Data-driven simulation for general-purpose multibody dynamics using Deep Neural Networks

Hee-Sun Choi1 · Junmo An2 · Seongji Han1 · Jin-Gyun Kim1 · Jae-Yoon Jung2 · Juhwan Choi3 · Grzegorz Orzechowski4 · Aki Mikkola4 · Jin Hwan Choi1

Received: 25 August 2019 / Accepted: 24 November 2020 / Published online: 14 December 2020 © The Author(s), under exclusive licence to Springer Nature B.V. part of Springer Nature 2020

Abstract In this paper, we introduce a machine learning-based simulation framework of general-purpose multibody dynamics (MBD). The aim of the framework is to construct a well-trained meta-model of MBD systems, based on a deep neural network (DNN). Since the main advantage of the meta-model is the enhancement of computational efficiency in returning solutions, the modeling would be beneficial for solving highly complex MBD problems in a short time. Furthermore, for dynamics problems, not only the accuracy but
also the smoothness in time of motion solutions, such as displacement, velocity, and acceleration, are essential aspects to consider. We analyze and discuss the influence of training data structures on both aspects of solutions. As a result of the introduced approach, the meta-model provides motion estimation of system dynamics without solving an analytical equation of motion or a numerical solver. Numerical tests demonstrate the performance of the proposed meta-modeling for representing several MBD systems.

Keywords Multibody dynamics · Meta-model · Deep neural network · Feed forward network · Data-driven simulation

1 Introduction

Using Machine Learning (ML) with big data is an important subject matter in science and engineering. This is because ML is effective to handle and interpret big data sets for the purpose of finding certain patterns from the data. In particular, Deep Neural Networks (DNNs), which are based on an Artificial Neural Network (ANN) with multiple hidden layers between input and output layers, allows users to handle complex shapes with nonlinear functions with multi-dimensional input data. DNNs have been successfully used in a large number of practical applications.

A well trained neural network can provide precise solutions in real-time based on solution-patter recognition trained from data sets. This makes them ideal for building surrogate models of mechanical systems. As indicated by Eldered et al. [8], surrogate models can be categorized as data-driven and physics-based. Data-driven models can be further subdivided according to the number of data points employed: local approximation based on single point, multi-point approximation based on small number of data points, and global approximation with points from the whole domain of interest. For handling general-purpose nonlinear systems, only the last option is viable and DNN is one of the data-driven global approximation methods. In case of the physics-based surrogates, the original model is replaced by the one with lower fidelity. A notable members of this family are model order reduction methods [1, 5, 14], where the full order model is replaced by the system with smaller number of degrees of freedom using techniques like modal truncation and Krylov subspace based reduction. It is worth to point out that multiple local, often linear approximations can represent nonlinear system by combining numerous pieces over the domain of interest, using both physics-based [23] and data-driven [2] approaches.

ML features, such as big data recognition and real-time estimation of nonlinear functions, are attractive to dynamics and control engineers who are handling nonlinear system dynamics. There have been several previous studies on applying ML, DNN, or other big-data handling techniques to rigid multibody system problems. For example, a Bayesian formulation [6, 17, 26] in combination with Markov random field approximation, a Kalman filter, a Gaussian process modeling [21], or a particle filter has been applied to various multibody dynamics (MBD) problems to handle noise data effectively in real-life applications, generate reliable modeling with efficient computational cost, estimate multibody systems in a probabilistic sense, and identify nonlinear parameters in governing equations. ML approaches [3, 11, 18, 19, 27] such as regression methods, reinforcement learning algorithms, and surrogate models have also been employed. There are many different types of regression methods that
can be performed in ML. In addition to the simple linear regression model, one can select and use techniques such as polynomial regression, support vector regression, decision tree regression, and random forest regression to address a given problem. Based on the investigated input-label values, surrogate models perform a probabilistic estimate for an unknown objective function. This is an approach that uses an interpretable model to describe complex models. The most commonly used model in surrogate models is the Gaussian process. Previously proposed methods have method has enhanced accuracy of prediction, especially in the long time scales, and increased computational efficiency in simulating the dynamic response of multibody systems. Moreover, neural networks \([3, 7, 9, 16, 20]\) have been suggested as effective alternatives to multibody dynamics simulation in comparison with conventional algorithms such as recursive formulations \([13]\) and reduced-order modeling techniques \([1, 14]\). The approaches have been proved to be fast and reliable to describe and predict characteristics of multibody systems.

It is important to note that previous studies are limited to particular MBD problems, for example, on contact, railways, vehicles, gaits, robotics, or tracking. A general MBD problem has not been introduced and analyzed through DNN techniques.

The main contributions of this study are that it broadens the application of DNN techniques for multibody systems and introduces a procedure of generating a general-purpose data-driven solver. The resulting data-driven solver, or a surrogate model, shows its potential for real-time simulations of MBD systems. Moreover, we describe and compare two different approaches to generating a data-driven solver. Both solve the same MBD problems based on different ways of handling for a time variable. It turns out that the predicted solutions from the consequent data-driven solvers have a large difference in the smoothness in time. Thus, we conclude that an approach producing smooth solutions in time would be more beneficial for time-transient dynamic problems.

In respect of training a surrogate model, among the various ML methods, a supervised learning technique is used for the mathematical and/or numerical data set of the MBD model in the training process. The data preparation and training process are referred to as the off-line stage, and the trained result is known as a meta-model. Using the meta-model, the time-varying results can be estimated such as displacement, velocity, and acceleration of the multibody system without directly solving the governing equations of MBD, and then this estimation process is called the on-line stage. This is distinct from the conventional recursive and/or reduced-order modeling approaches \([1, 13, 14]\) that have to handle ordinary differential equations using numerical integrations.

In particular, the feed forward networks (FFN) with hidden layers and non-linear activation functions are employed among the various DNN methods since they can efficiently represent continuous functions. To obtain a reliable meta-model, a sufficient and accurate data for MBD systems are required, and random search is also important to define appropriate hyper-parameters of MBD problems such as the number of hidden layers, the size of batches, the number of epochs, optimizer, etc.

In Sect. 2, the governing equations of MBD are reviewed. In Sect. 3, an overview of neural networks of MBD and the meta-modeling process is presented. It should be noted that the framework of the proposed meta-modeling provides fundamental ideas of handling experimental or real-world data and exploiting their structures and relations to understand the dynamics of general multibody systems. Not depending on the complexity of MBD systems, the present meta-modeling helps us to achieve real-time and robust simulations with accurate motion results. Section 4 describes the case studies of the meta-modeling process. Four representative MBD and kinematics problems, such as a single pendulum, double pendulum, slider–crank mechanisms, and a vibrating transmission system with contact, were
considered to evaluate the performance of the proposed DNN-based meta-modeling framework. Conclusions are given in Sect. 5.

2 Brief review of common general-purpose MBD governing equations

Multibody system dynamics offers a straightforward approach to constructing and solving equations of motion for mechanical systems. Multibody system dynamics includes a large number of procedures that can be categorized based on the used coordinates [13, 25]. In topological approaches, such as semi-recursive formulation, relative coordinates between the bodies are used. In the global approaches, in turn, the set of coordinates defines each body of the system. It is important to note that although topological and global approaches both lead to identical dynamic responses, the numerical performance differs. In this section often used global methods are reviewed.

In the augmented formulation, constraint equations are accounted in the equation of motion by employing Lagrange multipliers. In this approach the equations of motion can be written as

\[
\begin{bmatrix}
  M & C_q^T \\
  C_q & 0
\end{bmatrix}
\begin{bmatrix}
  \ddot{q} \\
  \lambda
\end{bmatrix} =
\begin{bmatrix}
  F_a \\
  F_c
\end{bmatrix}
\]

(1)

where \(M\) is the mass matrix, \(C_q\) is the Jacobian matrix that can be obtained by taking partial differentiation of the constraint vector \(C\) with respect to the vector of generalized coordinates \(q\), \(F_a\) is the vector that can be obtained by combining generalized applied forces and quadratic velocity vector, and \(F_c\) is the vector that can be obtained by differential constraint twice with respect to time. The equation of motion is solved to obtain the generalized coordinates \(q\) and the Lagrange multipliers \(\lambda\).

The other commonly used form of equations of motion for multibody system can be derived by applying the embedding technique to global coordinates (1). The embedding technique reduces the generalized coordinates to be solved from \(\ddot{q}\) to a set of independent generalized \(\ddot{q}_{ind}\). In practice, this reduction can be accomplished using a coordinate partitioning and a transformation matrix \(T\), in particular, as follows:

\[
\ddot{q} = T \ddot{q}_{ind} + T_c,
\]

(2)

where \(T_c\) is a vector that can be obtained using an inverse of the partitioned Jacobian matrix associated to the dependent coordinates and vector \(F_a\). Substituting (2) into the augmented system (1) yields

\[
\begin{align*}
  MT \ddot{q}_{ind} + MT_c + C_q^T \lambda &= F_a, \\
  C_q T \ddot{q}_{ind} + C_q T_c &= F_c.
\end{align*}
\]

(3)

By applying the identity \(T^T C_q = 0\), Eq. (3) can be simplified to

\[
\tilde{M} \ddot{q}_{ind} = \tilde{F},
\]

(4)

where

\[
\begin{align*}
  \tilde{M} &:= T^T M T, \\
  \tilde{F} &:= T^T F_a - T^T M T_c.
\end{align*}
\]
The equations of motion based on the augmented formulation can be also modified to be expressed using relative joint coordinates. To this end, a relationship between the Cartesian velocities and the relative joint velocity can be established as follows:

\[ \dot{\mathbf{q}} = \mathbf{R}\dot{z} \]  

where \( \mathbf{R} \) is the velocity transformation matrix and \( \dot{z} \) is the relative joint velocity. This relation is acceleration level can be written as

\[ \ddot{\mathbf{q}} = \mathbf{R}\ddot{z} + \dot{\mathbf{R}}\dot{z}. \]  

By employing relative joint coordinates, equations of motion for open loop system can be written as

\[ \mathbf{R}^T\mathbf{M}\ddot{z} = \mathbf{R}^T(\mathbf{F}_a - \mathbf{M}\dot{\mathbf{R}}\dot{z}). \]  

A closed loop system can be accounted by utilizing the cut-joint method. Constrain equations the associated to the cut-joint by employing the augmented Lagrangian method.

### 3 Deep neural network for multibody dynamics systems

In this section, a brief introduction of DNN that will be used in numerical examples is presented, and training of the DNN for MBD systems is also described.

Machine Learning (ML) aims to analyze complex patterns of system solutions based on a big amount of data and to provide accurate predictions. ML allows important tasks to be performed. ML has already powered many aspects of modern society from web searches and item recognition to image classification, speech recognition, and cyber-physical systems (CPSs).

Being a part of ML, Artificial Neural Networks (ANNs) are clusters of nodes (or neurons) to mimic the decision-making process of the human brain; see Fig. 1. There are an input layer, multiple hidden layers, and an output layer. Each layer consists of multiple nodes (or neurons). The connections or links between the nodes represent linear transformations of information through weights. The main purpose of an ANN is to find the best set of weights to optimize the performance of the neural network. An error back-propagation algorithm developed by Rumelhart et al. [24] suggests an efficient way of optimizing the weights.
To describe and represent more complicated and intricate data, more than one hidden layer can be considered. In this case, the ANN is referred to as a Deep Neural Network (DNN). The increased number of hidden layers increases the number of nodes and weights, which requires an expensive computational cost and makes it difficult to train a model. Despite these shortcomings, DNN yields better meta-models for solving complex nonlinear problems.

The structure of the DNNs can be specified in more detail by the hyper-parameters such as the number of layers, the number of nodes for each layer, the batch size, the activation functions, the regulatory method, and the optimizers. The performance of the DNN highly relies on the proper choice of hyper-parameters. Some important hyper-parameters mentioned in the numerical tests (Sect. 4) are briefly summarized as follows:

- **Batch size**
  The batch size is the number of training data samples in one pass for updating weights. Due to memory limitations, it is not recommended to perform training with all available data samples at once. As the batch size gets larger, the fewer iterations of training need in an epoch, and thus it requires less computation cost.

- **Activation function**
  In the DNN, values specified to nodes of a layer are not transferred directly to the next layer, but transformed through a nonlinear function, called an activation function. This helps the values of nodes not to diverge during training and makes it possible to solve complex problems with a small number of nodes. If an unsuitable activation function is chosen, gradients of DNN (in the error back-propagation process) can be vanishing, which slows learning speed severely. Activation functions, such as tanh, sigmoid, and ReLU, are popular choices.

- **Optimizers**
  Weights of the DNN are found by error back-propagation processes, which sequentially updates the weights to minimize a loss function defined by a given error, such as $E_{mse}$ described in (10). In this process, a local minimum problem needs to be solved and an efficient optimizer helps to reduce the solution time. Representative techniques are stochastic gradient descent, and the Adam [15], and RMSprop [12] methods.

### 3.1 Overview of neural networks for MBD

**Meta-model using neural networks** ML methods can be categorized from the viewpoint of learning styles into three groups: supervised, unsupervised, and reinforcement learning. Supervised learning trains a meta-model by considering both reference response features called labels and predictive features, and by gradually improving the model to fit the given training data. There are mainly classification and regression methods in supervised learning. In unsupervised learning, contrary to supervised learning, label (or reference) features are not designated. It focuses on how training data is structured. Reinforcement learning is an effective algorithm for optimization analysis. It learns data by making decisions to maximize a user-specified reward. Users need to design appropriate model conditions such as the environments, actions, and rewards.

MBD problems can be mainly dealt with using supervised or reinforcement learning techniques since many MBD problems aim to seek robust and optimal design considering a set of design parameters.

To apply supervised learning, training data need to be prepared in advance for learning the model. The training data for MBD meta-models can be obtained in a in a few ways, usually by computational methods. In the case of reinforcement learning, a multibody systems
Data-driven multibody dynamics simulation

Simulation environment is requisite to train an agent according to cumulated reward for each action. Both learning approaches require time-consuming tasks to learn the meta-models of MBD: a data preparation task for supervised learning and simulation task for reinforcement learning. However, once the meta-model is built, it resolves MBD problems in real-time and yields dynamics responses.

In this research, the supervised learning of a MBD meta-model based on training data is mainly considered. Supervised learning finds an approximation function \( M \) that minimizes a loss \( L(x; M) \) over samples \( x \). An algorithm \( A \) produces \( M \) for a training set \( X^{\text{train}} \) through the optimization of a training criterion with respect to a set of given hyper-parameters \( \alpha \) [4]. The built function

\[
M = A(X^{\text{train}}; \alpha)
\]

is called a meta-model in this research.

A neural network algorithm is one of the powerful machine learning algorithms of minimizing the loss:

\[
L(x; A(X^{\text{train}}; \alpha)).
\]

Specifically, the algorithm uses a network structure and optimizes the parameters of the networks, i.e., weights and biases, by utilizing the back-propagation algorithms, which are an extension of the gradient descent method for neural network structures.

In this research, neural networks are adopted to build the meta-models of MBD problems, since they are subject to be generalized to fit various shapes of nonlinear functions with multi-dimensional input data. In particular, feed forward networks (FFNs) with hidden layers and nonlinear activation functions, which are the universal approximators that can represent effectively continuous functions, are considered. Owing to the characteristics of FFNs, they are a powerful candidate for implementing the meta-models of general-purposed MBD problems. Moreover, many techniques for DNNs including accelerated activation functions such as ReLU, dropout, regularization, and batch normalization have strengthened the potential of FFNs with deep layers for modeling general-purpose MBD problems.

The flowchart in Fig. 2 presents brief outlines of meta-modeling of MBD problems and its benefits.

**Design of neural networks for meta-models**

MBD problems rarely have high dimensionality of input or output data, compared to common DNN applications such as image, speech, and text data. Rather than high dimensionality, in general, MBD considers complicated nonlinear functions and requires accurate and robust solutions.

If an MBD problem is given, the design of input and output layers is typically decided. For example, each variable of input (or output) data is mapped to a single node of the input (or output) layer in cases where the variable is numeric one, but if the variable is nominal one, it should be mapped to multiple nodes through one-hot encoding. In statistical modeling, a nominal variable refers to a variable that has multiple categories, where there is no clear intrinsic ordering to them. Because of the lack of order, all categories in a nominal variable have equal and independent values. Gender and color are typical examples of the nominal variable. In the MBD system, material, loading, or geometry can be examples of nominal variables. In one-hot encoding, each value of the nominal variable is transformed to one of one-hot vectors,

\[
\{(1, 0, \ldots, 0), (0, 1, \ldots, 0), \ldots, (0, 0, \ldots, 1)\}.
\]
Different from input and output layers, the best design of hidden layers is hardly predictable a priori. The number of hidden layers and the number of nodes are the most critical hyper-parameters, and their optimal design must be decided along with other hyper-parameters at the step of hyper-parameter tuning. Empirically, it is known that deeper hidden layers are more effective than larger nodes of shallower hidden layers if two FFN models have similar numbers of parameters such as weights and biases.

To build expressive MBD meta-models, FFN models with enough width and depth are necessary. However, proper regularization methods such as $L_1$ and $L_2$ regularization, dropout and batch-normalization are required to achieve the generalized meta-models because FFN models with too many parameters are often overfitted to the given training data [10].

Optimization of hyper-parameters optimization of meta-models Similar to typical ML algorithms, the neural network algorithm does not provide a method to find the optimal hyper-parameters $\alpha$. Hyper-parameters of the DNN are critical to the accuracy and robustness of the meta-model. Unfortunately, there is no perfect scheme of building the most accurate and robust DNN model from a given training data. One must search the best set of hyper-parameters such as the number of hidden layers, the number of nodes in each hidden layers, activation function, optimization function, learning rate, and the number of epochs.

Generally, two kinds of search methods are often used for the purpose of hyper-parameter optimization; a given set of candidate values for each hyper-parameter are investigated with the grid search method, or randomly selected values for hyper-parameters are evaluated with the random search method. It is known that random search is more efficient to find optimal hyper-parameters than grid search [4]. Recently, AutoML has been actively researched in academic and practical fields to find the best DNN design. Once the AutoML techniques are
mature, it is expected that the optimal design of the DNN-based meta-models can be found in an easier and faster manners.

**Generation of MBD training data** In this paper, it is assumed that one can obtain as many MBD sample data as is needed to train the meta-model and achieve a reliable model. In other words, a case with an insufficient training data set is not considered. Nevertheless, since the process of MBD data collection takes a long time in the case of complex multibody systems, a more efficient manners of collecting training data is required.

First, the amount of training samples can be determined according to some criteria. Incremental learning methods can be applied to learn the meta-models. For instance, a certain level of performance measures such as the root mean-squared errors or the mean absolute percentage errors can be adopted for the criteria to stop feeding more samples to the meta-model. In the case of the random search method, simply more random samples can be provided to the less trained meta-model, and in the case of the grid search method, finer-grained grid samples can be applied.

Second, an appropriate range of a design parameter (where an arbitrary data sample is selected) can be adjusted or determined after multiple trials of training a meta-model. For example, if the sensitivity of a solution to a design parameter is highly sensitive, then a wide range for its sampling can deteriorate the performance of the meta-model. In this case, we need to narrow the sampling range. This is done under the assumption that the model complexity is often different in many ranges of nonlinear hyperplanes. In such cases, adaptive sampling methods such as focused grid search can be less exhaustive than uniform design of the typical grid search method [22].

### 3.2 Detailed assumptions and conditions for meta-modeling process

The following are some assumptions and comments on the meta-modeling that is developed for MBD problems. The same conditions are applied to the numerical tests in Sect. 4.

1. **Input and ouput parameters**
   - The input parameters for the MBD meta-modeling can be any factors that may affect the dynamics of the MBD system, e.g., the mass of the bodies, damping coefficients, the lengths of shafts, or external forces. However, not all the parameters are selected as inputs of the meta-model. We can assume various parameters are fixed as given constants. Only the parameters that we are interested in as regards their influences on the system dynamics are chosen as inputs. As an example, in this study, we do not consider the parameters related to a MBD solver such as parameters used for step size control in the integrator, a value that comes into play in the stopping criteria for an iterative solver, etc., as inputs of the meta-model.
   - Similarly, the output parameters can be any dynamic responses that we are interested in and want to know about, e.g., the time-transient displacements of masses and their time derivatives.

2. **Training data**
   - **Sufficiently many sets of training data**
     As mentioned in Sect. 3.1, it is assumed that there are as many sets of data for training and tests as one requires. Since the most important objective of this research is to achieve a highly accurate meta-model, other issues such as computational efficiency and problems of insufficient training data are not considered here.
Table 1  Structure of training data set for DNN, where time variable $t$ is considered as an input. This type of training data structure is denoted by $S_{\text{full}}$. In this case, a single meta-model is generated. Here, $L$ and $c$ denote the length of the massless rod and the damping coefficient, respectively; see Fig. 8

| Input | Output |
|-------|--------|
| $L$   | $c$    | $t$   | $\theta$ | $\dot{\theta}$ |
| 0.10  | 0.05   | 0.00  | 1.57080  | 0.00000  |
| 0.10  | 0.05   | 0.01  | 1.56590  | −0.97936 |
| 0.10  | 0.05   | 0.02  | 1.55122  | −1.95540 |
| 0.20  | 0.10   | 0.00  | 1.57080  | 0.00000  |
| 0.20  | 0.10   | 0.01  | 1.56590  | −0.97773 |
| 0.20  | 0.10   | 0.02  | 1.55126  | −1.94890 |

- **Uniform meshes**
  Training data for input parameters are uniformly meshed in a given finite range. For example, consider a single pendulum problem and assume the length of the massless rod $L$ is an input of a meta-model. Then, to get data samples for the training of the meta-model, we set a finite range $[L_1, L_2]$ and use the uniformly meshed samples $L_i = L_1 + i \Delta L$, $i = 0, 1, \ldots, n$, as training data, where $\Delta L := (L_2 - L_1)/n$ for an integer $n$.

- **Validation set**
  Out of the uniformly meshed data, 80% randomly chosen data is used to train a DNN model and is referred to as *training data*. The other 20%, referred to as *validation data*, is used to monitor the performance of the DNN model.

- **Data without noise**
  Training data for output responses such as displacements, velocities, or accelerations are exactly calculated from governing equations for MBD problems. In other words, training data are artificially generated without any noise.

- **Time variable and structures of training data**
  An important question in meta-modeling for dynamic problems is whether the time variable $t$ needs to be handled as an input parameter or not.

  Table 1 shows an example of a training data set, where the time variable $t$ is considered as an input. All the discrete time instants are contained in the set of training data. On the other hand, if the time variable is not considered as an input parameter, there are $\#\{t_n\}$ sets of training data, where time is fixed to $t = t_n$, as shown in Table 2. The two types of training data structures are referred to as $S_{\text{full}}$ and $S_{\text{fixed}}$.

  It may seem that $S_{\text{fixed}}$ is simpler than $S_{\text{full}}$, in that the former considers a fixed time instant $t = t_n$ and has a much smaller size of training data set compared to $S_{\text{full}}$, especially when the number of discrete time instants is very large. However, handling the time variable as a non-input ($S_{\text{fixed}}$) is not adequate for MBD analysis in the two following major aspects:

  (a) It is necessary to make as many meta-models as the number of discrete time instants $t_n$, $n = 0, 1, \ldots$. Moreover, if a grid search is performed for each meta-model to find the best hyper-parameters, this approach can be computationally infeasible.

  (b) Each resulting meta-model provides predictions only for a specific time $t = t_n$, which makes it difficult to figure out the time-varying tendency of MBD.

  Thus, in this research, it is concluded that a meta-model for MBD problems should be generated from training data of form $S_{\text{full}}$, where the time variable is considered as an input. More details of the training data structure and its results are described in Sect. 4.
Table 2 Structure of training data set for DNN, where time variable $t$ is fixed and not considered as an input. This type of training data structure is denoted by $S_{\text{fixed}}$. In this case, $\#\{n_t\}$ numbers of meta-models are generated corresponding to $\#\{n_t\}$ sets of training data. Here, $L$ and $c$ denote the length of the massless rod and the damping coefficient, respectively; see Fig. 8.

| Input | Output |
|-------|--------|
| $L$   | $c$    | $\theta$ | $\dot{\theta}$ |
| $t = 0.00$ |       |        |                 |
| 0.10  | 0.05   | 1.57080 | 0.00000         |
| 0.20  | 0.10   | 1.57080 | 0.00000         |
| $t = 0.01$ |       |        |                 |
| 0.10  | 0.05   | 1.57080 | 0.00000         |
| 0.20  | 0.10   | 1.56590 | -0.97773        |
| $t = 0.02$ |       |        |                 |
| 0.10  | 0.05   | 1.55122 | -1.95540        |
| 0.20  | 0.10   | 1.55126 | -1.94890        |

3. **Test data**

- **Unseen data**
  Samples, called test data, evaluate the performance of a finalized meta-model. They are independent of and unseen from training data. The size of the test data is the same as the uniformly meshed data set.

- **Randomly distributed data**
  Unlike training data, input parameters for the test are not uniformly meshed. They are randomly distributed in the same given range.

4. **Grid search and hyper-parameters**

A grid search is performed to find appropriate hyper-parameters for each MBD example, which helps to yield a highly accurate meta-model. From a grid search, the number of hidden layers, the number of nodes for each layer, the size of batches, the number of epochs, optimizer, and loss functions need to be decided. Still, there can be other sets of hyper-parameters that result in similar or better performance.

5. **Evaluation of performance**

In a training process, the performance of a meta-model $M$ is evaluated with the validation data in terms of two measures: $R$-squared value and absolute mean-squared error (MSE), denoted by $R^2$ and $\varepsilon_{\text{mse}}$, respectively. When an output label $y$ is given for a set of test data, and the meta-model $M$ yields a prediction $\hat{y}$ for the test set, the performance measures are defined by

$$
R^2(y, \hat{y}) := 1 - \frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2},
$$

(9)
Fig. 3 Slider–crank mechanism. The constant $\tau$, the length of crankshaft $r$ [m], the ratio of the lengths between the connecting rod and the crankshaft $L/r$ are arbitrarily determined within given ranges.

\[ \mathcal{E}_{\text{mse}}(y, \hat{y}) := \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2, \]

(10)

where $y = (y_1, \ldots, y_N)$, $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_N)$, $\bar{y} := \sum_{i=1}^{N} y_i / N$, and $N$ denotes the number of data samples of the output $y$. Here, for simplicity we assume $y$ is 1-dimensional. As the solution $\hat{y}$ of the meta-model predicts the label $y$ more accurately, the value of $R^2$ approaches 1, and the error $\mathcal{E}_{\text{mse}}$ approaches 0.

Remark 1 As many studies have developed various methods of neural networks and machine learning, other techniques can yield a surrogate model for FMBD with better accuracy or efficiency. However, since the purpose of this paper is to suggest a framework of real-time FMBD simulations based on a data-driven model, we adopt the most elementary form of a deep neural network.

4 Case studies

In this section, three fundamental MBD examples, a single pendulum, double pendulum, slider–crank mechanism, are investigated. For a more practical problem, we consider a vibrating transmission with bearing contact. For each example, a data-driven meta-model is generated through the FFN, and its performance is evaluated in various ways, as described in Sect. 3.2.

4.1 A kinematic example: slider–crank mechanism

Consider a slider–crank in Fig. 3, where the parameters $(r, L, \theta(t), \phi(t))$ represent, respectively, the length of the massless crankshaft [m], the length of the massless connecting rod [m], the angle of the crankshaft [rad], and the angle of the connecting rod [rad]. The initial angle $\theta^0$ [rad] and the initial velocity $\dot{\theta}^0$ [rad/s] are assumed as zeros, and the angular acceleration of the crankshaft $\ddot{\theta}(t)$ [rad/s$^2$] is given as

\[
\begin{align*}
\theta(t) &= \theta^0 = 0, \quad \text{where } t = 0, \\
\dot{\theta}(t) &= \dot{\theta}^0 = 0, \quad \text{where } t = 0, \\
\ddot{\theta}(t) &= \sin(\tau t), \quad \text{where } t \in [0, t_f],
\end{align*}
\]

(11)

for some constant $\tau \in \mathbb{R}$.
Table 3 Summary on parameters of slider–crank problem. In $S_{\text{fixed}}$, a fixed time instant is considered. In $S_{\text{full}}$, all the time instants are treated as inputs

| Parameters | Ranges | Meshsizes for training data | Meshsizes for test data |
|------------|--------|-----------------------------|------------------------|
| Fixed constants | $\theta^0$ [rad] | 0 | . | . |
| Inputs | $\tau$ | [1, 2] | $\Delta \tau = 0.1$ | arbitrary (not uniform) |
| | $r$ [m] | [1, 3] | $\Delta r = 0.2$ | arbitrary (not uniform) |
| | $L/r$ | [2.5, 3.5] | $\Delta (L/r) = 0.1$ | arbitrary (not uniform) |
| Time instants | $\{t_n\}$ [s] | [0, 5] | $\Delta t = 0.01$ ($t_0 = 0$) | $\Delta t = 0.01$ ($t_0 = 0$) |

The angle of the crankshaft $\theta(t)$ and its temporal derivatives can be rewritten explicitly, for $t \in [0, t_f]$,

$$\theta(t) = -\frac{1}{\tau^2} \sin(\tau t) + \frac{t}{\tau} + \theta^0 = -\frac{1}{\tau^2} \sin(\tau t) + \frac{t}{\tau},$$

$$\dot{\theta}(t) = -\frac{1}{\tau} \cos(\tau t) + \frac{1}{\tau} + \dot{\theta}^0 = -\frac{1}{\tau} \cos(\tau t) + \frac{1}{\tau}. \quad (12)$$

In DNN modeling, three independent parameters ($\tau, r, L/r$) are considered as inputs, while the time variable $t$ can be fixed to an instant ($S_{\text{fixed}}$) or considered as an input ($S_{\text{full}}$). More details on ranges and mesh sizes of parameters are summarized in Table 3.

Although the slider–crank mechanism is not a dynamic problem, this kinematic example is a good example because the kinematics should be treated as a special case of dynamic problems. To describe kinematics of the slider–crank, seven kinematic solutions, $\theta, \phi, \dot{\phi}, x_B, \dot{x}_B,$ and $\ddot{x}_B$, are considered as output parameters, where $x_B$ denotes the $x$-directional translation of the slider.

The output solutions other than $(\theta, \dot{\theta}, \ddot{\theta})$ can be found from kinematic equations as follows:

$$\begin{bmatrix} \phi \\ x_B \end{bmatrix} = \begin{bmatrix} \arcsin(-(r/L) \sin \theta) \\ r \cos \theta + L \cos \phi \end{bmatrix},$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{x}_B \end{bmatrix} = \begin{bmatrix} L \sin \phi \\ -L \cos \phi \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}^{-1} \begin{bmatrix} -r \dot{\theta} \sin \theta \\ r \dot{\theta} \cos \theta \end{bmatrix},$$

and

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{x}_B \end{bmatrix} = \begin{bmatrix} L \sin \phi \\ -L \cos \phi \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}^{-1} \left( \begin{bmatrix} -L \dot{\phi} \cos \phi \\ -L \dot{\phi} \sin \phi \end{bmatrix} \dot{\phi} + \begin{bmatrix} -r \dot{\theta} \cos \theta \\ -r \dot{\theta} \sin \theta \end{bmatrix} \right). \quad (15)$$

Two meta-models are generated from $S_{\text{fixed}}$ and $S_{\text{full}}$ types of training data, by employing the hyper-parameters found from grid searches for the case of $S_{\text{full}}$ shown in Table 4.

The scatter plots in Fig. 4 compares labels and predictions of the meta-model from $S_{\text{full}}$, and verifies that the meta-model produces almost accurate results. Its performance is much better than the other meta-models of previous examples, which seems to be caused by a simple form of kinematic equations (13) and a sufficient training data set. The $R^2$ values are over 0.999 for the kinematic responses $\theta, \phi, \dot{\phi}, x_B,$ and $\dot{x}_B$. 

 Springer
Table 4 Hyper-parameters for the slider–crank problem

| Hyper-parameters                  | Choice            |
|----------------------------------|-------------------|
| The number of hidden layers      | 2                 |
| The number of nodes in each layer| 128               |
| The size of batch                | 64                |
| The number of epochs             | 200               |
| Loss function                    | $\mathcal{E}_{mse}$ |
| Optimizer                        | Adam              |

Since the predictions for test data are highly accurate as confirmed in Fig. 4, Figs. 5, 6, and 7 present results only for a specific case of test data: $\tau = 1.780$, $r = 1.360$, $L/r = 3.050$. Figure 5 shows changes of translation and velocities of the slider mass $B$ in time $t$. As shown in Sects. 4.2.1 and 4.2.2, $S_{\text{fixed}}$ (left) shows oscillatory waves, while $S_{\text{full}}$ yields smooth solutions. The error $\mathcal{E}_{mse}$ compares the difference of their accuracies more clearly.

Figure 6 displays time-varying relations between the angle of the connecting rod $\phi(t)$ and its temporal derivatives ($\dot{\phi}(t), \ddot{\phi}(t)$). The oscillations from the case of $S_{\text{fixed}}$ (left) are observed. Figure 7 shows the relations between the displacement of slider $x_B$ and its derivatives. The performance of two training data set $S_{\text{fixed}}$ (left) and $S_{\text{full}}$ (Right) is clearer than Fig. 6. $S_{\text{full}}$ yields smoother and more accurate results than $S_{\text{fixed}}$.

4.2 Dynamic examples

4.2.1 Damped single pendulum

Using the embedding technique, equation of motion for a damped single pendulum problem shown in Fig. 8 can be expressed in the following mathematical governing equation:

$$\begin{cases}
\ddot{\theta} + \frac{g}{L} \sin(\theta) + \frac{c}{mL} \dot{\theta} = 0, & \text{where } \theta = \theta(t), \ t \in [0, t_f], \\
\theta(t) = \theta^0, \quad \dot{\theta}(t) = \dot{\theta}^0, & \text{where } t = 0,
\end{cases}
$$

(16)

where $g$ is the gravity acceleration, $L$ is the length of the massless rod, $m$ is the mass, and $c$ is the damping coefficient, respectively. The variables $\theta$ and $\dot{\theta}$ are the time-varying angle and its time derivative, whose initial values are specified as $\theta^0$ and $\dot{\theta}^0$, respectively.

Although all the input parameters ($g, L, m, c, \theta^0, \dot{\theta}^0$) affect dynamics of the simple pendulum in Fig. 8, we assume the parameters ($L, c, \theta^0$) are the major factors of interest. Thus, the other parameters ($g, m, \dot{\theta}^0$) are fixed to values ($9.81 \, \text{[m/s$^2$]}, 0.3 \, \text{[kg]}, \pi/2 \, \text{[rad/s]}$), while the parameters ($L, c, \theta^0$) are not determined specifically.

It is the objective of this example to generate a meta-model that yields the dynamics of the damped single pendulum as outputs when a particular set of input parameters ($L, c, \theta^0$) is given. It should be noted that one may build a meta-model that depends on other parameters.

For efficient learning, it is assumed that ($L, c, \theta^0$) are chosen within finite ranges:

$L \, \text{[m]} \in [0.1, 0.2] \quad (\Delta L = 0.01),$

$c \, \text{[kg} \cdot \text{m/s]} \in [0, 0.15] \quad (\Delta c = 0.01),$

$\theta^0 \, \text{[rad]} \in [-\pi/2, \pi/2] \quad (\Delta \theta^0 = \pi/10).$
Here, $(\Delta L, \Delta c, \Delta \theta^0)$ denote uniform meshsizes for training data. In evaluating a metamodel, the uniform meshes are not applied, and arbitrarily chosen input values are used.

To describe the dynamics of the damped single pendulum, the time-varying solutions $\theta(t), \dot{\theta}(t), \text{and } \ddot{\theta}(t)$ are achieved as outputs of a meta-model. For the time variable $t$, discrete
time instants \( \{ t_n \} \) with a uniform mesh size \( \Delta t \) is considered in an interval \( [0, t_f] \), where \( t_f = 2 \):

\[
t_n \text{ [s]} := n \Delta t \in [0, 2] \quad (\Delta t = 10^{-2}) .
\]

for \( n = 0, 1, \ldots, 200 \).
As described in Sect. 3.2, the time variable $t$ can be handled as an input ($S_{\text{full}}$) or fixed to a certain instant ($S_{\text{fixed}}$). In other words, the approach $S_{\text{full}}$ and $S_{\text{fixed}}$ consider four and three dimensional inputs, respectively. Thus, the $S_{\text{full}}$ case generates only one meta-model by using 389,136 number of training data, while the $S_{\text{fixed}}$ case develops $\# \{t_n\} = 201$ meta-models by using 1,936 training data for each model. Results from the two structures are compared.

Hyper-parameters for the approach $S_{\text{fixed}}$ found from the grid search are shown in Table 5. Figure 9 displays the scatter plots where labels, i.e. reference solutions, and predictions of outputs ($\theta, \dot{\theta}, \ddot{\theta}$) are compared. The results are achieved from a set of test data, which are unseen from training. The $R^2$ scores are around 0.997, which implies that the DNN model predicts the outputs with high accuracy.

Figure 10 shows the dynamics of angle ($\theta$) (top), angular velocity ($\dot{\theta}$) (middle), and angular acceleration ($\ddot{\theta}$) (bottom), for a specific case: $L = 0.113$ [m], $c = 0.093$ [kg · m/s], $\theta^0 = 0.716$ [rad]. Labels (blue dashed, crosses) and predictions (red solid, circles) are shown for each solution. Results of $S_{\text{fixed}}$ (left) and $S_{\text{full}}$ (right) are compared. Although both $S_{\text{fixed}}$
Fig. 7 Relations between dynamic responses of slider–crank problem when $\tau = 1.780$, $r = 1.360$, and $L/r = 3.050$: Labels (black dashed) and predictions (red solid, circles) are given. Results of $S_{\text{fixed}}$ (left) and $S_{\text{full}}$ (right) are compared (Color figure online)

Fig. 8 A damped single pendulum problem. The length of the massless rod $L$, the damping coefficient $c$, and the initial angle $\theta(t = 0) = \theta^0$ are arbitrarily determined within given ranges and $S_{\text{full}}$ yields highly accurate results, some oscillations are observed in the case of $S_{\text{fixed}}$ (left). On the other hand, $S_{\text{full}}$ (Right) gives relatively smooth solutions.

In Fig. 11, a performance comparison of $S_{\text{fixed}}$ (left) and $S_{\text{full}}$ (Right) for other input parameters (selected from test data) are summarized in Table 6: Similarly as in Fig. 10, oscillations...
Table 5  Hyper-parameters for the damped single pendulum problem for $S_{\text{fixed}}$

| Hyper-parameters                      | Choice |
|---------------------------------------|--------|
| The number of hidden layers           | 3      |
| The number of nodes in each layer     | 256    |
| The size of batch                     | 256    |
| The number of epochs                  | 100    |
| Loss function                         | $\ell_{\text{mse}}$ |
| Optimizer                             | Adam   |

Fig. 9  Labels vs. predictions for test data. The meta-model for the damped single pendulum problem is generated from a $S_{\text{full}}$ type of training set.

Torsion waves are observed in the case of $S_{\text{fixed}}$. Some are more severe than others, which makes prediction error greater. On the other hand, $S_{\text{full}}$ yields smooth and accurate predictions for all cases.
Hyper-parameters for $S_{\text{full}}$ and $S_{\text{fixed}}$

Since the performance of a DNN depends on its architecture or the hyper-parameters, it would be the best to carry out an independent grid search for each structure and each training data set, in comparing the results of $S_{\text{full}}$ and $S_{\text{fixed}}$. 

---

Fig. 10 Dynamic responses of the damped single pendulum for specific input $L = 0.113$, $c = 0.093$, $\theta^0 = 0.716$. Results of $S_{\text{fixed}}$ (left) and $S_{\text{full}}$ (right) are compared (Color figure online)
Fig. 11 Dynamic responses of single pendulum for multiple inputs; see Table 6. Results of $S_{\text{fixed}}$ (left) and $S_{\text{full}}$ (right) are compared (Color figure online)

For $S_{\text{full}}$, we need to perform a single grid search; see Table 5.

On the other hand, notice that the $S_{\text{fixed}}$ approach needs $\#(t_n) = 201$ number of grid searches for the $\#(t_n)$ training data sets. These independent grid searches would yield the best results for the $S_{\text{fixed}}$ approach. However, such grid searches require an extremely heavy
Table 6 Input parameters of multiple cases for Fig. 11

| Case  | \(L\) [m]     | \(c\) [kg·m/s] | \(\theta^0\) [rad] |
|-------|---------------|----------------|-------------------|
| Case 1| 0.143         | 0.071          | −1.210            |
| Case 2| 0.161         | 0.023          | 0.627             |
| Case 3| 0.180         | 0.131          | −0.599            |
| Case 4| 0.180         | 0.035          | 0.402             |

Table 7 Hyper-parameters for the training data sets of \(S_{\text{fixed}}\)

| Model for \(t = t_0\) | The number of hidden layers | The number of nodes per a hidden layer | The size of batch |
|------------------------|------------------------------|---------------------------------------|-------------------|
| \(t_0 \in [0, 50]\)    | 2                            | 64                                    | 2                 |
| \(t_0 \in [50, 100]\)   | 4                            | 64                                    | 8                 |
| \(t_0 \in [100, 150]\)  | 2                            | 128                                   | 8                 |
| \(t_0 \in [150, 200]\)  | 3                            | 256                                   | 8                 |

computational burden. To reduce the computation cost for a feasible training, we perform 4 independent grid searches for the training data sets for \(t = 50, 100, 150,\) and 200; see Table 7. Each hyper-parameter set achieved from \(t = 50, 100, 150,\) and 200 is used for the data sets for \(t \in [0, 50],\) \(t \in [50, 100],\) \(t \in [100, 150],\) and \(t \in [150, 200],\) respectively. In Figs. 10 and 11, we use the hyper-parameters.

It is worth noting that even with the grid searches, the results of \(S_{\text{fixed}}\) still show oscillations and less smooth solutions compared to the results of \(S_{\text{full}}\). Thus, the usage of possibly not optimal hyper-parameters for the \(S_{\text{fixed}}\) approach would not be a serious hindrance to comparing the performance of the \(S_{\text{full}}\) and the \(S_{\text{fixed}}\) approaches. For simplicity and computational feasibility, a simplified grid search is performed for \(S_{\text{fixed}}\) in the other numerical examples. More details are described in each numerical example.

4.2.2 Double pendulum

Equations of motion for a double pendulum system depicted in Fig. 13 can be again expressed using embedded technique. The equations of motion can be written as

\[
\begin{align*}
(m_1 + m_2)L_1\ddot{\theta}_1 + m_2L_2\ddot{\theta}_2 \cos(\theta_1 - \theta_2) + m_2L_2\ddot{\theta}_2^2 \sin(\theta_1 - \theta_2) + (m_1 + m_2)g \sin(\theta_1) &= 0, \\
m_2L_2\ddot{\theta}_2 + m_2L_1\ddot{\theta}_1 \cos(\theta_1 - \theta_2) - m_2L_1\ddot{\theta}_2 \sin(\theta_1 - \theta_2) + m_2g \sin(\theta_2) &= 0, \\
\dot{\theta}_i(t) &= \dot{\theta}_i^0, \quad \dot{\theta}_i(t) = \dot{\theta}_i^0, \quad \text{where} \quad t = 0, \quad i = 1, 2,
\end{align*}
\]

where \(\theta_i = \theta_i(t)\) and \(t \in [0, t_f]\), \(i = 1, 2\), represent the time-varying angles of the links, as shown in Fig. 13. The parameter \(g\) is the gravity constant, \(L_i\) is the length of the massless rod \(i\), \(m_i\) is the mass, \(\theta_0^i\) is the initial angle, \(\dot{\theta}_i^0\) is the initial angular velocity, and \(i = 1, 2\), body notation, respectively.

In the meta-modeling, it is assumed that \((L_1, L_2, \dot{\theta}_1^0, \dot{\theta}_2^0)\) are independent input parameters and \((\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2)\) are output parameters. As in the single pendulum problem (16), inputs are chosen within some ranges. The other parameters are fixed to given constants. More details on ranges and mesh sizes of parameters are summarized in Table 8.
Data-driven multibody dynamics simulation

Fig. 12 The damped single pendulum results achieved from $S_{\text{full}}$ when different initial configurations are given: $\theta_0 = \pi/2$ (left), $\theta_0 = -\pi/2$ (Right). Top: the initial configurations. Middle: when $c = 0$. Bottom: when $L = 0.2$ (Color figure online)

As in the previous numerical example, two types of training data, i.e. $S_{\text{fixed}}$ and $S_{\text{full}}$ are compared. For $S_{\text{fixed}}$, there are $\#\{t_n\} = 501$ meta-models, where each model is trained from 14,641 numbers of the data set. For $S_{\text{full}}$, there is only one meta-model trained from
Fig. 13 Double pendulum problem. The lengths of the massless rods $L_1, L_2$, and the initial angular velocities $\dot{\theta}_1^0, \dot{\theta}_2^0$ are arbitrarily determined within given ranges.

Table 8 Summary on parameters of double pendulum problem. In $S_{\text{fixed}}$, a fixed time instant is considered. In $S_{\text{full}}$, all the time instants are treated as inputs.

| Parameters | Ranges | Meshsizes for training data | Meshsizes for Test Data |
|------------|--------|-----------------------------|-------------------------|
| Fixed constants | $g$ [m/s²] | 9.81 | . | . |
| $m_1$ [kg] | 2.0 | . | . |
| $m_2$ [kg] | 1.0 | . | . |
| $\theta_1^0$ [rad] | 1.6 | . | . |
| $\theta_2^0$ [rad] | 1.6 | . | . |
| Inputs | $L_1$ [m] | [1, 2] | $\Delta L_1 = 0.1$ | arbitrary (not uniform) |
| $L_2$ [m] | [2, 3] | $\Delta L_2 = 0.1$ | arbitrary (not uniform) |
| $\dot{\theta}_1^0$ [rad/s] | [0, 0.1] | $\Delta \dot{\theta}_1^0 = 0.01$ | arbitrary (not uniform) |
| $\dot{\theta}_2^0$ [rad/s] | [0.3, 0.5] | $\Delta \dot{\theta}_2^0 = 0.02$ | arbitrary (not uniform) |
| Time instants | $\{t_n\}$ [s] | [0, 5] | $\Delta t = 0.01$ ($t_0 = 0$) | $\Delta t = 0.01$ ($t_0 = 0$) |

Table 9 Hyper-parameters for the double pendulum problem

| Hyper-parameters | Choice |
|------------------|--------|
| The number of hidden layers | 4 |
| The number of nodes in each layer | 64 |
| The size of batch | 1024 |
| The number of epochs | 400 |
| Loss function | $\ell_{\text{mse}}$ |
| Optimizer | Adam |

$14,641 \times 501 = 7,335,141$ numbers of the data set. For both $S_{\text{fixed}}$ and $S_{\text{full}}$ types of training data, hyper-parameters are found, as given in Table 9.

The scatter plots in Fig. 14 show that a meta-model from $S_{\text{full}}$ predicts output parameters $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2)$ with a great accuracy. The $R^2$ values are over 0.997 in all cases of solutions.
Fig. 14  Labels vs. predictions for test data. The meta-model for the double pendulum problem is generated from a $S_{\text{full}}$ type of training set.

Table 10  Input parameters of multiple cases for Figs. 15 and 16

| Case  | $L_1$ [m] | $L_2$ [m] | $\dot{\theta}_1$ [rad/s] | $\dot{\theta}_2$ [rad/s] |
|-------|-----------|-----------|--------------------------|--------------------------|
| Case 1 | 1.010     | 2.130     | 0.00                     | 0.300                    |
| Case 2 | 1.500     | 2.410     | 0.03                     | 0.330                    |
| Case 3 | 1.620     | 2.560     | 0.044                    | 0.384                    |
| Case 4 | 1.330     | 2.820     | 0.062                    | 0.412                    |
| Case 5 | 1.980     | 2.940     | 0.087                    | 0.470                    |

Performances of meta-models from $S_{\text{fixed}}$ and $S_{\text{full}}$ types of training data are compared in Fig. 15 and 16. The comparison shows dynamic changes of predictions (solid) from meta-models in comparison with their labels (dashed), for multiple cases as shown in Table 10.

As observed in the single pendulum cases shown in Figs. 10 and 11, the meta-model from $S_{\text{fixed}}$ shows many oscillations in its dynamic responses. Here the oscillations are quite severe, especially when $t$ is large. Though these results can be improved if more appropriate hyper-parameters are employed for each of $\#\{t_n\}$ number of meta-models, the grid searches
are computationally infeasible. On the other hand, the meta-model from $S_{\text{full}}$ yields more accurate and smoother dynamic responses.

The difference between the two training data sets $S_{\text{fixed}}$ and $S_{\text{full}}$ is shown more clearly in Fig. 17, where the trajectories of two masses $m_1$ and $m_2$ are shown. Labels ($m_1$: black solid, $m_2$: black dashed) and predictions ($m_1$: blue solid, circles, $m_2$: red solid, circles) are given for the results from $S_{\text{fixed}}$ (left) and $S_{\text{full}}$ (Right). Each plot is from particular input parameters: $L_1 = 1.500$ [m], $L_2 = 2.410$ [m], $\dot{\theta}_1^0 = 0.03$ [rad/s], $\dot{\theta}_2^0 = 0.330$ [rad/s] (top), $L_1 = 1.980$ [m], $L_2 = 2.940$ [m], $\dot{\theta}_1^0 = 0.087$ [rad/s], $\dot{\theta}_2^0 = 0.470$ [rad/s] (middle), $L_1 = 1.400$ [m], $L_2 = 2.500$ [m], $\dot{\theta}_1^0 = 0.060$ [rad/s], $\dot{\theta}_2^0 = 0.380$ [rad/s] (bottom).

### 4.2.3 Vibrating transmission with bearing contact

Accuracy of the DNN-based meta-modeling technique, particularly $S_{\text{full}}$ was well evaluated in the previous three examples. However, those are relatively simple, and thus those governing equations can also be analytically solved in real-time. Therefore, more practical multibody dynamics models should be considered to evaluate the real-time estimation.
performance of the proposed algorithm. To handle this issue, we here consider a vibrating transmission system shown in Fig. 18.

When the driving torque $\tau$ is imposed to the right shaft, the links transfer the rotating motion of the shaft to vibrate the cam on the left. The bearings are described as green bodies, and the yellow surfaces are contact surfaces between the bearings and the housing. The bearings and shafts are connected with bushing forces, which represent the joint clearance. The $\gamma$-directional rotational stiffness $K^\gamma_R$ of each shaft is zero. We fix the density of the housing $\rho_{\text{housing}}$, the rotational stiffnesses $K^x_R, K^z_R$, and the translational and rotational damping coefficients of the bushing $C_T, C_R$ as

$$\rho_{\text{housing}} = 7.85 \times 10^{-6} [\text{kg/mm}^3], \quad K^x_R = K^z_R = 5 \times 10^6 [\text{N-mm/rad}],$$
$$C_T = 10 [\text{N-s/mm}], \quad C_R = 5000 [\text{N-mm-s/rad}].$$

For the meta-modeling, we employ the numerical formulation in Sect. 2 to generate data samples, while the other numerical examples (in Sects. 4.1, 4.2.1, 4.2.2) use the analytic EOMs. For the $S_{\text{full}}$ approach, we construct a DNN model with 4-dimensional input $(t, \tau, \rho, K_T)$, where $t$ is time, $\tau$ is the driving torque, $\rho$ is density of the bodies except
the housing ($\rho_{\text{housing}}$), and $K_T = [K_T^x, K_T^y, K_T^z]$ is the translational stiffness of the bushing forces. The $S_{\text{fixed}}$ approach considers 3-dimensional input $(\tau, \rho, K_T)$. For both $S_{\text{full}}$ and $S_{\text{full}}$, the input parameters are chosen from finite ranges; see Table 11.
Fig. 18 Vibrating transmission system. The driving torque $\tau$, the density $\rho$ of the bodies except to the housing, and the translational bushing stiffness $K_T$ of the shafts and the bearings are arbitrarily chosen within given ranges.

Table 11 Summary on parameters of the vibrating transmission system. In $S_{\text{fixed}}$, a fixed time instant is considered. In $S_{\text{full}}$, all the time instants are treated as inputs.

| Parameters | Ranges | Meshsizes for training data | Meshsizes for test data |
|------------|--------|-----------------------------|-------------------------|
| Inputs     | $\tau$ [N·mm] $[300, 500]$ | $\Delta \tau = 22.2222$ arbitrary (not uniform) | $\Delta \tau = 22.2222$ arbitrary (not uniform) |
|            | $\rho$ [kg/mm$^3$] $[2.71 \times 10^{-6}, 7.85 \times 10^{-6}]$ | $\Delta \rho = 5.7111 \times 10^{-7}$ arbitrary (not uniform) | $\Delta \rho = 5.7111 \times 10^{-7}$ arbitrary (not uniform) |
|            | $K_T$ [N/mm] $[1000, 10000]$ | $\Delta K_T = 1000$ arbitrary (not uniform) | $\Delta K_T = 1000$ arbitrary (not uniform) |
| Time instants $\{t_n\}$ [s] $[0, 0.1]$ | $\Delta t = 0.0002 (t_0 = 0)$ | $\Delta t = 0.0002 (t_0 = 0)$ |

Table 12 Hyper-parameters of the $S_{\text{full}}$ approach for the vibrating transmission system.

| Hyper-parameters | Choice |
|------------------|--------|
| The number of hidden layers | 9 |
| The number of nodes in each layer | 512 |
| The size of batch | 2048 |
| The number of epochs | 500 |
| Loss function | $\epsilon_{\text{mse}}$ |
| Optimizer | Adam |

The DNN model has 9-dimensional output $(x_A, y_A, z_A, \dot{x}_A, \dot{y}_A, \dot{z}_A, \ddot{x}_A, \ddot{y}_A, \ddot{z}_A)$, where $x_A, y_A, z_A$ are the positions, $\dot{x}_A, \dot{y}_A, \dot{z}_A$ are velocities, and $\ddot{x}_A, \ddot{y}_A, \ddot{z}_A$ are accelerations of $A$ in the $x$-, $y$-, $z$-direction; see Fig. 18.

For training, the hyper-parameters are found from grid search as in Table 12.
Fig. 19  Labels vs. predictions for test data. The meta-model for the vibrating transmission system is generated from $S_{\text{full}}$ type of training set

The scatter plots in Fig. 19 show that a meta-model from $S_{\text{full}}$ predicts output parameters $(x_A, y_A, z_A, \dot{x}_A, \dot{y}_A, \dot{z}_A, \ddot{x}_A, \ddot{y}_A, \ddot{z}_A)$ with good accuracy. The $R^2$ values are over 0.994 for all outputs.

Figure 20 shows time-transient dynamics of positions $(x_A, y_A, z_A)$ for a specific case: $\tau = 316.85$ [N · mm], $\rho = 7.27e^{-6}$ [kg/mm$^3$], $K_T = 1858.4$ [N/mm]. Labels (blue dashed, crosses) and predictions (red solid, circles) are shown for each solution. The results of $S_{\text{fixed}}$ (left) and $S_{\text{full}}$ (Right) are compared. Although both $S_{\text{fixed}}$ and $S_{\text{full}}$ result in accurate results, some oscillations are observed in the case of $S_{\text{fixed}}$ (left). On the other hand, $S_{\text{full}}$ (Right) yields smoother and more accurate solutions.

The difference in the performances of $S_{\text{fixed}}$ and $S_{\text{full}}$ approaches is shown more clearly in Figs. 21 and 22, where time-transient velocities and accelerations are depicted for multiple cases; see Table 13.

Figure 23 shows trajectories of $A$ in time. Labels (black dashed) and predictions (red solid circles) are given for the results from $S_{\text{fixed}}$ (left) and $S_{\text{full}}$ (Right). Each plot is from particular input parameter cases; $\tau = 316.85$ [N · mm], $\rho = 7.83e^{-6}$ [kg/mm$^3$], $K_T =$
Fig. 20 Time-transient positions $x_A, y_A, z_A$ of the vibrating transmission system for specific input $\tau = 316.85, \rho = 7.27 \times 10^{-6}, K_T = 1858.4$ (Color figure online)

$2318.6$ [N/mm] (top), $\tau = 490.18$ [N mm], $\rho = 4.41 e^{-6}$ [kg/mm$^3$], $K_T = 3918$ [N/mm] (bottom).

Figure 24 shows computation clock times (CPU time [s]) for attaining a solution for specific inputs $\tau = 403.07, \rho = 3.04 \times 10^{-6}, K_T = 2318.6$. It compares CPU times of a numerical solver (black circles) and a DNN model from $S_{full}$ approach (red circles). For
Fig. 21 Time-transient velocities of the vibrating transmission system for multiple cases of input parameters; see Table 13. Labels (dashed) and predictions (solid) are given for test data. Results of $S_{\text{fixed}}$ (left) and $S_{\text{full}}$ (right) are compared (Color figure online)

301 time instants over an interval $[t_0, t_{300}]$ (left), it takes 28.4096 (s) and 0.2496 (s) for the numerical solver and the DNN model, respectively. The log-scaled CPU time comparison is also provided in the right figure. This clearly demonstrates the feasibility of applying a meta-model to real-time MBD simulations.
However, it is also clear that the learning time with the practical model becomes more significant with model complexity and the number of learning parameters. To overcome this
Table 13  Input parameters of multiple cases for Figs. 21 and 22

| Case | $\tau$   | $\rho$           | $K_T$  |
|------|----------|-----------------|--------|
| 1    | 312.08   | $7.27e^{-6}$    | 1105.1 |
| 2    | 326.87   | $3.04e^{-6}$    | 8733.9 |
| 3    | 403.07   | $7.83e^{-6}$    | 6680.3 |

issue, we can employ recent advanced smart sampling techniques or parametric reduced-order modeling in the proposed algorithm.

5 Conclusions

In this paper, a deep neural network-based meta-modeling technique is proposed for real-time prediction of time-varying responses in multibody dynamics simulation. The paper
Fig. 24 CPU time comparisons of the vibrating transmission system for a specific input \( \tau = 403.07 \), \( \rho = 3.04 \times 10^{-6} \), \( K_T = 2318.6 \). Left: CPU times for over a time interval \([t_0, t_{300}]\) (i.e., 301 time instants). Right: Log-scaled CPU times. Each circle implies a simulation trial through a solver (Color figure online)

contributes to data-driven modeling for multibody systems in two meaningful aspects. The first is that Deep Neural Network learning is applied, not to a specified particular type, but to a general multibody dynamic problem. The meta-model trained by DNN instantly provides time-varying responses of multibody dynamics without directly solving its governing equations. The generality also makes it possible for the proposed DNN algorithm to be employed for other multibody system problems in future research. The second is that the present work analyzes and suggests how training data should be structured for more effective DNN learning. In particular, it is found that treating the time variable as an input parameter enhances accuracy and smoothness of the resulting predictions. The observation is worthwhile to notice, since the smoothness of physical variables in the time direction is significant in dynamic problems. Through kinematics to multibody dynamics with contact, this paper demonstrates that accurate solutions can be achieved by the DNN procedure for general-purpose multibody dynamics.

Despite the introduced numerical results, the present data-based learning algorithm can be improved through further studies. First, performing smart sampling which decides more suitable ranges and non-uniform mesh sizes of data that will improve computational efficiency in generating a meta-model. Effective learning procedures with robust machine learning techniques become more significant to construct meta-models for flexible multibody dynamics problems with much larger dimensions and complexities. Moreover, to make fundamental progress in data-driven design of MBD, further studies are required on other various subjects, from theories on probability, uncertainties, and physics, to brand-new data-handling techniques.

Acknowledgements This research is supported by 2019 KyungHee University research program and Functionbay Inc., and the authors would like to acknowledge the support for Grzegorz Orzechowski from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie project No. 845600 (RealFlex).

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.
References

1. Agrawal, O.P., Shabana, A.A.: Dynamic analysis of multibody systems using component modes. Comput. Struct. 21(6), 1303–1312 (1985)
2. Angeli, A., Naets, F., Desmet, W.: Data-driven model order reduction for real-time multibody simulations. In: The 5th Joint International Conference on Multibody System Dynamics, IMSD (2018)
3. Ansari, H., Tupy, M., Datar, M., Negrut, D.: Construction and Use of Surrogate Models for the Dynamic Analysis of Multibody Systems. SAE International by Columbia Univ. (2018)
4. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. 13, 281–305 (2012)
5. Besselink, B., Tabuk, A., Lotowska, A., van de Wouw, N., Nijmeijer, H., Rixen, D.J., Hochstenbach, M.E., Schilders, W.H.A.: A comparison of model reduction techniques from structural dynamics, numerical mathematics and systems and control. J. Sound Vib. 332, 4403–4422 (2013)
6. Blanco-Claraco, J.L., Torres-Moreno, J.L., Giménez-Fernández, A.: Multibody dynamic systems as Bayesian networks: applications to robust state estimation of mechanisms. Multibody Syst. Dyn. 34, 103–128 (2015)
7. Byravan, A., Fox, D.: SE3-nets: learning rigid body motion using deep neural networks. In: IEEE International Conference on Robotics and Automation (ICRA) (2017)
8. Eldred, M., Giunta, A., Collis, S.: Second-Order Corrections for Surrogate-Based Optimization with Model Hierarchies. 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference. AIAA, Washington (2004)
9. Falomi, S., Malvezzi, M., Meli, E.: Multibody modeling of railway vehicles: innovative algorithms for the detection of wheel-rail contact points. Wear 271, 453–461 (2011)
10. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016)
11. Halloran, J.P., Erdemir, A., van den Bogert, A.J.: Adaptive surrogate modeling for efficient coupling of musculoskeletal control and tissue deformation models. J. Biomech. Eng. 131(1), 011014 (2009)
12. Hinton, G.: Neural Networks for Machine Learning—Overview of mini-batch gradient descent (2012)
13. Kim, S.S., Haug, E.J.: A recursive formulation for flexible multibody dynamics, part I: open-loop systems. Comput. Methods Appl. Mech. Eng. 71(3), 293–314 (1988)
14. Kim, J.G., Han, J.B., Lee, H., Kim, S.S.: Flexible multibody dynamics using coordinate reduction improved by dynamic correction. Multibody Syst. Dyn. 42(4), 411–429 (2018)
15. Kingma, D.P., Ba, J.J.: Adam: a method for stochastic optimization (2014). arXiv:1412.6980v9
16. Kraft, S., Causse, J., Martinez, A.: Black-box modelling of nonlinear railway vehicle dynamics for track geometry assessment using neural networks. Int. J. Veh. Mech. Mobil. (2018). https://doi.org/10.1080/00423114.2018.1497186
17. Lanz, O.: Approximate Bayesian multibody tracking. IEEE Trans. Pattern Anal. Mach. Intell. 28, 9 (2006)
18. Li, Y., Wu, J., Tedrake, R., Tenenbaum, J.B., Torralba, A.: Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In: ICLR (2019)
19. Lin, Y-C., Haftka, R.T., Queipo, N.V., Fregly, B.J.: Surrogate articular contact models for computationally efficient multibody dynamic simulations. Med. Eng. Phys. 32(6), 584–594 (2010)
20. Martin, T.P., Zaaazaa, K.E., Whitten, B., Tajaddini, A.: Using a multibody dynamic simulation code with neural network technology to predict railroad vehicle-track interaction performance in real time. In: Proceedings of the ASME 2007 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference (2007)
21. Negrut, D., Qian, Z.G.P., Khude, N.: Building Gaussian process based metamodels using variable-fidelity experiments for dynamic. In: Analysis of Mechanical Systems, ASME International Mechanical Engineering Congress and Exposition, ASME (2007)
22. Pontes, F.J., Amorim, G.F., Balestrassi, P.P., Paiva, A.P., Ferreira, J.R.: Design of experiments and focused grid search for neural network parameter optimization. Neurocomputing 186, 22–34 (2016)
23. Rewienski, M., White, J.: A trajectory piecewise-linear approach to model order reduction and fast simulation of nonlinear circuits and micromachined devices. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 22, 155–170 (2003)
24. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature 323, 533–536 (1986)
25. Shabana, A.A.: Dynamics of Multibody Systems. Cambridge University Press, Cambridge (2005)
26. Ting, J-A., Mistry, M., Peters, J., Schaal, S., Nakanishi, J.: A Bayesian approach to nonlinear parameter identification for rigid body dynamics. In: Robotics: Science and Systems II, pp. 247–254 (2007)
27. Tutsy, O., Brown, M., Wang, H.: Reinforcement learning algorithm application and multi-body system design by using MapleSim and Modelica. Int. J. Adv. Mechatron. Syst., 650–655 (2012)