption it models the deviation from the constancy as white Gaussian noise, s.t.:

$$\boldsymbol{Q}_c = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \Phi_s \tag{6.38}$$

In practice, it is difficult to estimate Φ_s. Therefore, it is often used as tuning parameter that is applicated in experiments. The covariance matrices $\boldsymbol{Q}_i$ for the filter order i are defined as follows:

$$\boldsymbol{Q}_0 = \begin{bmatrix}\Delta t\end{bmatrix} \Phi_s \qquad \boldsymbol{Q}_1 = \begin{bmatrix} \frac{\Delta t^3}{3} & \frac{\Delta t^2}{2} \\ \frac{\Delta t^2}{2} & \Delta t \end{bmatrix} \Phi_s \qquad \boldsymbol{Q}_2 = \begin{bmatrix} \frac{\Delta t^5}{20} & \frac{\Delta t^4}{8} & \frac{\Delta t^3}{6} \\ \frac{\Delta t^4}{8} & \frac{\Delta t^3}{3} & \frac{\Delta t^2}{2} \\ \frac{\Delta t^3}{6} & \frac{\Delta t^2}{2} & \Delta t \end{bmatrix} \Phi_s \tag{6.39}$$

Piecewise White Noise Model This model assumes a constant highest order term for each period of time, i.e., the discontinuities have random, uncorrelated values with zero mean. In contrast, the continuous model assumes continuously varying noise. In this model, the impact of the piecewise constant disturbance $\mathbf{w}$ is defined with the noise gain $\boldsymbol{G}$:

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{G}\mathbf{w} \tag{6.40}$$

The resulting process covariance matrix is defined as:

$$\boldsymbol{Q} = \mathbb{E}[\boldsymbol{G}\mathbf{w}\mathbf{w}^\top\boldsymbol{G}^\top] \tag{6.41}$$
$$= \boldsymbol{G}\mathrm{Cov}[\mathbf{w}]\boldsymbol{G}^\top \tag{6.42}$$

For example, in the constant velocity case, the noise affects the velocity as discontinuous jump in acceleration which yields $\boldsymbol{G} = [\frac{1}{2}\Delta t, \Delta t]^\top$ and the covariance matrix $\mathrm{Cov}[\mathbf{w}] = [\sigma_a^2]$ with the acceleration variance σ_a^2. As a rule of thumb, the author [24] suggests to choose σ_a between $\frac{1}{2}\Delta a$

and Δa, with Δa as the maximum change of acceleration between samples. In practice, the value is often applicated in experiments. The resulting covariance matrices $\boldsymbol{Q}_i$ for the filter order i are:

$$Q_0 = \begin{bmatrix}\Delta t^2\end{bmatrix}\sigma_v^2 \qquad Q_1 = \begin{bmatrix}\frac{\Delta t^4}{4} & \frac{\Delta t^3}{2}\\ \frac{\Delta t^3}{2} & \Delta t^2\end{bmatrix}\sigma_a^2 \qquad Q_2 = \begin{bmatrix}\frac{\Delta t^6}{36} & \frac{\Delta t^5}{12} & \frac{\Delta t^4}{6}\\ \frac{\Delta t^5}{12} & \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2}\\ \frac{\Delta t^4}{6} & \frac{\Delta t^3}{2} & \Delta t^2\end{bmatrix}\sigma_j^2 \tag{6.43}$$

Direct Discretization Model The direct discretization is similar to the piecewise model, but here the disturbance directly affects the highest order term in the filter. This gives the following covariance matrices $\boldsymbol{Q}_i$ for the filter order i:

$$Q_0 = \begin{bmatrix}1\end{bmatrix}\sigma_p^2 \qquad Q_1 = \begin{bmatrix}\Delta t^2 & \Delta t\\ \Delta t & 1\end{bmatrix}\sigma_v^2 \qquad Q_2 = \begin{bmatrix}\frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} & \frac{\Delta t^2}{2}\\ \frac{\Delta t^3}{2} & \Delta t^2 & \Delta t\\ \frac{\Delta t^2}{2} & \Delta t & 1\end{bmatrix}\sigma_a^2 \tag{6.44}$$

Simplification of $\boldsymbol{Q}$ In this approach, the process covariance matrix is strongly simplified and only models the highest order term as the variance σ^2. Given the constant acceleration assumption $\boldsymbol{Q}$ can be defines as:

$$Q = \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & \sigma_a^2\end{bmatrix} \tag{6.45}$$

Labbe [24] reasons, this represents an approximation of the previous approach, since for small Δt all other terms tend towards zero. Furthermore, in the prediction state covariance update the contribution of those components is negligible if the values of $\boldsymbol{Q}$ are small compared to $\boldsymbol{P}$. Although this approach is strictly speaking not correct, the author [24] highlights is usefulness. Nevertheless, he also warns that the developed filter might not work as intended and recommends to spend more validation effort as compensation. In [25] an example for a constant velocity KF is presented which uses a diagonal matrix as $\boldsymbol{Q}$. Thus, neglecting only the covariance terms is also a simplification option.

Comparison of $\boldsymbol{Q}$ Estimation Methods With a trick from Marchthaler and Dingler [26] the estimation models can be compared: the lower right entry of $\boldsymbol{Q}$ models by definition the variance of the noise of the highest order term. Therefore, the presented matrices can be normalized and compared. For the continuous white noise model $\Delta t\Phi_s$ is substituted with σ^2. The piecewise model is transformed with $\sigma_{i-1}^2 = \Delta t^2\sigma_i^2$ where i denotes the order of the noise term. The direct discretisation model already has the desired form.

Continuous White Noise Model:

$$Q_0 = \begin{bmatrix}1\end{bmatrix} \sigma_p^2 \qquad Q_1 = \begin{bmatrix} \frac{\Delta t^2}{3} & \frac{\Delta t}{2} \\ \frac{\Delta t}{2} & 1 \end{bmatrix} \sigma_v^2 \qquad Q_2 = \begin{bmatrix} \frac{\Delta t^4}{20} & \frac{\Delta t^3}{8} & \frac{\Delta t^2}{6} \\ \frac{\Delta t^3}{8} & \frac{\Delta t^2}{3} & \frac{\Delta t}{2} \\ \frac{\Delta t^2}{6} & \frac{\Delta t}{2} & 1 \end{bmatrix} \sigma_a^2 \tag{6.46}$$

Piecewise White Noise Model:

$$Q_0 = \begin{bmatrix}1\end{bmatrix} \sigma_p^2 \qquad Q_1 = \begin{bmatrix} \frac{\Delta t^2}{4} & \frac{\Delta t}{2} \\ \frac{\Delta t}{2} & 1 \end{bmatrix} \sigma_v^2 \qquad Q_2 = \begin{bmatrix} \frac{\Delta t^4}{36} & \frac{\Delta t^3}{12} & \frac{\Delta t^2}{6} \\ \frac{\Delta t^3}{12} & \frac{\Delta t^2}{4} & \frac{\Delta t}{2} \\ \frac{\Delta t^2}{6} & \frac{\Delta t}{2} & 1 \end{bmatrix} \sigma_a^2 \tag{6.47}$$

Direct Discretization Model:

$$Q_0 = \begin{bmatrix}1\end{bmatrix} \sigma_p^2 \qquad Q_1 = \begin{bmatrix} \Delta t^2 & \Delta t \\ \Delta t & 1 \end{bmatrix} \sigma_v^2 \qquad Q_2 = \begin{bmatrix} \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} & \frac{\Delta t^2}{2} \\ \frac{\Delta t^3}{2} & \Delta t^2 & \Delta t \\ \frac{\Delta t^2}{2} & \Delta t & 1 \end{bmatrix} \sigma_a^2 \tag{6.48}$$

For the zeroth order filter all methods give the same shape for the covariance matrix. For higher order filters the different approaches result in different scaling factors for the respective matrix elements.

Labbe [24] states that the estimation models are approximations. None of them has a significant benefit over the others. Nevertheless, one might perform better compared to the others, which has to be evaluated in experiments. A benefit of the piecewise model is that it can be intuitively derived from theoretically expected disturbances or measurements. With the normalization approach from [26], this benefit can be applied to the other methods as well. Thus, a value for the highest order variance σ_i^2 can be estimated and multiplied with the desired model matrix.

Covariance Estimation from Samples The covariance matrix is formally defined as [12]:

$$\begin{aligned} \mathrm{Cov}[\mathbf{x}, \mathbf{y}] &= \mathbb{E}[(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{y}^\top - \mathbb{E}[\mathbf{y}^\top])] && (6.49) \\ &= \mathbb{E}[\mathbf{x}\mathbf{y}^\top] - \mathbb{E}[\mathbf{x}]\mathbb{E}[\mathbf{y}^\top] && (6.50) \end{aligned}$$

For a zero mean random vector **w** this gives the following equation that can be approximated with the sample mean of measured values:

$$\begin{aligned} \mathrm{Cov}[\mathbf{w}] &= \mathrm{Cov}[\mathbf{w}, \mathbf{w}] && (6.51) \\ &= \mathbb{E}[\mathbf{w}\mathbf{w}^\top] - \underbrace{\mathbb{E}[\mathbf{w}]\mathbb{E}[\mathbf{w}^\top]}_{0} && (6.52) \\ &\approx \frac{1}{N} \sum_{i}^{N} \mathbf{w}_i \mathbf{w}_i^\top && (6.53) \end{aligned}$$

6.2.5 Filter Evaluation

The KF performance measure is different from other estimators. For the resulting estimation error classical metrics like the MSE or RMSE can be applied if the ground truth is known. These metrics are useful because the KF provides an optimal estimate in the least square sense. [24]

In addition the KF provides an estimate of the state estimation error in the form of the state covariance matrix $\boldsymbol{P}$. As already mentioned, $\boldsymbol{P}$ does not necessarily represent the error, because the filter can over- and underestimate the real deviation from the ground truth. A helpful tool for performance estimation is plotting the residuals and the estimated standard deviation. From the Gaussian perspective, approximately 68 % of the residuals should lie within the 1σ range. [24]

The Normalized Estimation Error Squared (NEES) is a useful metric for the performance evaluation if the ground truth signal is available. The NEES ϵ_k is defined as:

$$\tilde{\boldsymbol{x}}_k = \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k \tag{6.54}$$

$$\epsilon_k = \tilde{\boldsymbol{x}}_k^\top \boldsymbol{P}_k^{-1} \tilde{\boldsymbol{x}}_k \tag{6.55}$$

with the ground truth $\boldsymbol{x}_k$, the filter estimate $\hat{\boldsymbol{x}}_k$ and the estimation error $\tilde{\boldsymbol{x}}_k$. It can be interpreted more intuitively in the one-dimensional case:

$$\epsilon = \frac{\tilde{x}_k^2}{P_k} \tag{6.56}$$

$$= \frac{\tilde{x}_k^2}{\sigma_k^2} \tag{6.57}$$

Thus, the NEES describes the actual state residual normalized by the estimated variance. A random variable of the form $\tilde{\boldsymbol{x}}^\top \boldsymbol{P}^{-1} \tilde{\boldsymbol{x}}$ is χ^2 distributed with n degrees of freedom, where n is the dimension of $\boldsymbol{x}$. Therefore, its expected value is n. Consequently, the average $\bar{\epsilon}$ can be computed and compared to its expected value n. [24]

In cases where no ground truth is available, the Normalized Innovation Squared (NIS) can be applied analogously to the NEES but with the innovation $\boldsymbol{y}_k$ and innovation covariance matrix $\boldsymbol{S}_k$ [49]:

$$\epsilon_{\mathrm{v}_k} = \boldsymbol{y}_k^\top \boldsymbol{S}_k^{-1} \boldsymbol{y}_k \tag{6.58}$$

Labbe [24] suggests to compute the average and test whether $\bar{\epsilon} < n$. There exist advanced consistency evaluation approaches for the KF in form of hypothesis testing which are beyond the scope of this thesis [49].

6.2.6 Computational Complexity

Murphy [16] states, the computational complexity of the KF is in $O(N^3)$. Thrun et al. [50] further elaborate that the complexity of today's best known algorithms for inversion of a $d \times d$ matrix is approximately $O(d^{2.8})$. Thus, the complexity of the KF update is approximately cubic in size m of the measurement vector z. With n, the size of the state vector, the complexity is at least bound in $O(n^2)$ due to the matrix multiplication. Furthermore, Thrun et al. add that for applications where the measurement space is much lower than the state space, $m \ll n$, $O(n^2)$ dominates the computational cost.

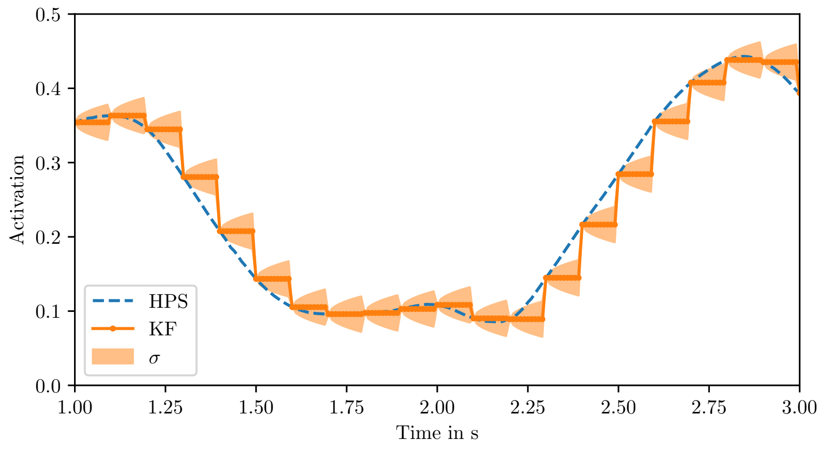
6.3 Kalman Filter Tracking

First, the KF is applied to Scenario I, where the goal is to estimate intermediate results when only sparse updates of the HPS model are available. The intermediate results are produced with the help of a tracking KF. The basic idea is to make use of the prediction step for time steps where no “measurement” from the HPS is available. In this scenario the KF is applied as time series forecasting model. Since the HPS is interpreted as ground truth model, the measurement covariance matrix $\boldsymbol{R}$ should be zero. Nonetheless, very small values are used to avoid numerical issues, e.g., a singular covariance matrix. Thus, $\boldsymbol{R}$ is chosen to be $\sigma_{\text{HPS}}\boldsymbol{I}$, e.g., with $\sigma_{\text{HPS}} = 1\text{e}{-9}$. This approximation is acceptable as long as $\boldsymbol{R}$ is reasonably small compared to the process covariance matrix $\boldsymbol{Q}$. The filter is initialized with $\boldsymbol{x} = \boldsymbol{0}$ and $\boldsymbol{P} = \sigma_{\text{init}}\boldsymbol{I}$ with a large enough σ_{init} to model the uncertainty. Due to the very low measurement covariance the KF converges nearly instantaneous on measurement arrival. Nonetheless, for implementation in the final application a proper initialization scheme, as described in Section 6.2.2, should be chosen. Because of the computational complexity, presented in Section 6.2.6, each target variable is tracked independently with an individual KF. In this way, the complexity is linear in the number of features instead of cubic. For independent targets this approach yields equivalent estimates.

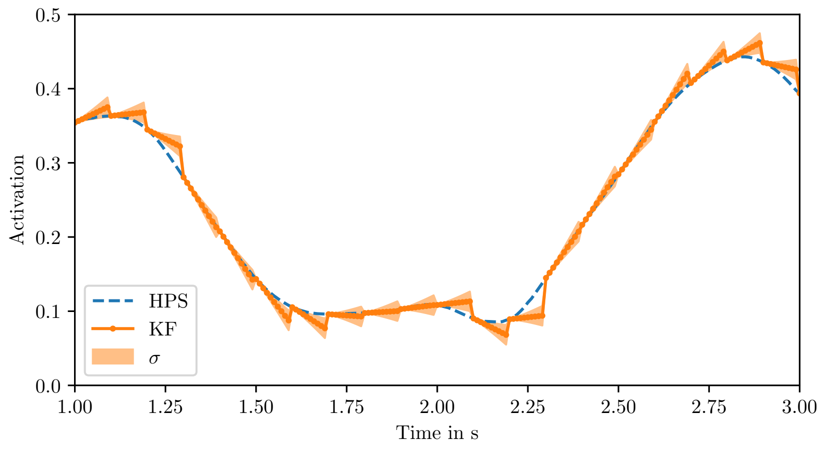
As black box kinematic system model the n-th order KF is applied as described in Section 6.2.3. The zeroth order KF or constant position model produces a constant prediction. This behavior is similar to the naïve forecasting, which simply predicts the latest measurement value [21]. The constant velocity model extends the prediction model with a velocity component. Thus, the predicted values are a linear function over time. A similar behavior can be acquired with the Holt’s forecasting method which uses the same prediction model, but applies exponential smoothing for state estimation [21]. Analogously, the constant acceleration model predicts a parabola-shaped course. An exemplary course of the different tracking filters is visualized in Figure 6.2. The plots show that each order has its own strengths and weaknesses. The filter performs best when the type of target motion is matching the constancy assumption, e.g., a first order filter performs best when the target velocity is actually constant and a second order filter performs better in phases with constant acceleration.

6.4 Kalman Filter Fusion

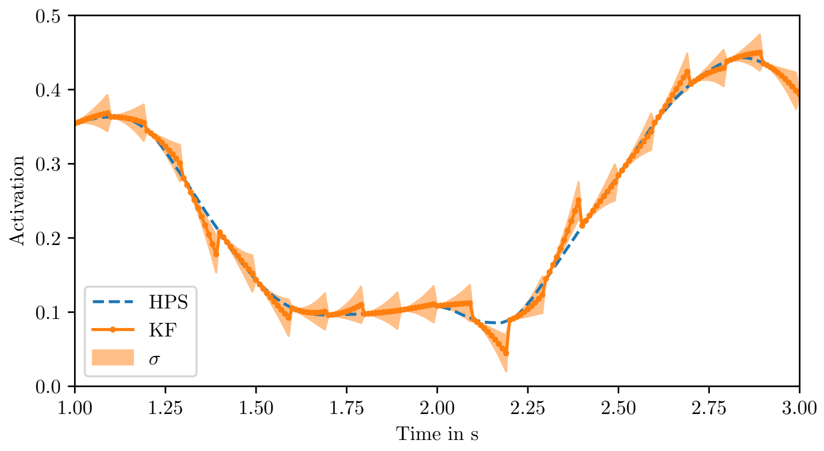
Next, the KF is applied in Scenario II, which extends Scenario I with a LPS model on the mobile device. Thus, the goal of the fusion is to combine simulation results from the LPS and HPS. The LPS results are available in every cycle, while the HPS updates arrive with a lower frequency.



(a) Example course of the constant position KF.



(b) Example course of the constant velocity KF.



(c) Example course of the constant acceleration KF.

Figure 6.2: Example courses of the first three n-th order KFs. The tracked target is the activation value of the biceps. The HPS update is fused every tenth time step. The KF has a sample time of $\Delta t = 10\,\text{ms}$. σ represents the estimated error standard deviation of the KF.

The standard way of fusing measurements from different sources is the augmented measurement vector. For example, given two uncorrelated measurements z_1 and z_2, the state space formulation of a zeroth order KF can be defined as:

$$\boldsymbol{x} = [x] \tag{6.59}$$

$$\boldsymbol{A} = [1] \tag{6.60}$$

$$\boldsymbol{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \tag{6.61}$$

$$\boldsymbol{R} = \begin{bmatrix} \sigma_{z_1}^2 & 0 \\ 0 & \sigma_{z_2}^2 \end{bmatrix} \tag{6.62}$$

$$\boldsymbol{C} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{6.63}$$

According to Brown and Hwang [51] it is also possible to fuse independent measurements in a sequential manner. This is convenient for the given scenario. Instead of augmentation of the KF matrices, the HPS measurement can be fused on arrival in a second update run. Each measurement is fused with its respective measurement covariance $\boldsymbol{R}$. Therefore, an execution sequence could look like the following example:

```
k = 0  KF.predict()
       KF.update($z_{\text{LPS}}$, $\boldsymbol{R}_{\text{LPS}}$)
k = 1  KF.predict()
       KF.update($z_{\text{HPS}}$, $\boldsymbol{R}_{\text{HPS}}$)
       KF.update($z_{\text{LPS}}$, $\boldsymbol{R}_{\text{LPS}}$)
k = 2  KF.predict()
       KF.update($z_{\text{LPS}}$, $\boldsymbol{R}_{\text{LPS}}$)
k = 3  ...
```

As system model, a desired order KF can be chosen as in Scenario I. As in Section 6.3, $\boldsymbol{R}_{\text{HPS}}$ is chosen with a very small value because it is the ground truth. The error covariance $\boldsymbol{R}_{\text{LPS}}$ can be estimated from the test set residuals or residuals over the simulated trajectory with Equation (2.17). To track the targets individually, independence is assumed, i.e., only the variances, the diagonal elements, are used. With the underlying zero mean assumption, the estimated variances are equivalent to the MSE. If the residuals do not have a true zero mean in practice, the values of the estimated variances are implicitly increased which is good because it indicates a higher uncertainty. However, a bias violates the KF requirements and can lead to divergence of the estimate. In fact, the underfitted polynomial regression LPS models with low degree show a high bias and low variance over the simulation trajectory. This issue is addressed next in Section 6.5.

6.5 Kalman Filter Fusion with a Biased Model

According to Marchthaler and Dingler [26], a common approach to deal with a biased measurement z_{LPS} is to augment the state vector. The key idea is to add the offset o as additional state. In this way, the KF estimates the offset during execution. An example for this approach is the offset compensation of an acceleration sensor with slow offset drift in the context of velocity and position estimation. A possible zeroth order KF can be defined as:

$$\boldsymbol{x} = \begin{bmatrix} x \\ x_{\text{offset}} \end{bmatrix} \tag{6.64}$$

$$\boldsymbol{z} = \begin{bmatrix} z_{\text{HPS}} \\ z_{\text{LPS}} \end{bmatrix} \tag{6.65}$$

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{6.66}$$

$$\boldsymbol{R} = \begin{bmatrix} \sigma^2_{z_{\text{HPS}}} & 0 \\ 0 & \sigma^2_{z_{\text{LPS}}} \end{bmatrix} \tag{6.67}$$

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix} \tag{6.68}$$

While this system is observable, an issue arises in the setting with lower frequent HPS updates. When the system is split in sequential updates as in Section 6.4 the LPS system is not observable anymore:

$$\boldsymbol{z} = [z_{\text{LPS}}] \tag{6.69}$$

$$\boldsymbol{R} = [\sigma^2_{z_{\text{LPS}}}] \tag{6.70}$$

$$\boldsymbol{C} = \begin{bmatrix} 1 & -1 \end{bmatrix} \tag{6.71}$$

According to the observability criterion of Kalman, a linear time invariant system is observable if and only if the observability matrix $\boldsymbol{O}$ has rank n, where n is the order of the system [26]:

$$\boldsymbol{O} = \begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{CA} \\ \vdots \\ \boldsymbol{CA}^{n-1} \end{bmatrix} \tag{6.72}$$

$$\tag{6.73}$$

Proof of (un)observability:

$$\boldsymbol{O}_{\text{full}} = \begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{CA} \end{bmatrix} \tag{6.74}$$

$$= \begin{bmatrix} 1 & 0 \\ 1 & -1 \\ 1 & 0 \\ 1 & -1 \end{bmatrix} \tag{6.75}$$

$$\text{rank}(\boldsymbol{O}_{\text{full}}) = 2 \tag{6.76}$$

$$\boldsymbol{O}_{\text{LPS}} = \begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{CA} \end{bmatrix} \tag{6.77}$$

$$= \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} \tag{6.78}$$

$$\text{rank}(\boldsymbol{O}_{\text{LPS}}) = 1 \tag{6.79}$$

For the system to be observable, the rank of $\boldsymbol{O}_{\text{LPS}}$ has to be two as the full system with $\boldsymbol{O}_{\text{full}}$. Since $\text{rank}(\boldsymbol{O}_{\text{LPS}}) = 1 \neq 2$ this concludes the proof that the system is not observable. A prototypical implementation shows that ignoring this fact results in a drift of the position and offset estimates in the unobservable phase without HPS updates. Therefore, an alternative solution is required.

The fusion strategy from Hubatscheck [4], presented and improved in Section 6.1, provides good results in the presence of delays. It is very similar to the polynomial regression delta forecasting approach, developed in [34], where the future values are predicted with the help of a polynomial regression surrogate model. The approaches are mathematically identical regarding the continued update, although they have been developed for different purposes. The key idea is that the prediction course is the result of the latest HPS sampling point plus the change of the LPS model Δ_l over time. For $\alpha = 1$ the forecast solely depends on the continued update c_{t+h}:

$$c_{t+h} = u_t + \underbrace{l_{t+h} - l_t}_{\Delta_l} \tag{6.80}$$

with the HPS update u_t and LPS result l_t. This concept can be converted in KF language. Assuming that the state x accurately represents the HPS result the change in the LPS Δ_l can be fed into the KF as control input $\boldsymbol{u}$, s.t.:

$$\boldsymbol{x} = \begin{bmatrix} x \end{bmatrix} \tag{6.81}$$

$$\boldsymbol{z} = \begin{bmatrix} z_{\text{HPS}} \end{bmatrix} \tag{6.82}$$

$$\boldsymbol{u} = \begin{bmatrix} \frac{z_{\text{LPS}_k} - z_{\text{LPS}_{k-1}}}{\Delta t} \end{bmatrix} \tag{6.83}$$

$$\boldsymbol{A} = \begin{bmatrix} 1 \end{bmatrix} \tag{6.84}$$

$$\boldsymbol{B} = \begin{bmatrix} \Delta t \end{bmatrix} \tag{6.85}$$

$$\boldsymbol{R} = \begin{bmatrix} \sigma^2_{z_{\text{HPS}}} \end{bmatrix} \tag{6.86}$$

$$\boldsymbol{C} = \begin{bmatrix} 1 \end{bmatrix} \tag{6.87}$$

We call this approach Kalman filter with constant position tracking and input augmentation fusion (KF$_{\text{CPU}}$). In fact, the special case $\alpha = 1$ of the continued update strategy is equivalent to the KF$_{\text{CPU}}$ with $\boldsymbol{R} = [0]$.

Proof:

Update:

$$\boldsymbol{S}_{k+1} = \underbrace{\boldsymbol{C}}_{\boldsymbol{I}} \boldsymbol{P}_{k+1|k} \underbrace{\boldsymbol{C}^\top}_{\boldsymbol{I}} + \underbrace{\boldsymbol{R}}_{[0]} \tag{6.88}$$

$$\boldsymbol{K}_{k+1} = \boldsymbol{C}\boldsymbol{P}_{k+1|k}\boldsymbol{C}^\top \boldsymbol{S}_{k+1}^{-1} \tag{6.89}$$

$$= \boldsymbol{P}_{k+1|k}\boldsymbol{S}_{k+1}^{-1} \tag{6.90}$$

$$= \underbrace{\boldsymbol{P}_{k+1|k}\boldsymbol{P}_{k+1|k}^{-1}}_{\boldsymbol{I}} \tag{6.91}$$

$$\boldsymbol{y}_{k+1} = \boldsymbol{z}_{k+1} - \boldsymbol{C}\hat{\boldsymbol{x}}_{k+1|k} \tag{6.92}$$

$$= \boldsymbol{z}_{k+1} - \hat{\boldsymbol{x}}_{k+1|k} \tag{6.93}$$

$$\hat{\boldsymbol{x}}_{k+1} = \hat{\boldsymbol{x}}_{k+1|k} + \boldsymbol{K}_{k+1}\boldsymbol{y}_{k+1} \tag{6.94}$$

$$= \hat{\boldsymbol{x}}_{k+1|k} + \boldsymbol{y}_{k+1} \tag{6.95}$$

$$= \hat{\boldsymbol{x}}_{k+1|k} + \boldsymbol{z}_{k+1} - \hat{\boldsymbol{x}}_{k+1|k} \tag{6.96}$$

$$= \boldsymbol{z}_{k+1} \tag{6.97}$$

$$\boldsymbol{P}_{k+1} = (\boldsymbol{I} - \boldsymbol{K}_{k+1}\boldsymbol{C})\boldsymbol{P}_{k+1|k} \tag{6.98}$$

$$= [0] \tag{6.99}$$

Prediction:

$$\hat{\boldsymbol{x}}_{k+1|k} = \boldsymbol{A}\hat{\boldsymbol{x}}_k + \boldsymbol{B}\boldsymbol{u}_k \tag{6.100}$$

$$= [\hat{x}_k] + [\Delta t]\left[\frac{z_{\text{LPS}_k} - z_{\text{LPS}_{k-1}}}{\Delta t}\right] \tag{6.101}$$

$$= [\hat{x}_k + z_{\text{LPS}_k} - z_{\text{LPS}_{k-1}}] \tag{6.102}$$

The update with $\boldsymbol{R} = [0]$ leads to $\hat{x}_{k+1} = z_{k+1}$ which is continued in the next prediction with the one step difference of the LPS $z_{\text{LPS}_k} - z_{\text{LPS}_{k-1}}$. This concludes the proof that KF_{CPU} with $\boldsymbol{R} = [0]$ is equivalent to the continued update strategy with $\alpha = 1$.

To this point using the KF looks like a very complicated variant of a simple approach. The benefit lies in the extension to higher order models. The control signal $\boldsymbol{u}$ contains the discrete derivative of z_{HPS} over time. Consequently, a second and third order KF can be defined analogously.

Kalman filter with constant velocity tracking and input augmentation fusion (KF$_{\mathrm{CVU}}$):

$$\boldsymbol{x} = \begin{bmatrix} x \\ \dot{x} \end{bmatrix} \tag{6.103}$$

$$\boldsymbol{z} = \begin{bmatrix} z_{\mathrm{HPS}} \end{bmatrix} \tag{6.104}$$

$$\boldsymbol{u} = \begin{bmatrix} \frac{\mathrm{d}^2 z_{\mathrm{LPS}}}{\mathrm{d}t^2} \end{bmatrix} \tag{6.105}$$

$$\boldsymbol{A} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \tag{6.106}$$

$$\boldsymbol{B} = \begin{bmatrix} \frac{\Delta t^2}{2} \\ \Delta t \end{bmatrix} \tag{6.107}$$

$$\boldsymbol{R} = \begin{bmatrix} \sigma^2_{z_{\mathrm{HPS}}} \end{bmatrix} \tag{6.108}$$

$$\boldsymbol{C} = \begin{bmatrix} 1, 0 \end{bmatrix} \tag{6.109}$$

Kalman filter with constant acceleration tracking and input augmentation fusion (KF$_{\mathrm{CAU}}$):

$$\boldsymbol{x} = \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} \tag{6.110}$$

$$\boldsymbol{z} = \begin{bmatrix} z_{\mathrm{HPS}} \end{bmatrix} \tag{6.111}$$

$$\boldsymbol{u} = \begin{bmatrix} \frac{\mathrm{d}^3 z_{\mathrm{LPS}}}{\mathrm{d}t^3} \end{bmatrix} \tag{6.112}$$

$$\boldsymbol{A} = \begin{bmatrix} 1 & \Delta t & \frac{\Delta t^2}{2} \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{bmatrix} \tag{6.113}$$

$$\boldsymbol{B} = \begin{bmatrix} \frac{\Delta t^3}{6} \\ \frac{\Delta t^2}{2} \\ \Delta t \end{bmatrix} \tag{6.114}$$

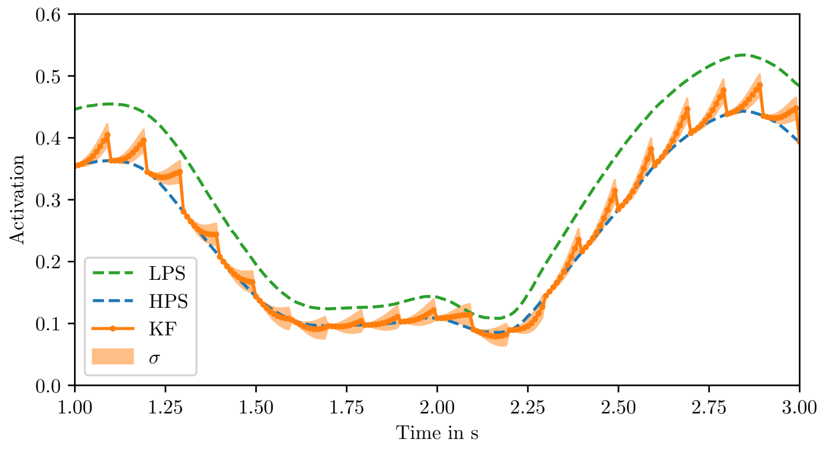
$$\boldsymbol{R} = \begin{bmatrix} \sigma^2_{z_{\mathrm{HPS}}} \end{bmatrix} \tag{6.115}$$

$$\boldsymbol{C} = \begin{bmatrix} 1, 0, 0 \end{bmatrix} \tag{6.116}$$

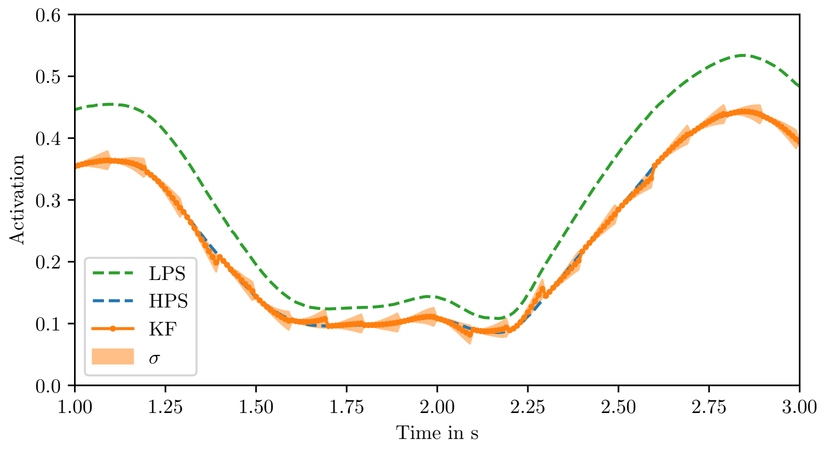
As in the zeroth order case, the derivatives of z_{LPS} can be approximated with the discrete difference approach. Higher orders can be defined analogously.

The process covariance matrix $\boldsymbol{Q}$ can be estimated from simulated or measured data in the following way. If $\boldsymbol{u}$ accurately captures the HPS dynamic, the estimate of the KF is perfect. Thus, the difference between $\boldsymbol{u}$ and the same order derivative of z_{HPS} is the origin of the process error one order above the filter order. Thus, $\boldsymbol{Q}$ can be shaped with the presented models in Section 6.2.4 and scaled with $\sigma_i^2 = \Delta t^2 \sigma_{i+1}^2$ where i is the filter order. The variance σ_{i+1}^2 can be computed with the difference of the derivatives and Equation (6.53).

The classical fusion, presented in Section 6.4, and the bias compensated KF$_{\mathrm{CVU}}$ are exemplary visualized in Figure 6.3. The LPS estimate suffers from a drifting bias compared to the HPS. If this bias is not compensated it makes the prediction even worse at some points because it draws



(a) Example course of the constant velocity KF with classical fusion.



(b) Example course of the constant velocity fusion $\mathrm{KF_{CPU}}$ with input augmentation fusion.

Figure 6.3: Example courses of first order KF fusion with a biased model. The tracked target is the activation value of the biceps. The HPS update is fused every tenth time step. The KF has a sample time of $\Delta t = 10\,\mathrm{ms}$. σ represents the estimated error standard deviation of the KF.

the state estimate away from the ground truth. In contrast, the $\mathrm{KF_{CVU}}$ successfully exploits change information from the LPS. A drawback of this approach is that it can drift over time, i.e., if HPS updates are rare or not available for some time a fallback strategy is advised.

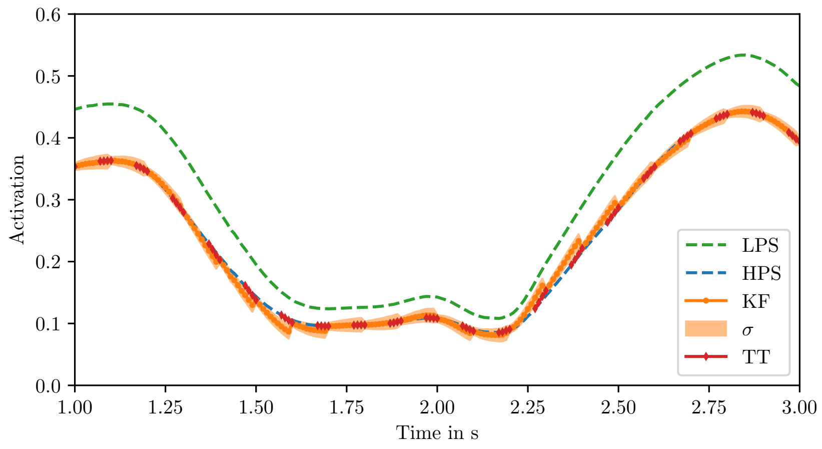


Figure 6.4: Example course of the KF_{CPU} in the presence of delay. The delay is compensated with the BTTF approach. The tracked target is the activation value of the biceps. The HPS update is fused every tenth time step with a constant delay of 30 ms. The KF has a sample time of $\Delta t = 10$ ms. σ represents the estimated error standard deviation of the KF. TT shows the hidden course of the time travel of the BTTF KF.

6.6 Incorporation of Delayed Measurements

In Scenario III transmission delays and message losses are added to the previous scenarios. Many approaches exist in the literature to deal with delayed measurements since it is a natural challenge that arises in real world multi-sensor systems. The delayed measurement problem is also known as Out-of-Sequence Measurement (OOSM) problem or negative-time measurement problem [52]. An overview of available OOSM approaches can be found in [53–55].

The classical KF requires measurements to arrive in order. For the AR application a minimum delay is preferable. Thus, buffering measurements, as suggested in [53], and waiting for the HPS arrival is not an option because it delays the output and therefore violates the low-latency requirement. Neglecting the HPS updates is obviously no option as well since it degrades the system to a LPS only execution.

According to [56], the recalculation of the KF through the delay period yields an optimal estimate. After arrival of the delayed measurement the KF produces the same estimates as if the measurement has not been delayed. Informally, the KF jumps back in time, fuses the measurement and relives the past to the current time step by replaying the history of inputs to the KF. Thus, it is called Back to the Future (BTTF) approach in this thesis. Especially for large delays the approach is computationally expensive and requires memory to store the history of states and relevant KF inputs. The BTTF approach is exemplary visualized in Figure 6.4. In this thesis the BTTF strategy is implemented s.t. it only remembers the latest measurement per time step. This is a reasonable simplification because the later arriving HPS update replaces the LPS measurement. It is acceptable since the HPS update is dominant with a near zero covariance matrix.

Another elegant but expensive approach is the so called state space augmentation, described by [55, 57, 58]. The basic idea is the usage of a fixed-lag smoother [48] with a time-varying measurement matrix $\boldsymbol{C}$. There exist different variants, the state augmentation, the measurement augmentation and the sample state augmentation [55]. The common concept is the explicit addition of lagged variables to the state vector $\boldsymbol{x}$. In case of the measurement augmentation lagged measurements are added to the state. This is helpful, if the measurement vector $\boldsymbol{z}$ is smaller than $\boldsymbol{x}$. The sample state augmentation dynamically augments the state on measurement creation and arrival, i.e., it adds and removes states from the state vector. Gopalakrishnan et al. [55] state that all three augmentation approaches are expected to give identical state estimates, equivalent to the filter recalculation, for a linear system. In this thesis, the Measurement Augmentation (MA) from [58] based on [57] is used, because in the given use case, the measurement vector size is always smaller or equal to the state vector dimension. The augmented system is defined as:

$$\boldsymbol{x}'_k = \begin{bmatrix} \boldsymbol{x}_k & \boldsymbol{y}_{k-1} & \dots & \boldsymbol{y}_{k-L} \end{bmatrix}^\top \tag{6.117}$$

$$\boldsymbol{A}' = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{0} & & \cdots & \boldsymbol{0} \\ \boldsymbol{C} & \boldsymbol{0} & & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} & & \boldsymbol{0} \\ \vdots & & \ddots & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & & & \boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \tag{6.118}$$

$$\boldsymbol{B}' = \begin{bmatrix} \boldsymbol{B} \\ \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \end{bmatrix} \tag{6.119}$$

$$\boldsymbol{C}'_k = \begin{bmatrix} \gamma_{0,k}\boldsymbol{C} & \gamma_{1,k}\boldsymbol{I} & \dots & \gamma_{L,k}\boldsymbol{I} \end{bmatrix} \tag{6.120}$$

$$\boldsymbol{Q}' = \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \tag{6.121}$$

$$\tag{6.122}$$

where $\boldsymbol{y}_{k-1}, \dots, \boldsymbol{y}_{k-L}$ are the delayed noise-free true outputs in the augmented state $\boldsymbol{x}$, L is the maximum allowed delay, $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ and $\boldsymbol{Q}$ are the matrices of the original state space system. Measurements with a delay greater than L are dropped. The mask parameter $\gamma_{l,k}$ is used to adjust $\boldsymbol{C}_k$ according to the respective measurement delay d:

$$\gamma_{l,k} = \begin{cases} 1 & l = d \\ 0 & \text{otherwise} \end{cases} \tag{6.123}$$

Note that $\boldsymbol{C}_k$ is time-varying. The augmented state system is applied with the classical KF equations.

There exist other, sub-optimal, but more efficient approximative methods for incorporation of delayed measurements. For example, Larsen et al. [56] propose an approach based on measurement extrapolation or Bar-Shalom et al. [52] present a solution based on retrodiction. Those and further methods remain to be investigates in future work.

7 Evaluation

In this chapter, the developed Kalman Filter (KF)-based approaches are evaluated in the context of three scenarios. Section 7.1 describes the experiment setup. Scenario I and II, as presented in Chapter 6, aim to temporary suspend the computationally expensive HPS execution. The KF can be applied as forecasting model which is investigated in Section 7.2. Scenario III adds the challenge of delays, jitter and losses to the previous scenarios. Consequently, the robustness of the developed approaches in the presence of the named challenges is evaluated in Section 7.3. Preliminary runtime analysis results are presented in Section 7.4, followed by the evaluation summary in Section 7.5.

7.1 Experiment Setup

The pervasive simulation use case is the bio-mechanical simulation of a human arm in an AR environment. The visualization is assumed to require inputs for a fixed frame rate f_V. The human target is tracked by a sensor with the frequency f_S. In this experiment, f_V and f_S are assumed to be equal, i.e., each sensed input produces a simulation result. For a real system f_V and f_S might be different. The data for this experiment has been recorded at a rate f_S of 100 Hz. This rate also lies in the desired range between 60 Hz and 120 Hz for today's AR devices.

The analyzed scenarios are described in detail in Chapter 6. Scenario I aims to temporary suspend the execution of the computationally expensive HPS on the mobile device with the help of forecasting models. Scenario II extends Scenario I with an additional LPS model that can be applied to improve the forecasting quality due to its explanatory nature. Scenario III extends the previous scenarios with delays, jitter and losses. In this distributed scenario, the HPS is deployed on a remote server.

The concrete architecture for the experiment is implemented with the distribution model, presented in Chapter 5. The architecture contains two nodes as depicted in Figure 5.1. The local node represents the mobile simulation device which cyclically schedules the simulation process. To match the frequency of the recorded data, the cycle time of the local process T_L is 10 ms. The remote node implements the simulation server on which the HPS is deployed in Scenario III. The processes communicate via messages. The process on the remote node is message invoked, i.e., it is executed on message arrival from the local process. This models a classical client-server architecture with a request-response pattern. The messages are delivered over uni-directional communication channels. The channels are implemented as head-drop FIFO channels, i.e., all but the newest message are dropped in case of congestion. The channels are assumed to be symmetric, i.e., the forward and backward channels need the same amount of time for message passing.

The emulation of the distributed system is executed with a cycle time T_E of 1 ms. The processing delays of the local process T_{P_L} and the remote process T_{P_L} are chosen the be 2 ms. The local processing time T_{P_L} is neglected in the evaluation between visualization and ground truth, i.e., only additional delays caused by the distribution are considered. This is acceptable because the local

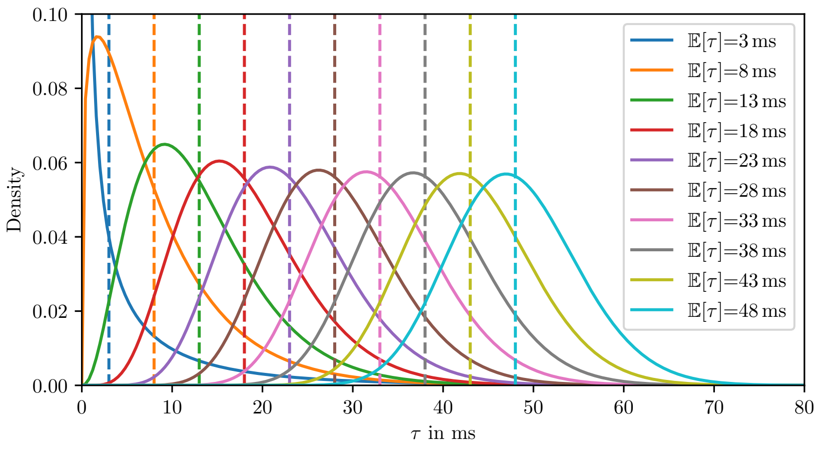


Figure 7.1: Gamma distributed channel delays τ in the experiment. The plot shows the probability density functions for different expected values $\mathbb{E}[\tau]$ for one channel. The variance is constant with $\mathbb{V}[\tau] = 50\,\text{ms}^2$. The dashed lines indicate the position of the expected value of the respective distribution.

process delay is much less than the local process cycle time: $T_{P_\text{L}} \ll T_\text{L}$. The channel delays T_C are derived from a desired Round-Trip Time (RTT) T_RTT:

$$T_\text{C} = \frac{T_\text{RTT} - T_{P_\text{R}}}{2} \tag{7.1}$$

Note that even in the static case, where the channel delays T_C are constant, the resulting RTT may vary due to the fixed cycle time of the local process, i.e., messages might have to wait until they are received, or sender caused congestion, i.e., if the sender overloads the communication system, messages might have to wait for the completion of the transmission of the preceding message.

To emulate jitter in the message delays, the channels are optionally implemented in a probabilistic way, as described in detail in Section 5.2. The message delays are assumed to be simply gamma distributed, i.e., $p_d(\tau \mid \text{not lost})$ follows a gamma distribution. As described in Section 5.2, the RTT can be decomposed in the forward and backward delay by halving the expected value and variance. Therefore, in the probabilistic case $\mathbb{E}[\tau]$ and $\mathbb{V}[\tau]$ of $p_d(\tau \mid \text{not lost})$ are chosen as:

$$\mathbb{E}[\tau] = T_\text{C} \tag{7.2}$$

$$\mathbb{V}[\tau] = \frac{\sigma_\text{s}^2}{2} \tag{7.3}$$

with $\sigma_\text{s} = 10\,\text{ms}$. The probability density functions for the randomized channel delays are exemplary visualized in Figure 7.1. For this experiment the message loss probability is chosen as $p_\text{l} = 0$, i.e., there is no additional probabilistic message loss. Except for the dropped messages due to newer available data in cases with sender caused congestion the channels are considered to be reliable.

For the experiment the five muscle activations are used as simulation target. The polynomial regression PR_{15} is applied as HPS. As LPS the linear regression model PR_1 is chosen. It is the weakest but also the smallest LPS available in this work. In theory arbitrary combinations of models can be chosen, which is beyond the scope of this work. For this experiment the sensor

Name	Description
$\mathrm{KF}_{\mathrm{CP}}$	Kalman filter with constant position tracking
$\mathrm{KF}_{\mathrm{CV}}$	Kalman filter with constant velocity tracking
$\mathrm{KF}_{\mathrm{CA}}$	Kalman filter with constant acceleration tracking
$\mathrm{KF}_{\mathrm{CPF}}$	Kalman filter with constant position tracking and fusion
$\mathrm{KF}_{\mathrm{CVF}}$	Kalman filter with constant velocity tracking and fusion
$\mathrm{KF}_{\mathrm{CAF}}$	Kalman filter with constant acceleration tracking and fusion
$\mathrm{KF}_{\mathrm{CPU}}$	Kalman filter with constant position tracking and input augmentation fusion
$\mathrm{KF}_{\mathrm{CVU}}$	Kalman filter with constant velocity tracking and input augmentation fusion
$\mathrm{KF}_{\mathrm{CAU}}$	Kalman filter with constant acceleration tracking and input augmentation fusion

Table 7.1: KF-based approaches investigated in the evaluation.

data is assumed to be ideal, i.e., there is no noise. The evaluation is based on a trajectory with arbitrary motion. It is chosen because it contains "unpredictable" movement patterns that are more challenging compared to the periodic measurements. The experiment for the muscle surface deformation remains for future work. Comparable results are expected for this case.

The KF-based approaches require the covariance matrices for process and measurement noise $\boldsymbol{Q}$ and $\boldsymbol{R}$. Those are estimated from simulation results of the given models on the evaluation trajectory with help of the methods described in Chapter 6, especially Section 6.2.4. The variances are acquired with the help of Equation (6.53). In this way the approaches are tuned on the given trajectory.

The evaluated KF-based approaches are listed in Table 7.1. The filters $\mathrm{KF}_{\mathrm{CP}}$, $\mathrm{KF}_{\mathrm{CV}}$ and $\mathrm{KF}_{\mathrm{CA}}$ are the zeroth, first and second order tracking KFs, as introduced in Section 6.3. $\mathrm{KF}_{\mathrm{CPF}}$, $\mathrm{KF}_{\mathrm{CVF}}$ and $\mathrm{KF}_{\mathrm{CAF}}$ represent the respective order tracking filters combined with the classical fusion, as presented in Section 6.4. $\mathrm{KF}_{\mathrm{CPU}}$, $\mathrm{KF}_{\mathrm{CVU}}$ and $\mathrm{KF}_{\mathrm{CAU}}$ are the respective order tracking filters with input augmentation fusion for LPS models with a bias as shown in Section 6.5.

The algorithms have been implemented, evaluated and visualized with the following software: Anaconda 4.13.0 with Python 3.10.4 and the following packages: c3d 0.5.1, matplotlib 3.5.2, numpy 1.23.1, pandas 1.4.3, scikit-learn 1.1.1, scipy 1.8.1, tensorflow 2.9.1. The evaluation has been performed on the following system: Intel Xeon X5650 @ 2.67 GHz, 6 Cores, 20 GB RAM, Windows 10 (22H2), 64 bit.

7.2 Kalman Filter-based Forecasting

This section evaluates the KF-based approaches in Scenario I and II in which the HPS and optionally a second LPS are deployed on the mobile device with the goal of forecasting intermediate simulation outputs while the expensive HPS execution is temporarily suspended. The temporary suspension of the HPS is specified by the request period T_{R}. For example, in the specified setup with a local process invocation each $T_{\mathrm{L}} = 10\,\mathrm{ms}$, $T_{\mathrm{R}} = 20\,\mathrm{ms}$ means HPS execution in every second cycle, $T_{\mathrm{R}} = 50\,\mathrm{ms}$ in every fifth cycle and so on. The request period is evaluted in the range $T_{\mathrm{R}} \in [20, 100]$ ms.

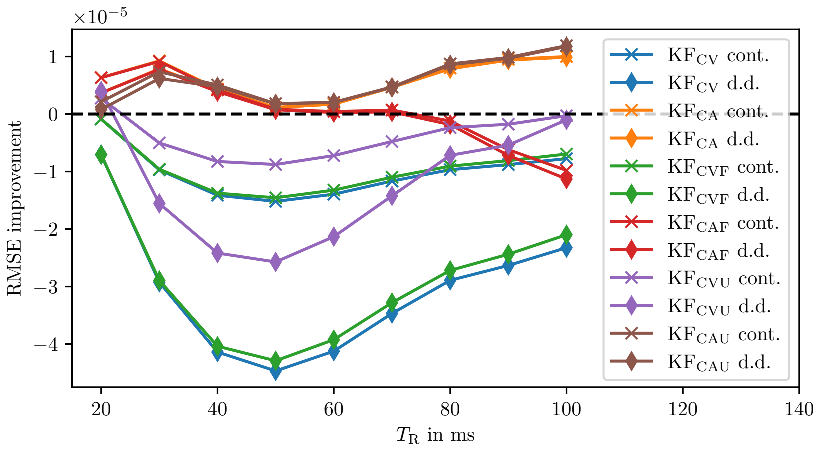


Figure 7.2: Comparison of process covariance estimation models. The plot shows the RMSE improvement of the continuous (cont.) and direct discretization (d.d.) model compared to the piecewise model (zero line) over the request period T_R. A higher improvement value indicates a lower RMSE and therefore a better filter.

Before the different KF-based approaches are benchmarked, the different models for process covariance matrix estimation are selected for each type of filter. The estimation models are presented in detail in Section 6.2.4. In this section, the piecewise white noise model, the continuous white noise model and the direct descretization model are evaluated in the given scenarios. As described in Section 6.2.4, all models are approximations of the process noise and can be represented in a way s.t. they can be interpreted as differently shaped base matrices that are scaled with the underlying variance. The piecewise white noise model yields a direct way of variance estimation from the simulation results and is therefore applied for variance estimation. Afterwards, the estimated variance is used to scale the normalized covariance matrices from the other estimation models.

There is no difference in scaling for the zeroth order KFs. Thus, the estimation models result in identical covariance matrices. An analysis of the RMSE over the request periods $T_R \in [20, 100]$ ms indicates that all covariance estimation models lead to a filter performance with an RMSE of similar order. Figure 7.2 shows the RMSE improvement of the estimation models over T_R. The RMSE over the range of T_R is shown in Table 7.2. The results suggest that the piecewise model performs best for the constant velocity models on average over T_R. For the constant acceleration models, the results indicate an average advantage of the continuous estimation model over the other models. As already mentioned, the difference in the RMSE is relative small compared to the absolute RMSE. The covariance matrices are not fine tuned, i.e., the variances are only estimated from the seen data. Thus, the effects of the covariance matrix estimation model and potential fine tuning of the variance can overlap. Since fine tuning is out of scope of this experiment, the piecewise model is applied for the zeroth and first order KFs and the continuous model for the second order KFs for the rest of the evaluation.

With the preselected covariance estimation approaches, the performance of the KF-based approaches is evaluated over $T_R \in [20, 100]$ ms. The results are presented in Figure 7.3. Figure 7.3a shows the resulting RMSE of the different KF-based approaches. The performance of the polynomial regression models PR$_6$ and PR$_{14}$ are represented as dashed horizontal lines for comparison. The

Filter	Average RMSE		
	Piecewise	Continuous	D.d.
KF_{CP}	0.01383635	0.01383635	0.01383635
KF_{CV}	0.00432850	0.00433869	0.00435925
KF_{CA}	0.00490315	0.00489703	0.00489756
KF_{CPF}	0.01259582	0.01259582	0.01259582
KF_{CVF}	0.00451206	0.00452175	0.00454139
KF_{CAF}	0.00569089	0.00569041	0.00569130
KF_{CPU}	0.00577393	0.00577393	0.00577393
KF_{CVU}	0.00290152	0.00290549	0.00291381
KF_{CAU}	0.00408435	0.00407840	0.00407878

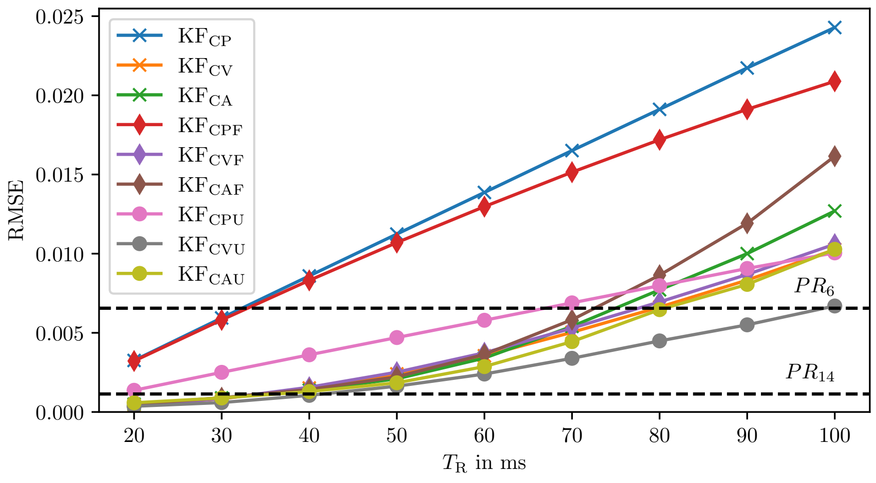
Table 7.2: Comparison of process covariance estimation models. Average RMSE over $T_R \in [20, 100]$ ms of the piecewise, continuous and direct discretization (d.d.) approaches.

KF_{CP} represents the naïve forecasting performance, i.e., it outputs the latest estimate as constant forecast. In the zeroth order case, the fusion with the KF_{CPF} cannot significantly improve the performance for low request periods. For higher request periods the fusion slightly improves the estimates but compared to the other approaches the filter performance is weak. In contrast, the KF_{CPU} exploits the additional LPS information and shows a significant improvement in RMSE. The first and second order tracking filters KF_{CV} and KF_{CA} show a similar performance till $T_R \approx 60$ ms. Beyond that point the KF_{CA} performance drops compared to the KF_{CV}. For first and second order filters, the classical fusion without bias correction even increases the RMSE. While the performance is nearly identical for small request periods, the performance degrades over T_R. In contrast, the fusion with input augmentation shows a significant improvement for all filter orders. The advantage is small for low request periods and grows with T_R. In one specific case $T_R = 20$ ms the performance of KF_{CAU} is slightly worse than KF_{CA} but in a similar range. The best performing filter is the KF_{CPU} which shows the least RMSE for all request periods.

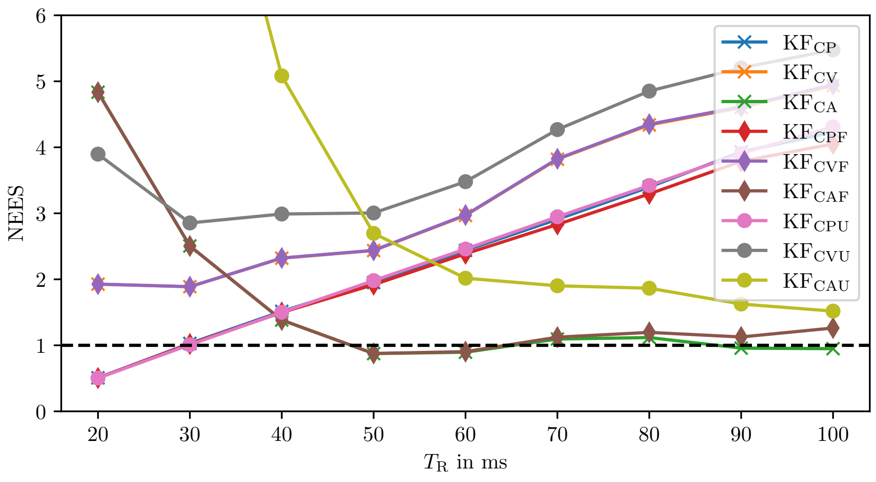
Figure 7.3b shows the average NEES over T_R. The NEES is solely computed based on the position state and variance even for higher order filters because the position estimate is the state of relevance for this experiment. The estimates of for example velocity and acceleration are not required for the AR application and are therefore filter internal. Thus, the expected value of the average NEES should be one. The plot shows that the filters are mostly overconfident with a NEES higher than one in the analyzed scenarios, i.e., the actual error is higher than the estimated variance indicates. This result is no issue for the given use case because only the actual accuracy of the estimated position compared to the ground truth is relevant. For scenarios where the error variance estimation of the KF is actually used, the consistency of the error estimate becomes an important topic.

7.3 Kalman Filter-based Delay Compensation

This section evaluates Scenario III, where the presented KF-based approaches are challenged with delays, jitter and losses. Scenario III is more challenging compared to Scenario I and II because HPS updates are always delayed. In the previous Scenarios I and II the HPS results have been



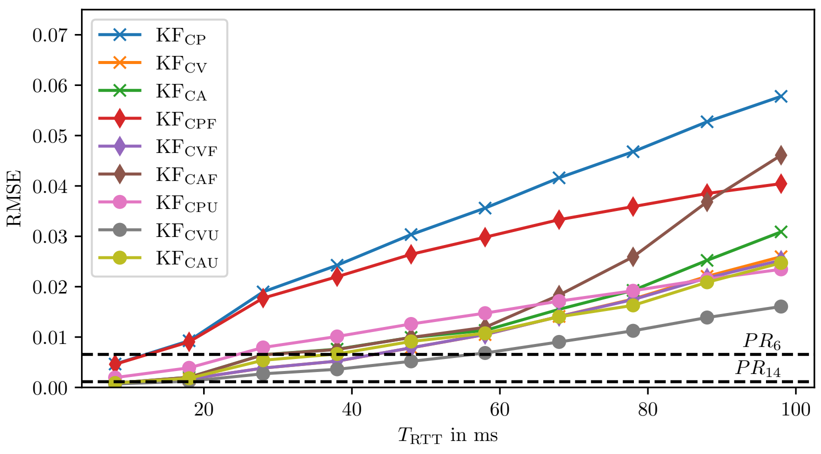
(a) The plot shows the RMSE over T_R.



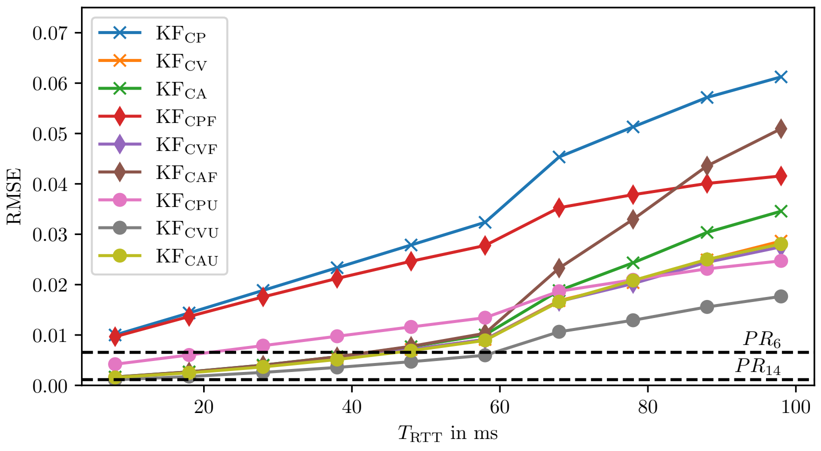
(b) The plot shows the average NEES over T_R.

Figure 7.3: Evaluation results of the KF-based forecasting approaches.

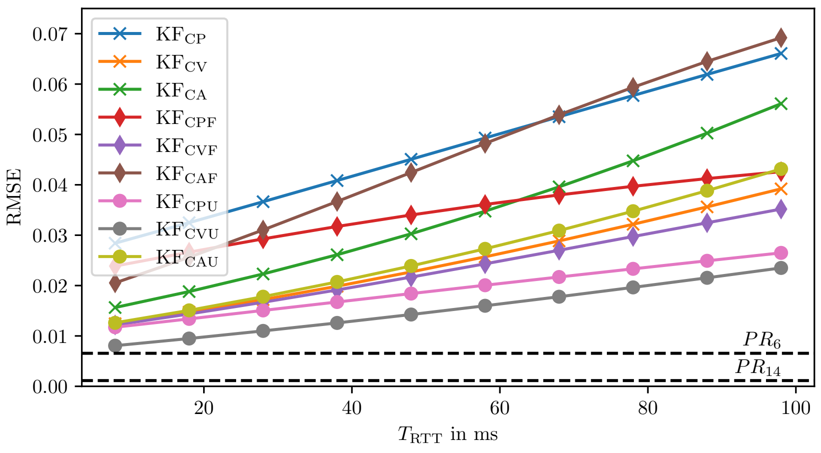
available instantaneously, i.e., results can be incorporated in the same cycle. In Scenario III the updates can be considered at the earliest in the next cycle, i.e., the minimum age of the HPS update is 10 ms. Thus, the KF is always in “forecasting mode” where the lookahead is determined by the age of the HPS updates. There also exists a challenge regarding the initialization. Since the first HPS update arrives delayed as well, the performance of the approaches is limited to available initial information. For example the models without LPS have no choice but to output an initial guess, e.g., zero. In general, the transient behavior of filters is an important topic. In the given use case with available high precision measurements from the HPS, the transient phase is very short and the steady-state performance is more important. Thus, the error metrics are acquired in a corrected way, i.e., the transient phase is ignored by cutting the first 40 samples. For this experiment, the request period is varied in the interval $T_\mathrm{R} \in [10, 100]$ ms and the desired round-trip time in the interval $T_\mathrm{RTT} \in [8, 98]$ ms.



(a) $T_R = 10$ ms.



(b) $T_R = 30$ ms.



(c) $T_R = 100$ ms.

Figure 7.4: Evaluation results of the KF-based approaches with delays and losses. The plot shows the RMSE over T_{RTT}.

For incorporation of delayed measurements, the BTTF and MA strategies have been presented in Section 6.6. Both approaches are expected to yield identical estimates for a linear system. The evaluation results show that the difference in the RMSE for both methods over all KFs, request periods and RTTs is negligible as expected. The mean absolute difference in RMSE is 7.867e−17 and the maximum absolute difference is 1.522e−14 which are probably caused by numerical inaccuracies or the BTTF behavior that replaces old LPS measurements with HPS results instead of keeping both. In this experiment both methods are applicated to store a history of the past 20 values. In this way, delayed measurement with a maximum age of 200 ms can be incorporated. The effective maximum age of a delivered message in the experiment has been measured with 190 ms. Therefore, all delayed updates haven been incorporated by the fusion algorithms.

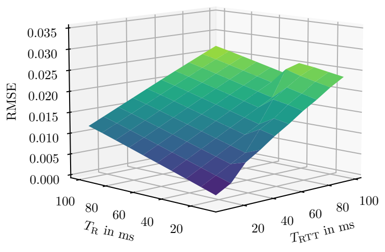
In Section 6.5 equivalence of the continued update strategy and the $\mathrm{KF}_{\mathrm{CPU}}$ has been proofed for $\boldsymbol{R} = [0]$ and $\alpha = 1$. In this experiment $\boldsymbol{R} = [1\mathrm{e}{-9}]$ is used. Nevertheless, the results show that both approaches reach a nearly identical RMSE values with a mean absolute difference of 4.023e−7 and maximum absolute difference 5.132e−7. Therefore, the $\mathrm{KF}_{\mathrm{CPU}}$ is used as representative for the continued update strategy.

The plots in Figure 7.4 show the RMSE over T_{RTT} for three exemplary values of $T_{\mathrm{R}} \in [10, 30, 100]$ ms. For $T_{\mathrm{R}} = 10$ ms in Figure 7.4a, the courses of the approaches are qualitatively similar to the forecasting case but with a higher RMSE values. The strong increase of the RMSE between $T_{\mathrm{R}} = 20$ ms and $T_{\mathrm{R}} = 30$ ms is the edge of the avoid area which is explained later in this section. With increasing T_{R} the curves are slightly reshaped and shifted upwards. Furthermore, the edge of the avoid area moves to the right. The plots suggest that some methods are more robust against delays, losses and higher request periods than others. For example Figure 7.4c shows that $\mathrm{KF}_{\mathrm{CVF}}$, $\mathrm{KF}_{\mathrm{CPU}}$ and $\mathrm{KF}_{\mathrm{CVU}}$ do not degrade as strong as many others. The $\mathrm{KF}_{\mathrm{CVU}}$ outperforms all competitors in all scenarios.

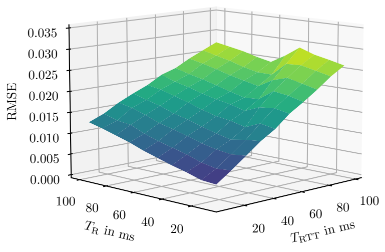
The experiment results for Scenario III with jitter qualitatively show the same results with slightly increased RMSE values. Figure 7.5 exemplarily shows the RMSE of $\mathrm{KF}_{\mathrm{CPU}}$ and $\mathrm{KF}_{\mathrm{CVU}}$ as three-dimensional plots over T_{R} and T_{RTT}. The plots on the left are the results based on the static channel delays. The plots on the right show the results based on the randomized channel delays which emulate jitter. In this visualization the avoid area clearly stands out. On the upper right of each RMSE plot, there is an area with increased RMSE for low request periods and high RTTs. The effect is caused by overload of the channels. The local process sends more requests than the channel can handle and causes congestion. Even though head-dropping is applied, i.e., only the newest message is actually transmitted as soon as the send queue is free, the messages have to wait which results in an increased delay. This increased delay is correlated with the increased RMSE. As a result, less frequent requests can improve the accuracy if congestion in the channels is avoided. Thus, it is named avoid area. The plots also indicate that delay is worth than sparse requests for the simulation accuracy.

7.4 Runtime Analysis

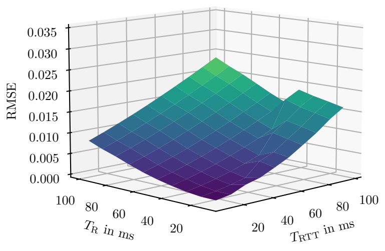
In this section, preliminary runtime evaluation results are presented. Table 7.3 shows the median runtime of 10000 model executions. Surprisingly, the lowest runtime is achieved by the NN [1 [4 , 256] 5] with 2573 parameters. In comparison the PR_1 has only 10 parameters and the PR_{15} 80. This effect has to be caused by a different degree of optimization or overhead in the implementation. Mathematically, the linear regression model requires the least number of



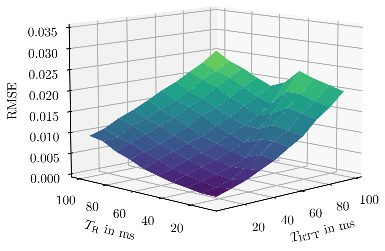
(a) $\mathrm{KF}_{\mathrm{CPU}}$.



(b) $\mathrm{KF}_{\mathrm{CPU}}$ with jitter.



(c) $\mathrm{KF}_{\mathrm{CVU}}$.



(d) $\mathrm{KF}_{\mathrm{CVU}}$ with jitter.

Figure 7.5: Evaluation results of the KF-based approaches with delays, jitter and losses. The plot shows the RMSE over T_{R} and T_{RTT}.

operations, in this case 5 multiplications and 5 additions. The PR models also have linear complexity in the polynomial degree and number of responses in this case because there is only one input feature. Nonetheless, all PR models require a similar amount of computation time. The increase of time from PR_1 to PR_{15} is only 16.9 %, while only the regression part without the polynomial feature creation requires 15 times more operations. Even the more expensive NN [1 [8, 128, 128] 5] with 18325 parameters needs only 24.3 % more time.

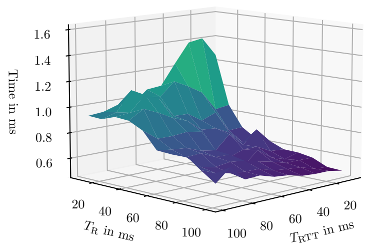
The mean execution times of the local process in Scenario III with static delays over all T_{R} and T_{RTT} are presented in Table 7.4. This local process contains a mocked LPS, i.e., it is implemented as data player. Thus, the times do not include the LPS execution. The values indicate that the filter types, i.e., tracking, fusion and input augmentation fusion, require a similar amount of time independent from the underlying tracking filter order. The tracking filters without fusion are also the most efficient ones. In the MA case, on average the input augmentation requires ≈ 1.4 times more time and the classical fusion requires ≈ 2.2 times more compared to the tracking. The MA shows a reduction of the mean execution time in all cases compared to the BTTF strategy. The improved continued update strategy has a mean execution time of 21 µs and is therefore much more efficient. Note that the KF-based approaches are applicated for a worst case delay of 200 ms.

Model	Median time in ms
NN [1 [8, 128, 128] 5]	0.455
PR_{15}	0.366
PR_{13}	0.363
PR_{14}	0.360
PR_{12}	0.354
PR_{11}	0.351
PR_{10}	0.348
PR_{9}	0.346
PR_{8}	0.341
PR_{7}	0.337
PR_{6}	0.333
PR_{5}	0.330
PR_{4}	0.328
PR_{3}	0.323
PR_{2}	0.320
PR_{1}	0.313
NN [1 [4, 256] 5]	0.266

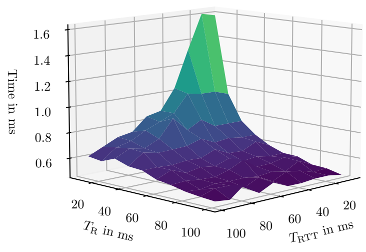
Table 7.3: Runtime of surrogate models. Ordered by time.

	Mean time in ms	
Filter	BTTF	MA
KF_{CP}	0.534	0.458
KF_{CV}	0.509	0.465
KF_{CA}	0.519	0.475
KF_{CPF}	1.612	0.997
KF_{CVF}	1.657	1.015
KF_{CAF}	1.672	1.045
KF_{CPU}	0.843	0.641
KF_{CVU}	0.772	0.650
KF_{CAU}	0.781	0.662

Table 7.4: Runtime of KF-based approaches.



(**a**) KF$_{CVU}$ with BTTF.



(**b**) KF$_{CVU}$ with MA.

Figure 7.6: Comparison of the KF delay handling methods. The plots show the mean execution time over T_R and T_{RTT}. Note that the axis are swapped compared to the previous plots.

A shorter history reduces the computational costs for both BTTF and MA. Figure 7.6 exemplarily shows the mean execution time of the KF$_{CPU}$ with the BTTF and MA strategies. In general both plots show a similar behavior. Both methods share a peak of runtime for small delays and request periods. The peak of the BTTF approach is slightly lower. The MA approach shows an advantage over BTTF for high delays and low request periods. The average runtime of the MA is mainly affected by the number of received HPS updates.

In summary, the KF-based approaches require more time on average compared to the execution of the PR$_{15}$ ground truth model. Note that the comparison is not fair. The model times have been acquired in a specific time evaluation setup with deactivated garbage collector. In contrast, the local process execution times have been measured in a simulation run where also other factors contribute, e.g., logging mechanisms, garbage collector or simulation overhead. Nonetheless, the improved continued update strategy is ≈ 31 times faster compared to the accuracy equivalent KF$_{CPU}$. A more detailed runtime optimization and evaluation of the presented approaches, especially for the AR application target hardware, are topics on their own and remain for future work.

7.5 Summary and Discussion

This section summarizes and discusses the most important evaluation results. The evaluation has shown that the developed KF-based approaches can be successfully applied in the context of distributed pervasive simulations.

In Scenarios I and II the KF-based approaches have been utilized as forecasting models to intermediately suspend the HPS. The constant velocity and constant acceleration tracking filters have a similar and significant advantage over the constant position tracking on the tested trajectory, especially for low request periods. For high request periods, the constant acceleration shows a stronger degradation compared to the constant velocity tracking. The classical fusion with a biased model can even increase the estimation error. The developed input augmentation fusion for biased models shows a significant improvement in accuracy. The KF$_{CVU}$, the constant velocity KF with input augmentation fusion, has the best performance in the tested scenarios. This significant improvement is achieved with the fusion of a weak linear regression LPS model. LPS models with better performance can be applied to further improve the estimation accuracy with the drawback of increased computational costs. In theory, the estimate of the input augmentation fusion approach can drift for long phases without HPS updates because those are required for bias correction. Nonetheless, this effect has not been visible in the evaluated cases.

In Scenario III the KF-based approaches have been successfully applied in the presence of delays, jitter and losses. As expected, the BTTF and MA approaches for incorporation of delayed measurements have shown practically identical results. Furthermore, the difference in accuracy between the continued update strategy and the $\mathrm{KF}_{\mathrm{CPU}}$ is negligible in practice. In general, the results of Scenario III are qualitatively similar to Scenario I and II. The performance degrades with increasing request period and RTT while the impact of the delay is more severe. The results also show that sender caused congestion in the communication channels, even with a head-drop channel, degrades the overall performance. Therefore, too frequent requests should be avoided in the presence of large delays. Again, the best performing model is the $\mathrm{KF}_{\mathrm{CVU}}$ and therefore better than the $\mathrm{KF}_{\mathrm{CPU}}$ and the continued update strategy in the evaluated cases. In Scenario III with jitter the $\mathrm{KF}_{\mathrm{CVU}}$ has an average improvement of 37.2 %, a maximum improvement of 69.6 % and a minimum improvement of 8.2 % in RMSE compared to the $\mathrm{KF}_{\mathrm{CPU}}$, and thus the continued update strategy, over the tested request periods and round-trip times.

Preliminary results of a runtime analysis have shown that the KF-based approaches are computationally more demanding compared to the light-weight PR surrogate models. Furthermore, the improved continued update strategy is $\approx$ 31 times faster compared to its KF-based equivalent $\mathrm{KF}_{\mathrm{CPU}}$. In summary, the continued update strategy is a light-weight and robust strategy in the presence of delays, jitter and losses with good performance. The KF-based approaches, especially the $\mathrm{KF}_{\mathrm{CVU}}$, outperform the continued update strategy at a higher computational cost. In general, whether the presented offloading strategies are worth applying depends on various factors, e.g., the computational costs and quality of available surrogate models, the cost and accuracy of the offloading strategy, the underlying network properties and the application requirements regarding accuracy and smoothness. All presented fusion methods introduce discontinuous jumps when the HPS update arrives. This effect becomes worse with longer suspension of the HPS or higher delays. Those jumps can be filtered for the visualization, e.g., with a low pass filter. On the downside this filtering introduces a delay-like phase shift which again conflicts with the desired minimum delay. Ultimately, the trade-off between smoothness and accuracy has to be applicated according to the respective application requirements. The KF framework provides a powerful toolbox to develop a problem specific offloading and fusion strategy.

8 Conclusion and Future Work

In this thesis, concepts to improve the robustness of distributed pervasive simulations, jointly executed on a mobile device and a supporting server infrastructure, have been designed with respect to delays, jitter and losses. Pervasive simulation envisions to deploy computationally expensive simulations on resource-constrained mobile devices, limited in available energy, computational power and memory. The interdisciplinary PerSiVal project targets to deploy the simulation of a bio-mechanical muscle model of the human arm as Augmented Reality (AR) application on mobile devices. The key for realization is the usage of surrogate models which approximatively represents relevant aspects of the original Finite Element Method (FEM) model. In previous work, Machine Learning (ML) methods, e.g., Neural Networks (NNs), have been utilized to reach that goal. While the efficiency of those surrogate models differs from the FEM by an order of magnitude, their real-time execution can still be challenging. Therefore, Hubatscheck [4] has presented a distribution approach, where a High-Performance Surrogate (HPS) is executed on a remote server. A second, light-weight Low-Performance Surrogate (LPS) is deployed on the wirelessly connected mobile device, which is used to counterweight inevitable delays caused by processing on the remote server and communication between devices. This thesis has been motivated by the question whether the fusion of the distributed surrogate models can be performed in a better way to improve robustness with respect to delays, jitter and losses.

As a first contribution and prerequisite for the rest of the work, an improved *surrogate model* has been designed. The surrogate model so far available in the project has suffered from strong discontinuities. During this thesis, a metric for smoothness quantification has been derived which allows the comparison of different surrogate model derivation strategies. The surrogate model for the AR application works inversely to the original FEM model, i.e., it estimates muscle activations and surface deformations given the elbow angle. This work has proposed new solutions to this inversion problem. Previous work [5] has applied the minimum activation strategy, motivated by the assumption that minimum activations represent the most efficient state to reach a specific angle. As a result, this thesis has shown that this strategy leads to strong discontinuities over the angular trajectory. Two alternative approaches have been proposed. The minimum deformation strategy has been considered best suited for the given task. It has shown a significant improvement in smoothness and does not require artificial assumptions about the activations in contrast to the other presented solutions. In the following surrogate models have been trained and evaluated. For this regression task NN and polynomial regression models have been benchmarked. The polynomial regression models have outperformed the investigated NNs in accuracy and number of parameters. Furthermore, an improved deployment strategy for the muscle surface estimation has been proposed. In comparison to the previous model chaining approach, a significant improvement in accuracy and reduction in size has been shown with the model stacking approach. In future work, further optimizations or different types of light-weight surrogate models can be explored, e.g., the reduction of polynomial terms in regression with the best subset selection [14] or sparse, kernelized approaches like support vector regression [12, 14].

As a second contribution, a *distribution model* has been developed that allows the deterministic analysis and evaluation of the distribution approaches. The model emulates a distributed system with concurrent processes on nodes that communicate via messages over first-in-first-out channels. To account for jitter in the communication delays the channels have been modeled in a probabilistic way. In this scenario, the delays have been assumed to be gamma distributed. The channels have been modeled with head-drop behavior, i.e., in case of sender caused congestion only the newest messages are transmitted. Thus, loss of messages occurs if the sender overloads the channel capacity. In summary, a useful tool for deterministic analysis and benchmark of the distributed algorithms under the modeled assumptions has been developed. In future work, the model can be applicated with parameters from the real target system to match the concrete deployment environment.

The third contribution of this work are different *Kalman filter-based concepts* to optimize the local and distributed execution of the pervasive simulation. Furthermore, this thesis has proposed an optimization of the continued update strategy from [4], which allows for a more efficient implementation. The Kalman Filter (KF) has been successfully applied for forecasting and delay compensation in the distributed pervasive simulation context. Different strategies for tracking, fusion and fusion with a biased LPS have been presented. For delay compensation the Back to the Future (BTTF) and Measurement Augmentation (MA) strategies have been successfully applied. The BTTF is also known as filter recalculation in the literature. The key idea is to save the history of the states and inputs to the KF. If a delayed update arrives, the filter is reset to the historic state that matches the point of time of the update. Subsequently, the filter is iterated with the historic inputs until the current point of time is reached again. Informally, this behavior can be described as a time travel of the filter. In contrast, the MA approach extends the KF state vector with lagged measurement values. This augmented filter is basically a time-varying fixed-lag smoother which is expected to give identical results as the BTTF strategy for linear systems. This work has also shown that the continued update strategy from [4] is closely related to the zeroth order KF with input augmentation. In fact, equivalence has been proven for a specific set of parameters. The KF variant additionally provides an estimate of the error. Furthermore, this variant can be extended to a higher order filter which has outperformed the continued update strategy in the presence of delays, jitter and losses in the evaluated scenarios. A general advantage of the proposed methods is their linear scaling in the number of independently tracked features. Due to the curse of dimensionality, the surrogate model complexity is expected to grow at least over-proportionate with the number of input features. In future work, given any constancy assumption of simulation results that can be exploited by the KF tracking, the presented approaches should be evaluated regarding their cost-performance trade-off. Conceptually, adaptive approaches like the interactive multiple model filter can be investigated [24]. This approach is usually applied for maneuvering target tracking and allows the combination of multiple KFs with individual system models to exploit the strengths of the best fitting model in the current scenario. Another interesting class of adaptive filters worth investigating, described in [24, 26], can estimate the process covariance during filter execution.

Evaluation results have shown that the developed KF-based approaches have been successfully applied in the distributed pervasive simulation context in the presence of delays, jitter and losses. Preliminary runtime analysis results indicate that the KF-based approaches are computationally more demanding than the optimized continued update strategy and the light-weight polynomial regression surrogate models. The improved continued update strategy implements a light-weight fusion strategy worth considering if the performance is sufficient. Nonetheless, the best performing constant velocity KF with augmented input fusion has shown a significant advantage regarding fusion accuracy in all evaluated scenarios. In the most challenging case with delays, jitter and losses it shows an average improvement of 37.2 %, a maximum improvement of 69.6 % and a minimum improvement of 8.2 % in the RMSE over the evaluated parameter space compared to the continued update strategy. For the presented use case, the bio-mechanical muscle simulation with one degree of freedom, the polynomial regression surrogate models of higher order are very accurate, small and have a nearly negligible impact on runtime. Thus, the success of having found very good and light-weight surrogate models degrades the usefulness of offloading with the KF-based approaches. It is worth mentioning that runtime optimization has not been the focus of this work. Instead, the focus lies on the potential analysis of the presented methods. Thus, the approaches are optimized for e.g. numerical stability or ease of use in the evaluation. Optimized algorithms exist for sub-problems with approximative solutions which could be integrated as part of future work. In general, it depends on various factors, e.g., the computational costs and quality of available surrogate models, the costs and accuracy of the offloading strategy, the underlying network properties and the application requirements regarding accuracy and smoothness, whether an offloading approach is worth considering. Ultimately, the trade-off between accuracy and conservation of resources has to be applicated according to the respective application requirements and constraints.

In conclusion, this thesis has successfully presented the application of the KF framework in the context of distributed pervasive simulations for continuous problems. It is a powerful toolbox that comes in many different flavors. The PerSiVal project is working on an extended muscle model with more degrees of freedom. In future work, the presented methods and solutions shall be applied and evaluated in the context of the new challenge.

Bibliography

[1] J. Kässinger, D. Rosin, F. Dürr, N. Hornischer, O. Röhrle, K. Rothermel. “Persival: Simulating Complex 3D Meshes on Resource-Constrained Mobile AR Devices Using Interpolation”. In: *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. 2022, pp. 961–971. DOI: 10.1109/ICDCS54860.2022.00097 (cit. on pp. 1, 18, 33).

[2] Stuttgart Center for Simulation Science. *PerSiVal: Pervasive Simulation and Visualization with Resource- and Time-Constraints*. 2022. URL: https://www.simtech.uni-stuttgart.de/exc/research/pn/pn7/pn7-1/ (visited on 10/27/2022) (cit. on p. 1).

[3] J. Valentin, M. Sprenger, D. Pflüger, O. Röhrle. “Gradient-based Optimization with B-splines on Sparse Grids for Solving Forward-Dynamics Simulations of Three-dimensional, Continuum-Mechanical Musculoskeletal System Models”. In: *International Journal for Numerical Methods in Biomedical Engineering* 34.5 (2018), e2965. DOI: 10.1002/cnm.2965 (cit. on pp. 1, 3, 4).

[4] T. Hubatscheck. “Distributed Neural Networks for Continuous Simulations on Mobile Devices”. B. S. thesis. Institute of Parallel and Distributed Systems, University of Stuttgart, 2021. DOI: 10.18419/opus-11556 (cit. on pp. 1, 2, 18, 20, 46, 47, 59, 77, 78).

[5] D. Rosin. *PerSiVal Project Documentation*. Institute for Modelling and Simulation of Biomechanical Systems, University of Stuttgart. 2022 (cit. on pp. 3–5, 25, 27, 31, 35, 77).

[6] O. Röhrle. “Skeletal Muscle Modelling”. In: *Encyclopedia of Continuum Mechanics*. Ed. by A. Altenbach Holm and Öchsner. Berlin, Heidelberg: Springer Berlin Heidelberg, 2020, pp. 2292–2301. ISBN: 978-3-662-55771-6. DOI: 10.1007/978-3-662-55771-6_39 (cit. on p. 3).

[7] C. P. Bradley, N. Emamy, T. Ertl, D. Göddeke, A. Hessenthaler, T. Klotz, A. Krämer, M. Krone, B. Maier, M. Mehl, et al. “Enabling Detailed, Biophysics-Based Skeletal Muscle Models on HPC Systems”. In: *Frontiers in physiology* 9 (2018), p. 816. DOI: 10.3389/fphys.2018.00816 (cit. on p. 3).

[8] B. Maier, N. Emamy, A. Krämer, M. Mehl. “Highly Parallel Multi-Physics Simulation of Muscular Activation and EMG”. In: *COUPLED VIII: proceedings of the VIII International Conference on Computational Methods for Coupled Problems in Science and Engineering*. CIMNE. 2019, pp. 610–621 (cit. on p. 3).

[9] A. W. Wiegner, M. M. Wierzbicka. “Kinematic Models and Human Elbow Flexion Movements: Quantitative Analysis”. In: *Experimental Brain Research* 88.3 (1992), pp. 665–673 (cit. on p. 5).

[10] T. Mitchell. *Machine Learning*. McGraw Hill, 1997 (cit. on p. 6).

[11] K. P. Murphy. *Probabilistic Machine Learning: An Introduction*. MIT Press, 2022. URL: https://probml.github.io/book1 (cit. on pp. 6, 15, 43).

[12] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer New York, NY, 2006. ISBN: 9780387310732 (cit. on pp. 6–9, 12, 14, 15, 25, 43, 53, 77, XXIII).

[13] I. Goodfellow, Y. Bengio, A. Courville. *Deep Learning*. MIT Press, 2016. URL: https://www.deeplearningbook.org (cit. on pp. 6–8, 12–14, 25).

[14] T. Hastie, R. Tibshirani, J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Vol. 2. 12th printing with corrections, Jan 13, 2017. Springer, 2009. URL: https://hastie.su.domains/ElemStatLearn/printings/ESLII_print12_toc.pdf (cit. on pp. 6, 8, 12, 77).

[15] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012. ISBN: 9780262018029 (cit. on p. 6).

[16] K. P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023. URL: https://probml.github.io/book2 (cit. on pp. 6, 48, 54).

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 6).

[18] scikit-learn. *Cross-validation: Evaluating Estimator Performance, scikit-learn 1.1.3*. 2022. URL: https://scikit-learn.org/stable/modules/cross_validation.html (cit. on pp. 8, 9).

[19] scikit-learn. *Preprocessing Data, scikit-learn 1.1.3*. 2022. URL: https://scikit-learn.org/stable/modules/preprocessing.html (cit. on pp. 10, 14).

[20] scikit-learn. *sklearn.neighbors.KNeighborsRegressor, scikit-learn 1.1.3*. 2022. URL: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html (cit. on p. 12).

[21] R. J. Hyndman, G. Athanasopoulos. *Forecasting: Principles and Practice, 3rd edition*. 2021. URL: https://OTexts.com/fpp3 (cit. on pp. 13, 55).

[22] scikit-learn. *Metrics and Scoring: Quantifying the Quality of Predictions, scikit-learn 1.1.3*. 2022. URL: https://scikit-learn.org/stable/modules/model_evaluation.html (cit. on p. 15).

[23] R. E. Kalman. “A New Approach to Linear Filtering and Prediction Problems”. In: *Journal of Basic Engineering* 82.1 (Mar. 1960), pp. 35–45. DOI: 10.1115/1.3662552 (cit. on p. 15).

[24] R. R. Labbe. *Kalman and Bayesian Filters in Python*. 2020. URL: https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python (cit. on pp. 15–17, 48–54, 78).

[25] P. Kim. *Kalman-Filter für Einsteiger*. Trans. by U. Schneider. 2016. ISBN: 9781502723789 (cit. on pp. 16, 51, 52).

[26] R. Marchthaler, S. Dingler. *Kalman-Filter: Einführung in die Zustandsschätzung und ihre Anwendung für eingebettete Systeme*. Springer Vieweg, 2017. ISBN: 9783658167288. DOI: 10.1007/978-3-658-16728-8 (cit. on pp. 16, 17, 48, 51–53, 58, 78).

[27] I. Reid. *Estimation II*. 2001. URL: https://www.robots.ox.ac.uk/~ian/Teaching/Estimation/LectureNotes2.pdf (cit. on pp. 16, 17).

[28] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, F. Kawsar. “DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices”. In: *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. 2016, pp. 1–12. DOI: 10.1109/IPSN.2016.7460664 (cit. on p. 18).

[29] T. Tan, G. Cao. "Efficient Execution of Deep Neural Networks on Mobile Devices with NPU". In: *Proceedings of the 20th International Conference on Information Processing in Sensor Networks (Co-Located with CPS-IoT Week 2021)*. IPSN '21. Nashville, TN, USA: Association for Computing Machinery, 2021, pp. 283–298. ISBN: 9781450380980. DOI: 10.1145/3412382.3458272 (cit. on p. 18).

[30] S. Teerapittayanon, B. McDanel, H. Kung. "Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices". In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 2017, pp. 328–339. DOI: 10.1109/ICDCS.2017.226 (cit. on p. 18).

[31] Y. Wang, J. Wang, W. Zhang, Y. Zhan, S. Guo, Q. Zheng, X. Wang. "A Survey on Deploying Mobile Deep Learning Applications: A Systemic and Technical Perspective". In: *Digital Communications and Networks* 8.1 (2022), pp. 1–17. ISSN: 2352-8648. DOI: 10.1016/j.dcan.2021.06.001 (cit. on p. 18).

[32] C. Dibak, A. Schmidt, F. Dürr, B. Haasdonk, K. Rothermel. "Server-Assisted Interactive Mobile Simulations for Pervasive Applications". In: *2017 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. 2017, pp. 111–120. DOI: 10.1109/PERCOM.2017.7917857 (cit. on p. 18).

[33] N. Hornischer. "Mesh Size Reduction and Interpolation of a Biomechanical Simulation for Neural Networks on Mobile Devices". Research Project. Institute of Parallel and Distributed Systems, University of Stuttgart, 2021 (cit. on p. 18).

[34] F. Belz, B. Mehler. "Prediction supported Continuous Simulations using Neural Networks". Research Project. Institute of Parallel and Distributed Systems, University of Stuttgart, 2022 (cit. on pp. 18, 20, 46, 59).

[35] I. N. Bronstein, K. Semendjajew, G. Musiol, H. Mühlig. *Taschenbuch der Mathematik*. 9th ed. Verlag Europa-Lehrmittel, 2013. ISBN: 9783808556719 (cit. on p. 22).

[36] B. K. Horn, B. G. Schunck. "Determining Optical Flow". In: *Artificial Intelligence* 17.1 (1981), pp. 185–203. ISSN: 0004-3702. DOI: 10.1016/0004-3702(81)90024-2 (cit. on p. 24).

[37] A. Bruhn. *Correspondence Problems in Computer Vision*. Script, Institute for Visualization and Interactive Systems, University of Stuttgart. 2021 (cit. on p. 24).

[38] NumPy. *numpy.gradient, NumPy 1.23*. 2022. URL: https://numpy.org/doc/stable/reference/generated/numpy.gradient.html (cit. on p. 24).

[39] Michigan Metrology, LLC. *Root Mean Square Surface Slope*. 2022. URL: https://michmet.com/glossary-term/root-mean-square-surface-slope/ (cit. on p. 24).

[40] KEYENCE CORPORATION. *Surface Roughness Parameters - Root Mean Square Slope*. 2022. URL: https://www.keyence.com/ss/products/microscope/roughness/line/root-mean-square-slope.jsp (cit. on p. 24).

[41] KEYENCE CORPORATION. *Area Roughness Parameters - Sdq (Root Mean Square Gradient)*. 2022. URL: https://www.keyence.com/ss/products/microscope/roughness/surface/sdq-root-mean-square-gradient.jsp (cit. on p. 24).

[42] W. Wilson. *Testing Distributed Systems w/ Deterministic Simulation*. Talk at Strange Loop Conference. 2014. URL: https://www.youtube.com/watch?v=4fFDFbi3toc (cit. on p. 38).

[43] A. Mukherjee. "On the Dynamics and Significance of Low Frequency Components of Internet Load". In: (1992) (cit. on pp. 41–43).

[44] A. Corlett, D. Pullin, S. Sargood. *Statistics of One-Way Internet Packet Delays*. Internet-Draft draft-corlett-statistics-of-packet-delays-00. Work in Progress. Internet Engineering Task Force, Feb. 2002. URL: https://datatracker.ietf.org/doc/draft-corlett-statistics-of-packet-delays/00/ (cit. on p. 42).

[45] K. Sui, M. Zhou, D. Liu, M. Ma, D. Pei, Y. Zhao, Z. Li, T. Moscibroda. "Characterizing and Improving WiFi Latency in Large-Scale Operational Networks". In: *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '16. Singapore, Singapore, 2016, pp. 347–360. DOI: 10.1145/2906388.2906393 (cit. on p. 42).

[46] P. Chou, Z. Miao. "Rate-Distortion Optimized Streaming of Packetized Media". In: *IEEE Transactions on Multimedia* 8.2 (2006), pp. 390–404. DOI: 10.1109/TMM.2005.864313 (cit. on pp. 42, 43).

[47] M. Kalman, B. Girod. "Modeling the Delays of Successively-transmitted Internet Packets". In: *2004 IEEE International Conference on Multimedia and Expo (ICME) (IEEE Cat. No.04TH8763)*. Vol. 3. 2004, 2015–2018 Vol.3. DOI: 10.1109/ICME.2004.1394659 (cit. on p. 42).

[48] B. D. O. Anderson, J. B. Moore. *Optimal Filtering*. Englewood Cliffs, NJ: Prentice-Hall, 1979. ISBN: 0136381227 (cit. on pp. 48, 64).

[49] Y. Bar-Shalom, X.-R. Li, T. Kirubarajan. *Estimation with Applications to Navigation and Tracking*. John Wiley & Sons, Inc., 2001. ISBN: 9780471416555 (cit. on pp. 48, 49, 54).

[50] S. Thrun, W. Burgard, D. Fox. *Probabilistic Robotics*. MIT Press, 2006. ISBN: 9780262201629 (cit. on p. 54).

[51] R. G. Brown, P. Y. C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering*. 4th ed. John Wiley & Sons, Inc., 2012. ISBN: 9780470609699 (cit. on p. 57).

[52] Y. Bar-Shalom, H. Chen, M. Mallick. "One-step Solution for the Multistep Out-of-Sequence-Measurement Problem in Tracking". In: *IEEE Transactions on Aerospace and Electronic Systems* 40.1 (2004), pp. 27–37. DOI: 10.1109/TAES.2004.1292140 (cit. on pp. 63, 64).

[53] M. Koplin, W. Elmenreich. "Analysis of Kalman Filter based Approaches for Fusing Out-of-Sequence Measurements corrupted by Systematic Errors". In: *2008 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*. 2008, pp. 175–180. DOI: 10.1109/MFI.2008.4648061 (cit. on p. 63).

[54] M. S. Mahmoud, H. M. Khalid. "Distributed Kalman Filtering: a Bibliographic Review". In: *IET Control Theory & Applications* 7.4 (2013), pp. 483–501. DOI: https://doi.org/10.1049/iet-cta.2012.0732 (cit. on p. 63).

[55] A. Gopalakrishnan, N. S. Kaisare, S. Narasimhan. "Incorporating Delayed and Infrequent Measurements in Extended Kalman Filter based Nonlinear State Estimation". In: *Journal of Process Control* 21.1 (2011), pp. 119–129. ISSN: 0959-1524. DOI: 10.1016/j.jprocont.2010.10.013 (cit. on pp. 63, 64).

[56] T. Larsen, N. Andersen, O. Ravn, N. Poulsen. "Incorporation of Time Delayed Measurements in a Discrete-time Kalman Filter". In: *Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No.98CH36171)*. Vol. 4. 1998, 3972–3977 vol.4. DOI: 10.1109/CDC.1998.761918 (cit. on pp. 63, 64).

[57] G. C. Goodwin, J. C. Agüero, A. Feuer. “State Estimation for Systems having Random Measurement Delays using Errors in Variables”. In: *IFAC Proceedings Volumes* 35.1 (2002). 15th IFAC World Congress, pp. 385–390. ISSN: 1474-6670. DOI: 10.3182/20020721-6-ES-1901.00475 (cit. on p. 64).

[58] M. Pohjola, H. Koivo. “Measurement Delay Estimation for Kalman Filter in Networked Control Systems”. In: *IFAC Proceedings Volumes* 41.2 (2008). 17th IFAC World Congress, pp. 4192–4197. DOI: 10.3182/20080706-5-KR-1001.00705 (cit. on p. 64).

[59] E. W. Weisstein. *Gamma Distribution*. From MathWorld–A Wolfram Web Resource. URL: https://mathworld.wolfram.com/GammaDistribution.html (cit. on p. XXIII).

All links were last followed on December 29, 2022.

A Kurzfassung

Die Vision der so genannten Pervasive Simulation ist die Verwendung von rechenaufwändigen Simulationen auf Ressourcen-beschränkten mobilen Endgeräten in alltäglichen Situationen. Das PerSiVal Projekt hat die Realisierung einer biomechanischen Simulation eines menschlichen Arms als Augmented Reality Applikation zum Ziel. Um die Ausführung solch teurer Simulationen auf Ressourcen-beschränkten Endgeräten zu ermöglichen, werden so genannte Surrogat-Modelle verwendet. Die Ausführung solcher Surrogat-Modelle kann weiterhin herausfordernd sein. Eine Lösung, um mit der Ressourcen-Beschränkung umzugehen, ist das Auslagern des Surrogat-Modells auf einen Server, der kabellos mit dem mobilen Endgerät verbunden ist. Hierbei sind unvermeidbare Verzögerungen, verursacht durch Prozessierung und Kommunikation, eine große Herausforderung. Um diesen entgegenzuwirken, ist in vorangegangener Arbeit ein Ansatz entwickelt worden, der ein zweites, leichtgewichtiges Surrogat-Modell mit niedrigerer Performance auf dem mobilen Endgerät ausführt. Das Ziel dieser Thesis ist das Design und die Evaluation von Ansätzen, basierend auf dem Kalman-Filter, für das Fusionsproblem in der Gegenwart von Verzögerungen, Jitter und Datenverlust. Diese Arbeit präsentiert folgende Kontributionen: eine verbesserte Strategie für die Surrogat-Model Ableitung, verbesserte und leichtgewichtige Surrogat-Modelle für die Muskelsimulation, ein Verteilungsmodell für die reproduzierbare Analyse und Evaluation von verteilten Algorithmen, eine verbesserte Variante des Fusionsansatzes aus vorheriger Arbeit und das Design und die Evaluation von Kalman-Filter-basierten Lösungen für das Fusionsproblem. Während diese rechenaufwändiger sind, haben sie signifikante Vorteile gegenüber Verzögerungen, Jitter und Datenverlust in den evaluierten Szenarien. Das Kalman-Filter erster Ordnung mit Eingangsaugmentation erbringt die besten Resultate in den getesteten Szenarien. Schlussendlich bietet das Kalman-Filter ein mächtiges Rahmenwerk, das erfolgreich im Kontext verteilter pervasiver Simulation für kontinuierliche Probleme angewendet worden ist.

B Proofs

Gamma Distribution Parameter Derivation Given Equation (B.1) and Equation (B.2) from [12] the parameters a and b can be computed as follows:

$$\mathbb{E}[\tau] = \frac{a}{b} \tag{B.1}$$

$$\mathbb{V}[\tau] = \frac{a}{b^2} \tag{B.2}$$

$$= \frac{\mathbb{E}[\tau]}{b} \tag{B.3}$$

$$b = \frac{\mathbb{E}[\tau]}{\mathbb{V}[\tau]} \tag{B.4}$$

$$a = \mathbb{E}[\tau] \cdot b \tag{B.5}$$

$$= \frac{\mathbb{E}[\tau]^2}{\mathbb{V}[\tau]} \tag{B.6}$$

Symmetric Split of Gamma Distributions The additive property of the gamma distribution states [59]: If $\tau_1, \tau_2, \dots, \tau_n$ are independent and gamma distributed with parameters $(a_1, b), (a_2, b), \dots (a_n, b)$, then $\sum_{i=1}^{n} \tau_i$ is gamma distributed with:

$$a = \sum_{i=1}^{n} a_i \tag{B.7}$$

$$b = b \tag{B.8}$$

Thus, a random variable τ_s can be symmetrically decomposed into two random variables τ_1 and τ_2, s.t. $\tau_s = \tau_1 + \tau_2$. For this purpose, the expected values and variances of the random variables τ_1 and τ_2 are identical and half of the expected value and variance of τ_s.

Proof:

$$\tau_1 \sim \text{Gamma}(a_1, b) \tag{B.9}$$

$$\tau_2 \sim \text{Gamma}(a_2, b) \tag{B.10}$$

$$\tau_s = \tau_1 + \tau_2 \sim \text{Gamma}(a_1 + a_2, b) \tag{B.11}$$

$$a_g = 2a_1 = 2a_2 \tag{B.12}$$

$$\frac{\mathbb{E}[\tau_s]^2}{\mathbb{V}[\tau_s]} = 2\frac{\mathbb{E}[\tau_1]^2}{\mathbb{V}[\tau_1]} \tag{B.13}$$

$$b_g = b_1 = b_2 \tag{B.14}$$

$$\frac{\mathbb{E}[\tau_s]}{\mathbb{V}[\tau_s]} = \frac{\mathbb{E}[\tau_1]}{\mathbb{V}[\tau_1]} \tag{B.15}$$

$$\frac{\mathbb{E}[\tau_s]\mathbb{E}[\tau_s]}{\mathbb{V}[\tau_s]} = 2\frac{\mathbb{E}[\tau_1]^2}{\mathbb{V}[\tau_1]} \tag{B.16}$$

$$\frac{\mathbb{E}[\tau_s]\mathbb{E}[\tau_1]}{\mathbb{V}[\tau_1]} = 2\frac{\mathbb{E}[\tau_1]^2}{\mathbb{V}[\tau_1]} \tag{B.17}$$

$$\mathbb{E}[\tau_s] = 2\mathbb{E}[\tau_1] \tag{B.18}$$

$$\mathbb{E}[\tau_1] = \frac{1}{2}\mathbb{E}[\tau_s] \tag{B.19}$$

$$\mathbb{E}[\tau_{1/2}] = \frac{1}{2}\mathbb{E}[\tau_s] \tag{B.20}$$

$$\frac{\mathbb{E}[\tau_s]}{\mathbb{V}[\tau_s]} = \frac{\mathbb{E}[\tau_1]}{\mathbb{V}[\tau_1]} \tag{B.21}$$

$$\frac{2\mathbb{E}[\tau_1]}{\mathbb{V}[\tau_s]} = \frac{\mathbb{E}[\tau_1]}{\mathbb{V}[\tau_1]} \tag{B.22}$$

$$\mathbb{V}[\tau_1] = \frac{1}{2}\mathbb{V}[\tau_s] \tag{B.23}$$

$$\mathbb{V}[\tau_{1/2}] = \frac{1}{2}\mathbb{V}[\tau_s] \tag{B.24}$$

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature