Implicit Contact Dynamics Modeling With Explicit Inertia Matrix Representation for Real-Time, Model-Based Control in Physical Environment

Takeshi D. Itoh
itoh.takeshi.ik4@is.naist.jp
Department of Brain Robot Interface, ATR Computational Neuroscience Laboratories, Kyoto, 619-0288, Japan, and Graduate School of Science and Technology, Nara Institute of Science and Technology, Nara, 630-0192, Japan

Koji Ishihara
ishihara-k@atr.jp
Department of Brain Robot Interface, ATR Computational Neuroscience Laboratories, Kyoto, 619-0288, Japan

Jun Morimoto[*]
xmorimo@atr.jp
Department of Brain Robot Interface, ATR Computational Neuroscience Laboratories, Kyoto, 619-0288, Japan and Graduate School of Informatics, Kyoto University, Kyoto, 606-8501, Japan

Model-based control has great potential for use in real robots due to its high sampling efficiency. Nevertheless, dealing with physical contacts and generating accurate motions are inevitable for practical robot control tasks, such as precise manipulation. For a real-time, model-based approach, the difficulty of contact-rich tasks that requires precise movement lies in the fact that a model needs to accurately predict forthcoming contact events within a limited length of time rather than detect them afterward with sensors. Therefore, in this study, we investigate whether and how neural network models can learn a task-related model useful enough for model-based control, that is, a model predicting future states, including contact events. To this end, we propose a structured neural network model predictive control (SNN-MPC) method, whose neural network architecture is designed with explicit inertia matrix representation. To train the proposed network, we develop a two-stage modeling procedure for contact-rich dynamics from a limited number of samples. As a contact-rich task, we take up a trackball manipulation task using a physical 3-DoF finger robot. The results showed that the SNN-MPC

[*]Jun Morimoto is the corresponding author.

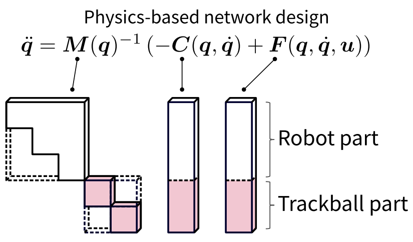
outperformed MPC with a conventional fully connected network model on the manipulation task.

1 Introduction

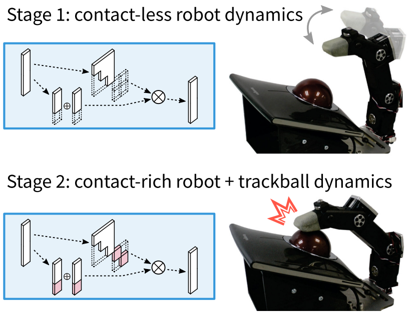
The application of reinforcement learning or optimal control methods using (deep) neural networks to robot control has been a popular research topic (Gu, Holly, Lillicrap, & Levine, 2017; Levine, Pastor, Krizhevsky, Ibarz, & Quillen, 2018). In particular, model-based policy learning has great potential for use in physical robots due to its sample efficiency (Deisenroth, Fox, & Rasmussen, 2015; Lenz, Knepper, & Saxena, 2015; Nagabandi, Kahn, Fearing, & Levine, 2018). However, most of the previous studies have evaluated their methods in either simulated environments (Pan & Theodorou, 2014; Gupta et al., 2020), real environments but without any external contacts (Deisenroth & Rasmussen, 2011; Gamboa Higuera, Meger, & Dudek, 2018; Lutter, Listmann, & Peters, 2019), or real environments with external contact but for tasks that do not require very precise movements (Deisenroth et al., 2015; Lenz et al., 2015). Conversely, dealing with external contacts and generating accurate motions are inevitable for real robot control tasks, such as precise manipulation. Particularly for a real-time, model-based approach, contact-rich tasks requiring precise movement pose a fundamentally difficult problem (Fazeli, Kolbert, Tedrake, & Rodriguez, 2017). Concretely, a model needs to accurately predict forthcoming contact events within a limited length of time rather than detect them afterward with sensors. Furthermore, we need to train such accurate models from a limited number of training samples because acquiring data from a physical system is costly and time-consuming. Therefore, in the study we present in this letter, we develop both a neural network structure for modeling contact-rich dynamics and a sample-efficient training procedure for the network.

First, we introduce a neural network architecture for modeling contact-rich dynamics. We designed the network architecture by tapping into knowledge on dynamics: the system acceleration is a product of an inverse inertia matrix and the total force, and the inertia matrix is necessarily symmetric. The network model needs to be trained with a limited amount of data yet provide accurate predictions of future states, including contact events within a limited amount of time. We evaluated the benefit of explicitly representing the inertia matrix in a network structure to model task-related dynamics (see Figure 1a) of a trackball-manipulation setup.

As a sample efficient training procedure, we develop a two-stage model learning method that utilizes human demonstration for contact-rich tasks. We train a contact-free dynamics model using random movement data in the first stage. In the second stage, we sample contact-rich data by making the robot track human-demonstrated contact-rich trajectories using the derived model-based controller from the first stage. We then used these


Physics-based network design
$\ddot{q} = M(q)^{-1}(-C(q,\dot{q}) + F(q,\dot{q},u))$
Robot part
Trackball part

(a) Structured neural network


Stage 1: contact-less robot dynamics
Stage 2: contact-rich robot + trackball dynamics

(b) Two-stage model learning

Figure 1: (a) Structured neural network architecture with explicit inertia matrix representation. Physics-based domain knowledge is introduced into the design of neural network architecture for modeling target dynamics. (b) Two-stage dynamics modeling method for contact-rich tasks. We investigated the control performance of real-time, model-based control methods on a trackball-manipulation task using a real 3-DoF finger robot.

movement data sampled through the trajectory tracking to train a model that implicitly includes contact dynamics (see Figure 1b).

We adopt model predictive control (MPC) as an online model-based control method to generate real-time robot movements using the neural network model. The models used in previous neural network–based MPC (NN-MPC) studies vary in both input domain and architecture; for example, they used camera images with convolutional neural networks (Lenz et al., 2015; Finn & Levine, 2017; Zhang et al., 2019) or joint angles and velocities with a fully connected network (Nagabandi et al., 2018). We propose to use the joint angles and velocities to model system dynamics and introduce physics-based structural constraints into the network architecture. Our structured NN-MPC (SNN-MPC) combines MPC with the structured neural network for dynamics modeling.

Introducing physical knowledge of rigid-body dynamics into neural network models has also been explored. Díaz Ledezma and Haddadin (2017) embedded a robotic system's kinematic structure into a neural network. They focused on learning an inverse dynamics model and did not apply their technique for model-based control. Lutter et al. (2019) proposed representing the Lagrangian mechanics as the neural network structure and applied the learned model to a real pendulum control. However, they did not consider contact forces from the environment. Also, since the Lagrangian formulation provides the inverse dynamics, the inertia matrix needs to be inverted to derive the forward model. Gupta, Menda, Manchester, and Kochenderfer (2020) additionally introduced a network to predict dissipative forces such as joint friction, while Lutter et al. (2019) needed to measure these forces, but evaluations were limited in simulation environments. Zhong, Dey, and Chakraborty (2021) proposed an approach to incorporate a contact model into energy-conserving neural networks that take advantage of physical knowledge of Lagrangian (Lutter et al., 2019) or Hamiltonian mechanics (Greydanus, Dzamba, & Yosinski, 2019; Finzi, Wang, & Wilson, 2020) with adopting an idea introduced in Agrawal et al. (2019). To activate the contact model, they needed to detect contact events. Moreover, in their evaluations, they focused on simulating target dynamics rather than controlling mechanical systems.

In our approach, we design a network to represent the forward dynamics to use with MPC. Inverse calculation for the inertia matrix is not necessary. Furthermore, we take into account the existence of the contact forces in our network structure. An interesting feature of our approach is that we do not need to explicitly detect the contact event to derive the contact forces. The proposed network rather predicts the contact forces from the current state and control variables. As a contact-rich task for experimental evaluation, we employed a trackball manipulation task using a real 3-DoF finger robot. Results show that the manipulation performance of the SNN-MPC, which has the constrained network models with explicit inertia-matrix representation, was better than that of a conventional fully connected network model. We also evaluated control performance with different control periods to clarify the importance of fast calculation of the network model, that is, the importance of a small control time step, for the precise manipulation task.

The contributions of our study are following:

1. We introduced physics-based structural constraints into the network architecture for the online and real-time MPC calculation. In our network model, we implicitly took into account the existence of the contact dynamics.
2. A two-stage network training procedure for our SNN-MPC was developed to learn a forward dynamics model for a contact-rich task efficiently.
3. We applied our proposed approach to the contact-rich trackball manipulation task using a real 3-Dof finger robot. Our SNN-MPC method outperformed MPC with a conventional fully connected network model.

The rest of this letter is organized as follows. Section 2 provides the background of the NN-MPC framework. Section 3 introduces the proposed SNN-MPC framework and its implementation. Section 4 explains the experimental system and procedure for evaluating the proposed methods. Section 5 demonstrates the performance of the network. Section 6 discusses the findings and summarizes the study.

2 Using Model Predictive Control with Neural Networks

This section summarizes the background of the neural-network-based model predictive control (NN-MPC) framework. Here we use MPC to derive the optimal control signal using a dynamics model at each time step.

Consider a state-space model with state variables $\boldsymbol{x}_t \in \mathbb{R}^{d_x}$ and control signals $\boldsymbol{u}_t \in \mathbb{R}^{d_u}$, where t denotes the time index, and d_x and d_u are the dimensions of state space and control space, respectively. In this study, we consider a discrete-time nonlinear dynamical system:

$$\boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t). \tag{2.1}$$

In the NN-MPC framework, an MPC approach uses a neural network model to represent a target dynamics $\boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t)$. Previous studies used fully connected networks (Nagabandi et al., 2018) or recurrent neural networks (Lenz et al., 2015).

Given a finite horizon T_{mpc}, a cost function $l(\boldsymbol{x}_t, \boldsymbol{u}_t)$, and a terminal cost $l^f(\boldsymbol{x}_{T_{\text{mpc}}})$, a control sequence $\boldsymbol{U}_0 \equiv \{\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{T_{\text{mpc}}-1}\}$ and an objective function $J(\boldsymbol{x}_0, \boldsymbol{U}_0)$ are defined as

$$J(\boldsymbol{x}_0, \boldsymbol{U}_0) \equiv \sum_{t=0}^{T_{\text{mpc}}-1} l(\boldsymbol{x}_t, \boldsymbol{u}_t) + l^f(\boldsymbol{x}_{T_{\text{mpc}}}). \tag{2.2}$$

The optimal control sequence $\boldsymbol{U}_0^*$ can be derived using this total cost function, which satisfies

$$\boldsymbol{U}_0^* \equiv \arg\min_{\boldsymbol{U}_0} J(\boldsymbol{x}_0, \boldsymbol{U}_0). \tag{2.3}$$

An MPC derives optimal control sequence under the system dynamics, equation 2.1, at each time step, with initial state $\boldsymbol{x}_0$ set to the currently observed state. Once the optimization is completed, we apply only $\boldsymbol{u}_0^*$, which is the first element of $\boldsymbol{U}_0^*$, to the system. When a new state is observed, the same procedure is repeated.

However, one cannot analytically derive the optimal control sequence, equation 2.3, under nonlinear system dynamics in general. Here we adopted an iterative linear quadratic regulator (iLQR; Li & Todorov, 2004) to approximately derive the optimal control sequence. In iLQR, the dynamics and the cost function are linearly and quadratically approximated. Since it efficiently computes the control sequence, iLQR is widely used in the MPC studies (Tassa, Erez, & Smart, 2008; Tassa, Erez, & Todorov, 2012).

To derive the linear system and the quadratic cost function for iLQR calculation, one has to compute the derivatives[1] of dynamics ($\boldsymbol{f_x}$, $\boldsymbol{f_u}$) and cost functions ($l_{\boldsymbol{x}}$, $l_{\boldsymbol{u}}$, $l_{\boldsymbol{xx}}$, $l_{\boldsymbol{xu}}$, and $l_{\boldsymbol{uu}}$). When a neural network is used as a dynamics model, the derivatives of dynamics can be computed on the network. The computation technique for those derivatives is provided in detail in section 3.3.

3 Proposed Method

This section introduces our structured NN-MPC (SNN-MPC) framework, which uses a neural network model with an explicit representation of the inertia matrix. We also explain how we train the neural network model in our SNN-MPC framework with the two-stage network training strategy.

3.1 Structured Neural Network for Modeling Robot Dynamics. Consider the multi-link rigid body dynamics in equation 2.1 for a robot with the state variable $\boldsymbol{x} \equiv \left[\boldsymbol{q}^\top, \dot{\boldsymbol{q}}^\top\right]^\top$, where $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ denote the generalized position and velocity, respectively. Given a time step Δt, the state for the next time index ($\boldsymbol{q}_{t+1}$ and $\dot{\boldsymbol{q}}_{t+1}$) is computed by numerically integrating the generalized acceleration:

$$\ddot{\boldsymbol{q}} = \boldsymbol{g}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) = \boldsymbol{M}(\boldsymbol{q})^{-1}\left(-\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{F}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u})\right), \tag{3.1}$$

where $\boldsymbol{M}$ is the inertia matrix, while $\boldsymbol{C}$ denotes gravity and the Coriolis forces and $\boldsymbol{F}$ contains the joint torque and external forces.

Previous studies exploited the flexible function approximation ability of neural networks in dynamics modeling for MPC (Lenz et al., 2015; Nagabandi et al., 2018). However, due to the large networks involved in those studies, their model was not suitable for real-time, high-frequency control. Moreover, large training data sets had to be prepared. These studies used unconstrained networks, such as the fully connected networks in Figure 2a, for modeling.

In this study, we designed structured neural networks to model the system dynamics $\boldsymbol{g}$ in equation 3.1, which explicitly represents the inverse inertia matrix $\boldsymbol{M}^{-1}$ and force-related terms $\boldsymbol{C}$ and $\boldsymbol{F}$ (see Figure 2b) that implicitly represents external forces from the environment. While the C-network and the F-network directly represent the generalized forces, the M-network leverages another feature of the inertia matrix: it is necessarily symmetric; thus, the inverse of the inertia matrix is also symmetric. Therefore, the M-network has to predict only the upper triangular elements $\boldsymbol{m} = [m_1 \cdots m_{k(k+1)/2}]^\top$ of the inverse inertia matrix,

[1]We use vector subscripts to denote partial derivatives: $\boldsymbol{f_x} \equiv \partial \boldsymbol{f}/\partial \boldsymbol{x}$ and $l_{\boldsymbol{xx}} \equiv \partial^2 l/\partial \boldsymbol{x}^2$.

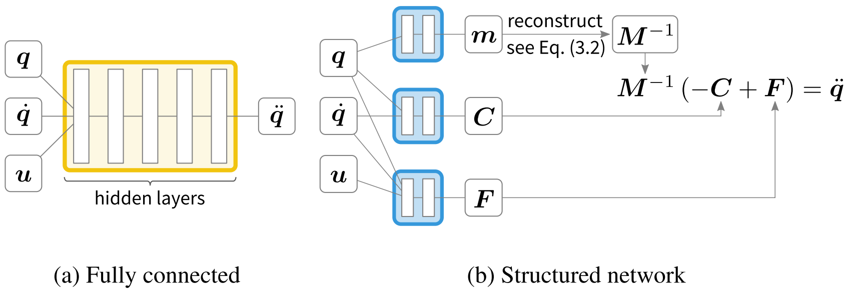


Figure 2: (a) Conventional fully connected network. (b) Proposed structured neural network with explicit inertia matrix representation. See equation 3.2 for inertia matrix reconstruction details. Note that all hidden layers in both fully connected and structured networks have ReLU activation.

$$
\boldsymbol{M}^{-1} = \begin{bmatrix} m_1 & \cdots & m_k \\ \vdots & \ddots & \vdots \\ m_k & \cdots & m_{k(k+1)/2} \end{bmatrix}, \tag{3.2}
$$

where k is the degree of freedom of the system. With those structural constraints, the neural network can learn a model of contact-rich robot dynamics with a limited number of hidden units. When there is additional knowledge available on the system, we can introduce them to the network structure as additional constraints. A practical example is shown in section 4.2.

3.2 Two-Stage Model Identification in Contact-Rich Environments. To train an accurate task-related dynamics model in a contact-rich environment, it is essential to have training data (i.e., robot motion data) recorded with appropriate contact force that militates to the system. However, acquiring such motion data with contact is often difficult on physical systems. Thus, we developed a two-stage model identification method in contact-rich environments (see Figure 1b):

1. In the first stage, we sampled the robot movement data by applying random torque command to the robot. Then we trained a *contact-free* model using these random movements of the robot system.
2. For the second stage, we first prepared a variety of *contact-rich* reference trajectories by kinesthetic teaching by a human operator. Then we controlled the robot to track the reference trajectories using SNN-MPC with the dynamics model acquired in the first stage. Finally, we trained a model that implicitly includes contact dynamics using movement data sampled through this tracking.

3.3 Deriving Controllers Using an Acquired Network Model. To derive an optimal control sequence $\boldsymbol{U}^*$ with iLQR, we have to compute the derivatives of system dynamics ($\boldsymbol{f}_x$ and $\boldsymbol{f}_u$) and the cost functions (l_x, l_{xx}, l_{xu}, and l_{uu}). The cost functions l are subject to our design; hence, we can make them analytically differentiable. Let us now consider the derivatives of the system dynamics. Although Carpentier and Mansard (2018) proposed a method to calculate analytical derivatives of a ridged-body dynamics model, this algorithm cannot be used with a contact model. If contact modeling is necessary and the number of DoF becomes larger, analytically differentiating a multiridged-body dynamics model (i.e., deriving $\boldsymbol{f}_x$ and $\boldsymbol{f}_u$) tends to be impractical. In such a case, using numerical differentiation is the standard approach. However, for a many-DoF system, the numerical differentiation process constitutes the most time-consuming part of the computation of optimal control signals (Erez et al., 2013). On the other hand, modeling the system dynamics using a neural network allows us to analytically compute the derivatives using backpropagation. Here, we assume that our neural network represents a function that has similar smoothness to the real dynamical system. With that assumption, by minimizing the error between the output of the network and that of the real system, we can expect that the error between the gradients of the network model and the real one will also be decreased. We empirically validated this assumption through evaluating the model-based control performances.

The discrete-time system dynamics $\boldsymbol{f}$ in equation 2.1 is computed by numerical integration of the generalized acceleration $\ddot{\boldsymbol{q}}$, and the derivatives of $\boldsymbol{f}$ are computed using the derivatives of $\ddot{\boldsymbol{q}} = \boldsymbol{g}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u})$ defined in equation 3.1. Here, $\boldsymbol{g}$ is approximated by a neural network; thus, we can analytically compute the derivatives of $\boldsymbol{g}$ (i.e., $\boldsymbol{g}_x$ and $\boldsymbol{g}_u$) via backpropagation on the neural network instead of time-consuming numerical differentiation. For brevity, we concatenate $\boldsymbol{g}_x$ and $\boldsymbol{g}_u$ into one matrix $\boldsymbol{g}_z$ by defining $\boldsymbol{z} \equiv \left[\boldsymbol{x}^\top, \boldsymbol{u}^\top\right]^\top$. Using backpropagation, we can compute the ith row of $\boldsymbol{g}_z$, which corresponds to the ith element of $\ddot{\boldsymbol{q}}$, at once. We first set the initial gradient vector $\boldsymbol{g}_{\ddot{q},i}$ on the output layer (i.e., $\ddot{\boldsymbol{q}}$) to satisfy $\boldsymbol{g}_{\ddot{q},i,j} = \delta_{i,j}$, where $\boldsymbol{g}_{\ddot{q},i,j}$ is the jth element of $\boldsymbol{g}_{\ddot{q},i}$ and $\delta_{i,j}$ is the Kronecker delta. We then compute the gradient vector $\partial\ddot{q}_i/\partial\boldsymbol{z}$ by backpropagating this initial gradient vector from the output layer to the input layer ($\boldsymbol{x}$ and $\boldsymbol{u}$), where $\ddot{q}_i$ denote the ith element of $\ddot{\boldsymbol{q}}$. By computing $\partial\ddot{q}_i/\partial\boldsymbol{z}$ for all i, we get $\boldsymbol{g}_z$, hence $\boldsymbol{g}_x$ and $\boldsymbol{g}_u$.

Here, there are a couple of caveats to implement a faster iLQR program with a neural network dynamics model:

1. Compute the derivatives of $\boldsymbol{g}$ in a batch of the whole nominal trajectory using matrix multiplication rather than computing them for each time step in a loop.
2. Because the computation of $\partial\ddot{q}_i/\partial\boldsymbol{z}$ is mutually independent with regard to i, we can parallelize them.

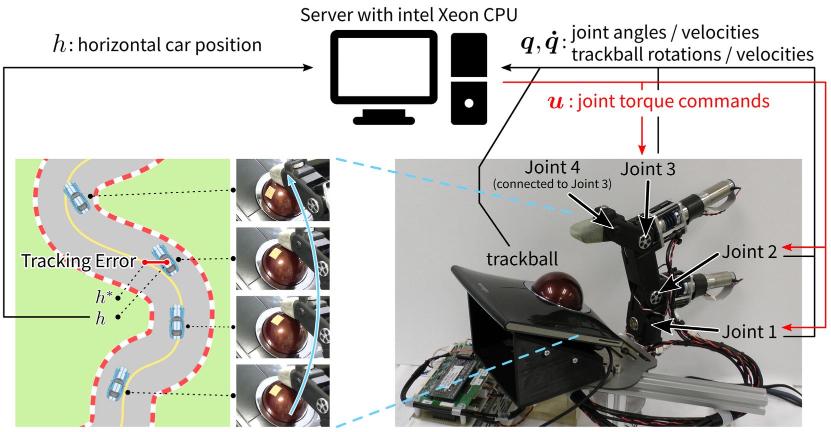

Server with intel Xeon CPU
h: horizontal car position
$\boldsymbol{q}, \dot{\boldsymbol{q}}$: joint angles / velocities trackball rotations / velocities
$\boldsymbol{u}$: joint torque commands
Joint 4 (connected to Joint 3)
Joint 3
trackball
Joint 2
Joint 1
Tracking Error
h^*
h

Figure 3: Experimental robot system composed of a 3-DoF finger robot and a 2-DoF trackball, which is used for controlling the car driving game. The trackball device was mounted in front of the robot. The horizontal position of the car was controlled by the lateral rotation of the trackball while the car was moved with a constant speed for the vertical direction. The combined state $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ of the finger robot and trackball state were sent to the computation server every 10 ms. The SNN-MPC implemented on the server derived an optimal control signal $\boldsymbol{u}$ for the next time step within 10 ms.

4 Experiments

To evaluate the proposed SNN-MPC approach, we compared the control performance of the SNN-MPC and an NN-MPC method with a fully connected network in a contact-rich driving game task using a 5-Dof system.

4.1 System Setup.

We developed a 5-DoF experimental system composed of a robot finger and a trackball (see Figure 3, right). The robot finger was composed of four links. Three joints were actuated by motors, while the third and fourth joints were connected by a mechanical link. The robot fingers were actuated by the electric motors, MAXON DCX19S KL 24V. The electric motor was equipped with the reduction gear MAXON GPX22 with a gear ratio of 326:1. To measure joint angle, we used the encoder MAXON ENX16 EASY (512 counts/rotation). A trackball device was mounted in front of the robot.

The system was controlled in hard real time. The state $\boldsymbol{x}_t$ composed of the robot joint angles, angular velocities, the trackball 2-D rotation, and rotational velocity was sampled every 10 ms. The SNN-MPC, implemented on the server equipped with two Intel Xeon E5-2690 v4 processors, derived an optimal control signal $\boldsymbol{u}$ for the next time step within 10 ms. This control signal represented the desired torque, and motor drivers converted to the desired current. Note that there were no contact sensors mounted on the system, but the F-network implicitly represents contact dynamics.

4.2 Training Dynamics Models. We implemented the structured neural network introduced in section 3.1 using the Chainer deep learning framework (Tokui, Oono, Hido, & Clayton, 2015). In this experiment, we applied additional constraints on the M-network by the nature of our robot-and-trackball system:

1. The robot and the trackball are physically unconnected; thus, the inertia matrix is separated into two submatrices, and the remaining elements are zero.
2. The trackball is a spherical rigid body and is geometrically symmetric; thus, the diagonal elements of the trackball submatrix take the same value.

Therefore, the relationship between the output of the M-network $\boldsymbol{m} = [m_1 \ldots m_8]^\top$ and the reconstructed full inverse inertia matrix M^{-1} is as follows:

$$\boldsymbol{M}^{-1} = \begin{bmatrix} m_1 & m_2 & m_3 & & \\ m_2 & m_4 & m_5 & \multicolumn{2}{c}{\mathbf{0}} \\ m_3 & m_5 & m_6 & & \\ & \mathbf{0} & & m_7 & m_8 \\ & & & m_8 & m_7 \end{bmatrix} \tag{4.1}$$

where the upper-left submatrix is for the robot and the lower-right submatrix is for the trackball, which shares the same values of the diagonal elements.

To train the neural network described above, we recorded the robot motion trajectories composed of tuples of $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\boldsymbol{u}$, and $\ddot{\boldsymbol{q}}$. We adopted our two-stage network training procedure introduced in section 3.2.

In the first stage, we sampled the movement data while applying random control sequences to the robot to acquire a contact-less model. We used the zero-order hold for 10 time steps. Concretely, we sampled the reference torque every 10 time steps (100 ms) from a normal distribution and fed the sampled signal to the robot for 10 time steps. We determined the variance of the normal distribution for generating random control sequences regarding the torque limits of the motors.

In the second stage, this contact-less model was used to control the robot around the contact surface. We handcrafted reference joint trajectories in which the robot and the trackball kept touch by manually moving the robot. We obtained a contact-rich trajectory data set by controlling the robot to follow these joint trajectories. By learning from this data set, the neural network ended up representing the contact-rich model.

We collected the training data sets only for 10 minutes (60000 data points) in both training stages. Eighty percent of these data sets were used for training, while the remaining 20% were used for validation. Note that we collected those movement data sets without considering the car driving task (explained in the subsequent section). This means the parameters of the network model were not specially tuned for the given task.

To compare the task performances between the proposed structured network and the conventional fully connected network, we trained various-sized networks with 1, 2, 3, 5, and 8 hidden layers, each of which had 20 to 500 hidden units. Note that some of these, such as the 8-layer 500-unit network, were omitted because the iLQR optimization on such large networks violated the 10 ms limitation of the MPC time step. For the car driving task, the best performance was achieved by a structured neural network model that had one layer with 20 units in each *M*-, *C*-, and *F*-network. The fully connected network that achieved the best performance among them had one layer and 75 units. Note that the fully connected networks with two hidden layers or more could not learn the dynamics model from the limited training data set. We used the above network models for the comparisons in the result section.

4.3 The Car Driving Task. We evaluated our real-time, model-based control method with the neural network models on the car driving game task (see Figure 3). The horizontal position of the car in the game was controlled by the lateral rotation of the trackball, while the car was moved with constant speed for the vertical direction. The task duration was 24 s. The yellow line at the center of the course in the figure is the desired trajectory for the car.

The objective function (see equation 2.2) for MPC in the car driving task was given as

$$J = \sum_{t=0}^{T_{\mathrm{mpc}}-1} l(\boldsymbol{x}_t, \boldsymbol{u}_t) + l^f(\boldsymbol{x}_{T_{\mathrm{mpc}}}). \tag{4.2}$$

The immediate and terminal costs were defined as

$$l(\boldsymbol{x}_t, \boldsymbol{u}_t) = w(h_t - h_t^*)^2 + \dot{\boldsymbol{q}}_t^\top \boldsymbol{Q} \dot{\boldsymbol{q}}_t + \boldsymbol{u}_t^\top \boldsymbol{R} \boldsymbol{u}_t, \tag{4.3a}$$

$$l^f(\boldsymbol{x}_t) = w(h_t - h_t^*)^2 + \dot{\boldsymbol{q}}_t^\top \boldsymbol{Q} \dot{\boldsymbol{q}}_t, \tag{4.3b}$$

where h denotes the horizontal car position, h^* denotes the desired course, and w, $\boldsymbol{Q}$, and $\boldsymbol{R}$ are weight parameters. T_{mpc} denotes the MPC horizon. Note that only the desired trajectory for the car position was given, but not for the robot finger's joint angles themselves. Therefore, the controller had to derive joint torque commands for the robot to manipulate the trackball through the contact force between the robot and the trackball.

To evaluate how the control frequency affects the performance of MPC, we conducted the experiments with three different control time steps: $\Delta t = 10, 20$, and 30 ms. In this task, we set the time horizon for MPC as $T_{\text{mpc}} \times \Delta t = 500$ ms. Note that the MPC horizon T_{mpc} was adjusted according to the step size Δt. Since the task duration was 24 s (= 24,000 ms), the total number of steps for one trial was $T_{\text{trial}} = 24{,}000/\Delta t$, that is, $T_{\text{trial}} = 2400, 1200$, and 800 for the trials with the time steps $\Delta t = 10, 20$, and 30 ms, respectively.

4.4 Experimental Evaluation. We evaluated the control performance of each model by the trajectory tracking error defined as

$$E = \sum_{t=0}^{T_{\text{trial}}-1} (h_t - h_t^*)^2 \Delta t. \tag{4.4}$$

We considered a driving trial as successful if the accumulated error defined above did not exceed a certain threshold ($E < 0.025$). This threshold was manually determined to detect obvious failure trails.

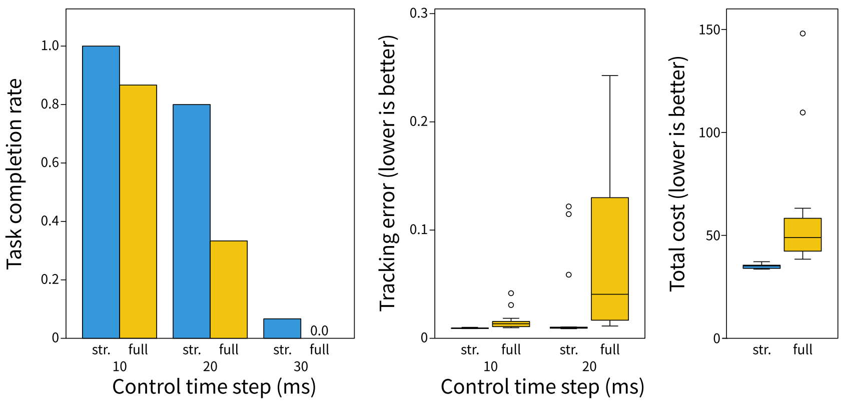
We also evaluated the accumulated cost score of the objective function,

$$J_{\text{total}} = \left\{ \sum_{t=0}^{T_{\text{trial}}-1} l(\boldsymbol{x}_t, \boldsymbol{u}_t) + l^f(\boldsymbol{x}_{T_{\text{trial}}}) \right\} \Delta t. \tag{4.5}$$

Finally, we evaluated the accuracy of the learned dynamics models with the time step $\Delta t = 10$ ms. We first prepared the driving-game experiment data with the moving time window of 500 ms. This window size was equivalent to the time horizon for MPC. For each dataset in a time window, we took the state at the first time step $\boldsymbol{x}_0$ and control signals for all time steps in the segment $\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{T_{\text{pred}}-1}$. Then the neural network dynamics models predicted the trajectory $\hat{\boldsymbol{x}}_0, \ldots, \hat{\boldsymbol{x}}_{T_{\text{pred}}}$, where $\hat{\boldsymbol{x}}_0 = \boldsymbol{x}_0$, by iteratively input the previously predicted state and the control signal for the corresponding time step. We repeated this computation for all movement data sets. We evaluated the prediction performance by the mean squared error of the predicted trajectory from the actual trajectory, computed as

$$E_{\text{pred}} = \frac{1}{T_{\text{pred}}} \sum_{t=1}^{T_{\text{pred}}} ||\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t||^2 \Delta t, \tag{4.6}$$

where $T_{\text{pred}} = 500/\Delta t = 50$ was the number of time steps to predict the system state.



(a) Task completion rate (b) Tracking error (c) Total cost

Figure 4: Performance comparison among models and time steps. (a) Task completion rate for each setting. (b) Overall trajectory tracking error. We compared only with 10 ms and 20 ms time steps since completion rates were very low for 30 ms. (c) Total cost for 10 ms trials.

5 Results

We evaluated both the proposed SNN-MPC and the standard NN-MPC techniques with a fully connected network on the car driving game. We conducted the driving game using MPCs with each network model for 15 trials.

5.1 Task Performance Evaluation. Figure 4a shows the task completion rate. As we explained in the previous section, we defined trials with an accumulated tracking error less than 0.025 as successful. While SNN-MPC completed all the trials, a fully connected network failed a couple of trials even with the shortest time step of 10 ms. The completion rates for both methods dropped as the time step increased from 10 ms to 30 ms, and the fully connected one was affected more severely.

Figure 4b shows the trajectory tracking error for each trial. Here, we omitted the result with the 30 ms time step because the task completion rates were very low. We observe that our SNN-MPC outperformed the fully connected network. With a 20 ms time step, the SNN-MPC failed in three trials, showing three outliers in the box plot. The error is much larger for the fully connected network.

We performed a two-way ANOVA on these results. We observed significant effects on both network type ($p < 10^{-3}$) and time step ($p < 10^{-3}$). We also noticed a relationship between the network type and the time step

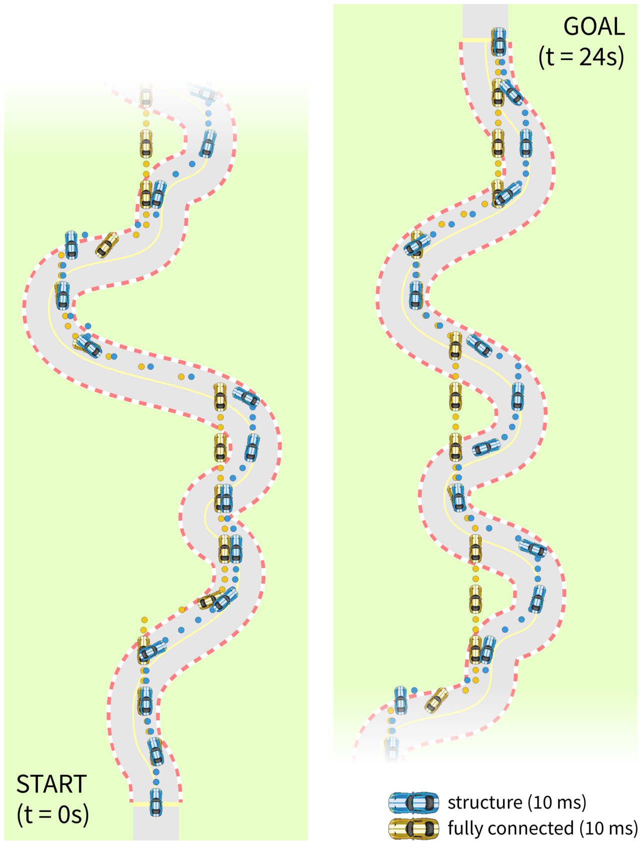
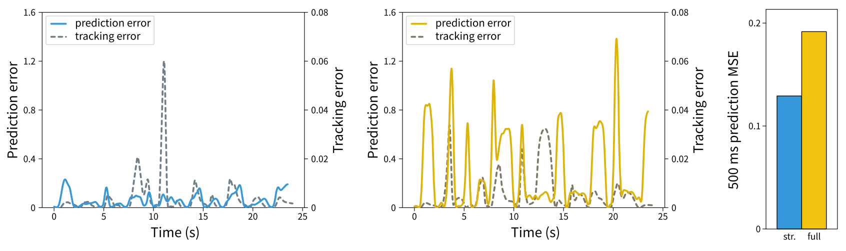


Figure 5: A full-length trajectory of the car for each network architecture in the driving game experiments. Blue car: structured network, Yellow car: fully connected network.

($p = 0.005$). This indicates that the performance of the fully connected network deteriorated more rapidly than the structured neural network as the time step increased.

Figure 4c shows the total cost score in equation 4.5 for both networks tested with 10 ms time steps because, from Figure 4b, they showed reasonable control performances only with the shortest time step of 10 ms. The SNN-MPC method with a 10 ms control time step achieved the best median performance score (35.2). It outperformed the NN-MPC method with a fully connected network, which failed to follow the course in two trials out of 15 and whose median score was 48.9.

Figure 5 shows the comparison between the typical example trajectories generated by the proposed and the standard MPCs. Our SNN-MPC (blue



(a) Prediction error (str.) (b) Prediction error (full) (c) MSE

Figure 6: (a, b) 500 ms trajectory prediction accuracy along the trajectory shown in Figure 5. (a) Structured network. (b) Fully connected network. (For visualization, the time series are smoothed using a Butterworth filter with a cutoff frequency of 2.5 Hz.) (c) Mean 500 ms trajectory prediction accuracy for 14 trajectories.

car) demonstrated better course tracking performance than the NN-MPC with a fully connected network (yellow car).

5.2 Prediction Performances of Learned Network Models. Finally, we evaluated the accuracy of the neural network dynamics models. Figure 6a (structured) and Figure 6b (fully connected) show the mean squared error of the prediction for the 500 ms time window (blue and yellow lines) at each time step and the absolute trajectory tracking error (dashed black lines), computed along the trajectory shown in Figure 5. As depicted in the both figures, the trajectory tracking errors coincided with the prediction errors. This result further indicates that the prediction performance of our structured network contributed to the control performance of the robot system.

Figure 6c shows the average 500 ms trajectory prediction error for 14 different trajectories. The prediction error of the structured neural network is smaller than that of the fully connected network. These results support that the performance of the SNN-MPC method was better than that of the fully connected one because it modeled task dynamics more accurately.

6 Conclusion

In this letter, we proposed the structured neural network model predictive control (SNN-MPC) technique and the two-stage training procedure to model contact-rich task-related dynamics. The structured network model implicitly predicted the contact dynamics while explicitly represented the inertial matrix.

In the driving game experiments, the SNN-MPC outperformed the standard NN-MPC method with a fully connected network. This result is possibly attributed to the fact that unconstrained, fully connected networks tend to have redundant representations with deeper network structures and are susceptible to overfitting when trained with limited data sets. Furthermore, shorter control time steps resulted in performances superior to those with the lower-frequency ones. We showed that the structured neural network achieved a smaller trajectory prediction error than the fully connected network, which reflects the accuracy of the modeled dynamics. This result also validated the effectiveness of our approach to introduce the structural constraint into the network architecture.

If a robot system is equipped with reliable contact sensors, we can possibly combine our network model with the contact model proposed by Zhong et al. (2021) to further improve the prediction performance. We empirically showed that such physics-based structural constraints for neural networks could improve robot control. Further research on the theoretical validity of such constraints is necessary.

Conflict of Interest

The authors declare no competing financial interests.

Acknowledgments

This research was supported by JSPS KAKENHI grants JP16H06565 and JP19J20669 and also partially supported by the New Energy and Industrial Technology Development Organization (NEDO) JPNP20006, JST-Mirai Program Grant JPMJMI18B8 and JPMJI21B1, MIC-SCOPE (182107105), Tateishi Science and Technology Foundation, and ImPACT Program of Council for Science, Technology and Innovation (Cabinet Office, Government of Japan).

References

Agrawal, A., Amos, B., Barratt, S. T., Boyd, S. P., Diamond, S., & Kolter, J. Z. (2019). Differentiable convex optimization layers. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems, 32* (pp. 9558–9570). Red Hook, NY: Curran.

Carpentier, J., & Mansard, N. (2018). Analytical derivatives of rigid body dynamics algorithms. In *Proceedings of the Robotics: Science and Systems XIV.* https://rislab.github.io/rss2018website/

Deisenroth, M. P., Fox, D., & Rasmussen, C. E. (2015). Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *37*(2), 408–423. 10.1109/TPAMI.2013.218, PubMed: 26353251

Deisenroth, M., & Rasmussen, C. (2011). PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning* (pp. 465–472). Madison, WI: Omnipress.

Díaz Ledezma, F., & Haddadin, S. (2017). First-order-principles-based constructive network topologies: An application to robot inverse dynamics. In *Proceedings of the IEEE-RAS 17th International Conference on Humanoid Robotics* (pp. 438–445). Piscataway, NJ: IEEE.

Erez, T., Lowrey, K., Tassa, Y., Kumar, V., Kolev, S., & Todorov, E. (2013). An integrated system for real-time model predictive control of humanoid robots. In *Proceedings of the 13th IEEE-RAS International Conference on Humanoid Robots* (pp. 292–299). Piscataway, NJ: IEEE.

Fazeli, N., Kolbert, R., Tedrake, R., & Rodriguez, A. (2017). Parameter and contact force estimation of planar rigid-bodies undergoing frictional contact. *International Journal of Robotics Research*, *36*(13–14), 1437–1454. 10.1177/0278364917698749

Finn, C., & Levine, S. (2017). Deep visual foresight for planning robot motion. In *Proceedings of the IEEE International Conference on Robotics and Automation* (pp. 2786–2793). Piscataway, NJ: IEEE.

Finzi, M., Wang, K. A., & Wilson, A. G. (2020). Simplifying Hamiltonian and Lagrangian neural networks via explicit constraints. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems*, *33* (pp. 13880–13889). Red Hook, NY: Curran.

Gamboa Higuera, J. C., Meger, D., & Dudek, G. (2018). Synthesizing neural network controllers with probabilistic model-based reinforcement learning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 2538–2544). Piscataway, NJ: IEEE.

Greydanus, S., Dzamba, M., & Yosinski, J. (2019). Hamiltonian neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems*, *32* (pp. 15353–15363). Red Hook, NY: Curran.

Gu, S., Holly, E., Lillicrap, T., & Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Proceedings of the IEEE International Conference on Robotics and Automation* (pp. 3389–3396). Piscataway, NJ: IEEE.

Gupta, J. K., Menda, K., Manchester, Z., & Kochenderfer, M. J. (2020). Structured mechanical models for robot learning and control. In *Proceedings of the Second Annual Conference on Learning for Dynamics and Control* (pp. 328–337).

Lenz, I., Knepper, R., & Saxena, A. (2015). DeepMPC: Learning deep latent features for model predictive control. In *Proceedings of the Robotics: Science and Systems XI.*

Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., & Quillen, D. (2018). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *International Journal of Robotics Research*, *37*(4–5), 421–436. 10.1177/0278364917710318

Li, W., & Todorov, E. (2004). Iterative linear quadratic regulator design for nonlinear biological movement systems. In *Proceedings of the First International Conference on Informatics in Control, Automation and Robotics* (pp. 222–229). Setúbal, Portugal: SciTePress.

Lutter, M., Listmann, K., & Peters, J. (2019). Deep Lagrangian networks for end-to-end learning of energy-based control for under-actuated systems. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 7718–7725). Piscataway, NJ: IEEE.

Nagabandi, A., Kahn, G., Fearing, R. S., & Levine, S. (2018). Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *Proceedings of the IEEE International Conference on Robotics and Automation.* Piscataway, NJ: IEEE.

Pan, Y., & Theodorou, E. (2014). Probabilistic differential dynamic programming. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems*, *27* (pp. 1907–1915). Red Hook, NY: Curran.

Tassa, Y., Erez, T., & Smart, W. D. (2008). Receding horizon differential dynamic programming. In J. Platt, D. Koller, Y. Singer, & S. Roweis (Eds.), *Advances in neural information processing systems*, *20* (pp. 1465–1472). Red Hook, NY: Curran.

Tassa, Y., Erez, T., & Todorov, E. (2012). Synthesis and stabilization of complex behaviors through online trajectory optimization. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 4906–4913). Piscataway, NJ: IEEE.

Tokui, S., Oono, K., Hido, S., & Clayton, J. (2015). Chainer: A next-generation open source framework for deep learning. In *Proceedings of the Workshop on Machine Learning Systems in the 29th Annual Conference on Neural Information Processing Systems*.

Zhang, M., Vikram, S., Smith, L., Abbeel, P., Johnson, M., & Levine, S. (2019). SOLAR: Deep structured representations for model-based reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning* (pp. 7444–7453).

Zhong, Y. D., Dey, B., & Chakraborty, A. (2021). *A differentiable contact model to extend Lagrangian and Hamiltonian neural networks for modeling hybrid dynamics*. arXiv:2102.06794.

Received April 20, 2021; accepted September 3, 2021.