ARTICLE OPEN

Exploration-based model learning with self-attention for risk-sensitive robot control

DongWook Kim[1,2,3], Sudong Lee[4], Tae Hwa Hong[1,2,3] and Yong-Lae Park[1,2,3]✉

Model-based reinforcement learning for robot control offers the advantages of overcoming concerns on data collection and iterative processes for policy improvement in model-free methods. However, both methods use exploration strategy relying on heuristics that involve inherent randomness, which may cause instability or malfunction of the target system and render the system susceptible to external perturbations. In this paper, we propose an online model update algorithm that can be directly operated in real-world robot systems. The algorithm leverages a self-attention mechanism embedded in neural networks for the kinematics and the dynamics models of the target system. The approximated model involves redundant self-attention paths to the time-independent kinematics and dynamics models, allowing us to detect abnormalities by calculating the trace values of the self-attention matrices. This approach reduces the randomness during the exploration process and enables the detection and rejection of detected perturbations while updating the model. We validate the proposed method in simulation and with real-world robot systems in three application scenarios: path tracking of a soft robotic manipulator, kinesthetic teaching and behavior cloning of an industrial robotic arm, and gait generation of a legged robot. All of these demonstrations are achieved without the aid of simulation or prior knowledge of the models, which supports the proposed method's universality for various robotics applications.

npj Robotics (2023)1:7 ; https://doi.org/10.1038/s44182-023-00006-5

INTRODUCTION

The kinematics and the dynamics of robots are essential elements for precision control that guarantee robust and stable task completion[1–3]. Most robot control schemes rely on models for different tasks, including motion optimization, task scheduling, prioritized control, and the construction of accurate models has thus been one of the top priorities in conventional control. The models for rigid-bodied robots typically rely on the geometries and the energy method, which estimate their shapes and positions[1,4–7]. However, calculating the dynamic features of every component is often tedious and always involves errors in the model[8–15]. Alternatively, the techniques of black-box model identification using machine intelligence have been proposed for obtaining models without considering mechanical characteristics. Moreover, its control strategy can also be trained by employing reinforcement learning (RL). Meanwhile, model-based RL methods are frequently studied in robotics applications due to the efficiency of data usage and avoiding computational burdens induced by iterative policy evaluation and improvement[16–24].

Despite the advantages, the control policy generated using model-based methods still gives rise to unexpected motions or undesired output behaviors during policy learning. Thus, it is risky to apply RL directly to real-world robots without preprocessing in a simulated environment[15,25–29]. This has led to the enhancement of RL methods that can be implemented in real-world applications, including simulation-to-real (sim2real) methods[13,30], uncertainty-based modeling[31], exploration techniques[32–34], and robust optimal control[35,36]. For instance, adding Kullback–Leibler divergence to the optimization objective limits sudden changes in the desired trajectory[18]. Employing a Gaussian process-based model provides the degrees of uncertainties during model learning for determining the exploration level or the robustness of the controller[37,38]. Domain randomization[39,40] or adversarial training[41] also provides robustness by learning from various situations or incorporating worst-case scenarios[40–42].

In this paper, we demonstrate how the self-attention mechanism can be employed in model learning for real-world robotics applications without simulation. Two types of models, namely the kinematics and the dynamics models, depict the behavior of robot motion. Our method utilizes a self-attention layer in both models to overcome the issues that can arise from the direct application of RL methods in a real-world environment. Specifically, the self-attention layer in the kinematics model determines the exploration region by adjusting the cost function for the robot's movement range. On the other hand, the self-attention in the dynamics model detects possible perturbations during the learning process and manages the quality of the dataset.

Assuming trajectory tracking control using model-based RL, the cost function can be defined as the sum of errors between the reference and the executed trajectory. During the initial optimization phase, the resulting control policy could be inaccurate due to a poorly learned model. Using the control policy directly on the robot may be inadequate since the output behavior of the control is uncertain. To address this limitation, we propose the incorporation of a self-attention layer into the kinematics model, aiming to enhance the control performance and provide more predictable behaviors[43,44]. Although the kinematics model is not explicitly dependent on time, constructing a self-attention chain based on time allows us to look closer on the data with low self-referential rates, which are considered as abnormal data or regions that are insufficiently learned. The scaling of the desired trajectory can be determined from the self-attention matrix of the kinematics model.

Additionally, by constructing a self-attention chain for dynamics, we can detect perturbations in the robot caused by

[1]Department of Mechanical Engineering, Seoul National University, Seoul 08826, Korea. [2]Institute of Advanced Machines and Design (IAMD), Seoul National University, Seoul 08826, Korea. [3]Institute of Engineering Research, Seoul National University, Seoul 08826, Korea. [4]CREATE Lab, Institute of Mechanical Engineering, Swiss Federal Institute of Technology in Lausanne (EPFL), 1015 Lausanne, Switzerland. ✉email: ylpark@snu.ac.kr

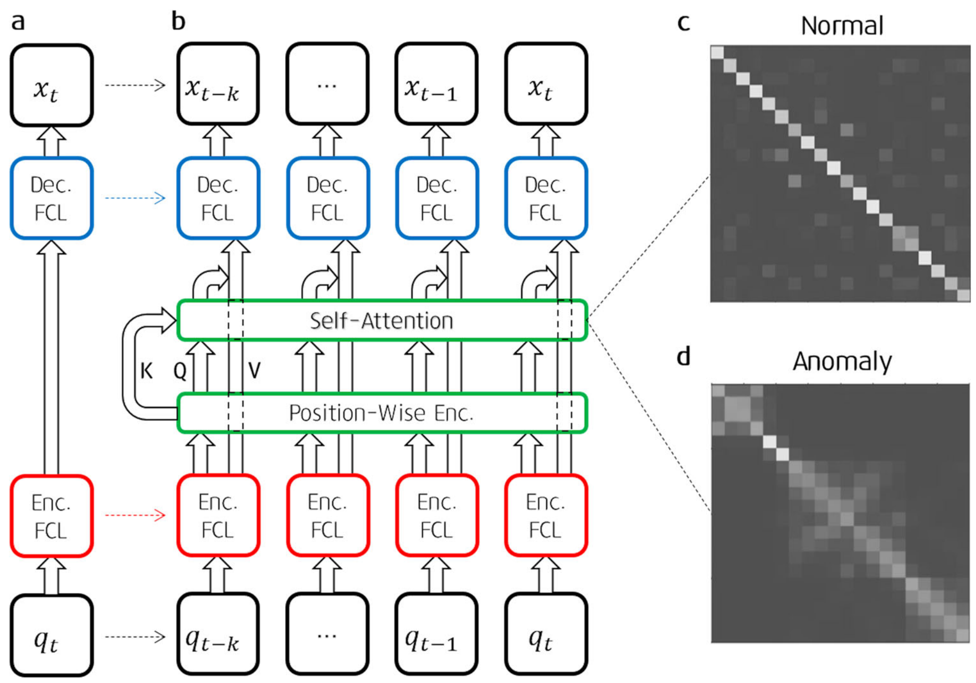

Fig. 1 Overall scheme of the kinematics network and its self-attention model for adjusting the exploration region. a The kinematics model consists of a fully connected layer with a radial basis activation function. **b** When we expand this model over the past *k* timesteps, we can correlate their encoded features using a self-attention mechanism. **c** If the model is normally working, the self-attention matrix will exhibit a large value in the diagonal position. The brighter color indicates the larger value. **d** If abnormal data are detected, the self-attention matrix will show a blurred image.

unintended external forces. The dynamics model learned in our proposed method considers the relationship between the control inputs and the robot's configuration states while excluding the effects of external forces[45–47]. Consequently, the addition of external forces leads to a decrease in self-referential rates within the self-attention matrix of the dynamics model.

To implement the proposed self-attention mechanism, the time series of the input is connected to the encoder network, which is connected to the decoder in series, using feedforward neural networks (FNNs). The self-attention layer exists between the encoder and the decoder networks, as shown in Fig. 1b. If a single FNN connecting the input and the output at a single timestep (Fig. 1a) is cloned and arranged in parallel over time, and the self-attention layer is connected between the outputs of the networks (Fig. 1b), then the self-attention matrix will eventually be a diagonal matrix when the model is fully trained (Fig. 1c). In other words, each input component primarily refers to itself. Now, imagine the procedure mentioned earlier, i.e., the first-time exploration and confronting perturbations. The corresponding input-output pair will never be shown to the network models. In this case, referring to itself will not be beneficial since the abnormal input data are not sufficiently learned to represent the hidden features. Instead, the self-attention mechanism will refer to the other inputs to extract meaningful features; in other words, a macroscopic change will occur in the self-attention matrix (Fig. 1d). Thus, by analyzing the trace values of the self-attention matrix for the kinematics and dynamics models, we can adjust the exploration task space or detect disturbances. That is, a sudden change in the kinematics attention matrix results in freezing the searching space and commanding the robot to move in its proximity. On the other hand, a change in the dynamics attention matrix prevents the current dataset from being learned, since it may have been contaminated by disturbances.

We evaluated the performance for each network structure and learning method through an ablation study in a robot simulation environment. In addition, we applied the proposed algorithm to actual robot systems, showing that they were able to learn the local models in real time and complete target tasks. More specifically, our algorithm was used to teach an industrial robot arm to play the piano, control the motion of a soft robotic manipulator and generate a waking gait of a quadrupedal robot. Each robot was able to perform its desired task within 5 min in real time without any assistance from the simulated environment or prior knowledge of the robot. The results indicate that the proposed algorithm can be used universally in various robotic applications.

RESULTS

Algorithm overview

The overall architecture of the robust online model identification algorithm consists of various types of neural networks and their derivative operations with detection modules. Figure 2 schematically summarizes the flow of the algorithm, indicating that it consists of a clockwise outer loop and multiple branch lines.

When dealing with the limited knowledge of a robot's kinematics and dynamics, the algorithm aims to effectively control the robot to achieve a predefined task. However, there are three essential pieces of information that must be accessible either through sensors or computations. First, information regarding the energy used for robot actuation, such as the magnitude of the applied force, the input current to the motor, or the input pressure to the pneumatic or hydraulic system should be available. Second, the robot's joint state resulting from actuation, commonly referred to as the configuration-space state, needs to be measured using encoders or similar sensors. Lastly, the task-space state should correspond directly to the ongoing task being performed. The goal of the algorithm is to ensure that the robot's task state is in line with the desired trajectory, using these three types of information, typically obtained through path planning. Throughout the rest of this paper, we will specifically refer to these three components as the control input, the configuration state, and the task state, respectively.

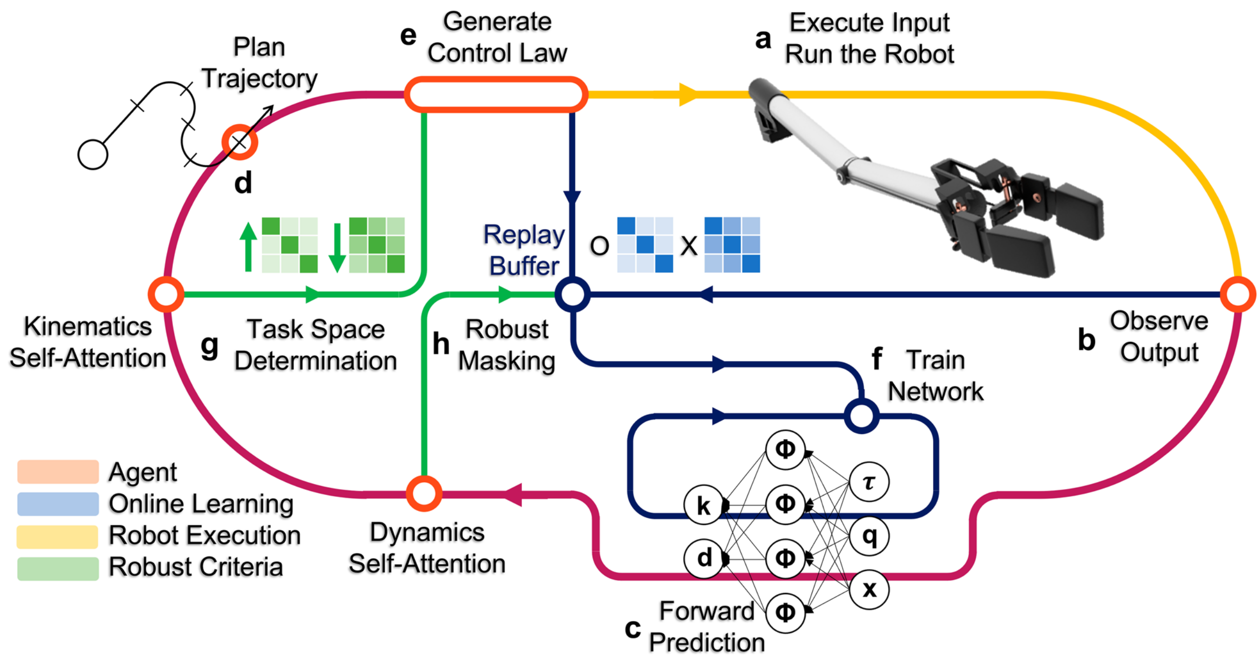

Fig. 2 The overall schematic of the algorithm that consists of the main loop for execution of the robot (yellow line) and the calculation through the agent (red line). The kinematics and the dynamics network are trained using a radial basis neural network sampled from the replay buffer (blue line). The self-attention network for both models is utilized for the robust masking and constraining of the task space. **a** Control input is applied to the real-world robot. **b** Input, state, and output are collected and then sent to the replay buffer. **c** Forward model prediction is done for calculating self-attention matrix. **d** Next desired task-space trajectory is given. **e** Control law is generated, reflecting the updated model from **f**. From self-attention matrices, **g** task-space determination and **h** rejection of perturbed dataset are possible.

The aforementioned three types of information are used as variables of the kinematics and the dynamics models. Specifically, the kinematics encompasses the correlation between the configuration state and the task state, whereas the dynamics model captures the interplay between the control input and the configuration state. Both of these models adopt a neural network architecture, as shown in Fig. 1a. At each timestep, new input and state information is acquired, which is then stored in a replay buffer to further train the neural network approximations of the kinematics and the dynamics models.

Detailed computation process of outer loop of Fig. 2 is explained from now. First, the robot receives control inputs and executes them for a single timestep (Fig. 2a). Then, the agent observes the robot's states, namely the configuration and task states (Fig. 2b). Using the observed states and given inputs, the kinematics and dynamics equation components (matrices J, M, C, and g) are computed (Fig. 2c). The agent then prepares the next target task state that was predefined from path planning (Fig. 2d). Finally, the control inputs for the next timestep are determined from the model-based feedback control law, reflecting the desired task state, the kinematics model and the dynamics model (Fig. 2e). The next target task states should be prepared in advance from planning algorithms, such as tree-based searching or demonstration by kinesthetic teaching[48,49].

When constructing a control law in the outer loop (Fig. 2e), we employ the approximated kinematics and dynamics models. A two-step control process is used; first, from the desired task state $x_{t+1,des}$, a desired configuration state $q_{t+1,des}$ is derived from the kinematics model, and then the actual control input $\tau_{t+1,des}$ is obtained from the dynamics model. However, the approximated model remains uncertain due to insufficient learning, particularly due to their black-box nature. This may yield peculiar output values in regions that are insufficiently trained, or with the existence of perturbation.

Two branch lines indicating internal operations, colored green and blue, is the realization of incorporating safe-aware control design. The blue line updates the kinematics and dynamics models. Input–output tuples for training are sampled from the replay buffer to avoid temporal correlations and for regularization (Fig. 2f). The two branch lines are shown in green, considering both the kinematics and the dynamics self-attention models, as shown in Fig. 1b. The self-attention matrix of the kinematics model constrains the task space by adjusting the damping factor of the kinematic Jacobian matrix (Fig. 2g). This process changes the possible range of $q_{t,des}$ even if the range of desired $x_{t,des}$ is intact. On the other hand, the self-attention matrix of the dynamics model detects any external perturbations and passes the result to the blue line, determining whether the current input-output tuple is valid or not (Fig. 2h). If an anomaly is detected, then the current input-output tuple is rejected and not included in the replay buffer.

In summary, a total of four neural networks are used in this algorithm. There is one network for modeling the kinematics and another for modeling the dynamics. Additionally, there are self-attention networks of the kinematics and the dynamics that have been expanded to be used for determination of the task space determination and robust masking, respectively. In the following sections, we will provide a detailed explanation of these networks.

Network structures and objectives

Kinematics model. The kinematics model in the algorithm (Fig. 1) were fitted with a function approximator based an FNN. To explain the model, we note that three measurable variables are used: the input $\tau \in \mathbb{R}^m$, the configuration state $q \in \mathbb{R}^m$, and the task state $x \in \mathbb{R}^m$.

The rigid-body kinematics make a connection between the configuration state and the task state. Their forward kinematics can be expressed as $x = f(q)$, where $f : \mathbb{R}^m \to \mathbb{R}^n$ maps from the configuration state to the task state. Its differential form in continuous-time domain is generally given as

$$\dot{x} = J(q)\dot{q}, \tag{1}$$

where an overdot means the derivative with respect to time, and $J \in \mathbb{R}^{n\times m}$ is the kinematic Jacobian matrix. We use a discretized time-domain kinematics as

$$\frac{x_{t+1}-x_t}{T} = J(q_t)\frac{q_{t+1}-q_t}{T}, \tag{2}$$

where $T \in \mathbb{R}^+$ is a sampling interval for discretization that does not raise aliasing. Rearranging the discretized form in terms of x_{t+1} yields

$$x_{t+1} = [f(q_t) - J(q_t)q_t] + J(q_t)q_{t+1} =: J_0(q_t) + J(q_t)q_{t+1} \tag{3}$$

where we replace first terms as J_0 for the final result.

Results of the discretized form of Kinematics is approximated by employing a radial basis neural network (RBNN) that contains a single hidden layer with a radial-basis activation function in its structure. The proposed structure can be interpreted as a concatenation of a local linear model. We analytically define the hidden features as

$$\Phi(q_t) = \begin{bmatrix} \Phi_1(q_t) \\ \Phi_2(q_t) \\ \dots \\ \Phi_N(q_t) \end{bmatrix} \in \mathbb{R}^N, \tag{4}$$
$$\Phi_i(q_t) = \exp(-(q_t - c_i)^\top S^{-1}(q_t - c_i)) \in \mathbb{R}, i = 1, 2, \dots, N,$$

where $S \in \mathbb{R}^{m\times m}$ is the positive-definite covariance matrix, $c_i \in \mathbb{R}^m$ is the center of basis, and a total of N features are given so that $\Phi \in \mathbb{R}^N$ where its components are given with a subscript i. The center c_i is a trainable variable which is first initialized by sampling from the uniform distribution, where the covariance matrix is defined as a constant matrix $S = 2I$ (see Supplementary Document 1, Appendix A for analysis). The output layer of the RBNN is the weighted sum of the hidden features. Now, the Jacobian is expressed as

$$J_{\Phi_J}(q_t) = \begin{bmatrix} \phi_{J,1}\Phi(q_t) & \phi_{J,2}\Phi(q_t) & \dots & \phi_{J,m}\Phi(q_t) \end{bmatrix},$$
$$J_{0\Phi_J}(q_t) = \phi_{J,0}\Phi(q_t), \tag{5}$$

where $\phi_{J,k} \in \mathbb{R}^{n\times N}$ for $k = \{k \in \mathbb{Z} | 0 \le k \le m\}$ is trainable weights for the functions J and J_0. The usage of the radial basis activation function also prevents unexpected peaks during the initial phase of online learning. The in-depth analysis will be discussed in a later section. We use the subscript expression J_{Φ_J} and $J_{0\Phi_J}$ so the Jacobian model is approximated using the trainable weights.

A replay buffer is used to store the data τ, q, x and train the model for every timestep. Using the stored variables q_t and q_{t+1} and the kinematics model Eq. (3), prediction $\hat{x}_{t+1}$ can be derived. The objective for kinematics model is a mean-squared-error (MSE) between $\hat{x}_{t+1}$ and x_{t+1}, as

$$L_{kin} = ||\hat{x}_{t+1} - x_{t+1}||^2 = ||J_{0\Phi_J}(q_t) + J_{\Phi_J}(q_t)(q_{t+1} - q_t) - x_{t+1}||^2. \tag{6}$$

Dynamics model. Similar to the kinematics model, dynamics model shows the evolution of the configuration state from the input. The vector equation of dynamics model is given as

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau, \tag{7}$$

where an overdot means the derivative with respect to time, $M \in \mathbb{R}^{m\times m}$ is the inertia matrix, $C \in \mathbb{R}^{m\times m}$ is the Coriolis matrix, and $g \in \mathbb{R}^m$ is the vector term for compensating gravity, in which $M, C,$ and g are unknown and need to be identified using an FNN. Acceleration and velocity terms are also transformed into discrete-time format. In particular, the following model is used for learning:

$$M_{\phi_M}(q_t)\frac{q_{t+1} - 2q_t + q_{t-1}}{T^2} + C_{\phi_C}(q_t, q_{t-1})\frac{q_t - q_{t-1}}{T} + g_{\phi_g}(q_t) = \hat{\tau}_{t+1} \tag{8}$$

where we do not make the equation into affine format like kinematics model. Since the Coriolis term uses both q_t and q_{t-1} for the input, the feature vector Φ used in the kinematics model is expanded to handle both terms as

$$\Phi_C(q_t) = \begin{bmatrix} \Phi_{1C}(q_t, q_{t-1}) \\ \Phi_{2C}(q_t, q_{t-1}) \\ \dots \\ \Phi_{NC}(q_t, q_{t-1}) \end{bmatrix} \in \mathbb{R}^N, \tag{9}$$

$$\Phi_{iC}(q_t, q_{t-1}) = \exp\left(-([q_t, q_{,t-1}] - c_i')^\top S'^{-1}([q_t, q_{t-1}] - c_i')\right) \in \mathbb{R}, i = 1, 2, \dots, N,$$

where $S' \in \mathbb{R}^{(2m)\times(2m)}$ is the positive-definite covariance matrix, and $c_i' \in \mathbb{R}^{2m}$ is the center of basis. Finally, the approximation for M, C and g is given as

$$M_{\phi_M}(q_t) = \begin{bmatrix} \phi_{M,1}\Phi(q_t) & \phi_{M,2}\Phi(q_t) & \dots & \phi_{M,m}\Phi(q_t) \end{bmatrix},$$
$$C_{\phi_C}(q_t, q_{t-1}) = \begin{bmatrix} \phi_{C,1}\Phi_C(q_t, q_{t-1}) & \phi_{C,2}\Phi_C(q_t, q_{t-1}) & \dots & \phi_{C,m}\Phi_C(q_t, q_{t-1}) \end{bmatrix},$$
$$g_{\phi_g}(q_t) = \phi_g\Phi(q_t), \tag{10}$$

where $\phi_{M,k} \in \mathbb{R}^{m\times N}, \phi_{C,k} \in \mathbb{R}^{m\times N}$ for $k = 1,2,\dots,m$ and $\phi_g \in \mathbb{R}^{m\times N}$ are trainable weights for matrices M, C, and g, respectively.

The optimization objective for the dynamics model is given as

$$L_{dyn} = ||\hat{\tau}_{t+1} - \tau_{t+1}||^2, \tag{11}$$

where $\hat{\tau}_{t+1}$ is defined in Eq. (8), and $\tau_{t+1}, q_t, q_{t-1}, q_{t+1}$ are restored from the replay buffer.

Kinematics self-attention model. We define the self-attention network for the kinematics, which considers the original kinematics model but expand its network structure to utilize time correlation using self-attention structure. After then, the self-attention score of the kinematics model is used for determining the task space, constraining the robot's reachable area as shown in Fig. 2g.

The self-attention network for the kinematics uses the configuration state q as an input that passes through the encoder layer, as shown in Fig. 1a. Then, a total of k duplicates of the encoder layer are constructed, in which the input of each encoder layer becomes an input q from time t to time $t - k$, as shown in Fig. 1b. The duplicated encoder layers share the hidden weights and the gradients. In the matrix form, a total of $k + 1$ recent configuration states are concatenated to be expressed as a matrix $q_{t-k:t} \in \mathbb{R}^{m\times(k+1)}$. The matrix passes through a fully connected encoder layer ($\theta_{E,kin}$) and a position-wise encoding layer (θ_{PE}), indicating that each input originates from a different timestep. The weight parameters for the fully connected encoder layer and the position-wise encoding layer are given as

$$W_{\theta_{E,kin}} \in \mathbb{R}^{h\times m}, b_{\theta_{E,kin}} \in \mathbb{R}^{h\times 1},$$
$$\theta_{PE} \in \mathbb{R}^{h\times(k+1)}, \theta_{PE}(x,y) = \begin{cases} \sin(x/10000^{y/h}) : \text{y even}, \\ \cos(x/(10000^{y-1/h}) : \text{y odd}, \end{cases} \tag{12}$$

where θ_{PE} is not trainable but originates from the transformer structure, indicating the relative position of each vector component[39]. The results of encoding are then given as

$$q_{enc,Q} = \tanh\left(W_{\theta_{E,kin}} q_{t-k:t} + \text{Cat}_{k+1}\left(b_{\theta_{E,kin}}\right)\right) + \theta_{PE} \in \mathbb{R}^{h\times(k+1)},$$
$$q_{enc,K} = \tanh\left(W_{\theta_{E,kin}} q_{t-k:t} + \text{Cat}_{k+1}\left(b_{\theta_{E,kin}}\right)\right) + \theta_{PE} \in \mathbb{R}^{h\times(k+1)},$$
$$q_{enc,V} = \tanh\left(W_{\theta_{E,kin}} q_{t-k:t} + \text{Cat}_{k+1}\left(b_{\theta_{E,kin}}\right)\right) \in \mathbb{R}^{h\times(k+1)},$$
$$\text{Cat}_{k+1}\left(b_{\theta_{E,kin}}\right) := \left[b_{\theta_{E,kin}}, b_{\theta_{E,kin}}, \dots, b_{\theta_{E,kin}}\right] \in \mathbb{R}^{h\times(k+1)}, \tag{13}$$

in which the encodings are for query, key, and value, respectively, for the self-attention layer, and the operator Cat_{k+1} means that the input column vector is concatenated $k + 1$ times so that $\text{Cat}_{k+1}\left(b_{\theta_{E,kin}}\right)$ becomes a matrix. In a typical transformer structure, different weight matrices are multiplied to the encoding of query, key, and value to prevent an excessive self-attention score, given to itself. In contrast, our approach aims to give an excessive self-attention score to itself since the kinematics model can accurately infer the output from its own input without time-correlation.

The encoding matrix now passes through a self-attention structure. The self-attention matrix and the output of the self-attention layer are given as

$$\begin{aligned}&\mathrm{Attn}_{\mathrm{Score,kin}}\left(q_{enc,Q}, q_{enc,K}\right) = \mathrm{softmax}\left(\frac{q_{enc,Q}^{\top} q_{enc,K}}{\sqrt{d}}\right) \in \mathbb{R}^{(k+1)\times(k+1)},\\ &\mathrm{Attn}_{Out,\mathrm{kin}}(q_{enc}) = q_{enc,V}\mathrm{Attn}_{\mathrm{Score,kin}}\left(q_{enc,Q}, q_{enc,K}\right) \in \mathbb{R}^{h\times(k+1)},\end{aligned} \tag{14}$$

which is a dot-product attention with a scalable parameter d[50]. Finally, the self-attention output passes through the decoder layer which consists of two fully connected layers, predicting the task state given as

$$\begin{aligned}&W_{\theta_{D1,kin}} \in \mathbb{R}^{l\times h}, b_{\theta_{D1,kin}} \in \mathbb{R}^{l\times 1},\\ &W_{\theta_{D2,kin}} \in \mathbb{R}^{n\times l}, b_{\theta_{D2,kin}} \in \mathbb{R}^{n\times 1},\\ &\hat{x}_{t-k:t} = W_{\theta_{D2,kin}}\left(\mathrm{ReLU}\left(W_{\theta_{D1,kin}}\mathrm{Attn}_{\mathrm{Out,kin}}(q_{enc}) + \mathrm{Cat}_{k+1}\left(b_{\theta_{D1,kin}}\right)\right)\right)\\ &\qquad + \mathrm{Cat}_{k+1}\left(b_{\theta_{D2,kin}}\right) \in \mathbb{R}^{n\times(k+1)},\end{aligned} \tag{15}$$

where ReLU refers to the rectified linear unit activation function[51]. Similar to the encoder layer, the duplicated decoder layers share the weights and the gradients.

Optimization objective for the kinematics self-attention network is composed of two terms. First, an MSE is used to calculate difference between the predicted $\hat{x}_{t-k:t}$ and the ground truth $x_{t-k:t}$. Next, discrepancy between the self-attention score matrix and the identity matrix is added, written as

$$\begin{aligned}&L(\hat{x}_{t-k:t}, x_{t-k:t}) := ||\mathrm{vec}(\hat{x}_{t-k:t} - x_{t-k:t})||^2\\ &+\mathrm{tr}\left[(\mathrm{Attn}_{\mathrm{Score,kin}}(q_{enc,Q}, q_{enc,K}) - I)(\mathrm{Attn}_{\mathrm{Score,kin}}(q_{enc,Q}, q_{enc,K}) - I)^{\top}\right],\end{aligned} \tag{16}$$

where the operator vec(·) is a vectorization of the given matrix. Minimizing only the MSE for output prediction ultimately means learning the kinematics model without considering self-attention connection. When using only the former term as the learning criterion, the inclusion of a self-attention layer becomes redundant. Consequently, the learning outcome of the self-attention network solely based on MSE minimization can yield multiple local optima. Our objective is to converge towards a scenario where all self-attention results become self-referential, representing one of the local optimal solutions. As previously discussed, this is accomplished by ensuring that the encoding weights for query, key, and value are uniformly applied, prioritizing higher self-attention scores when referring to the same time step rather than other time steps. Furthermore, if the self-attention score decreases, i.e., if the input that depends on the information from other time steps rather than one's own encoding information is entered, we treat it as an abnormal situation.

Dynamics self-attention model. The aforementioned self-attention structure can be applied to the dynamics model as well. In the dynamics model, we map from the input τ to the configuration state q. In particular, Eqs. (12)–(16) can be reorganized for the dynamics self-attention network as

$$W_{\theta_{E,dyn}} \in \mathbb{R}^{h\times m}, b_{\theta_{E,dyn}} \in \mathbb{R}^{h\times 1}, \tag{17}$$

$$\begin{aligned}&\tau_{enc,Q} = \tanh\left(W_{\theta_{E,dyn}}\tau_{t-k:t} + \mathrm{Cat}_{k+1}\left(b_{\theta_{E,dyn}}\right)\right) + \theta_{PE} \in \mathbb{R}^{h\times(k+1)},\\ &\tau_{enc,K} = \tanh\left(W_{\theta_{E,dyn}}\tau_{t-k:t} + \mathrm{Cat}_{k+1}\left(b_{\theta_{E,dyn}}\right)\right) + \theta_{PE} \in \mathbb{R}^{h\times(k+1)},\\ &\tau_{enc,V} = \tanh\left(W_{\theta_{E,dyn}}\tau_{t-k:t} + \mathrm{Cat}_{k+1}\left(b_{\theta_{E,dyn}}\right)\right) \in \mathbb{R}^{h\times(k+1)},\\ &\mathrm{Cat}_{k+1}\left(b_{\theta_{E,dyn}}\right) := \left[b_{\theta_{E,dyn}}, b_{\theta_{E,dyn}}, \ldots, b_{\theta_{E,dyn}}\right] \in \mathbb{R}^{h\times(k+1)},\end{aligned} \tag{18}$$

in which the input of the encoding layer is now changed into τ. Then, the attention score and the attention output can be written as

$$\begin{aligned}&\mathrm{Attn}_{\mathrm{Score,dyn}}\left(\tau_{enc,Q}, \tau_{enc,K}\right) = \mathrm{softmax}\left(\frac{\tau_{enc,Q}^{\top}\tau_{enc,K}}{\sqrt{d}}\right) \in \mathbb{R}^{(k+1)\times(k+1)},\\ &\mathrm{Attn}_{\mathrm{Out,dyn}}(\tau_{enc}) = \tau_{enc,V}\,\mathrm{Attn}_{\mathrm{Score,dyn}}\left(\tau_{enc,Q}, \tau_{enc,K}\right) \in \mathbb{R}^{h\times(k+1)},\end{aligned} \tag{19}$$

and the prediction layer as

$$\begin{aligned}&W_{\theta_{D1,dyn}} \in \mathbb{R}^{l\times h}, b_{\theta_{D1,dyn}} \in \mathbb{R}^{l\times 1},\\ &W_{\theta_{D2,dyn}} \in \mathbb{R}^{m\times l}, b_{\theta_{D2,dyn}} \in \mathbb{R}^{m\times 1},\\ &\hat{q}_{t-k:t} = W_{\theta_{D2,dyn}}\left(\mathrm{ReLU}\left(W_{\theta_{D1,dyn}}\mathrm{Attn}_{\mathrm{Out,dyn}}(\tau_{enc}) + \mathrm{Cat}_{k+1}\left(b_{\theta_{D1,dyn}}\right)\right)\right)\\ &\qquad + \mathrm{Cat}_{k+1}\left(b_{\theta_{D2,dyn}}\right) \in \mathbb{R}^{m\times(k+1)},\end{aligned} \tag{20}$$

where we predict q for the output of the dynamics self-attention network. Finally, the optimization objective of the dynamics self-attention network is given as

$$\begin{aligned}&L(\hat{q}_{t-k:t}, q_{t-k:t}) := ||\mathrm{vec}(\hat{q}_{t-k:t} - q_{t-k:t})||^2\\ &+\mathrm{tr}\left[(\mathrm{Attn}_{\mathrm{Score,dyn}}(\tau_{enc,Q}, \tau_{enc,K}) - I)(\mathrm{Attn}_{\mathrm{Score,dyn}}(\tau_{enc,Q}, \tau_{enc,K}) - I)^{\top}\right].\end{aligned} \tag{21}$$

Utilizing self-attention structure for control and learning

Task space determination by a self-attention kinematics network. We aim to utilize the changes in values of the self-attention matrix over time to monitor the learning status of the kinematics network and to detect abnormal situations, such as incomplete learning or perturbations. This will help us expand or narrow the task space of the robot accordingly. The kinematics of the robot is given as $x_{t+1} = J_0(q_t) + J(q_t)q_{t+1}$ under an assumption of discretization, the current state x_t is only dependent on the variables x_{t-1}, q_t, and q_{t-1}. Returning to the self-attention network, if the model is sufficiently learned minimizing the objective Eq. (16), then the self-attention matrix is similar to the diagonal matrix (see Fig. 1c). In other words, a query of the specific time does not need information far from itself. However, if the robot moves slightly out of the area where it was previously moving, the new data that have not been seen in the training dataset become part of the input of the neural network. As a result, the encoded feature for the new input may produce a random output value since the new input has not been trained before, and the self-attention matrix will then no longer be diagonal. This outcome now serves as an indicator of the new region in the task space that has not been frequently visited.

Utilizing the property of the self-attention matrix, we design a way of constraining the task space region to ensure safe exploration and avoid unexpected movements of the robot. Let our desired task trajectory be given as $x_{1:T,des} \in \mathbb{R}^{n\times T}$ Our optimization objective is to make the robot follow the desired trajectory, but depending on the learning process, we restrict the robot's range of motion using the following setting:

$$\min_{q_{t+1,des}} ||x_{t+1,des} - J_{0\phi_J}(q_t) - J_{\phi_J}(q_t)q_{t+1,des}||^2 + \lambda_t||q_{t+1,des}||^2, \tag{22}$$

where we define $\lambda_t \in \mathbb{R}^+$ as a safety factor. Setting $\lambda_t \to 0$ yields the desired configuration state to recover the desired task state. On the other hand, leveraging λ_t gives penalty when the desired configuration state becomes larger (i.e., the robot moves in a larger region). A solution for the optimization is then given as

$$q_{t+1,des} = J_{\phi_J}(q_t)^{\top}\left(J_{\phi_J}(q_t)J_{\phi_J}(q_t)^{\top} + \lambda_t I\right)^{-1}\left[x_{t+1,des} - J_{0\phi_J}(q_t)\right], \tag{23}$$

which is a linear-quadratic optimization solution[2]. We notice that the larger λ_t makes the task space the smaller, and the smaller λ_t makes the robot the more easily move to the other space.

Thus, simply setting λ_t inversely proportional to the trace value of the self-attention matrix can be a solution. However, this does not handle a long-term effect, where the safety factor should finally decrease to zero to complete the task, but the trace value of the self-attention matrix has an upper limit of $k+1$ (i.e., the softmax operator makes the sum of the column be equal to 1). In other words, the trace value walks sideways but is still capable of detecting the outliers. To address this issue, we suggest the following update rule:

$$\begin{aligned} \alpha_t &:= \mathrm{tr}\left(\mathrm{Attn}_{\mathrm{Score,kin}}\left(q_{enc,Q}, q_{enc,K}\right)\right), \\ \lambda_t &= \lambda_{t-1}\left(1+\gamma\times\frac{\alpha_t-\alpha_{t-1}}{\alpha_{t-1}}\right), \end{aligned} \tag{24}$$

where a constant $\gamma = \{\gamma\in\mathbb{R}|\gamma<-1\}$ is called a leveraging factor. For example, if α_t increases by 1%, then λ_t decreases by $|\gamma|$%, following the inverse relationship.

The proposed update rule expresses a long-term decay of λ_t by the volatility drag effect. Since modeling the behavior of α_t is nontrivial, making it difficult to predict the behavior, but will be related to the current value, let α_t follow the geometric Brownian motion with a stochastic differential equation (SDE) as

$$d\alpha_t = \mu\alpha_t dt + \sigma\alpha_t dW_t \tag{25}$$

with $\mu, \sigma\in\mathbb{R}$, and W_t is Wiener noise. Then, the N-step solution of the λ_t, considering the SDE and Eq. (24), is given as

$$\lambda_N = \lambda_0\left(\frac{\alpha_N}{\alpha_0}\right)^{\gamma}\exp\left(-\frac{(\gamma^2-\gamma)\sigma^2 NT}{2}\right), \tag{26}$$

where $T\in\mathbb{R}^+$ is the real-time interval between timesteps. The solution indicates that even if the index value α_t goes sideways with a rectangular pattern, the long-term value of λ_t will decrease and asymptotically converge to zero. The safety factor still captures the local behavior of α_t that observes the abnormal data in the task space.

Perturbation detection by a self-attention dynamics network. We considered the dynamics self-attention network to filter out the perturbed training data. Note that this method captures input-output outliers and is used to train the kinematics and the dynamics networks but not related to robust control. The filtering method uses a simple threshold. If the trace value of the dynamics attention score matrix exceeds a certain threshold, the dataset of that time is included in the replay buffer as

$$\begin{aligned} &\beta_t := \mathrm{tr}\left(\mathrm{Attn}_{\mathrm{Score,dyn}}\left(\tau_{enc,Q}, \tau_{enc,K}\right)\right), \\ &\mathrm{ReplayBuffer} \leftarrow \mathrm{Append}\begin{cases} \{\tau_t, q_t, x_t\} \text{ if } \beta_i \ge \epsilon, \forall i = \{i\in\mathbb{N}, |, t-\delta\le i\le t\}, \\ \text{None, else,} \end{cases} \end{aligned} \tag{27}$$

where δ is the perturbation buffer time and ϵ is the threshold to determine the outlier. If it does not exceed the threshold, it is considered to have been affected by external disturbances, such as external forces, which correspond to the term in the dynamics equation that cannot be observed, as likely given f_{ext} in

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau + J(q)^{\top} f_{ext}. \tag{28}$$

If a perturbation is detected, then δ samples after the perturbation are neglected since the robot needs time to recover to the stable point. In addition, since the kinematics attention score α_t is also affected by the perturbation, we do not update the safety factor using Eq. (26) while the perturbation is detected. New data is stored in the replay buffer, which has a predetermined fixed size. Once the stored data exceeds the predetermined size, the oldest data is replaced with the new incoming data in a sequential manner. If a threshold is set and only the datasets that meet the condition are added to the replay buffer, many data may not be stored in the replay buffer in the very early stages of training. However, since the same encoding feature is used for query, key, and value, even in the early stages where the learning is not well-established, the trace value is expected to reach a certain level. As a result, the threshold value ϵ in Eq. (27) is determined to be at the average level of the initial short period.

Control strategy. From the planned desired task state $x_{t,des}$, we find the desired configuration state $q_{t,des}$ from Eq. (23) (Fig. 2e). Then, we define the state error $e_t := q_t - q_{t,des}$ and the augmented velocity error $r_t := \dot{e}_t + \Lambda e_t$ with a positive-definite gain matrix $\Lambda\in\mathbb{R}^{m\times m}$. Then, we can define a passivity-based control as

$$\tau_t = M_{\phi_M}(q_t)\left[\ddot{q}_{t,des} - \Lambda\dot{e}_t\right] + C_{\phi_C}(q_t,\dot{q}_t)\left[\dot{q}_{t,des} - \Lambda e_t\right] + g_{\phi_g}(q_t) - K\dot{e}_t - K\Lambda e_t \tag{29}$$

with a positive-definite gain matrix K, to make the closed-loop dynamics as

$$M_{\phi_M}(q_t)\dot{r}_t + C_{\phi_C}(q_t,\dot{q}_t)r_t + Kr_t = \Gamma\left(\phi_M,\phi_C,\phi_g\right). \tag{30}$$

where Γ is the residual term induced by the modeling error of the neural network approximation[2]. If $\Gamma = 0$, then the error exponentially converges to zero. However, the existence of the model error guarantees the boundedness of the error, given the boundedness of Γ. Hence, feedback control will be heavily dependent on the feedback terms in Eq. (29). To avoid unexpected movements induced by the feedforward terms, we initialize ϕ_M,ϕ_C, and ϕ_g to zero. Finally, the algorithm described in Fig. 2 can be summarized by Algorithm 1.

Algorithm 1. Robust Online Model Identification

```
Zero-Initialize $\phi_M$,$\phi_C$, $\phi_g$, and Random Initialize $\phi_J, c_i, c_i'$
Initialize extended model parameters $\theta_{E,kin}$, $\theta_{D,kin}$, $\theta_{E,dyn}$, and $\theta_{D,dyn}$
for each timestep $t$ do
    Ready for the next desired task state $x_{t+1,des}$
    $q_{t+1,des} = J_{\phi_J}(q_t)^{\top}\left(J_{\phi_J}(q_t)J_{\phi_J}(q_t)^{\top} + \lambda_t I\right)^{-1}\left[x_{t+1,des} - J_{0\phi_J}(q_t)\right]$,
    $\tau_{t+1} = M_{\phi_M}(q_t)\left[\ddot{q}_{t+1,des} - \Lambda\dot{e}_t\right] + C_{\phi_C}(q_t,\dot{q}_t)\left[\dot{q}_{t+1,des} - \Lambda e_t\right] + g_{\phi_g}(q_t) - K\dot{e}_t - K\Lambda e_t$
    Obtain self-attention matrices.
    $\alpha_t = \mathrm{tr}\left(\mathrm{Attn}_{\mathrm{Score,kin}}\left(q_{enc,Q}, q_{enc,K}\right)\right)$,
    $\lambda_t = \lambda_{t-1}\left(1+\gamma\times\frac{\alpha_t-\alpha_{t-1}}{\alpha_{t-1}}\right)$,
    $\beta_t = \mathrm{tr}\left(\mathrm{Attn}_{\mathrm{Score,dyn}}\left(\tau_{enc,Q}, \tau_{enc,K}\right)\right)$,
    Save the result into the replay buffer using Eq. (27)
    for $i$ samples $\{x,q,\tau\}$ from the replay buffer do
        $L_{kin} = ||\hat{x}_{t+1} - x_{t+1}||^2$
        $L_{dyn} = ||\hat{\tau}_{t+1} - \tau_{t+1}||^2$
        $L_{kin,attn} = ||\mathrm{vec}(\hat{x}_{t-k:t} - x_{t-k:t})||^2 + \mathrm{tr}[(\mathrm{Attn}_{\mathrm{Score,kin}}(q_{enc,Q}, q_{enc,K}) - I)(\mathrm{Attn}_{\mathrm{Score,kin}}(q_{enc,Q}, q_{enc,K}) - I)^{\top}]$
        $L_{dyn,attn} = ||\mathrm{vec}(\hat{q}_{t-k:t} - q_{t-k:t})||^2 + \mathrm{tr}[(\mathrm{Attn}_{\mathrm{Score,dyn}}(\tau_{enc,Q}, \tau_{enc,K}) - I)(\mathrm{Attn}_{\mathrm{Score,dyn}}(\tau_{enc,Q}, \tau_{enc,K}) - I)^{\top}]$
        Optimize $L_{kin}$, $L_{dyn}$, $L_{kin,attn}$, $L_{dyn,attn}$ and update parameters.
    end
end
```

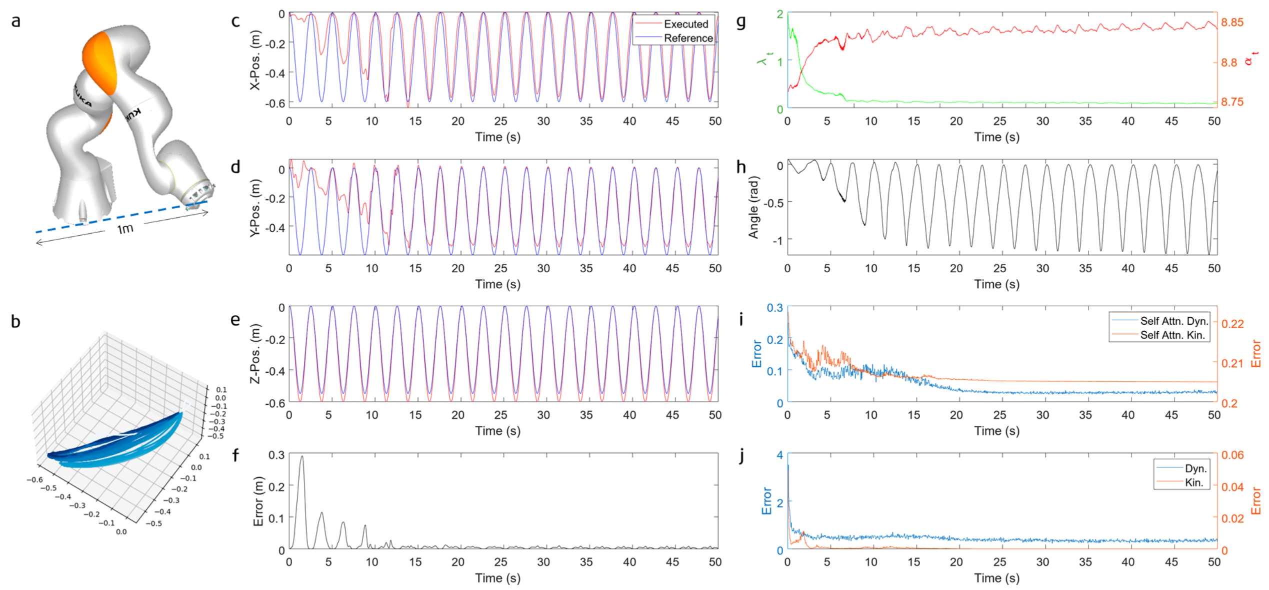

Fig. 3 Validation results in simulated environment of Task 1. a Round-trip task between two points (The model of the robot is from ref. [63]). **b** Overall positions of the end-effector over time for the trajectory tracking, where the darker color indicates the more recent time. **c–e** Three-dimensional position of the end-effector in *x*, *y*, and *z* axes, respectively. **f** Mean-squared error between the reference and the actual trajectories. **g** Auxiliary indicators λ_t and α_t. **h** One of the desired configuration states $q_{t,des}$, showing that the desired state expands over time. **i**, **j** Learning curve of four neural networks, self-attention kinematics and dynamics for determining α_t and β_t, and kinematics and dynamics network for determining $q_{t,des}$ and τ_t.

Validation in the simulation

A virtual robot control environment (PyBullet) was used to show the effectiveness of our approach in the simulation[52]. The robot was asked to make two different motions. One is to travel back and forth between two points (Task 1, Fig. 3a), and the other is to follow a predefined circular trajectory (Task 2, Supplementary Fig. 1a). The overall evolutionary trajectories over time for both cases are displayed in three-dimensional plots (50 s in simulation time), as shown in Fig. 3b and Supplementary Fig. 1b, where the darker curves indicate the more recent activities. The result of path tracking control with simultaneous model learning is shown in Fig. 3c–e, displaying the three-axis positions of the end-effector, also with the mean-squared tracking error in Fig. 3f. Figure 3g shows the values of λ_t and α_t over time. λ_t decays and converges near zero over time and α_t shows gradual increase with fluctuations. From the auxiliary indicators (e.g., λ_t and α_t), the desired configuration state $q_{t,des}$ is determined to move more widely after time passes, as shown in Fig. 3h. Figure 3i, j shows the learning curve of four kinds of neural networks. Zero-regulation results of orientation (i.e., roll, pitch, and yaw angles) is given in Supplementary Fig. 2.

The effect of the outlier detection algorithm is shown in Fig. 4. In this task, we applied the same task as Task 1 to the robot, in which the tracking results with errors are shown in Fig. 3a–d. However, impact perturbations are applied at a certain timestep, as shown with the shaded areas in the curve. Factors λ_t and α_t are also influenced by the perturbation, as shown in Fig. 4e, in which λ_t increases when a perturbation is applied to the robot. Figure 4f shows the value of β_t, indicating that the value decreases almost to the initial value. Nevertheless, the learning curve of the self-attention networks are not interrupted since the outliers are eliminated from the replay buffer (Fig. 4g). Finally, to verify the feasibility of learning the kinematics, Fig. 4h shows the difference between the approximated Jacobian J_{ϕ_J} and the robot's real Jacobian J provided from the simulated environment by calculating the Frobenius norm. As the kinematics network converges relatively fast, as shown in Fig. 3j, learning the Jacobian matrix can approximate the real Jacobian matrix when passing through the desired trajectory.

Application 1: trajectory tracking of a soft robotic manipulator

A custom-built soft robotic manipulator was used to test the proposed algorithm as an application. The soft manipulator is composed of three actuator modules, and each module is made of a pneumatic origami muscle actuator (POMA) encapsulated by a flexible 3D-printed mesh structure, as shown in Fig. 5a[53,54]. A laser pointer was attached to the tip of the manipulator so that the tip position was projected on a two-dimensional surface distant from the tip. A red-green-blue (RGB) camera was used to read the position of the laser projection on the screen. The target task was to track an 'S'-shaped curve on the screen in this application.

The robot was able to follow the reference trajectory, as shown in Fig. 5b. In the first trial, the robot barely made a movement. When the kinematics and the dynamics models were sufficiently trained, the robot was able to follow the full trajectory in real-time. Figure 5c–e show the secondary indicators: the kinematics manipulability (i.e., $\det(JJ^T)$), the tracking error, and the safety factor λ_t for determining the exploration region, respectively. The safety factor λ_t decreases over time, resulting in an increase in the kinematic manipulability. The results of real-time trajectory tracking with model learning and its input-output data logs are provided in Supplementary Materials (Supplementary Video 2 and Supplementary Data 1, respectively).

A comparison study was conducted to determine the effect of the proposed algorithm. Soft actor-critic (SAC), a widely used RL method, was employed as a comparison algorithm[55]. A reward function for the RL setting was given as

$$r_t = 2\exp\left(-\left(x_t - x_{t,des}\right)^2\right) - 0.05\max\left(q_t - q_{t-1}, 10\right)^2, \qquad (31)$$

where the first term is related to the tracking error, the second term penalizes rapid volatility of the configuration state, and the numbers 2, 0.05, and 10 are determined by the best-performance combination. We compare three use cases: our method, SAC with no prior, and SAC with a previously initialized network using our

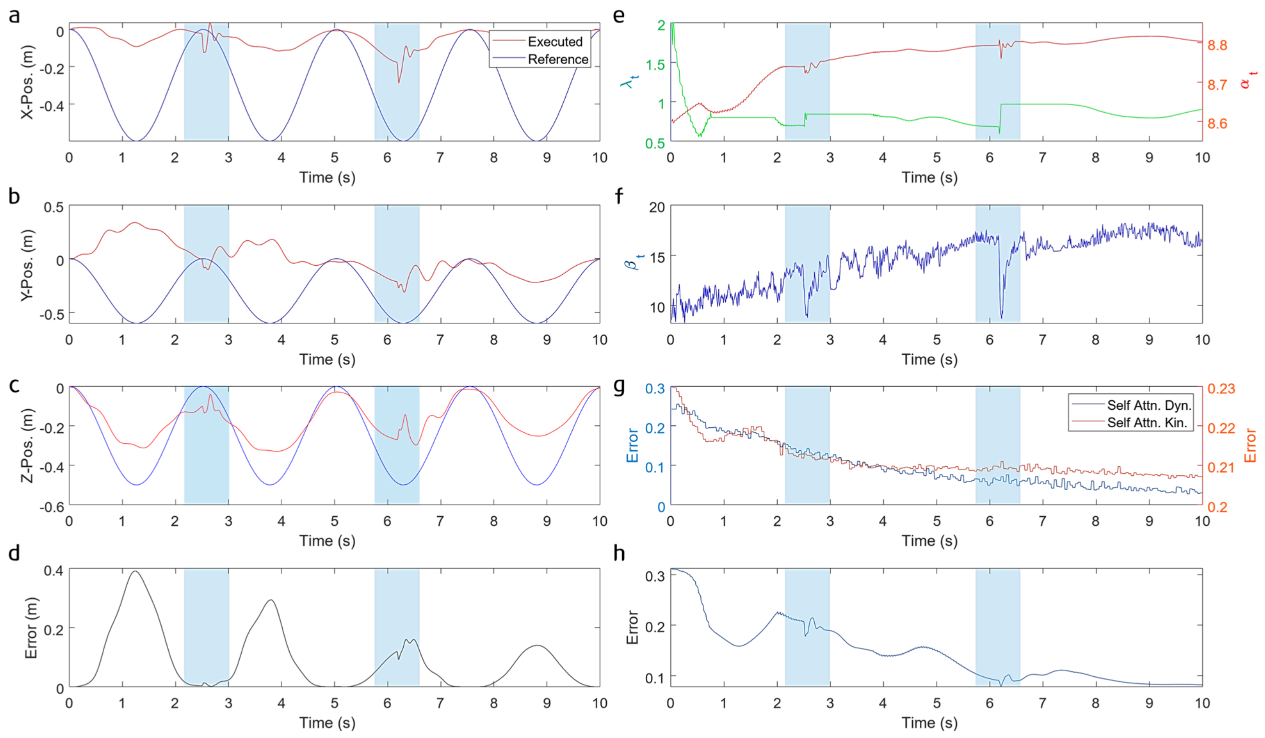

Fig. 4 Results of perturbation test. While performing trajectory tracking, an external perturbation was applied to the end-effector. **a**–**c** Three-dimensional position of the end-effector in *x*, *y*, and *z* axes, respectively. **d** Mean-squared error between the reference and the actual trajectories. **e** Auxiliary indicators λ_t and α_t. **f** Indicator β_t determining the perturbation. **g** Learning curve of the self-attention network. **h** Mean-squared error between the approximated Jacobian J_{ϕ_J} and the real Jacobian of the robot.

method. The reward obtained by each method over time is shown in Fig. 5f. The result shows that SAC took a long period of time to receive a high reward, while our method quickly obtained a high reward. The tracking performance of the RL algorithm over a long period of time is visualized in Supplementary Materials (Supplementary Fig. 3). Merging the two methods eventually surpassed the performance of the two individual cases when the SAC network was sufficiently trained. The result implies that our algorithm can be used as an accelerator for the RL algorithm to solve high-level tasks, such as grasping, interaction, or tasks involved with model-based RL problems.

Application 2: autonomous manipulation of a robotic arm

An industrial robotic arm (UR5e, Universal Robots) was used for another application, that is, play the piano. In this task, it was assumed that there was no information on the kinematics of the robot available. Instead, the kinematics model was learned by the proposed algorithm and used to perform the task. However, the dynamics model was not trained, since the robotic arm used in this experiment did not support real-time force control. The desired trajectory for playing the piano was obtained by kinesthetic teaching; a human expert moved the manipulator and demonstrated how to play the melody, and the orientation of the end-effector was sequentially saved in the buffer. A single task episode carried 200 timesteps, and 25 episodes (total 5000 timesteps) were executed.

The overall result is summarized in Fig. 6. Figure 6a shows the global coordinate and the experimental setting for the kinesthetic teaching. The human expert's trajectory is provided in Supplementary Materials (Supplementary Data 2). Figure 6b shows the contour analysis of the robot movement in timesteps $t = 1000$, 2000, and 3000. A contour volume of the robot at $t = 0$ was valued at V_0, and the volume ratio increased over time and reached 1.50 at $t = 3000$. In other words, the robot initially moved in the vicinity of its origin but finally broadened the task space. The full data can be found in Supplementary Materials (Supplementary Data 3). Figure 6c shows the reference and actual motions of the last episode. Figure 6d, e show the safety factor and the tracking error, respectively, both of which decreased over time. The overall process of kinesthetic teaching, model learning, and performance testing is visualized in Supplementary Materials (Supplementary Video 3).

The analysis of the results in detail shows that there exists a small peak in the actual trajectory, shown in Fig. 6c, i. In addition, the actual trajectory lagged behind the schedule of the reference, as shown in Fig. 6c, iii, meaning that the closed-loop feedback system had a boundedness property. Let the governing kinematics equation for the robot be $x_{t+1} = f(x_t) + g(x_t)q_t$, which is the first-order approximation for the kinematic Jacobian. However, we know the approximated functions $\hat{f}$ and $\hat{g}$ where the discrepancy errors are defined as $\hat{f} := f + \delta$ and $\hat{g} := g + \epsilon$, respectively. Then, the feedback control becomes $q_{t,des} = (g + \epsilon)^{+}(x_{t+1,des} - f - \delta)$ using the approximated neural network model. The expected execution is given as $x_{t+1,exp} = f + gq_{t,des}$. The tracking error of the next state is now given as

$$\begin{aligned} e_{t+1} &= x_{t+1,des} - x_{t+1,\exp} \\ &= \delta + \epsilon(g+\epsilon)^{+}\left(x_{t+1,des} - f - \delta\right) \\ &= \delta + \epsilon\hat{g}^{+}\left[x_{t+1,des} - \hat{f}\right]. \end{aligned} \tag{32}$$

As a result, the tracking error is proportional to the model inaccuracies, δ and ϵ. If we choose a sufficiently rich neural network model, then we can minimize the model inaccuracy by the online model update algorithm. However, there exists a trade-off between the tracking error and the operating frequency, since a complex neural network model degrades the performance of the real-time controller.

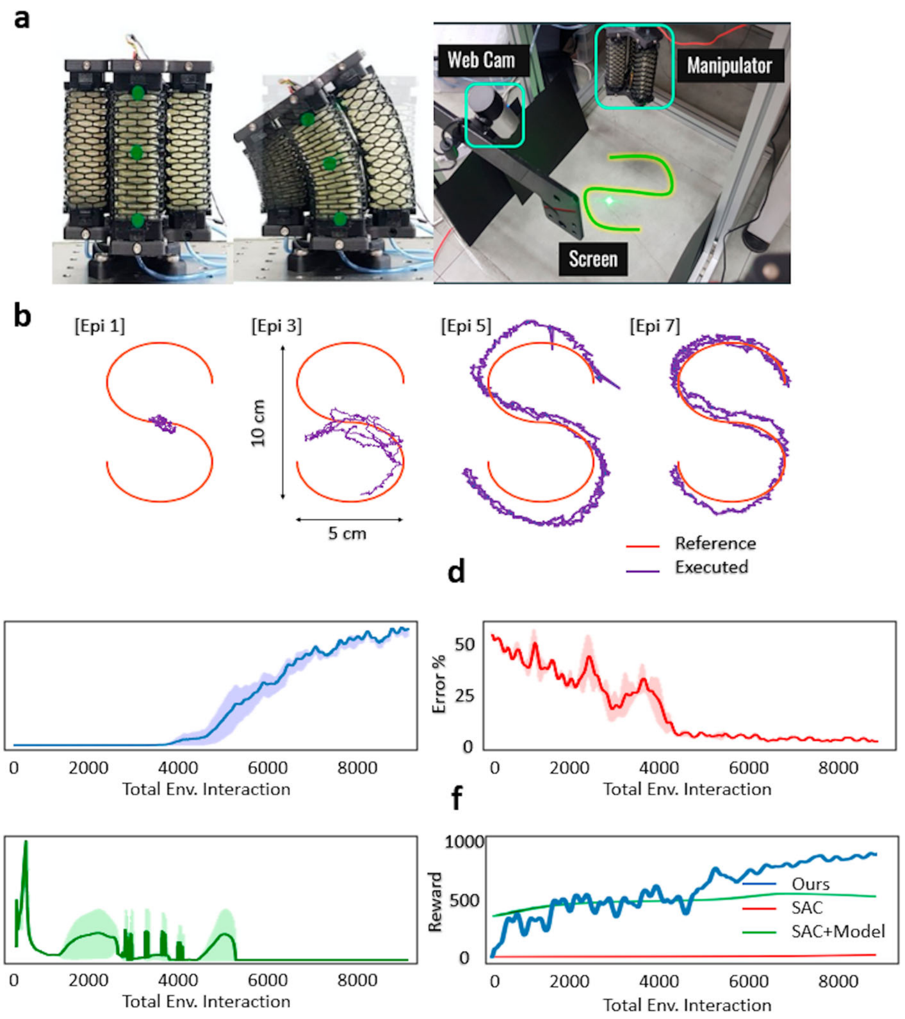

Fig. 5 Results of tracking control of a soft robotic manipulator. a Origami-based soft robotic manipulator for bending with pneumatic pressure inputs[50]. The manipulator was hung by fixing the base to the structure, and the laser module installed at the tip of the manipulator projected the light on the screen. A camera detects the trajectory of the projection and sends the positional information on the trajectory to the computer. **b** Reference and executed trajectories of the Episodes 1, 3, 5, and 7. **c**–**e** Auxiliary indicators, meaning the kinematic manipulability, the tracking error, and λ_t, respectively. **f** The result of the comparison study (SAC).

In conclusion, the proposed algorithm accomplished the task of online tracking of a complex trajectory with the assistance of a human expert. Although a commercial robotic arm was used in this demonstration, the overall approach can be extended to any unknown or custom-built robot.

Application 3: gait training of quadruped robot

Controlling legged robots, such as quadrupedal robots or humanoids, often requires high performance controllers that consider the stability of the center of mass, the walking balance, and phase matching[17,56]. Unfortunately, our approach based on kinematics and dynamics did not fully utilize those properties. Instead, we copied the walking behavior of a quadrupedal creature and implemented it as the reference trajectory. We used a commercial quadruped robot (Laikago, Unitree) for a validation experiment. The goal was to make the robot walk straight. The result is presented in Fig. 7. Figure 7a, b summarize the coordinate settings and the transmission protocol for the agent. Figure 7c shows the horizontal and vertical motions of each foot. It is observed that the motions became larger as time passed, indicating that the overall trajectory was affected by the safety factor that decreased over time. This is also observed in Fig. 7d, where the vertical position gradually increased over time. The real-time operation of the robot can be found in Supplementary Materials (Supplementary Video 4), and the Cartesian position of the robot foot of a single operation can also be found in Supplementary Materials (Supplementary Data 4).

In this application, the robot was unable to sufficiently explore the kinematics due to the relatively large weight of the battery attached to the robot, and we artificially created an exploration term that did not influence the overall performance of the robot, using the null-space exploration term. Let the feedback control be $q_{t,des} = \hat{g}^+(x_{t+1,des} - \hat{f})$. Then, we define the null-space of the Jacobian $\hat{g}$ as $[I - \hat{g}^+\hat{g}]$. We applied the desired configuration state as

$$q_{t,des} = \hat{g}^+\left(x_{t+1,des} - \hat{f}\right) + \left[I - \hat{g}^+\hat{g}\right]c_t, \tag{33}$$

where we leave c_t as an exploration constant decaying over time. By applying this control, the null-space exploration term eventually generates the internal motion that does not degrade the overall performance (see Supplementary Document 1, Appendix B).

Last, as shown in Fig. 7a, the actual trajectories did not exactly follow a cycloid curve, as drawn at the top of Fig. 7a. We observed that this result came from the high friction between the foot and the ground. The friction effect shows a highly nonlinear behavior that is tedious to model, even with the aid of a neural network, since friction model functions are discontinuous (e.g., Newton's

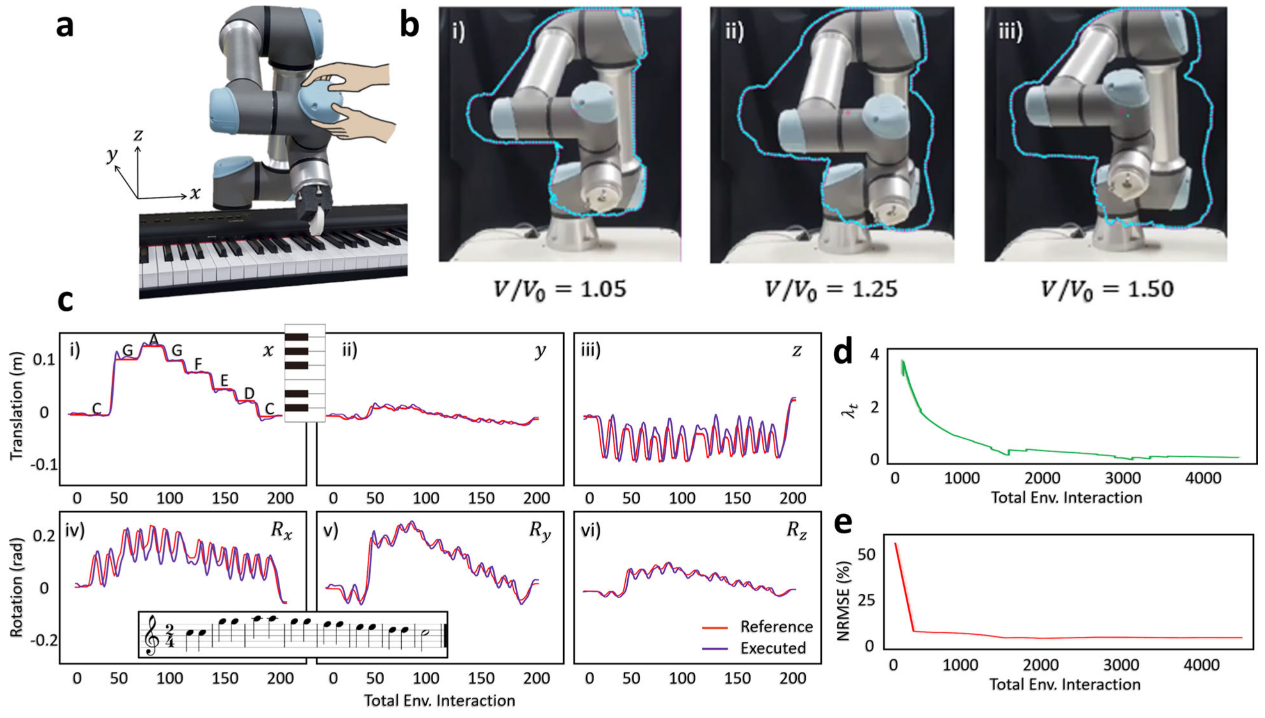

Fig. 6 Results of autonomous manipulation of a robotic arm. a The robot is first taught how to play the piano, and it records the trajectory of the end-effector. **b** While training, the robot moves in the vicinity of its original point at first but gradually broadens its task space. V_0 is the initial volume of the robot body locus, and V is the current volume. **c** Orientation of the end-effector. **d**, **e** Auxiliary indicators: the safety factor and the tracking error, respectively.

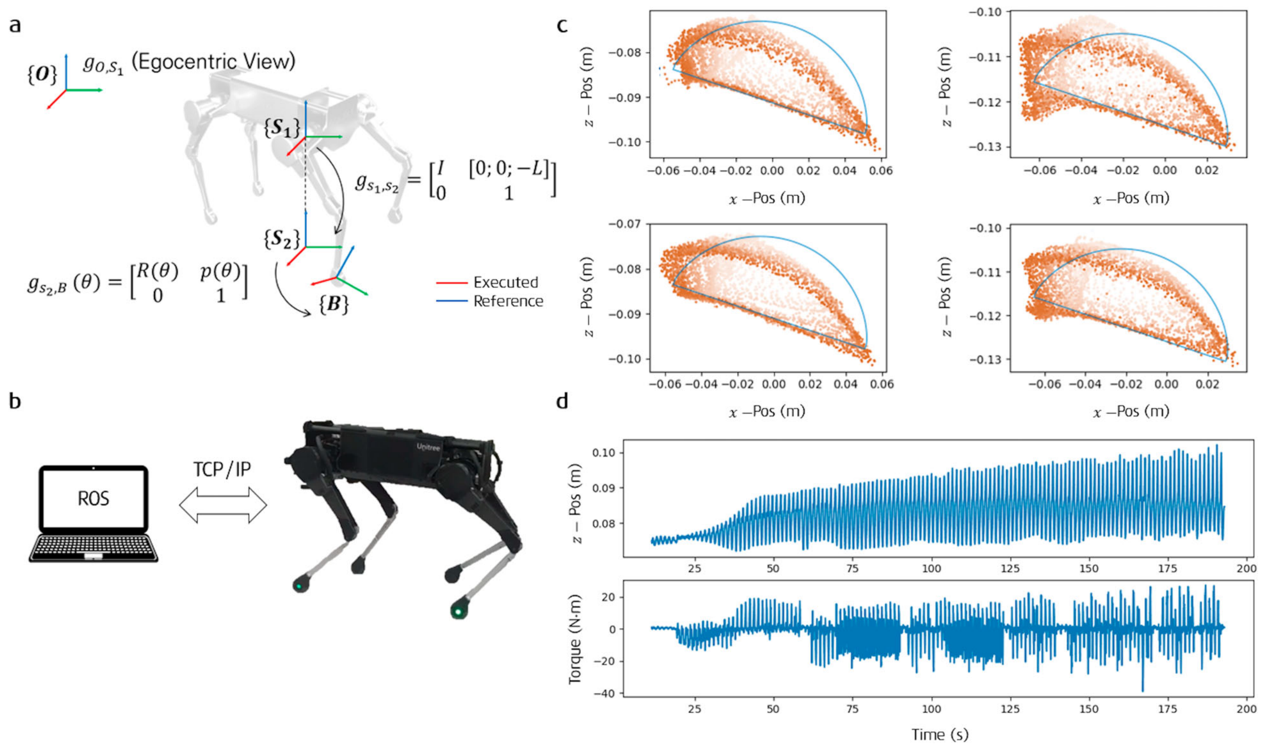

Fig. 7 Results of gait training of quadruped robot. a Overall schematic of the coordinate transformation for the agent. **b** Commercial quadrupedal robot communicating with the computer via TCP/IP using ROS. **c** Comparison between the reference and the actual trajectories. The darker color indicates the more recent time. **d** Vertical position of a single foot over time, together with the applied torque to the leg.

Table 1. Results of the ablation study and the results of the statistical tests.

Cases ($n = 8$)	Statistics		Mann–Whitney test		Two-sample *F*-test	
	Mean	Std.	*p* value	Asterisks	*p* value	Asterisks
Full Algorithm	0.0518	0.0132	–	–	–	–
No Masked Attention	0.0615	0.0154	0.3282	ns	0.7043	ns
No Leveraging λ_t	0.0729	0.0044	0.0003	***	0.0095	*
No Pos-Wise Encoding	0.0913	0.0409	0.0002	***	0.0081	*
Zero-Init. of ϕ_J	0.3316	0.0186	0.0002	***	0.3886	ns

block model[57]). This effect was inevitable in our model setting, indicating the limitation of the proposed algorithm. Nevertheless, our approach was able to successfully achieve the desired locomotion in three minutes from scratch.

DISCUSSION

Ablation study

We performed an ablation study to evaluate the effect of the four specific methods employed in the algorithm: masking for self-attention, use of the leveraging factor $\gamma < -1$, position-wise encoding for self-attention, and random initialization. The results of the study are presented in Table 1 and visualized in Fig. 8. Statistical tests on the mean and the standard deviation values were applied to the four ablation cases.

The effect of masking in the self-attention mechanism was first tested. Masking is a widely-used method to avoid referring to future inputs while training since a time series has a causality property. However, the result with masked self-attention showed no significant difference from that with non-masked self-attention because the proposed algorithm was used to find the abnormality in the datasets rather than to predict or generate them.

The next test was on the use of the leveraging factor. Updating the safety factor λ_t by following the leveraged ratio of the kinematics attention score α_t was used for gradually downsizing the factor to accelerate the exploration. The leveraging factor was fixed to $\gamma = -1$, and the performance was compared with that of the original algorithm. The update rule for λ_t using the leveraged ratio showed significant differences in both the mean and the standard deviation values, implying the acceleration of both the exploration and convergence of the model with the leveraging strategy. In addition, leveraging improved the reproducibility of the experiment as a result of a low standard deviation value.

The effect of position-wise encoding for self-attention was also tested. In this case, the self-attention layer worked as a time-recurrency of the system and thus played an essential role in analyzing both the kinematics and the dynamics models. Therefore, position-wise encoding also showed a significant improvement in performance compared to the case without position-wise encoding.

Finally, the effect of initializing the kinematics network by sampling from the normal distribution was checked, and it was confirmed that initialization contributed to determining the starting point of exploration. Two cases of initialization, namely, random initialization and zero initialization, were compared. The result indicates that random-initialization of the kinematics network facilitated the system to reduce the tracking error and to accelerate the convergence. After all, if the network uses zero initialization, then the kinematics network is less willing to explore a new region at the initial operation.

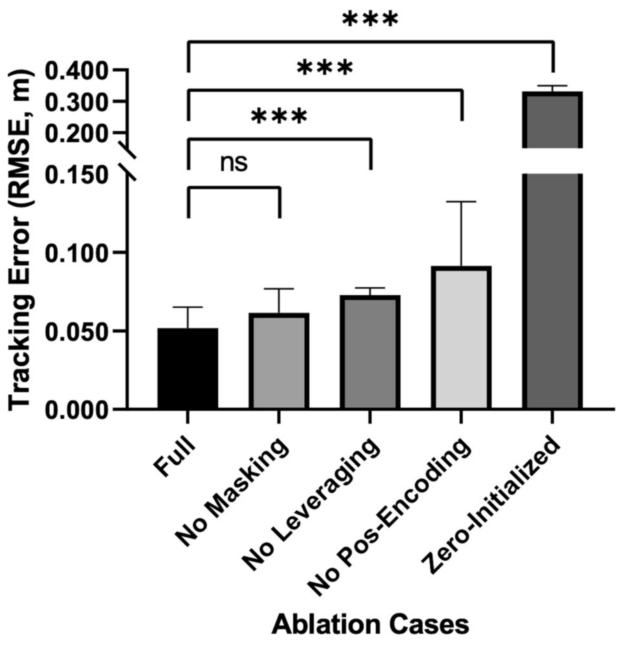

Fig. 8 Results of the ablation study. Result of the ablation study showing no significance between the complete algorithm and the unmasked self-attention but showing significance among the other three ablation cases that are leveraging the safety factor, applying position-wise encoding, and randomly initializing the kinematics network.

Effect of random initialization

In this section, we explain why an RBNN was chosen rather than a multilayer perceptron structure. We describe the behavior of the output using a vanilla example. Let us consider an artificial neural network as a network that consists of multidimensional inputs, a single hidden layer with hidden units N_H, and a scalar output. We can then express the mapping from the input $x \in \mathbb{R}^n$ to the output $y \in \mathbb{R}$ as

$$y = f(x) := b_2 + \sum_{i=1}^{N_H} w_{2,i}\sigma\left(w_{1,i}^\top x + b_{1,i}\right), \tag{34}$$

where $b_1 \in \mathbb{R}^{N_H}$ and $b_2 \in \mathbb{R}$ are the bias terms, $w_1 \in \mathbb{R}^{n \times N_H}$ and $w_2 \in \mathbb{R}^{N_H}$ are the weights, and $\sigma(\cdot)$ is a nonlinear activation function. We denote all the weights (i.e., [b_1;b_2;vec(w_1); w_2]) by w, initialized by sampling from the prior distribution. If the sampling distribution is chosen as

$$b_2 \sim N(0, \sigma_{b_2}^2), w_2 \sim N(0, \sigma_{w_2}^2), b_1 \sim N(0, \sigma_0^2), \mathrm{vec}(w_1) \sim N(0, \sigma_0^2), \tag{35}$$

then it is straightforward to calculate the following property:[58]

$$\mathbb{E}_w[f(x)] = 0,$$
$$\mathbb{E}_w[f(x)f(x')] = \sigma_{b_2}^2 + N_H\sigma_{w_2}^2 \cdot \tfrac{2}{\pi}\sin^{-1}\left(\frac{2x^\top \mathrm{diag}(\sigma_0^2)x'}{\sqrt{\left(1+2x^\top \mathrm{diag}(\sigma_0^2)x'\right)\left(1+2x'^\top \mathrm{diag}(\sigma_0^2)x\right)}}\right), \tag{36}$$

when a hyperbolic tangent is used as an activation function. The result indicates a kernel between x and x' and thus a Gaussian process when $N_H \to \infty$. However, when we use an RBNN, we can derive the covariance as,

$$\mathbb{E}_{\mathbf{w}}[f(x)f(x')] = \exp\left(-\frac{x^\top x}{2\sigma^2}\right)\exp\left(-\frac{||x-x'||^2}{2s^2}\right)\exp\left(-\frac{x'^\top x'}{2\sigma^2}\right) \tag{37}$$

which is an enveloped decaying Gaussian function[58]. Comparing Eqs. (36) and (37), we can infer that both networks behave like a Gaussian process when their weights are randomly initialized, which is the same situation as when a robot is about to start exploration and model learning. We showed that random initialization of the kinematics network gradually improved the tracking performance in the ablation study. However, an FNN defines the similarity between two inputs x and x' as an inner product, where the RBNN defines the similarity as the Euclidean distance. Therefore, the covariance of the RBNN is consistent within small input evolution, where the robot should exercise caution in the initial learning phase since no information is given to the robot.

Random initialization was not performed for the M, C, and g matrices, but zero-initialization was used instead. This is because we employed a two-step control strategy. The desired configuration state $q_{t,des}$ is first obtained using the desired task state $x_{t,des}$, and then the control input τ_t is further derived. When examining the equation for obtaining the control input using the desired configuration state, the inertia term and the Coriolis force term are multiplied by the acceleration and the velocity terms of the desired configuration state, respectively. During the initial stages of training, when the value of λ_t is high, the acceleration and the velocity components exhibit diminished magnitudes. Consequently, the gradients associated with the matrices M and C are also small in magnitude. Therefore, the matrix g is trained relatively quickly in the initial stages. This prevents unexpected movements resulting from the inertial terms when the learning has just started.

Network structure and component

As previously mentioned in the results section, the proposed algorithm used a discretized kinematics model instead of an original continuous-time Jacobian model. By using this first-order model and the update rule by gradient descent, the overall model worked as a piecewise linear model that was able to best express the local behavior near the bin. Functions f and g are approximated by the RBNN with a predefined Gaussian activation function. When operating the robot, the kinematics model is similar to the locally linearized model near the closest center of the Gaussian activations. When updating the model, the weight update ratio considers the distance between the current state and the center. If the center is far from the current state, then the weights of the neural network corresponding to the center will rarely be updated. On the other hand, if the center is close to the current state, then the weights will be periodically updated, realizing continual learning of our algorithm. If we need to perform multiple tasks at once, then all we need is to update the network sequentially by task rather than reconstructing a new model.

Both the dynamics model and the dynamics self-attention model consider only the relationship between the control input and the configuration state, neglecting the influence of external forces and the geometric constraints inherent in the actual dynamics of a robot. While it is feasible to equip the robot with force sensors, external perturbations may appear on any locations on the robot, making them undetectable by the sensors. As a result, these outliers result in reduction in the dynamics self-attention score and are consequently excluded from the replay buffer. However, focusing solely on the relationships within the dynamics model can be influenced if any self-collisions occur during training. While the dynamics attention model can detect the onset of the self-collision through the temporal patterns of the dynamics self-attention matrix, it is difficult to distinguish these patterns from the pattern from the external perturbations. As a result, dataset exclusions from replay buffer may also occur if there are self-collisions. This implies the importance of meticulously designing the desired task to prevent such issues. In this work, the desired task trajectories are designed as conservatively as possible to ensure the robot does not enter self-collision state, and use automatic safety functions embedded in the robot platform.

Relation to previous methods

We compare our approach with previously developed RL algorithms that consider model-based methods or control-oriented methods. Model-based RL methods by using a specific cost function regarding its trajectory constraints or incorporating model uncertainties have previously been developed[18,59]. In particular, an iterative linear quadratic regulator (iLQR) was used as a controller to generate guided samples[18], and nonlinear lifting was used to depict highly nonlinear terms with model predictive control[59]. Both approaches use iterative optimization based on local linearization. Our method can be rearranged as an iLQR form by

$$\begin{aligned} & x_{t+1} = x_t + J_{\phi_J}(q_t)(q_{t+1} - q_t), \\ & r(x_{t+1}, q_{t+1}) = ||x_{t+1,des} - x_{t+1}||^2 + \lambda_t ||q_{t+1}||^2, \end{aligned} \tag{38}$$

which yields iLQR recursive relation to calculate Q-function and value function as

$$\begin{aligned} & Q_{xx,t} = 1, Q_{qq,t} = \lambda_t + J^\top V_{xx,t+1} J, Q_{qx,t} = 0, \\ & Q_{x,t} = x_t - x_{t+1,des}, Q_{q,t} = J^\top V_{x,t+1}, \\ & V_{x,t} = x_t - x_{t+1,des}, V_{xx,t} = 1, \end{aligned} \tag{39}$$

where subscripts mean Jacobians and Hessians with respect to x and q. The results finally derive the same solution as Eq. (23) by defining feedback control $q_{t+1,des} = q_t - Q_{qq,t}^{-1} Q_{q,t}$. As a result, our method follows the flow of model-based RL, where employing λ_t and planning its value over time is the unique property of our method.

METHODS

This section discusses the environmental settings for the three validation experiments. All the source codes were written in Python 3, and all the hyperparameters on the experiments were determined by sweeping the parameters and selecting the best combination.

Learning implementation

For initial setting, we designated an initial position with position control and then changed it to a force control mode. We clipped the measured force, velocity, and acceleration values with a hyperbolic tangent function to prevent any undesired behaviors in the deep learning model. The learning rate for each neural network models was 0.005 with an Adam optimizer. The centers c_i of the RBNNs were also updated using the gradient descent. The size of the replay buffer was set to be 6000 samples, and the model was updated every 20 timestep with minibatch size of 32. The size of the self-attention matrix was 20×20, in which we refer to 20 past samples for detecting the anomaly. A leveraging factor $\gamma = -3$ was determined from its best-performance combination, but we clipped the maximum volatility to be 20% of its previous value to prevent unexpected motions. Moreover, a lower limit for the safety factor λ_t exists to avoid a kinematic singularity, which is often inevitable during the control manipulator (see Supplementary Document 1, Appendix C). To calculate the average value of β_t for determining ϵ in Eq. (27), approximately 1500 ms are required in the simulated environment and 10 s in the real-world environment. We performed a gradient descent of the kinematics and the dynamics models from scratch for rapid online calculation and simplicity. On the other hand, we received the assistance of an automated back-propagation tool while optimizing the self-attention models (Adam optimizer[60]). The selected hyperparameters were applied to both the simulated environment and the actual robot environments, but their values were slightly adjusted to account for differences in operation frequency, as summarized in Table 2.

Table 2. List of hyperparameters used in each experiment.

Parameter	Simulation	Soft robot	Manipulator	Quadruped
Initial safety factor λ_0	2.0	2.0	3.5	2.0
Number of center (Eq. (4))	25	30	30	30
Learning rate	5e−3	5e−3	5e−3	3e−3
Batch size	32	6	32	6
Attention matrix size	20	15	10	20
Leveraging factor γ	−3	−3	−3	−3
Average period of ϵ (Eq. (27))	1500 ms	10 s	–	10 s
Sliding feedback gain	0.35	0.5	0.5	0.35
Buffer size	6000	6000	2000	5000

The dynamics model requires the learned inertia matrix to adhere to the constraint of being symmetric positive-definite. To ensure this constraint, the Cholesky decomposition technique was employed during the learning process. This involves constructing a lower triangular matrix with positive diagonal elements, which can be multiplied by its transpose to yield a symmetric positive-definite matrix. Consequently, the weight utilized in computing the inertia matrix is specifically designed to obtain a triangular matrix to match the Cholesky decomposition requirements.

Simulation environment

In the simulation for validation, we used the open-sourced PyBullet physics engine compatible with the Python environment. A six degree-of-freedom (DOF) robotic arm (KUKA) with a default setting in the PyBullet engine was used in the simulation. The overall environment was exposed to a gravitational field ($-9.81\hat{k}\text{m/s}^2$). The PyBullet engine used a fixed timestep of 0.001 s in the simulation world.

Trajectory tracking was performed for the validation task. Two tasks, named Task 1 and Task 2, were executed, as shown in Fig. 3a and Supplementary Fig. 1a, respectively. In Task 1, a trajectory of back-and-forth motions composed of a sine wave over time was given, and in Task 2, a circular trajectory in the $x - y$ plane was given. A feedback gain for force control was given as 25 in the diagonal terms of the matrix.

For the statistical test in the ablation study, we used statistics and visualization tools (Prism 9, GraphPad) for analysis and plotting. We assumed that each ablation group was independent of the others, and the dependent variables were the ordinals. Therefore, we conducted a Mann–Whitney test (i.e., Wilcoxon rank-sum test) to compare the mean values and a non-parametric two-sample t-test. In addition, a two-sample F-test was performed to distinguish the repeatability of each experiment. A total of eight samples were used for each ablation case ($n = 8$), the asterisks indicate $^*P < 0.05$, $^{**}P < 0.01$, $^{***}P < 0.001$ and "ns" indicates non-significance between two groups.

Soft manipulator environment

A simple soft robotic manipulator, composed of three linear soft pneumatic actuators was used in this experiment[50]. Each actuator was made of an origami air chamber that was able to contract and expand along its length. The air chamber was fabricated with a 150-polyethylene terephthalate (PET) film and sealed with a polyamide film. Three independent linear motions allow the entire structure to bend in three different directions, eventually enabling the manipulator to span three-dimensional space. Therefore, the structure can be used for localizing the position of the end effector in three-dimensional space. For the conciseness of the system, we selected the task to track the designated path in a two-dimensional plane. At the tip of the robot structure, a laser module ($\lambda_t = 650$ nm, $P = 5$ mW, Epxfavjd, Amyta) was attached so that the manipulator projected the laser point on the test plane. The projected laser point was then captured by an RGB camera (Q2F-00014), and the data of the relative pixel positions were collected by using an open-source computer vision library (cv2) with a scale-invariant feature transform.

A pneumatic input source for each actuator attached to the robot was controlled by a pressure regulator (ITV2011-212BS5, SMC Corp.) connected to a microcontroller (Arduino Mega). A Python environment in the laptop communicated with the microcontroller via the transmission control protocol (TCP) in real-time. The input $\tau \in \mathbb{R}^3$ was an applied voltage to the pressure regulator, the configuration state $q \in \mathbb{R}^3$ was a current pressure level applied to the actuators, and the task state $x \in \mathbb{R}^2$ was the two-dimensional projection of the laser module.

Robotic arm environment

We validated the algorithm with a six-DOF industrial robotic arm. The robot was controlled in real-time by the transmission control protocol/internet protocol (TCP/IP) communication via the Python URBasic library[61]. The desired joint input was applied through the command for simultaneous position servoing in URBasic, which is a real-time torque control targeting a smoothed trajectory. The overall operation frequency of the system was 25 Hz, in which the computed torque control achieved the desired motion in the look-ahead time of 0.15 s with a sliding feedback gain of 0.5. The robot performed only position control, so we neglected the dynamics model and considered position control with the kinematics model.

The desired trajectory of the task was obtained by collecting the end-effector trajectory through kinesthetic teaching. To perform teaching, admittance control (i.e., free-drive mode) was used with the least target mass and damper setting. The robot was complied with a safety setting in case of an emergency.

We demonstrated playing the piano. A human operator taught the robot to play the melody of the music (W. A. Mozart, "Ah, vous dirai-je, maman K. 265, TEMA) with a piano (Yamaha P-125). The end-effector for pressing the keyboards of the piano was made from silicone rubber (EcoFlex-0030). A tempo for teaching was 45 beats-per-minute (bpm), where the actual demonstration was in the tempo of 60 bpm by downsampling the trajectory. A contour analysis is shown in Fig. 6b, a postprocessed image using a contour analysis video processing program (ProAnalyst, Xcitex).

Quadruped robot environment

In the last application, we used a commercial 12-DOF quadruped robot (Laikago Pro, Unitree) and tried to make it walk. The robot was controlled by the torque command on each motor torque in real time. The overall communication was performed by the robot operating system (ROS) with the Python 2 environment via the user datagram protocol (UDP) communication. This communication was implemented by the Unitree libraries with an operation frequency of 500 Hz[62]. The states of the robot were also transmitted to the computer through the UDP communication. We consider each of the four legs as an individual agent. An input $\tau \in \mathbb{R}^3$, a configuration state $q \in \mathbb{R}^3$, and a task state $x \in \mathbb{R}^3$ are the motor torque, the joint angle, and the Cartesian position of the foot, respectively. In order to account for the variation in motor torque due to the contact between the robot foot and the ground, we used both the actual torque applied to the motor and the target motor torque as the input values of the dynamics network. A cycloid-type curve was used as a reference trajectory, but it was slightly modified to overcome the relatively high friction between the foot and the ground. For the overall gait motions of the quadruped robot, we referred to the previous literature[17].

DATA AVAILABILITY

Source codes are available at github.com/mochacoco/npjselfattn.

Received: 6 March 2023; Accepted: 11 October 2023;
Published online: 07 December 2023

REFERENCES

1. Spong, M. W., Hutchinson, S. & Vidyasagar, M. *Robot Modeling and Control* (Wiley, 2020).
2. Lynch, K. M. & Park, F. C. *Modern Robotics* (Cambridge University Press, 2017).
3. Polydoros, A. S. & Nalpantidis, L. Survey of model-based reinforcement learning: applications on robotics. *J. Intell. Robot. Syst* **86**, 153–173 (2017).
4. Kwon, J., Choi, K. & Park, F. C. Kinodynamic model identification: a unified geometric approach. *IEEE Trans. Rob.* **37**, 1100–1114 (2021).
5. Jaquier, N., Rozo, L., Caldwell, D. G. & Calinon, S. Geometry-aware manipulability learning, tracking, and transfer. *Int. J. Robot. Res.* **40**, 624–650 (2021).
6. Abu-Dakka, F. J., Huang, Y., Silvério, J. & Kyrki, V. A probabilistic framework for learning geometry-based robot manipulation skills. *Rob. Auton. Syst.* **141**, 103761 (2021).
7. Park, Y.-L. et al. Design and control of a bio-inspired soft wearable robotic device for ankle-foot rehabilitation. *Bioinsp. Biomim.* **9**, 016007 (2014).
8. George Thuruthel, T., Renda, F. & Iida, F. First-order dynamic modeling and control of soft robots. *Front. Robot. AI* **7**, 95 (2020).
9. Kim, D. et al. Review of machine learning methods in soft robotics. *PLoS ONE* **16**, e0246102 (2021).
10. Rus, D. & Tolley, M. T. Design, fabrication and control of soft robots. *Nature* **521**, 467–475 (2015).
11. Chin, K., Hellebrekers, T. & Majidi, C. Machine learning for soft robotic sensing and control. *Adv. Intell. Syst.* **2**, 1900171 (2020).
12. Li, M., Kang, R., Branson, D. T. & Dai, J. S. Model-free control for continuum robots based on an adaptive kalman filter. *IEEE/ASME Trans. Mechatron.* **23**, 286–297 (2017).
13. Huang, W., Huang, X., Majidi, C. & Jawed, M. K. Dynamic simulation of articulated soft robots. *Nat. Commun.* **11**, 1–9 (2020).
14. Park, M., Jeong, B. & Park, Y.-L. Hybrid system analysis and control of a soft robotic gripper with embedded proprioceptive sensing for enhanced gripping performance. *Adv. Intell. Syst.* **3**, 2000061 (2021).
15. Kim, D., Kwon, J., Jeon, B. & Park, Y.-L. Adaptive calibration of soft sensors using optimal transportation transfer learning for mass production and long-term usage. *Adv. Intell. Syst.* **2**, 1900178 (2020).
16. Abraham, I. & Murphey, T. D. Active learning of dynamics for data-driven control using koopman operators. *IEEE Trans. Robot.* **35**, 1071–1083 (2019).
17. Hwangbo, J. et al. Learning agile and dynamic motor skills for legged robots. *Sci. Robot.* **4**, eaau5872 (2019).
18. Levine, S. & Koltun, V. Guided policy search. In *Proc. International Conference on Machine Learning* 1–9 (PMLR, 2013).
19. Cheng, L., Hou, Z.-G. & Tan, M. Adaptive neural network tracking control for manipulators with uncertain kinematics, dynamics and actuator model. *Automatica* **45**, 2312–2318 (2009).
20. Lyu, S. & Cheah, C. C. Data-driven learning for robot control with unknown jacobian. *Automatica* **120**, 109120 (2020).
21. Mnih, V. et al. Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015).
22. Kober, J., Bagnell, J. A. & Peters, J. Reinforcement learning in robotics: a survey. *Int. J. Robot. Res.* **32**, 1238–1274 (2013).
23. Giusti, A. et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robot. Autom. Lett* **1**, 661–667 (2015).
24. George Thuruthel, T., Picardi, G., Iida, F., Laschi, C. & Calisti, M. Learning to stop: a unifying principle for legged locomotion in varying environments. *R. Soc. Open Sci.* **8**, 210223 (2021).
25. Kormushev, P., Calinon, S. & Caldwell, D. G. Reinforcement learning in robotics: applications and real-world challenges. *Robotics* **2**, 122–148 (2013).
26. Lee, T., Lee, B. D. & Park, F. C. Optimal excitation trajectories for mechanical systems identification. *Automatica* **131**, 109773 (2021).
27. Mitchell, S., Potash, E., Barocas, S., D'Amour, A. & Lum, K. Algorithmic fairness: choices, assumptions, and definitions. *Annu. Rev. Stat. Appl.* **8**, 141–163 (2021).
28. Molnar, C. *Interpretable Machine Learning* (Lulu.com, 2020).
29. Covert, I., Lundberg, S. & Lee, S.-I. Explaining by removing: a unified framework for model explanation. *J. Mach. Learn. Res.* **22**, 9477–9566 (2021).
30. Hofer, S. et al. Sim2Real in robotics and automation: applications and challenges. *IEEE Trans. Autom. Sci. Eng.* **18**, 398–400 (2021).
31. Osogami, T. Robustness and risk-sensitivity in markov decision processes. *Proc. Adv. Neural Inf. Process. Syst.* **25**, 233–241 (2012).
32. Guo, Z. D. et al. BYOL-Explore: exploration by bootstrapped prediction. *Proc. Adv. Neural Inf. Process. Syst.* **35**, 31855–31870 (2022).
33. Pathak, D. et al. Curiosity-driven exploration by self-supervised prediction. In *Proc. Int. Conf. Mach. Learn.* 2778–2787 (2017).
34. Moldovan, T. M. & Abbeel, P. Safe exploration in Markov decision processes. In *Proc. Int. Conf. Mach. Learn.* 1451–1458 (2012).
35. Brunke, L. et al. Safe learning in robotics: from learning-based control to safe reinforcement learning. *Annu. Rev. Control Robot. Auton. Syst.* **5**, 411–444 (2022).
36. Omer, M., Ahmed, R., Rosman, B. & Babikir, S. F. Model predictive-actor critic reinforcement learning for dexterous manipulation. In *2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)* 1–6 (IEEE, 2021).
37. Berkenkamp, F. & Schoellig, A. P. Safe and robust learning control with gaussian processes. In *Proc. European Control Conference (ECC)* 2496–2501 (IEEE, 2015).
38. Turchetta, M., Berkenkamp, F. & Krause, A. Safe exploration in finite markov decision processes with Gaussian processes. *Proc. Adv. Neural Inf. Process. Syst.* **29**, 4312–4320 (2016).
39. Horvath, D., Erdos, G., Istenes, Z., Horvath, T. & Foldi, S. Object detection using sim2real domain randomization for robotic applications. *IEEE Trans. Robot.* https://doi.org/10.1109/TRO.2022.3207619 (2022).
40. Dulac-Arnold, G. et al. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Mach. Learn.* 1–50 (2021).
41. Chow, Y., Ghavamzadeh, M., Jason, L. & Pavone, M. Risk-constrained reinforcement learning with percentile risk criteria. *J. Mach. Learn. Res.* **18**, 6070–6120 (2017).
42. Park, K. M., Kim, J., Park, J. & Park, F. C. Learning-based real-time detection of robot collisions without joint torque sensors. *IEEE Robot. Autom. Lett.* **6**, 103–110 (2020).
43. Vaswani, A. et al. Attention is all you need. In *Proc. Advances in Neural Information Processing Systems* 5998–6008 (NIPS, 2017).
44. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. Bert: pre-training of deep bidirectional transformers for language understanding. In *Proc. 2019 Conf. North American Chap. Assoc. Comp. Ling.: Human Lang. Tech.* 4171–4186 (2019).
45. Kim, D., Park, M. & Park, Y.-L. Probabilistic modeling and bayesian filtering for improved state estimation for soft robots. *IEEE Trans. Robot.* **37**, 1728–1741 (2021).
46. Razmjooei, H. & Shafiei, M. H. A new approach to design a finite-time extended state observer: uncertain robotic manipulators application. *Int. J. Robust Nonlinear Control.* **31**, 1288–1302 (2021).
47. Choi, S., Lee, K. & Oh, S. Robust learning from demonstrations with mixed qualities using leveraged gaussian processes. *IEEE Trans. Robot.* **35**, 564–576 (2019).
48. Ravichandar, H., Polydoros, A. S., Chernova, S. & Billard, A. Recent advances in robot learning from demonstration. *Annu. Rev. Control. Robot. Auton. Syst.* **3**, 297–330 (2020).
49. Steffi, D. D., Mehta, S., Venkatesh, K. & Dasari, S. K. In *Data Science and Security* 211–219 (Springer, 2021).
50. Luong, M.-T., Pham, H. & Manning, C. D. Effective approaches to attention-based neural machine translation. In *Proc. 2015 Conf. Emp. Meth. Nat. Lang. Proc.* 1412–1421 (2015).
51. Agarap, A. F. Deep learning using rectified linear units (RELU). Preprint at https://arxiv.org/abs/1803.08375 (2018).
52. Coumans, E. & Bai, Y. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org (2016).
53. Hong, T. H., Park, S.-H., Park, J.-H., Paik, N.-J. & Park, Y.-L. Design of pneumatic origami muscle actuators (pomas) for a soft robotic hand orthosis for grasping assistance. In *Proc. IEEE International Conference on Soft Robotics* 627–632 (IEEE, 2020).
54. Kim, T. et al. Heterogeneous sensing in a multifunctional soft sensor for human-robot interfaces. *Sci. Robot.* **5**, eabc6878 (2020).
55. Haarnoja, T., Zhou, A., Abbeel, P. & Levine, S. Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proc. International Conference on Machine Learning* 1861–1870 (PMLR, 2018).
56. Lee, J., Hwangbo, J., Wellhausen, L., Koltun, V. & Hutter, M. Learning quadrupedal locomotion over challenging terrain. *Sci. Robot.* **5**, eabc5986 (2020).
57. Meyers, M. A. & Chawla, K. K. *Mechanical Behavior of Materials* (Cambridge University Press, 2008).
58. Rasmussen, C. E. Gaussian processes in machine learning. In *Summer School on Machine Learning* 63–71 (Springer, 2003).
59. Bruder, D., Fu, X., Gillespie, R. B., Remy, C. D. & Vasudevan, R. Data-driven control of soft robots using Koopman operator theory. *IEEE Trans. Robot.* **37**, 948–961 (2020).
60. Kingma, D. P. & Ba, J. Adam: a method for stochastic optimization. In *Proc. Int. Conf. Learn. Rep.* (2015).

61. UniversalRobots. UR interface. https://bitbucket.org/RopeRobotics/ur-interface/src/master/ (2021).
62. Unitree. Laikago working with ROS. https://github.com/unitreerobotics/laikago_ros (2021).
63. RoboDK. RoboDK for web. https://web.robodk.com/web?--OpenTab=t-4268 (2022).

ACKNOWLEDGEMENTS

This work was supported in part by the National Research Foundation Grant (RS-2023-00208052) and in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant (2021-0-00896) both funded by the Korean Government (MSIT).

AUTHOR CONTRIBUTIONS

D.W.K. conceived the overall algorithm, designed a deep neural network for both kinematics and dynamics model, conducted the simulation and the application experiments, performed an ablation study, and wrote the manuscript. S.L. provided a virtual environment for the PyBullet and ROS system and assisted in the experiment of the quadruped robot. T.H.H. provided a hardware environment for soft robot and assisted in the experiment of the soft robot. Y.-L.P. directed the overall research and organized and wrote the manuscript.

COMPETING INTERESTS

The authors declare no competing interests.

ADDITIONAL INFORMATION

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s44182-023-00006-5.

Correspondence and requests for materials should be addressed to Yong-Lae Park.