

Torque tracking position control of DLR-HIT II robotic hand using a real-time physics-informed neural network

Ali Al-Shahrabi^{a,b}, Masoud J. Javid^a, Ashraf A. Fahmy^a, Christian A. Griffiths^a, Chunxu Li^a,*

^a Department of Mechanical Engineering, Swansea University, Fabian way, Crymlyn Burrows, Swansea, SA1 8EN, Wales, United Kingdom

^b College of Engineering, Al-Nahrain University, Baghdad, Iraq

ARTICLE INFO

Keywords:

Machine learning in robotics
Computed torque control
Real-time applications of dexterous manipulation
Neural network-based control
DLR-HIT II hand
Normalization method

ABSTRACT

This paper presents a novel approach for controlling the DLR-HIT II robotic hand by leveraging physics-informed neural networks (PINNs) for torque and position control. This method eliminates the need for additional control inputs or external controllers, achieving high precision and simplified dynamics, which is validated through extensive simulations that closely replicate experimental conditions, demonstrating the system's ability to handle external disturbances and maintain accurate trajectory tracking. The strategy only requires time and joint position data as inputs, allowing the network to compute velocity and acceleration internally. Time normalization enhances the model's ability to generalize across different time scales and ensures stable training. The method demonstrates strong generalization from a limited training set and successfully performs across diverse trajectory types. This simplification significantly reduces computational complexity and facilitates real-time control in advanced robotic applications.

1. Introduction

The primary limitation of general Deep learning algorithms is sampling incompetence, which results from the requirement to extract every detail of a task from the dataset [1,2]. Significant findings have been accomplished across a variety of scientific disciplines, including the recognition of images [3], cognitive sciences [4], and biology [5]. These achievements can be attributed to the exponential rise of available data and computational resources. However, the cost of acquiring data becomes expensive when analysing complicated physical systems [6]. Another challenge faced by machine learning algorithms arises when a neural network is trained on a specific range of data, such as certain time periods. In such cases, the network becomes adept at identifying and simulating the dynamics and behaviours within the confines of that particular time frame [7]. Therefore, the neural network may lack the capacity to discern patterns effectively, leading to imprecise predictions when confronted with diverse data intervals, such as varying time scales and time steps [6,8].

Control systems often use discrete feedback due to digital processing, which impacts real-time applications [9]. While this study uses continuous-time analysis, adapting to discrete feedback is essential for practical use and introduces challenges like sampling delays that affect stability and tracking. Stability can be verified using discrete Lyapunov functions, as discussed in [10], and neural

* Corresponding author.

E-mail addresses: a.m.a.al-shahrabi@swansea.ac.uk (A. Al-Shahrabi), masoud.javid.j@gmail.com (M.J. Javid), a.a.fahmy@swansea.ac.uk (A.A. Fahmy), c.a.griffiths@swansea.ac.uk (C.A. Griffiths), chunxu.li@swansea.ac.uk (C. Li).

<https://doi.org/10.1016/j.apm.2025.116110>

Received 14 September 2024; Received in revised form 17 March 2025; Accepted 18 March 2025

Available online 22 March 2025

0307-904X/© 2025 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

network performance may require adjustments for discrete implementation, as shown in [11]. Discrete feedback also poses computational trade-offs; shorter sampling intervals improve accuracy but increase processing demands, while longer intervals reduce precision. Insights from [12] suggest balancing these trade-offs for real-time use. Future work should include discrete-time stability verification and retraining the PINN for sampled data to ensure robust, real-world performance.

In the field of robotics, using low-dimension inputs and limited features in Deep Learning algorithms are challenging. Most machine learning methods demand multiple inputs to capture a complex system such as the dynamic of the robots as the robotic dynamic equation is a non-convex function [13]. Specifically, the torque control requires high dimensional inputs since the torque depends on multiple parameters. Therefore, fewer inputs might not be enough to interpret the non-linearity of the dynamics system's behaviour [14]. Extrapolating deep learning algorithms to unseen data poses one of the most challenging obstacles within the robotic community [15], Achieving generalization in a neural network to predict a robot's trajectory across multiple types of trajectories demands a large and diverse dataset. It necessitates numerous scenarios for each trajectory type to enable the neural network to discern patterns effectively and produce accurate predictions.

Most contemporary machine learning approaches such as deep recurrent, and convolutional neural networks are not robust and are unable to ensure convergence over a small range of datasets [16]. As a result, there are an increasing tendency to improve the effectiveness of robot learning by explicitly incorporating physical laws into data-driven methods to solve these challenges. PINNs integrate physics equations into the neural network's structure [17]. Hence, they are especially beneficial in situations where the data is inadequate. Some research endeavors utilize (PINNs) to model manipulator dynamics, enhancing these networks with innovative methodologies and techniques. For instance, one approach involves the integration of a hybrid-PINN, combining a recurrent neural network with Runge-Kutta cells [18]. Another avenue explores the fusion of traditional PINNs with memetic nonlinear transform layers, utilizing liquid layers to create an equation-embedded neural network [19]. On a different note, additional research endeavors focus on utilizing PINNs for robot control. This involves integrating PINNs with other processes, such as merging them with non-linear model predictive control [20]. Additionally, researchers explore the integration of multiple neural networks to address dynamic equation terms. Subsequently, these terms are utilized within the PINNs framework to derive Lagrange or Hamilton equations, providing a comprehensive solution for effective robot control [21]. However, these works still have not been addressed sufficient solutions for the all mentioned challenges. Moreover implementing PINNs with multiple processes to enhance their performance can be computationally expensive. The presented work in this paper introduces a novel solution to overcome the mentioned barriers. Here, the dynamic equation serves as a foundation for implementing torque and position controls, employing PINNs model without the necessity of additional control inputs. Importantly, the integration of PINNs with other controllers is deemed superfluous for task accomplishment. In addition, minimum variables time, desired positions, and actual position are used as inputs for PINNs, harnessing the ability of the PINNs to compute the derivatives of actual and desired positions instead of feeding them as an input. Furthermore, the time is normalized to be in range [0 1] second for any time input for the neural network in order to generalize the PINNs for different time steps and time periods as well as make the training process more stable and faster. Despite being trained on a relatively modest dataset of 13,314 samples and exclusively utilizing a fifth-degree polynomial trajectory during training, the PINN demonstrate impressive generalization capabilities across multiple trajectory types. Notably, the model exhibits accurate predictions for the specified trajectory and has undergone rigorous testing on various trajectories, including third-degree polynomials, seventh-degree polynomials, third-degree splines, fifth-degree splines, seventh-degree splines, and sinusoidal trajectories. To validate its performance, Mean Absolute Position Error (MAPE), Mean Absolute Torque Error (MATE), Maximum Absolute Position Error (MaxAPE), and Maximum Absolute Torque Error (MaxATE) were computed to compare the developed PINN with PD computed torque control across these various trajectories. Although this study focuses on simulations conducted in MATLAB/Simulink, these simulations are designed to mimic experimental conditions closely, including the introduction of external disturbances to evaluate the controller's robustness. The results provide a strong foundation for future physical implementation and testing.

In the following sections, Modelling of the DLR-HIT II hand is provided in section 2. After that, section 3 demonstrates the proposed work and section 4 illustrates the experimental studies. Finally, the main conclusions of the paper are presented in Section 5.

The novelty of the paper can be summarized as follows:

- A new control strategy using physics-informed neural networks for robotic hand control, eliminating external control inputs.
- Internal computation of velocity and acceleration by the network, reducing the need for high-dimensional input data.
- Enhanced generalization capability from a limited training dataset across multiple trajectory types.
- Simplified and efficient handling of dynamic equations for multi-joint robotic systems.
- Effective performance under simulated external disturbances, showing robustness.

2. Modelling the DLR-HIT II hand

This section will cover the steps involved in building a robotic hand kinematic model along with its dynamic equation. As Fig. 1 illustrates the DLR-HIT II hand has a total of fifteen degrees of freedom (DOFs). Every finger is built symmetrically with four joints and three DOFs, ensuring a high degree of flexibility. A coupling mechanism that maintains constant angles is used to mechanically connect the final two joints of each finger via metallic cables [22]. Table 1 displays the joint constraints for each finger [22].

In order to simplify the forward kinematic, Denavit-Hartenberg method (DH) and composite method are used. The DH model makes use of the system's geometrical structure to express the correlation between the joints and links of the robotic hand by utilising the geometrical structure of the system [24]. This method determines the coordinate on every robot's link. The relationship between two neighbouring links is produced by the homogeneous transformation matrix. The robot's kinematic solution can be calculated

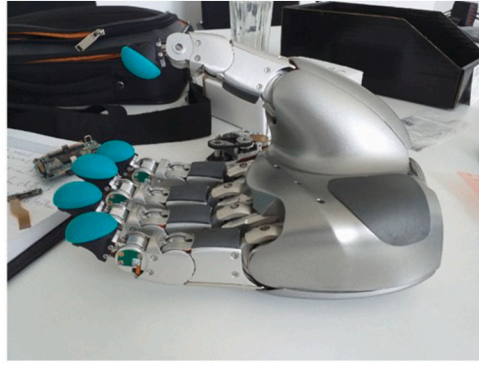


Fig. 1. The DLR-HIT II hand [23].

Table 1
Joint constraints and DH parameters for one finger. Unspecified values in DH parameters are zero.

Joint Constraints		DH Parameters				
Joint Angle	Constraint (rad)	Frames	α_i	a_i	θ_i	d_i
θ_1	$[-\frac{\pi}{9}, \frac{\pi}{9}]$	1			θ_1	
θ_2	$[0, \frac{\pi}{2}]$	2			θ_2	
θ_3	$[0, \frac{\pi}{2}]$	3		L_1	θ_3	
$\theta_4 = \theta_3$	$[0, \frac{\pi}{2}]$	4		L_2	$\theta_4 = \theta_3$	
		5		L_3		

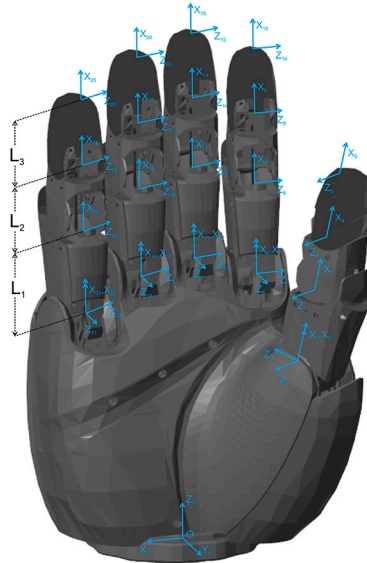


Fig. 2. The coordinate system for the DLR-HIT II hand.

through implementing the transformation sequence to obtain the end-effector’s pose in regards to the base frame [25]. In this work, the fixed robot frame is assumed to align with the first frame, which is attached to the first joint of each finger. Thus, the same DH parameter table can be used for all fingers as shown in Table 1. Knowing that all the fingers of DLR-HIT II hand have the same link’s length (L_1, L_2, L_3). By doing so, the dynamic equation will be simplified, as detailed later in Section 3 A.

Fig. 2 shows the coordinate system for the DLR-HIT II hand. As the palm contains no motors or joints, only the fingers need to be considered in its structure. The length of a link between two adjacent finger’s joints is expressed by L_i , and the fixed robotic frame coordinate frame is represented by the letter “o”. Finding the DH parameters for each finger makes it simple to build the DLR-HIT II hand framework. This enables a comprehensive comprehension of the kinematic structure of the hand and promotes efficient execution. By utilizing the general DH transformation matrix (1), the transformation matrix for each frame is derived.

$$\mathbf{T}_i^{i-1} = \begin{bmatrix} \mathbf{R}_i^{i-1} & \mathbf{P}_i^{i-1} \\ \hline \cos \theta_i & -\sin \theta_i & 0 & a_{i-1} \\ \sin \theta_i \cos \alpha_{i-1} & \cos \theta_i \cos \alpha_{i-1} & -\sin \alpha_{i-1} & -d_i \sin \alpha_{i-1} \\ \sin \theta_i \sin \alpha_{i-1} & \cos \theta_i \sin \alpha_{i-1} & \cos \alpha_{i-1} & d_i \cos \alpha_{i-1} \\ \hline 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

hence, $\mathbf{T}_i^{i-1} \in \mathbb{R}^{4 \times 4}$ is the homogeneous transformation matrix between two frames, the translation between two frames is denoted by $\mathbf{P}_i^{i-1} \in \mathbb{R}^{3 \times 1}$, while the rotational matrix is represented by $\mathbf{R}_i^{i-1} \in \mathbb{R}^{3 \times 3}$. Hence, the following equation can be used to multiply the calculated transformation matrix for each frame in order to determine the finger pose's tip with respect to its first joint:

$$\mathbf{T}_5^1 = \mathbf{T}_1^1 \mathbf{T}_2^1 \mathbf{T}_3^1 \mathbf{T}_4^1 \mathbf{T}_5^1 \quad (2)$$

\mathbf{T}_5^1 is the transformation matrix between the tip of the finger with respect to first joint, subscript ($r = 1, 2, \dots, 5$) indicates which finger the rotation matrix of the first joint belongs to, where $r=1$ represents the thumb, $r=2$ the index finger, $r=3$ the middle finger, $r=4$ the ring finger, and $r=5$ the pinky finger. Whereas, the composite method [26] is utilised to find the relationship between fixed robot frame and the first joint of each finger. Then, the forward kinematic of each finger can be obtained through multiplying the transformation matrix \mathbf{T}_5^1 with the transformation matrix that describes the pose of the first joint of the corresponding finger relative to the fixed robot frame as follows:

$$\mathbf{T}_5^0 = \mathbf{T}_1^0 \mathbf{T}_5^1 \quad (3)$$

where \mathbf{T}_1^0 is the transformation matrix between fixed robot frame and the first joint, which it will be different for each finger depending on its position and orientation to the fixed robot frame. The general transformation matrix between fixed robot frame and first joint of each finger can be stated as follows:

$$\mathbf{T}_{1_r}^0 = \begin{bmatrix} R_{11_r} & R_{12_r} & R_{13_r} & P_{x_r} \\ R_{21_r} & R_{22_r} & R_{23_r} & P_{y_r} \\ R_{31_r} & R_{32_r} & R_{33_r} & P_{z_r} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4)$$

The order of operation in composite method matters. For instance, the transformation matrix between fixed robot frame and first joint of the thumb finger can be as follows:

$$\mathbf{T}_{1_1}^0 = ROT(Z, \alpha_1) TRANS(X, Y, Z, P_{x_1}, P_{y_1}, P_{z_1}) ROT(Y, \beta_1) ROT(X, \phi_1) \quad (5)$$

$ROT(Z, \alpha_1)$ denotes the rotation around Z-axis with angle α_1 , $TRANS(X, Y, Z, P_{x_1}, P_{y_1}, P_{z_1})$ presents the translation among X-axis, Y-axis and Z-axis with P_{x_1} , P_{y_1} , and P_{z_1} respectively, where $ROT(Y, \beta_1)$ and $ROT(X, \phi_1)$ are rotation around Y-axis and Z-axis with angles β_1 and ϕ_1 correspondingly. Since the dynamics of each finger can differ due to variations in transformation matrix $\mathbf{T}_{1_r}^0$ between the fixed robot frame and the first joint of each finger, the dynamic equation will be derived separately for each finger rather than for the entire hand collectively. This finger-by-finger approach allows for a more precise and modular dynamic analysis. The dynamic equation for each finger of the DLR-HIT II hand will be calculated using the general Lagrange equation:

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{\mathbf{q}}} \right) - \frac{\partial L}{\partial \mathbf{q}} = \boldsymbol{\tau} \quad (6)$$

where;

$$\mathbf{q} = \begin{bmatrix} q_1 \\ \vdots \\ q_n \end{bmatrix}, \quad \dot{\mathbf{q}} = \begin{bmatrix} \dot{q}_1 \\ \vdots \\ \dot{q}_n \end{bmatrix} \quad (7)$$

$\mathbf{q} \in \mathbb{R}^{n \times 1}$ is the vector of joints' angles, n is the number of joints ($n=4$ as each finger has four joints). Here $q_1 = \theta_1$, $q_2 = \theta_2$, and $q_4 = q_3 = \theta_3$ (because of the mechanical coupling of the last two joint in each finger). $\dot{\mathbf{q}} \in \mathbb{R}^{n \times 1}$ is the vector of joints' angular velocities, $\boldsymbol{\tau} \in \mathbb{R}^{n \times 1}$ is the torque vector, and $L = K - U$, where K is the system's kinetic energy for one finger and U is its potential energy. Since \mathbf{q} is the only factor influencing the potential energy U , consequently, the Lagrange equation will be as follows:

$$\frac{d}{dt} \left(\frac{\partial K}{\partial \dot{\mathbf{q}}} \right) - \frac{\partial K}{\partial \mathbf{q}} + \frac{\partial U}{\partial \mathbf{q}} = \boldsymbol{\tau} \quad (8)$$

On the other hand, the total kinetic energy for one finger can be expressed as follows:

$$K = \sum_{j=1}^3 K_j = \frac{1}{2} \dot{\mathbf{q}}^T \mathbf{M}(\mathbf{q}) \dot{\mathbf{q}} \quad (9)$$

$\mathbf{M}(\mathbf{q}) \in \mathbb{R}^{n \times n}$ represents the inertia matrix for one finger, the index ($j = 1, 2, 3$; as each finger has three links) refers to each individual link within the system. Each term K_j represents the kinetic energy contribution of the j -th link. The partial derivative of the kinetic energy with respect to the joint velocities $\dot{\mathbf{q}}$ represents the generalized momentum associated with each joint's velocity:

$$\frac{\partial K}{\partial \dot{\mathbf{q}}} = \frac{\partial}{\partial \dot{\mathbf{q}}} \left[\frac{1}{2} \dot{\mathbf{q}}^T \mathbf{M}(\mathbf{q}) \dot{\mathbf{q}} \right] = \mathbf{M}(\mathbf{q}) \dot{\mathbf{q}} \tag{10}$$

The time derivative of the partial derivative of kinetic energy with respect to $\dot{\mathbf{q}}$ describes how the generalized momentum changes over time. This term incorporates both the inertia matrix and its time derivative:

$$\frac{d}{dt} \left(\frac{\partial K}{\partial \dot{\mathbf{q}}} \right) = \frac{d}{dt} (\mathbf{M}(\mathbf{q}) \dot{\mathbf{q}}) = \mathbf{M}(\mathbf{q}) \ddot{\mathbf{q}} + \dot{\mathbf{M}}(\mathbf{q}) \dot{\mathbf{q}} \tag{11}$$

$\ddot{\mathbf{q}} \in \mathbb{R}^{n \times 1}$ is the vector of joints' accelerations. Finally, the partial derivative of the kinetic energy with respect to the joint positions \mathbf{q} captures how the kinetic energy varies with changes in the joint positions:

$$\frac{\partial K}{\partial \mathbf{q}} = \frac{1}{2} \begin{bmatrix} \dot{\mathbf{q}}^T \frac{\partial \mathbf{M}(\mathbf{q})}{\partial q_1} \dot{\mathbf{q}} \\ \vdots \\ \dot{\mathbf{q}}^T \frac{\partial \mathbf{M}(\mathbf{q})}{\partial q_n} \dot{\mathbf{q}} \end{bmatrix} \tag{12}$$

Thereafter, by substituting (10), (11), and (12) in (8):

$$\mathbf{M}(\mathbf{q}) \ddot{\mathbf{q}} + \dot{\mathbf{M}}(\mathbf{q}) \dot{\mathbf{q}} - \frac{1}{2} \begin{bmatrix} \dot{\mathbf{q}}^T \frac{\partial \mathbf{M}(\mathbf{q})}{\partial q_1} \dot{\mathbf{q}} \\ \vdots \\ \dot{\mathbf{q}}^T \frac{\partial \mathbf{M}(\mathbf{q})}{\partial q_n} \dot{\mathbf{q}} \end{bmatrix} = \boldsymbol{\tau} - \mathbf{G}(\mathbf{q}); \quad \mathbf{G}(\mathbf{q}) = \frac{\partial U}{\partial \mathbf{q}} \tag{13}$$

$\mathbf{G}(\mathbf{q}) \in \mathbb{R}^{n \times 1}$ is the gravity vector. That leads to the following equation:

$$\mathbf{M}(\mathbf{q}) \ddot{\mathbf{q}} + \mathbf{H}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \boldsymbol{\tau} \tag{14}$$

where:

$$\mathbf{H}(\mathbf{q}, \dot{\mathbf{q}}) = \dot{\mathbf{M}}(\mathbf{q}) \dot{\mathbf{q}} - \frac{1}{2} \begin{bmatrix} \dot{\mathbf{q}}^T \frac{\partial \mathbf{M}(\mathbf{q})}{\partial q_1} \dot{\mathbf{q}} \\ \vdots \\ \dot{\mathbf{q}}^T \frac{\partial \mathbf{M}(\mathbf{q})}{\partial q_n} \dot{\mathbf{q}} \end{bmatrix} \tag{15}$$

$\mathbf{H}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^{n \times 1}$ is the Coriolis and Centrifugal vector. For every finger's j -link, the kinetic energy K_j is:

$$K_j = \frac{1}{2} (\mathbf{v}_{c_j}^T m_j \mathbf{v}_{c_j} + \boldsymbol{\omega}_j^T I_{c_j} \boldsymbol{\omega}_j) \tag{16}$$

the mass of the j -th link is m_j , and its centre's linear and angular velocities are $\mathbf{v}_j \in \mathbb{R}^{3 \times 1}$ and $\boldsymbol{\omega}_j \in \mathbb{R}^{3 \times 1}$, respectively, and I_{c_j} denotes the mass centre of the j -link's inertia in the direction of the rotation axis. Thereafter the total kinetic energy for one finger is computed to find its inertia matrix $\mathbf{M}(\mathbf{q})$, by substituting (16) in (9):

$$\sum_{j=1}^3 \frac{1}{2} (\mathbf{v}_{c_j}^T m_j \mathbf{v}_{c_j} + \boldsymbol{\omega}_j^T I_{c_j} \boldsymbol{\omega}_j) = \frac{1}{2} \dot{\mathbf{q}}^T \mathbf{M}(\mathbf{q}) \dot{\mathbf{q}} \tag{17}$$

Since

$$\mathbf{v}_{c_j} = \mathbf{J}_{v_{c_j}} \dot{\mathbf{q}}, \quad \boldsymbol{\omega}_j = \mathbf{J}_{\omega_j} \dot{\mathbf{q}} \tag{18}$$

$\mathbf{J}_{v_{c_j}} \in \mathbb{R}^{3 \times n}$ is the linear velocity Jacobin matrix of the mass centre of the finger's j -link, whereas $\mathbf{J}_{\omega_j} \in \mathbb{R}^{3 \times n}$ is the angular velocity Jacobin matrix of the finger's j -link. Consequently, substituting (18) into (17) yields the following:

$$\frac{1}{2} \dot{\mathbf{q}}^T \left[\sum_{j=1}^3 (\mathbf{J}_{v_{c_j}}^T m_j \mathbf{J}_{v_{c_j}} + \mathbf{J}_{\omega_j}^T I_{c_j} \mathbf{J}_{\omega_j}) \right] \dot{\mathbf{q}} = \frac{1}{2} \dot{\mathbf{q}}^T \mathbf{M}(\mathbf{q}) \dot{\mathbf{q}} \tag{19}$$

then

$$\mathbf{M}(\mathbf{q}) = \left[\sum_{j=1}^3 (\mathbf{J}_{v_{c_j}}^T m_j \mathbf{J}_{v_{c_j}} + \mathbf{J}_{\omega_j}^T I_{c_j} \mathbf{J}_{\omega_j}) \right] \tag{20}$$

After calculating inertia matrix $\mathbf{M}(\mathbf{q})$, $\mathbf{H}(\mathbf{q}, \dot{\mathbf{q}})$ can be obtained using equation (15). The general potential energy can be calculated as follows:

$$U_j = -m_j g h_j. \tag{21}$$

h_j is the height of the centre of mass of the link relative to a reference point, typically the ground, g represents the acceleration due to gravity. In order to calculate the total potential energy for a finger with m links, such as in the case of the DLR-HIT II robotic hand (which has 3 links per finger), the total potential energy for one finger can be computed as follows:

$$U = \sum_{j=1}^3 U_j = \sum_{j=1}^3 -m_j \left[\mathbf{p}_{c_j}^T \mathbf{g} \right], \quad (22)$$

the $\mathbf{p}_{c_j} \in \mathbb{R}^{3 \times 1}$ expresses a point located at the centre of the mass distribution on the j -th link of the finger, $\mathbf{g} \in \mathbb{R}^{3 \times 1}$ ($\mathbf{g} = [0 \ 0 \ g]^T$) is the vector of gravity. h_j is substituted with the term $\mathbf{p}_{c_j}^T \mathbf{g}$ since the height is in the direction of Z-axis of fixed robot frame as shown in Fig. 2. The gravity term $\mathbf{G}(\mathbf{q})$ of one finger can be computed by finding the derivative of the potential energy with respect to joint variables \mathbf{q} :

$$\mathbf{G}(\mathbf{q}) = \frac{\partial U}{\partial \mathbf{q}} = - \sum_{j=1}^3 \left[\frac{\partial \mathbf{p}_{c_j}}{\partial \mathbf{q}} \right]^T m_j \mathbf{g} \quad (23)$$

here, the term $\frac{\partial \mathbf{p}_{c_j}}{\partial \mathbf{q}}$ represents the Jacobian matrix that describes how the position of the j -th link's centre of mass of the finger changes with respect to the joint variables. This term is the linear velocity Jacobian matrix of the centre of mass position with respect to the joint variables:

$$\frac{\partial \mathbf{p}_{c_j}}{\partial \mathbf{q}} = \mathbf{J}_{v_{c_j}} \quad (24)$$

Thus, substituting (23) into (22) yields the gravitational force vector $\mathbf{G}(\mathbf{q})$ as:

$$\mathbf{G}(\mathbf{q}) = - \begin{bmatrix} \mathbf{J}_{v_{c_1}}^T & \mathbf{J}_{v_{c_2}}^T & \dots & \mathbf{J}_{v_{c_m}}^T \end{bmatrix} \begin{bmatrix} m_1 \mathbf{g} \\ m_2 \mathbf{g} \\ \vdots \\ m_m \mathbf{g} \end{bmatrix}, \quad (25)$$

$$\mathbf{G}(\mathbf{q}) = - \left[\mathbf{J}_{v_{c_1}}^T m_1 \mathbf{g} + \mathbf{J}_{v_{c_2}}^T m_2 \mathbf{g} + \dots + \mathbf{J}_{v_{c_m}}^T m_m \mathbf{g} \right],$$

Since the last two joints are coupled, the DLR-HIT II hand's finger has three actuators despite having four joints; meaning, each finger of the robotic hand has three torques. Therefore, the dynamic equation (14) should be represented in matrix form to account for the four torques acting on the finger, as follows:

$$\begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_{3'} \\ \tau_{4'} \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31'} & m_{32'} & m_{33'} & m_{34'} \\ m_{41'} & m_{42'} & m_{43'} & m_{44'} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \\ \ddot{q}_3 \\ \ddot{q}_4 \end{bmatrix} + \begin{bmatrix} H_1 \\ H_2 \\ H_{3'} \\ H_{4'} \end{bmatrix} + \begin{bmatrix} G_1 \\ G_2 \\ G_{3'} \\ G_{4'} \end{bmatrix} \quad (26)$$

where the prime (') notation indicates terms associated with the coupling effects. The actual torque for the third actuator is then calculated by adding $\tau_{3'}$ and $\tau_{4'}$, since the third and fourth joints are driven by the same motor ($q_3 = q_4$). Hence, $\tau_{3'} + \tau_{4'}$ can be expressed as τ_3 , leading to the following equation:

$$\begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \\ \ddot{q}_3 \end{bmatrix} + \begin{bmatrix} H_1 \\ H_2 \\ H_3 \end{bmatrix} + \begin{bmatrix} G_1 \\ G_2 \\ G_3 \end{bmatrix} \quad (27)$$

where the new values for the third torque term τ_3 are derived as follows:

$$\begin{aligned} m_{31} &= m_{31'} + m_{41'}, \\ m_{32} &= m_{32'} + m_{42'}, \\ m_{33} &= m_{33'} + m_{43'}, \\ m_{34} &= m_{34'} + m_{44'}, \\ H_3 &= H_{3'} + H_{4'}, \\ G_3 &= G_{3'} + G_{4'}. \end{aligned} \quad (28)$$

3. The methodology

3.1. Simplification of the dynamic equation

In this subsection, the dynamic equation will be simplified using a novel mathematical method. As stated in section 2, the fixed robot frame is considered to be aligned with the first joint frame of each finger. Then, its assumed that the transformation matrix $\mathbf{T}_{1_r}^0$ has only translation without any orientation:

$$\mathbf{T}_{1_r}^0 = \begin{bmatrix} 1 & 0 & 0 & P_{x_r} \\ 0 & 1 & 0 & P_{y_r} \\ 0 & 0 & 1 & P_{z_r} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (29)$$

Then, the new translation for \mathbf{T}_3^0 of the mass centre for any finger's first link will be:

$$\mathbf{T}_3^0 = \mathbf{T}_{1_r}^0 \mathbf{T}_3^1 = \begin{bmatrix} 1 & 0 & 0 & X_{c_1} + P_{x_r} \\ 0 & 1 & 0 & Y_{c_1} + P_{y_r} \\ 0 & 0 & 1 & Z_{c_1} + P_{z_r} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (30)$$

where $(X_{c_1}, Y_{c_1}, Z_{c_1})$ is the location of mass centre for first link of any finger. Next, the linear velocity Jacobin matrix for the mass centre of the first link can be determined as follows:

$$\mathbf{J}_{v_{c_1}} = \begin{bmatrix} \frac{\partial(X_{c_1} + P_{x_r})}{\partial q_1} & \frac{\partial(X_{c_1} + P_{x_r})}{\partial q_2} & \frac{\partial(X_{c_1} + P_{x_r})}{\partial q_3} & \frac{\partial(X_{c_1} + P_{x_r})}{\partial q_4} \\ \frac{\partial(Y_{c_1} + P_{y_r})}{\partial q_1} & \frac{\partial(Y_{c_1} + P_{y_r})}{\partial q_2} & \frac{\partial(Y_{c_1} + P_{y_r})}{\partial q_3} & \frac{\partial(Y_{c_1} + P_{y_r})}{\partial q_4} \\ \frac{\partial(Z_{c_1} + P_{z_r})}{\partial q_1} & \frac{\partial(Z_{c_1} + P_{z_r})}{\partial q_2} & \frac{\partial(Z_{c_1} + P_{z_r})}{\partial q_3} & \frac{\partial(Z_{c_1} + P_{z_r})}{\partial q_4} \end{bmatrix} \quad (31)$$

As long as P_{x_r} , P_{y_r} , and P_{z_r} are constant value, then their derivatives will be zero. Thus the linear velocity Jacobian matrix states as follows:

$$\mathbf{J}_{v_{c_1}} = \begin{bmatrix} \frac{\partial X_{c_1}}{\partial q_1} & \frac{\partial X_{c_1}}{\partial q_2} & \frac{\partial X_{c_1}}{\partial q_3} & \frac{\partial X_{c_1}}{\partial q_4} \\ \frac{\partial Y_{c_1}}{\partial q_1} & \frac{\partial Y_{c_1}}{\partial q_2} & \frac{\partial Y_{c_1}}{\partial q_3} & \frac{\partial Y_{c_1}}{\partial q_4} \\ \frac{\partial Z_{c_1}}{\partial q_1} & \frac{\partial Z_{c_1}}{\partial q_2} & \frac{\partial Z_{c_1}}{\partial q_3} & \frac{\partial Z_{c_1}}{\partial q_4} \end{bmatrix} \quad (32)$$

Subsequently, by multiplying the orientation matrix $\mathbf{R}_{1_r}^0 \in \mathbb{R}^{3 \times 3}$ from the fixed robot frame to the first joint by the linear velocity Jacobian matrix from first joint to the mass centre of the first link $\mathbf{J}_{v_{c_1}} \in \mathbb{R}^{3 \times n}$, we obtain:

$$\mathbf{J}_{v_{c_1_{new}}} = \mathbf{R}_{1_r}^0 \mathbf{J}_{v_{c_1}} \quad (33)$$

where $\mathbf{J}_{v_{c_1_{new}}} \in \mathbb{R}^{3 \times n}$ is the linear Jacobian matrix from the fixed robot frame to the mass centre of first link. And same goes for the angular velocity Jacobian $\mathbf{J}_{\omega_1} \in \mathbb{R}^{3 \times n}$:

$$\mathbf{J}_{\omega_{1_{new}}} = \mathbf{R}_{1_r}^0 \mathbf{J}_{\omega_{c_1}} \quad (34)$$

hence, $\mathbf{J}_{\omega_{c_1_{new}}} \in \mathbb{R}^{3 \times n}$ is the angular velocity Jacobian. Same process will be implemented for each link. In order to find the simplified \mathbf{M} inertia matrix, the new Jacobians will be substitution in equation (20):

$$\mathbf{M}(\mathbf{q}) = \left[\sum_{j=1}^3 (\mathbf{J}_{v_{c_j}}^T \mathbf{R}_{1_r}^{0T} m_j \mathbf{R}_{1_r}^0 \mathbf{J}_{v_{c_j}} + \mathbf{J}_{\omega_j}^T \mathbf{R}_{1_r}^{0T} I_{c_j} \mathbf{R}_{1_r}^0 \mathbf{J}_{\omega_j}) \right] \quad (35)$$

The mass m_j and the inertia I_{c_j} are scalar values, as the parallel axis theorem is applied to calculate the inertia solely around the z-axis (the axis of rotation). This ensures that the multiplication of matrices is unaffected by those values. One of the properties of a rotation matrix is that when you multiply it by its transpose, the result is the identity matrix. Thus, the inertia matrix \mathbf{M} can be expressed as follows:

$$\mathbf{M}(\mathbf{q}) = \left[\sum_{j=1}^3 (\mathbf{J}_{v_{c_j}}^T m_j \mathbf{J}_{v_{c_j}} + \mathbf{J}_{\omega_j}^T I_{c_j} \mathbf{J}_{\omega_j}) \right] \quad (36)$$

The Centrifugal and Coriolis matrix $\mathbf{H}(\mathbf{q}, \dot{\mathbf{q}})$ is depending on the inertia matrix as stated in equation (15), whereas, the gravity term $\mathbf{G}(\mathbf{q})$ can be computed as follows:

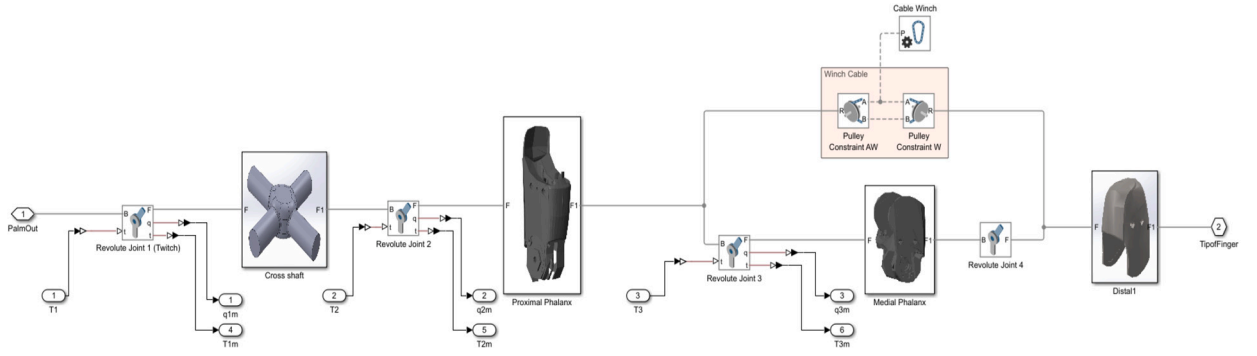


Fig. 3. The structure of DLR-HIT II finger model in Simulink.

Table 2
DLR-HIT II robotic hand dimensions and weights.

Parameters	Length (mm)	Width (mm)	Weight (g)
Distal Phalanx	25.0	19.2	24.0
Medial Phalanx	25.0	20.0	25.0
Proximal Phalanx	55.0	20.0	61.0
Hand Palm	140.4	126.2	-

$$\mathbf{G}(\mathbf{q}) = - \left[\mathbf{J}_{v_{c1}}^T m_1 + \mathbf{J}_{v_{c2}}^T m_2 + \dots + \mathbf{J}_{v_{cm}}^T m_m \right] \mathbf{A}_r \quad (37)$$

where

$$\mathbf{A}_r = \mathbf{R}_{1_r}^{0T} \mathbf{g} = \begin{bmatrix} R_{11_r} & R_{21_r} & R_{31_r} \\ R_{12_r} & R_{22_r} & R_{32_r} \\ R_{13_r} & R_{23_r} & R_{33_r} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} = \begin{bmatrix} g R_{31_r} \\ g R_{32_r} \\ g R_{33_r} \end{bmatrix} \quad (38)$$

The r here is only two values 1 and 2 because the x-axis of index, middle, ring, and pinky fingers align with z-axis of fixed robot frame, which means the last row of rotation matrix is the same for all the mentioned fingers, consequently, all the fingers of the DLR-HIT II hand excluding the thumb finger has the same dynamic. Therefore, $r = 1$ is for thumb finger and $r = 2$ is for all other fingers. This method is highly beneficial for simplifying the dynamics of complex systems, such as parallel robots and multi-fingered robotic hands. By reducing the computational complexity, it allows for more efficient and accurate modelling of these intricate systems. This simplification is crucial for real-time control and simulation, enabling the implementation of advanced control algorithms and improving the overall performance of robotic systems. Additionally, it facilitates the design and optimization processes, making it easier to analyze and fine-tune the mechanical and control aspects of complex robotic structures.

3.2. Simulation of the DLR-HIT II hand in MATLAB

In robotics, simulation is an essential tool for system evaluation to reduce the risks involved in testing and deployment in the real world. It makes it easier to simulate intricate situations, making it possible to optimise designs and experiment successfully and economically with various control methods. Additionally, simulations speed up development and improve the resilience and dependability of robotic solutions by offering crucial insights into system performance under various circumstances. MATLAB is a powerful platform widely used in robotics for its robust capabilities in algorithm development, data analysis, and simulation which is employed as an evaluating platform. It offers a powerful toolkit for simulating dynamic environments, creating and executing complex control systems, and optimising robotic algorithms. Accurate simulations and analyses are essential for assessing the operation of robotic systems under varied settings, and MATLAB's numerical computation precision guarantees them. MATLAB's integrated support for Simulink makes it easy to simulate robotic models. This allows for design testing and refinement before real-world deployment, which improves the reliability and efficacy of robotic solutions. Therefore, for a more comprehensive evaluation, we utilize the CAD files of the DLR-HIT II hand, which are sourced from a GitHub repository, ensuring detailed and accurate modelling of the robotic hand [27]. The DLR-HIT II robotic hand physical characteristics are shown in Table 2 and some other specifications of the robotic hand are mentioned in section 2. The work in [22] declares that each finger weight is 220 g including motors, electronic boards, and other components, and the whole robotic hand weight is 1.5 Kg. Simulink and specifically the Simscape Multibody library can provide robust robotics simulation capabilities that make it possible to precisely model, visualise, and analyse intricate mechanical systems and their dynamic interactions inside of a virtual environment. To simulate each finger of the DLR-HIT II robotic hand in MATLAB/Simulink, the Simscape Multibody library of Simulink is employed for modelling the physical model of the whole hand as shown in Fig. 2. The DLR HIT II has an intricate mechanical construction whose capabilities and proficiency are intended to be similar to those of the human hand [28]. As shown in Fig. 2, each finger can rotate around two axes from its base, which requires a

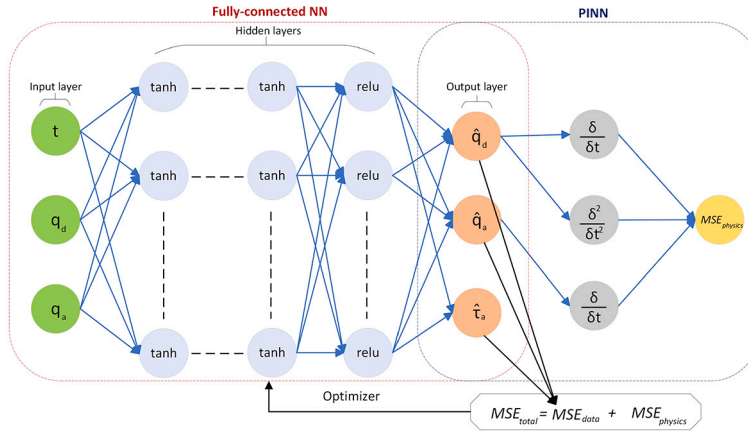


Fig. 4. The architecture of the PINN with 9 inputs (one input for time t , 3 inputs for the desired position \mathbf{q}_d , and 3 inputs for the feedback \mathbf{q}_a), 4 hidden layers (the first three hidden layers have 64 neurons for each with tanh activation function and the last hidden layer has 80 neurons with relu activation function), and 9 neurons at the output (3 outputs for the predicted desired positions $\hat{\mathbf{q}}_d$, 3 outputs for the prediction of tracking the positions $\hat{\mathbf{q}}_a$, and the last three outputs are the predicted actual torque $\hat{\boldsymbol{\tau}}_a$).

universal joint to be modelled. There is a universal joint block provided in the Simscape library, but this block is designed to rotate around the X and Y axes which is not suitable for our model as rotation around the Z axis is required. By using a cross shaft between two axes with similar weight and dimension, this 2 DOF joint is identical to the actual DLR-HIT II robotic hand. As mentioned in section 2, the other complexity in each finger belongs to the final two joints which coupled together using a metallic cable. A cable and pulley constraints mechanism considered for the final joints, so actuation is only applied to the third joint and the final joint will rotate using the torque applied to the third joint. Fig. 3 represents the structure of each finger of DLR-HIT II robotic hand made in Simulink. The damping coefficients for the joints are set as follow: 0.011 N·m·s/rad for the first joint, 0.02 N·m·s/rad for the second, and 0.0145 N·m·s/rad for the third. These values are selected to optimize the performance of the PINNs, which will be discussed in detail in the following section.

3.3. Employing the PINN to implement PD torque control

In this section, two PINNs are employed to solve the dynamic equation of all the fingers and to control the robotic hand. One is for all the four fingers excluding the thumb of the DLR-HIT II hand since they share the same dynamics, and one for the thumb finger. The architecture of the developed PINN is shown in Fig. 4, the input of the PINN is the time, the desired joints positions, and the actual joints positions (t , \mathbf{q}_d , and \mathbf{q}_a) $\in \mathbb{R}^{7 \times 1}$, while the outputs are the desired and actual positions of the finger joints, and the torques ($\hat{\mathbf{q}}_d$, $\hat{\mathbf{q}}_a$, and $\hat{\boldsymbol{\tau}}_a$) $\in \mathbb{R}^{9 \times 1}$. Four hidden fully-connected layers are utilized, the first three hidden layers have 64 neurons for each layer with hyperbolic tangent activation function (tanh) while the last hidden layer has 80 neurons and its activation function is rectified linear unit (relu), it is trained over 2500 epochs. All weights and biases employ the regularization method L2 [29] to prevent data overfitting. In order to avoid the limitation of multi-dimensionality in PINNs, each output has its own physics loss equation. The foremost three outputs values of PINN are assumed as follows:

$$\begin{aligned} \hat{y}_1(i) &= \hat{q}_{1_d}(i), \\ \hat{y}_2(i) &= \hat{q}_{2_d}(i), \\ \hat{y}_3(i) &= \hat{q}_{3_d}(i). \end{aligned} \quad (39)$$

$\hat{q}_{1_d}(i)$, $\hat{q}_{2_d}(i)$, and $\hat{q}_{3_d}(i)$ are the predicted desired position, $i = 1, \dots, B_f$; where $B_f \subset \mathcal{N}_f$, representing the subset of samples used for the physics loss function, which is updated every 100 epochs to enhance training variability. The first three outputs ($\hat{q}_{1_d}(i)$, $\hat{q}_{2_d}(i)$, $\hat{q}_{3_d}(i)$) are crucial for solving the system's dynamic equations. They help determine the desired joint positions and their derivatives, which are then substituted into other equations. In the first equation, $q_{1_d}(i)$ and its derivatives ($\ddot{q}_{1_d}(i)$, $\dot{q}_{1_d}(i)$) are assumed the only unknown parameters and the other variables $q_{2_d}(i)$ and $q_{3_d}(i)$ are known from the input of the PINN. In the second and third equation, the same concept is implemented, $q_{2_d}(i)$ and its derivatives ($\ddot{q}_{2_d}(i)$, $\dot{q}_{2_d}(i)$), $q_{3_d}(i)$ and its derivatives ($\ddot{q}_{3_d}(i)$, $\dot{q}_{3_d}(i)$) are unknown while $q_{1_d}(i)$, $q_{3_d}(i)$ and $q_{1_d}(i)$, $q_{2_d}(i)$ are the inputs for second and third equations respectively. Hence, the first three equation would be:

$$\begin{aligned} f_1(i) &= \epsilon (\mathbf{M}_d(1, :) \hat{\mathbf{q}}_d(i) + H_1(\mathbf{q}_d(i), \dot{\mathbf{q}}_d(i)) + G_1(\mathbf{q}_d(i))), \\ f_2(i) &= \epsilon (\mathbf{M}_d(2, :) \hat{\mathbf{q}}_d(i) + H_2(\mathbf{q}_d(i), \dot{\mathbf{q}}_d(i)) + G_2(\mathbf{q}_d(i))), \\ f_3(i) &= \epsilon (\mathbf{M}_d(3, :) \hat{\mathbf{q}}_d(i) + H_3(\mathbf{q}_d(i), \dot{\mathbf{q}}_d(i)) + G_3(\mathbf{q}_d(i))). \end{aligned} \quad (40)$$

ϵ is a constant to maximize the torque owing to the fact that the torques of the fingers' joints of the DLR-HIT II hand is low value considering the measured unit for the torque is Newton.meter (N.m) according to the International System of units (SI) [30] and this small value is difficult to be predicted by neural network since the neural network is an approximation algorithm, therefore, a constant factor ϵ is multiplied by the equations to keep the consistency of the system compatible with SI. $\mathbf{M}_d(1, :)$, $\mathbf{M}_d(2, :)$, and $\mathbf{M}_d(3, :)$ represent the first, second, and third row of the inertia matrix $\mathbf{M}(\mathbf{q}_d(i))$ correspondingly, whilst $\ddot{\mathbf{q}}_d(i)$ and $\dot{\mathbf{q}}_d(i)$ can be expressed as follows:

$$\ddot{\mathbf{q}}_d(i) = \begin{bmatrix} \frac{\partial^2 \hat{q}_{1_d}(i)}{\partial t^2} \\ \frac{\partial^2 \hat{q}_{2_d}(i)}{\partial t^2} \\ \frac{\partial^2 \hat{q}_{3_d}(i)}{\partial t^2} \\ \frac{\partial^2 \hat{q}_{3_d}(i)}{\partial t^2} \end{bmatrix}, \dot{\mathbf{q}}_d(i) = \begin{bmatrix} \frac{\partial \hat{q}_{1_d}(i)}{\partial t} \\ \frac{\partial \hat{q}_{2_d}(i)}{\partial t} \\ \frac{\partial \hat{q}_{3_d}(i)}{\partial t} \\ \frac{\partial \hat{q}_{3_d}(i)}{\partial t} \end{bmatrix} \tag{41}$$

In equation (41), the third and fourth components of $\ddot{\mathbf{q}}_d(i)$ and $\dot{\mathbf{q}}_d(i)$ are intentionally repeated, as they represent identical values in the system. This redundancy is due to mechanical coupling between the third and fourth joints. The PINN can incorporate the derivative from one equation into other equations because all physical equations are related to the same system as seen in Equations (40). The physics loss for the first three outputs will be:

$$\begin{aligned} MSE_{f_1}(i) &= \left| f_1(i) - \tau_{1_d}(i) \right|^2, \\ MSE_{f_2}(i) &= \left| f_2(i) - \tau_{2_d}(i) \right|^2, \\ MSE_{f_3}(i) &= \left| f_3(i) - \tau_{3_d}(i) \right|^2. \end{aligned} \tag{42}$$

$\tau_{1_d}(i)$, $\tau_{2_d}(i)$, and $\tau_{3_d}(i)$ are the desired torque for each joint of one finger. The same idea is applied on the equations of tracking position harnessing the derivatives of the desired values from the Equations (40). As a result, the following three output values of PINN are presumed to track the actual joints' position:

$$\begin{aligned} \hat{y}_4(i) &= \hat{q}_{1_a}(i), \\ \hat{y}_5(i) &= \hat{q}_{2_a}(i), \\ \hat{y}_6(i) &= \hat{q}_{3_a}(i). \end{aligned} \tag{43}$$

Then, the subsequent equations are formulated to trace the joints' positions:

$$\begin{aligned} f_4(i) &= \epsilon \left(\mathbf{M}_a(1, :) (\ddot{\mathbf{q}}_d(i) + K_P \mathbf{e}(i) + K_D \dot{\mathbf{e}}(i)) + H_1(\mathbf{q}_a(i), \dot{\mathbf{q}}_a(i)) + G_1(\mathbf{q}_a(i)) \right), \\ f_5(i) &= \epsilon \left(\mathbf{M}_a(2, :) (\ddot{\mathbf{q}}_d(i) + K_P \mathbf{e}(i) + K_D \dot{\mathbf{e}}(i)) + H_2(\mathbf{q}_a(i), \dot{\mathbf{q}}_a(i)) + G_2(\mathbf{q}_a(i)) \right), \\ f_6(i) &= \epsilon \left(\mathbf{M}_a(3, :) (\ddot{\mathbf{q}}_d(i) + K_P \mathbf{e}(i) + K_D \dot{\mathbf{e}}(i)) + H_3(\mathbf{q}_a(i), \dot{\mathbf{q}}_a(i)) + G_3(\mathbf{q}_a(i)) \right). \end{aligned} \tag{44}$$

The rows of the actual inertia matrix $\mathbf{M}(\mathbf{q}_a(i))$ are indicated by $\mathbf{M}_a(1, :)$, $\mathbf{M}_a(2, :)$, and $\mathbf{M}_a(3, :)$, K_p and K_d are the proportional and derivative gain of the controller, $\mathbf{e}(i) = \mathbf{q}_d(i) - \mathbf{q}_a(i)$ denotes the position error and $\dot{\mathbf{e}}(i) = \dot{\mathbf{q}}_d(i) - \dot{\mathbf{q}}_a(i)$ refers to the velocity error, $\dot{\mathbf{q}}_a(i)$ is a vector of the actual joints' velocities and can be stated as shown in the next equation:

$$\dot{\mathbf{q}}_a(i) = \begin{bmatrix} \frac{\partial \hat{q}_{1_a}(i)}{\partial t} \\ \frac{\partial \hat{q}_{2_a}(i)}{\partial t} \\ \frac{\partial \hat{q}_{3_a}(i)}{\partial t} \\ \frac{\partial \hat{q}_{3_a}(i)}{\partial t} \end{bmatrix} \tag{45}$$

The third and fourth terms of $\dot{\mathbf{q}}_a(i)$ in equation (45) are identical due to the mechanical coupling between these two joints, as previously explained. The following equation stand for the physics loss of tracking position:

$$\begin{aligned} MSE_{f_4}(i) &= \left| f_4(i) - \tau_{1_a}(i) \right|^2, \\ MSE_{f_5}(i) &= \left| f_5(i) - \tau_{2_a}(i) \right|^2, \\ MSE_{f_6}(i) &= \left| f_6(i) - \tau_{3_a}(i) \right|^2. \end{aligned} \tag{46}$$

$\tau_{1_a}(i)$, $\tau_{2_a}(i)$, and $\tau_{3_a}(i)$ are the actual torques for the joints of one finger. The actual torques of the joints are presumed as illustrated in the subsequent equation:

$$\begin{aligned}
\hat{y}_7(i) &= \hat{\tau}_{1_a}(i), \\
\hat{y}_8(i) &= \hat{\tau}_{2_a}(i), \\
\hat{y}_9(i) &= \hat{\tau}_{3_a}(i).
\end{aligned} \tag{47}$$

The actual torques' values are determined by leveraging the desired and actual joints positions ($\mathbf{q}_d(i)$, $\mathbf{q}_a(i)$) and their derivatives ($\dot{\mathbf{q}}_d(i)$, $\dot{\mathbf{q}}_a(i)$, $\ddot{\mathbf{q}}_a(i)$) that are computed in the Equations (40) and (44). Subsequently, the outputs' equation of the predicted torque would be:

$$\begin{aligned}
f_7(i) &= \epsilon \left(\mathbf{M}_a(1, :) \left(\ddot{\mathbf{q}}_d(i) + K_P \mathbf{e}(i) + K_D \dot{\mathbf{e}}(i) \right) + H_1(\mathbf{q}_a(i), \dot{\mathbf{q}}_a(i)) + G_1(\mathbf{q}_a(i)) - \hat{\tau}_{1_a}(i) \right), \\
f_8(i) &= \epsilon \left(\mathbf{M}_a(2, :) \left(\ddot{\mathbf{q}}_d(i) + K_P \mathbf{e}(i) + K_D \dot{\mathbf{e}}(i) \right) + H_2(\mathbf{q}_a(i), \dot{\mathbf{q}}_a(i)) + G_2(\mathbf{q}_a(i)) - \hat{\tau}_{2_a}(i) \right), \\
f_9(i) &= \epsilon \left(\mathbf{M}_a(3, :) \left(\ddot{\mathbf{q}}_d(i) + K_P \mathbf{e}(i) + K_D \dot{\mathbf{e}}(i) \right) + H_3(\mathbf{q}_a(i), \dot{\mathbf{q}}_a(i)) + G_3(\mathbf{q}_a(i)) - \hat{\tau}_{3_a}(i) \right),
\end{aligned} \tag{48}$$

and the physics loss for the computed torque control is described as follows:

$$\begin{aligned}
MSE_{f_7}(i) &= |f_7(i) - 0|^2, \\
MSE_{f_8}(i) &= |f_8(i) - 0|^2, \\
MSE_{f_9}(i) &= |f_9(i) - 0|^2,
\end{aligned} \tag{49}$$

by setting the target value to zero, the loss function encourages the predicted control torques $\hat{\tau}_{1_a}(i)$, $\hat{\tau}_{2_a}(i)$, and $\hat{\tau}_{3_a}(i)$ to satisfy the system's physical equations. This formulation ensures that the predicted torques correspond to an equilibrium state. The total physics loss can be written as shown in the following expression:

$$MSE_{physics} = \frac{1}{B_f} \sum_{j=1}^9 \sum_{i=1}^{B_f} MSE_{f_j}(i) \tag{50}$$

while the loss fitting data is obtained using the following equation:

$$MSE_{data} = \frac{1}{B_d} \sum_{j=1}^9 \sum_{k=1}^{B_d} \left(\hat{y}_j(k) - \mu_{data_j}(k) \right)^2 \tag{51}$$

here, MSE_{data} describes a Euclidean distance measure, which by definition produce a scalar quantity, it represents the sum of squared differences between actual and predicted data for each k . The set $B_d \subset \mathcal{N}_d$ denotes the number of samples used for the data loss function, which is updated every 100 epochs to ensure diversity in training. $\hat{y}_j(k)$ represents the predicted outputs from the PINN, which correspond to the predicted joints' desired positions, actual positions, and actual torques: $\hat{\mathbf{q}}_d(k)$, $\hat{\mathbf{q}}_a(k)$, and $\hat{\tau}_a(k)$, respectively. While $\mu_{data_j}(k)$ contains the actual output whose samples are carefully selected to satisfy the constraints of the robotic hand's finger joints. Then the total loss of PINN would be the combination of the physics loss and data fitting loss as illustrated in the following equation:

$$MSE_{total} = MSE_{physics} + MSE_{data} \tag{52}$$

where MSE_{total} is the total loss function throughout all dimensions and samples. The following lines of pseudocode illustrate the steps to achieve convergence for the PINNs:

4. Experimental studies

4.1. Data generation and preprocessing

The training data for the physics loss are generated using a fifth-order polynomial equation, with a final time of 10 seconds and a time step of 0.01 seconds, across 12 different trajectory scenarios (totalling 12,012 samples, including the initial time at 0). This data is then input into the dynamic equations of the DLR-HIT II hand to determine the desired torque. Simultaneously, 12 distinct points for the actual joint positions and zero velocities of the artificial hand's joints are substituted into the computed torque control equation to track the actual torque and trajectory. For the loss function related to the fitting data, 1,302 samples are carefully selected to encompass the constraints of the robotic hand's joints, covering both desired and actual values, using the same polynomial equation. The positions and velocities are updated using the following equations:

$$\begin{aligned}
\ddot{\mathbf{q}}(:, i) &= \mathbf{M}(\mathbf{q}(i))^{-1} \left(\frac{\boldsymbol{\tau}(i)}{\epsilon} - \mathbf{H}(\mathbf{q}(i), \dot{\mathbf{q}}(i)) - \mathbf{G}(\mathbf{q}(i)) \right), \\
\dot{\mathbf{q}}(:, i+1) &= \dot{\mathbf{q}}(:, i) + \ddot{\mathbf{q}}(:, i) \Delta t, \\
\mathbf{q}(:, i+1) &= \mathbf{q}(:, i) + \dot{\mathbf{q}}(:, i) \Delta t + \frac{1}{2} \ddot{\mathbf{q}}(:, i) \Delta t^2
\end{aligned} \tag{53}$$

Algorithm 1 PINN to Control the Hand.

Input: $(t(i), \mathbf{q}_d(i), \mathbf{q}_a(i)), (t(k), \mathbf{q}_d(k), \mathbf{q}_a(k)), i \in \mathcal{N}_f, k \in \mathcal{N}_d$
Output: $(\hat{\mathbf{q}}_d(i), \hat{\mathbf{q}}_a(i), \hat{\tau}_a(i)), (\hat{\mathbf{q}}_d(k), \hat{\mathbf{q}}_a(k), \hat{\tau}_a(k))$

- 1: $\theta \sim \text{Random}$ {Neural network parameters}
- 2: Set $\xi > 0$ {Convergence threshold}
- 3: Set B_f, B_d ; $B_f \subset \mathcal{N}_f, B_d \subset \mathcal{N}_d$ {Mini-Batch sizes for physics and data loss}
- 4: epoch $\leftarrow 0$
- 5: converged $\leftarrow \text{False}$
- 6: **while** $MSE_{\text{total}} \geq \xi$ **do**
- 7: **if** epoch mod 100 == 0 **then**
- 8: Select mini-batch $B_f \subset \mathcal{N}_f$ {Sample mini-batch for physics loss}
- 9: Select mini-batch $B_d \subset \mathcal{N}_d$ {Sample mini-batch for data loss}
- 10: **end if**
- 11: $\hat{\mathbf{q}}_d(i), \hat{\mathbf{q}}_a(i), \hat{\tau}_a(i) \leftarrow \text{NN}_\theta(t(i), \mathbf{q}_d(i), \mathbf{q}_a(i)), i \in B_f$ {NN $_\theta$ represents the neural network parameterized by θ }
- 12: $MSE_{f_j} = \begin{cases} |f_j - \tau_{j_d}|^2, & j \in \{1, 2, 3\}, \text{ (Eq. (42))} \\ |f_j - \tau_{j_a}|^2, & j \in \{4, 5, 6\}, \text{ (Eq. (46))} \\ |f_j - 0|^2, & j \in \{7, 8, 9\}, \text{ (Eq. (49))} \end{cases}$
- 13: $MSE_{\text{physics}} = \frac{1}{B_f} \sum_{j=1}^9 \sum_{i=1}^{B_f} MSE_{f_j}(i)$, (Eq. (50))
- 14: $\hat{\mathbf{q}}_d(k), \hat{\mathbf{q}}_a(k), \hat{\tau}_a(k) \leftarrow \text{NN}_\theta(t(k), \mathbf{q}_d(k), \mathbf{q}_a(k)), k \in B_d$
- 15: $MSE_{\text{data}} = \frac{1}{B_d} \sum_{j=1}^9 \sum_{k=1}^{B_d} (\hat{y}_j(k) - \mu_{\text{data}}(k))^2$, (Eq. (51))
- 16: $MSE_{\text{total}} = MSE_{\text{data}} + MSE_{\text{physics}}$, (Eq. (52))
- 17: $\theta \leftarrow \theta - \eta \nabla_\theta MSE_{\text{total}}$ {Update parameters using gradient descent}
- 18: epoch \leftarrow epoch + 1
- 19: converged $\leftarrow (MSE_{\text{total}} < \xi)$
- 20: **end while**

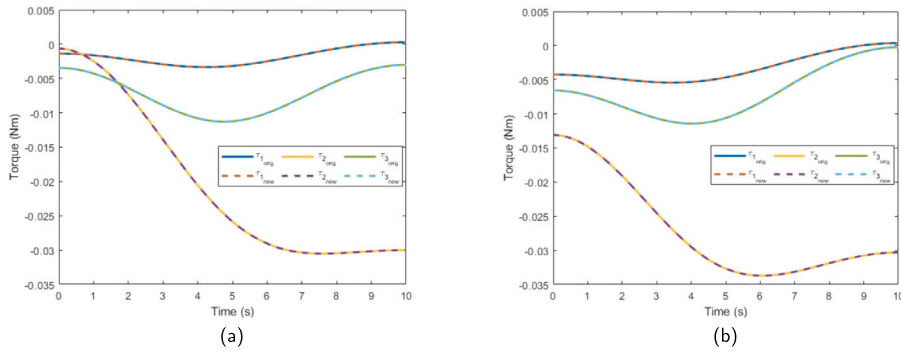


Fig. 5. (a) Comparison of the simplified dynamic and the original dynamic of the thumb finger, where $\tau_{1_{org}}, \tau_{2_{org}}, \tau_{3_{org}}$ represent the torques of original dynamic, and $\tau_{1_{new}}, \tau_{2_{new}}, \tau_{3_{new}}$ represent the torques of simplified dynamic. (b) Comparison of the simplified dynamic and the original dynamic of other fingers, where $\tau_{1_{org}}, \tau_{2_{org}}, \tau_{3_{org}}$ denote the torques of original dynamic, and $\tau_{1_{new}}, \tau_{2_{new}}, \tau_{3_{new}}$ denote the torques of simplified dynamic.

where Δt is the time step, which is $t(i+1) - t(i)$. In our formulation, Δt is constant throughout the trajectory, ensuring uniform time discretization. This assumption is standard in robotic trajectory planning and numerical integration, facilitating consistent control updates and real-time execution. Since Δt does not carry an index, it remains fixed for the entire trajectory. After collecting the data, the time for each trajectory is normalized to the range $[0, 1]$ s using the normalization method [25] for all training samples, as defined in Equation (54). This scaling ensures that the PINN can effectively generalize to different time horizons and steps.

$$t_n(i) = \frac{t(i) - t_{\min}}{t_{\max} - t_{\min}} \quad (54)$$

Here, the normalized time is denoted as $t_n(i)$, where i represents the index within the normalized time array. Similarly, $t(i)$ refers to the time value at index i , with i corresponding to the length of the time array that defines the temporal evolution of a trajectory, given by $\mathbf{t} = \{t(1), t(2), \dots, t(\mathcal{N})\}$. This array consists of discrete time values sampled at regular intervals of Δt starting from the initial time $t(1)$ and progressing up to the final time $t(\mathcal{N})$. The terms t_{\min} and t_{\max} represent the minimum and maximum time values within the array \mathbf{t} , respectively. Applying the normalisation method on the time lessens the training time, stabilizes the training, improves PINNs' ability for generalisation as well as the loss function converges faster.

The proposed control strategy was tested through comprehensive MATLAB/Simulink simulations. The Simscape Multibody library was employed to accurately model the physical structure of the DLR-HIT II robotic hand, replicating its kinematics and dynamics. The simulations involved various trajectory types, including polynomial and sinusoidal paths, to evaluate the controller's performance under typical conditions. To enhance the realism of the collected sample set and simulate practical operating conditions, we assured our data generation process to include external disturbances. Specifically, the simulations were subjected to disturbances in the form

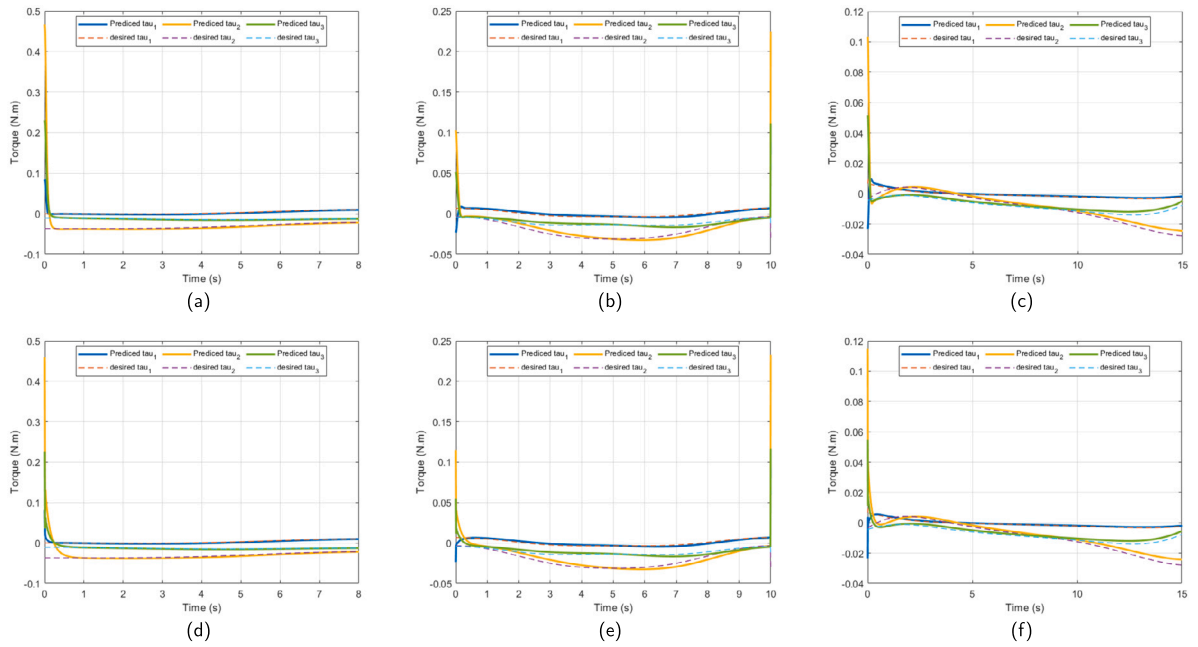


Fig. 6. (a), (b), and (c) show the predicted torque by the PINN for the cubic, sinusoidal, and spline trajectories with seven control points, respectively, compared to the desired torque over time periods of 8s, 10s, and 15s, with time steps of 0.004, 0.002, and 0.005 seconds. In contrast, (d), (e), and (f) present the torque generated by the PD computed torque control for the same trajectories, time intervals, and time steps.

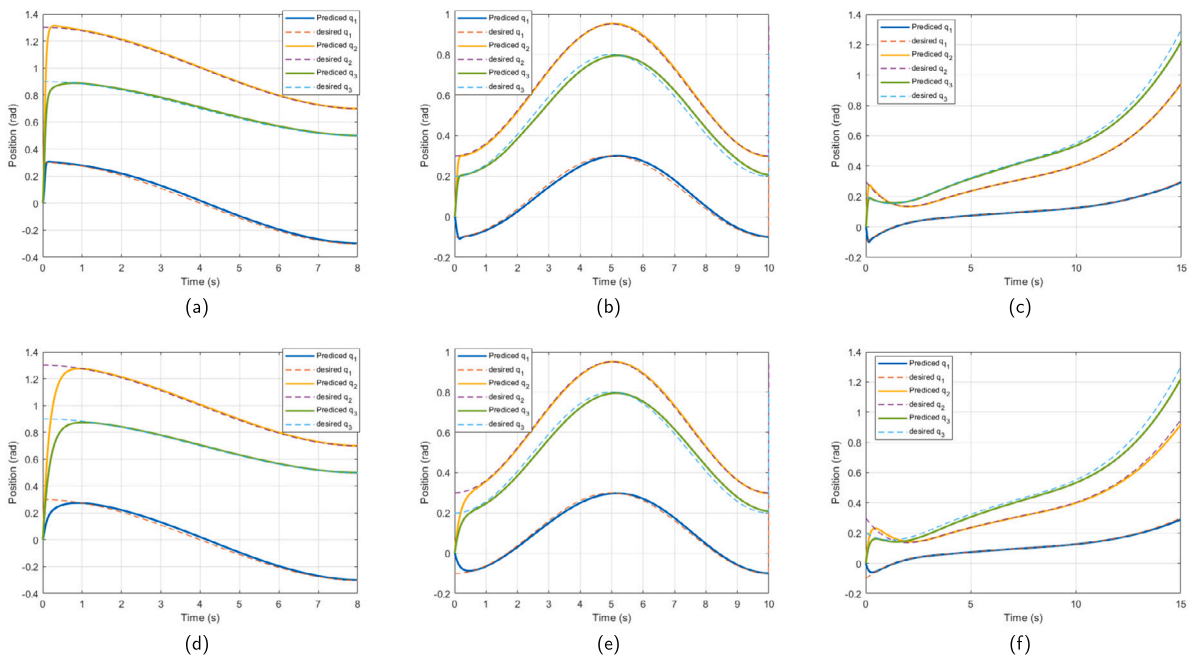


Fig. 7. (a), (b), and (c) depict the position tracking performance of the PINN for the cubic, sinusoidal, and spline trajectories with seven control points, respectively, compared to the desired trajectory over time periods of 8s, 10s, and 15s, with time steps of 0.004, 0.002, and 0.005 seconds. Meanwhile, (d), (e), and (f) illustrate the position tracking of the PD computed torque control for the same trajectories, time durations, and time steps.

of Gaussian noise applied to the joint torques and external forces acting on the robotic hand. These disturbances were modelled with a mean of zero and a standard deviation of 5%, 10%, 20%, and 30% of the joint's maximum torque, simulating typical noise and perturbations experienced in real-world scenarios.

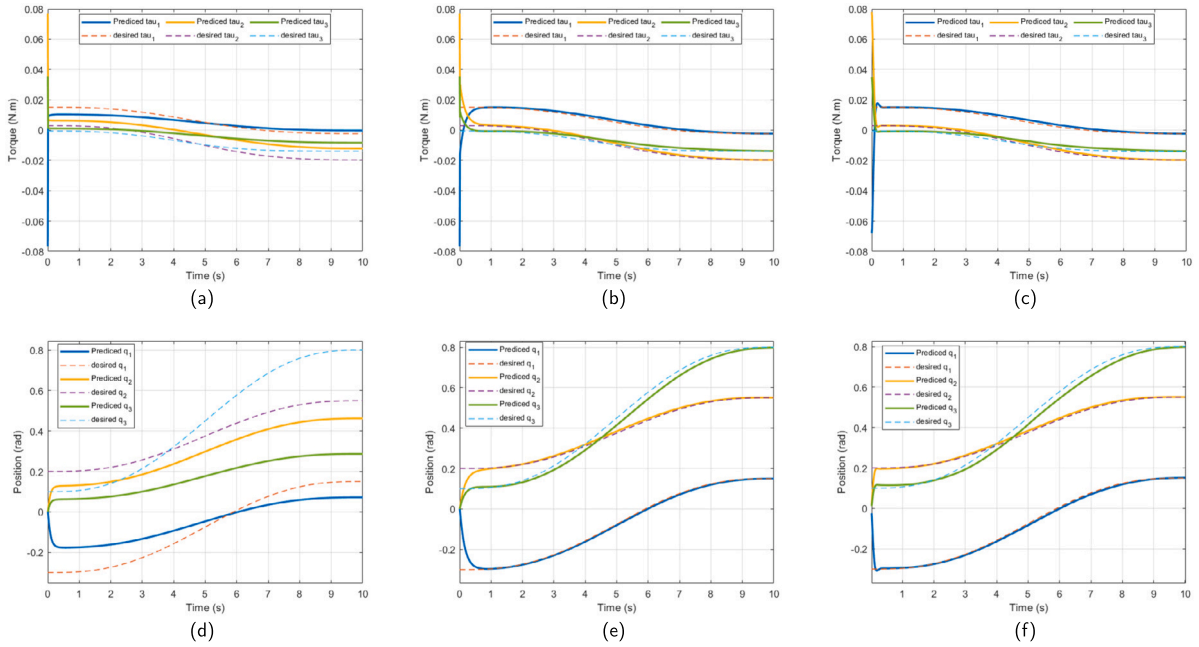


Fig. 8. (a) and (d) show the torque and tracking position of the PD computed torque control for the fifth polynomial trajectory, without velocity feedback, relying on the velocity generated from Equation (53). (b) and (e) display the torque and tracking position with actual velocity feedback for the same trajectory under PD computed torque control. Meanwhile, (c) and (f) present the predicted torque and tracking position using PINNs for the same trajectory, without velocity feedback, relying on low-dimensional inputs.

Table 3
MAPE, MATE, MAXAPE, and MAXATE for cubic trajectory.

Joints	MAPE		MATE		MaxAPE		MaxATE	
	PD	PINN	PD	PINN	PD	PINN	PD	PINN
1	0.0193	0.0143	0.0012	0.0015	0.3000	0.3000	0.0823	0.0848
2	0.0341	0.0155	0.0054	0.0055	1.3000	1.3000	0.4962	0.5032
3	0.0257	0.0162	0.0022	0.0026	0.9000	0.9000	0.2370	0.2408

Table 4
MAPE, MATE, MAXAPE, and MAXATE for sinusoidal trajectory.

Joints	MAPE		MATE		MaxAPE		MaxATE	
	PD	PINN	PD	PINN	PD	PINN	PD	PINN
1	0.0091	0.0110	0.0013	0.0014	0.3994	0.3994	0.1016	0.0921
2	0.0102	0.0073	0.0040	0.0039	0.6520	0.6522	0.2637	0.5260
3	0.0210	0.0210	0.0024	0.0024	0.5921	0.5942	0.1299	0.1244

Table 5
MAPE, MATE, MAXAPE, and MAXATE for spline trajectory.

Joints	MAPE		MATE		MaxAPE		MaxATE	
	PD	PINN	PD	PINN	PD	PINN	PD	PINN
1	0.0027	0.0017	0.0003	0.0004	0.1000	0.1000	0.0303	0.0310
2	0.0103	0.0032	0.0019	0.0019	0.3000	0.3000	0.1185	0.1067
3	0.0271	0.0176	0.0014	0.0014	0.2000	0.2000	0.0593	0.0559

4.2. Results and discussion

This section presents the findings and analysis of the novel method for simplifying the dynamic equations of the robotic hand and the application of PINNs for executing PD torque control. The dynamic equations are simplified using our novel method, which is mathematically refined and detailed in Section 3-A. The simplified dynamic is tested with fifth-order polynomial trajectories and integrated into the dynamic equations. This process is illustrated in Fig. 5a and 5b, which show that the curve for the simplified

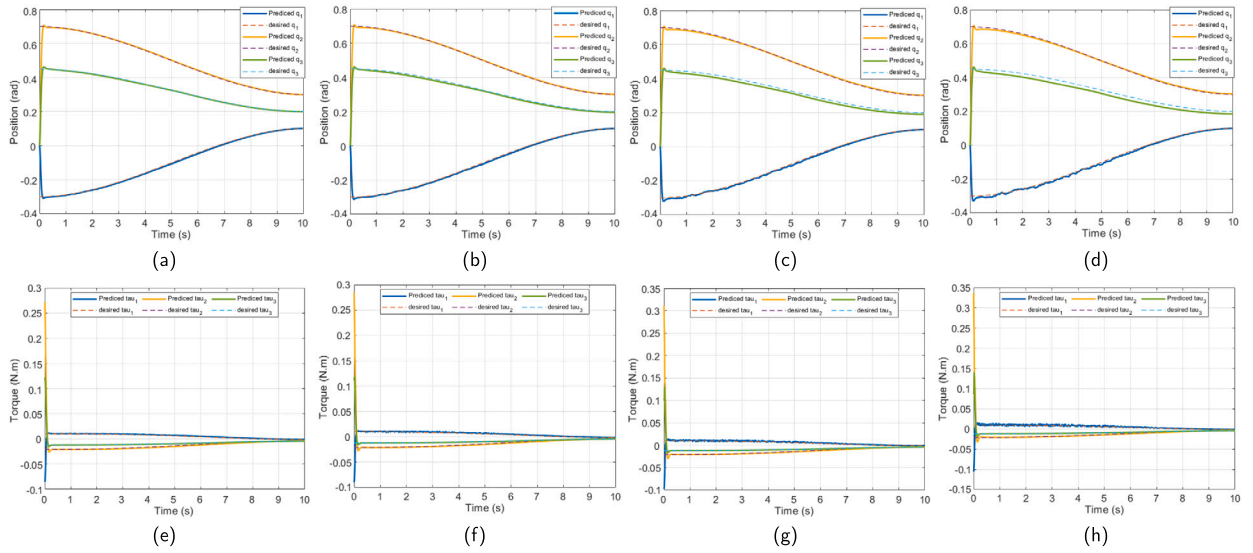


Fig. 9. (a), (b), (c), and (d) show the position tracking by the PINN for a cubic polynomial trajectory under Gaussian noise disturbances, with zero mean and standard deviations of 5%, 10%, 20%, and 30% of the joints’ maximum torque, respectively. These results are compared to the desired position over a 10-second duration with a time step of 0.001 seconds. In parallel, (e), (f), (g), and (h) present the torque predictions by the PINN under the same Gaussian noise conditions, trajectory, duration, and time step.

Table 6

MAPE by the PINN under varying disturbance percentages of Gaussian noise for a cubic trajectory.

Joints	MAPE for tracking position (PINN) under noise levels				
	0%	5%	10%	20%	30%
1	0.0051	0.0051	0.0054	0.0062	0.0069
2	0.0067	0.0059	0.0060	0.0070	0.0082
3	0.0056	0.0047	0.0087	0.0157	0.0214

Table 7

MATE by the PINN under varying disturbance percentages of Gaussian noise for a cubic trajectory.

Joints	MATE for predicting torque (PINN) under noise levels				
	0%	5%	10%	20%	30%
1	0.0011	0.0011	0.0012	0.0014	0.0018
2	0.0026	0.0024	0.0024	0.0023	0.0022
3	0.0013	0.0011	0.0011	0.0010	0.0009

dynamics exactly matches the original dynamic curve for thumb finger and other fingers, thereby demonstrating the method’s effectiveness. This study presents a streamlined approach that employs two PINNs to address the dynamics and control of a robotic hand. The developed PINNs successfully predict torques and tracks positions without requiring velocity input $\dot{\mathbf{q}}$. This is achieved because the desired torques τ_d and actual torques τ_a , as shown in Equations (42) and (46), respectively, serve as targets in the physical loss Equations (40) and (44). The time required for the finger’s joint to move from the initial to the final position is normalized between 0 and 1 second, as illustrated in Equation (54). Consequently, the PINNs are generalized to accommodate various time steps and periods, enhancing their applicability and robustness as shown in sub-Figs. 6c, 6a, and 6b for predicted torque and sub-Figs. 7a, 7b, and 7c for tracking position. The PINNs perpetually follow the trajectory even if the initial velocities differ from zero, even though they were trained with zero initial velocities. This flexibility of PINNs to disruptions guarantees steady performance, allowing it to stay on course even when it counters unforeseen changes in initial circumstances. Furthermore, the approach utilizes low-dimensional inputs, including desired joint positions \mathbf{q}_d , actual joint positions \mathbf{q}_a , and time t , without requiring feedback of the actual joint velocities $\dot{\mathbf{q}}_a$. This leverages the PINN’s ability to internally generate the derivatives of the desired joint positions ($\dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d$) and the derivative of the actual joint positions $\dot{\mathbf{q}}_a$, enabling precise control without relying on additional feedback. The developed PINN, although trained on a limited dataset of just 13,314 samples from a fifth polynomial trajectory. The PINNs are validated by comparing their output with the output of the PD computed torque function. This comparison is made by generating the velocity using Equation (53) without relying on actual feedback velocity. The PD controller fails to follow the desired trajectory under these conditions as shown in

sub-Figs. 8a and 8c. However, despite being trained on the same equation with the same generated velocity, the PINNs successfully predict the torques, demonstrating their effectiveness and accuracy. Additionally, the results are validated by comparing them with the PD computed torque control, this time incorporating the actual velocity feedback. Sub-Figs. 8b and 8e show the generated torque position tracking for PD computed torque control using fifth polynomial trajectory, while sub-Figs. 8c and 8f show the predicted torque by PINNs and its tracking position for the same trajectory. The PINNs' ability to generalize across different trajectory classes can be attributed to two key factors. First, the network is trained using normalized time and a carefully sampled dataset to cover the joints constraints. Second, the first three physics-informed equations of the PINNs (Equations (40)) encode the fundamental physical laws governing the system. This interpretative capability enables the PINNs to accurately predict outputs across various trajectory types. The PINNs demonstrate strong generalization across diverse trajectory types. Fig. 6 and 7 demonstrate the performance of the PINNs compared to PD computed torque control in real-time simulation across various trajectory types, highlighting both position tracking and the generated torque by the PINNs and PD control. MAPE, MATE, MaxAPE, and MaxATE were calculated for both PD computed torque control and the developed PINNs across the cubic trajectory (Table 3), sinusoidal trajectory (Table 4), and spline trajectory with seven control points (Table 5). The real-time error comparison reveals that the PINNs outperformed PD computed torque control for the spline trajectory, while achieving comparable performance in the sinusoidal and cubic trajectories. To evaluate the robustness of the PINN-based control system against external disturbances, a series of simulations were conducted with varying levels of Gaussian noise perturbations. The noise was introduced with a zero mean and standard deviations set at 5%, 10%, 20%, and 30% of the joints' maximum torque. These levels simulate real-world uncertainties and external forces that the control system might encounter. Fig. 9 illustrates the torque response and trajectory tracking performance under these noisy conditions. Despite the increasing intensity of the noise, the control system consistently maintained an error rate within acceptable bounds, effectively mitigating the effects of the perturbations. This highlights the inherent stability and adaptability of the PINN-based controller. Furthermore, Table 6 compares MAPE under noise-free and noisy conditions, while Table 7 presents a similar comparison for MATE. The results indicate that, while the introduction of noise led to a slight increase in tracking error, the overall performance of the PINN-based controller remained robust. These findings demonstrate the controller's resilience and its ability to handle unexpected external forces with minimal degradation in accuracy.

5. Conclusion

This study highlights the effectiveness of the developed PINNs in controlling robotic tasks that are either costly or have limited data availability. Unlike purely data-driven algorithms, the developed neural network requires significantly less training data. Moreover, the PINNs can model the nonlinear relationships inherent in complex robotic dynamics while incorporating prior knowledge of the system's physics directly into its learning process. This approach results in more accurate control by enabling the neural network to inherently understand and govern the underlying physical principles. Two PINNs are employed to solve the dynamics of the robotic hand and to implement computed torque control with position tracking, all without the need for integration with additional controllers or sub-networks. Furthermore, treating each output with its own equation effectively avoids the multi-dimensional limitations typically encountered in PINNs. By applying a normalization method to scale time and carefully selecting training data to cover the constraints of the robotic hand's joints, the PINNs are able to generalize across various trajectory classes, even when the training dataset is limited and belong to one type of trajectory. A low-dimensional input is fed into the PINNs, which leverages its capability to internally derive joint velocities, thus eliminating the need for external velocity feedback. The validity of this approach is further supported by the findings from the simplified dynamic equations. These equations were shown that accurately match the original dynamics of the robotic hand's thumb and other fingers. This alignment confirms both the accuracy and robustness of our method. The PINNs were validated through real-time simulation of the DLR-HIT II hand and demonstrated the potential to outperform traditional PD computed torque control. This is particularly evident in spline trajectory as well as when the PD control relies on generated velocities in the absence of actual velocity feedback. By integrating theoretical stability proofs and simulation results, we confirmed that the proposed method ensures stability and convergence. The simulations, which included external disturbances, demonstrated that the control approach is robust and suitable for real-time applications. Future work will focus on validating these findings with physical experiments, paving the way for practical deployment in real-world robotic tasks.

CRedit authorship contribution statement

Ali Al-Shahrabi: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Data curation, Conceptualization. **Masoud J. Javid:** Writing – original draft, Visualization, Validation, Software, Investigation, Data curation. **Ashraf A. Fahmy:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Investigation, Funding acquisition, Formal analysis. **Christian A. Griffiths:** Writing – review & editing, Supervision, Resources, Project administration, Investigation, Funding acquisition, Formal analysis. **Chunxu Li:** Writing – review & editing, Validation, Investigation, Formal analysis.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

The authors express their sincere gratitude to the Iraqi Ministry of Higher Education for their support. Special thanks to Al-Nahrain University for the financial assistance that enabled this study. We also extend our appreciation to Swansea University, particularly to Professor Johann Sienz, Deputy Executive Dean of Science and Engineering, for providing essential facilities, workspace, and financial support. Additionally, we would like to thank Professor Dunhui Xiao and Associate Professor Dr. Rowan Brown for reviewing the mathematical aspects of our work and providing valuable feedback as experts in mathematics.

Data availability

Data will be made available on request.

References

- [1] J. Liu, P. Borja, C. Della Santina, et al., Physics-informed neural networks to model and control robots: a theoretical and experimental investigation, *Adv. Intell. Syst.* 6 (2024) 2300385.
- [2] G. Antonelli, S. Chiaverini, P. Di Lillo, et al., On data-driven identification: is automatically discovering equations of motion from data a chimera?, *Nonlinear Dyn.* 111 (2023) 6487–6498.
- [3] A. Almomani, K. Nahar, M. Alauthman, et al., Image cyberbullying detection and recognition using transfer deep machine learning, *Int. J. Cogn. Comput. Eng.* 5 (2024) 14–26.
- [4] K. Sandbrink, C. Summerfield, Modelling cognitive flexibility with deep neural networks, *Curr. Opin. Behav. Sci.* 57 (2024) 101361.
- [5] G. Novakovsky, N. Dexter, M.W. Libbrecht, W.W. Wasserman, S. Mostafavi, Obtaining genetics insights from deep learning via explainable artificial intelligence, *Nat. Rev. Genet.* 24 (2023) 125–137.
- [6] E.A. Antonelo, E. Camponogara, L.O. Seman, et al., Physics-informed neural nets for control of dynamical systems, *Neurocomputing* 579 (2024) 127419.
- [7] R.R. Faria, B. Capron, A.R. Secchi, et al., A data-driven tracking control framework using physics-informed neural networks and deep reinforcement learning for dynamical systems, *Eng. Appl. Artif. Intell.* 127 (2024) 107256.
- [8] L. Bonnasse-Gahot, Interpolation, extrapolation, and local generalization in common neural networks, *arXiv preprint, arXiv:2207.08648*, 2022.
- [9] J. Barry-Straume, A. Sarshar, A.A. Popov, et al., Physics-informed neural networks for PDE-constrained optimization and control, *arXiv preprint, arXiv:2205.03377*, 2022.
- [10] G. Chen, J. Xia, J.H. Park, et al., Robust sampled-data control for switched complex dynamical networks with actuators saturation, *IEEE Trans. Cybern.* 52 (2022) 10909–10923.
- [11] G. Chen, G. Du, J. Xia, et al., Controller synthesis of aperiodic sampled-data networked control system with application to interleaved flyback module integrated converter, *IEEE Trans. Circuits Syst. I, Regul. Pap.* 70 (2023) 4570–4580.
- [12] G. Chen, J. Xia, J.H. Park, et al., Sampled-data synchronization of stochastic Markovian jump neural networks with time-varying delay, *IEEE Trans. Neural Netw. Learn. Syst.* 33 (2022) 3829–3841.
- [13] S. Cuomo, V.S. Di Cola, F. Giampaolo, et al., Scientific machine learning through physics-informed neural networks: where we are and what's next, *J. Sci. Comput.* 92 (2022) 88.
- [14] H.F. Al-Selwi, A.A. Aziz, F.S. Abas, Z. Zyada, et al., Reinforcement learning for robotic applications with vision feedback, in: 2021 IEEE 17th International Colloquium on Signal Processing & Its Applications (CSPA), IEEE, 2021, pp. 81–85.
- [15] P. Yadav, A. Mishra, J. Lee, et al., A survey on deep reinforcement learning-based approaches for adaptation and generalization, *arXiv preprint, arXiv:2202.08444*, 2022.
- [16] A. Farea, O. Yli-Harja, F. Emmert-Streib, et al., Understanding physics-informed neural networks: techniques, applications, trends, and challenges, *AI* 5 (2024) 1534–1557.
- [17] Z. Zou, X. Meng, G.E. Karniadakis, et al., Correcting model misspecification in physics-informed neural networks (PINNs), *J. Comput. Phys.* 505 (2024) 112918.
- [18] X. Yang, Y. Du, L. Li, et al., Physics-informed neural network for model prediction and dynamics parameter identification of collaborative robot joints, *IEEE Robot. Autom. Lett.* 8 (2023) 8462–8469.
- [19] W. Deng, F. Ardiani, K.T. Nguyen, et al., Physics Informed Machine Learning Model for Inverse Dynamics in Robotic Manipulators, Available at SSRN 4542725, 2023.
- [20] J. Nicodemus, J. Kneifl, J. Fehr, B. Unger, Physics-informed neural networks-based model predictive control for multi-link manipulators, *IFAC-PapersOnLine* 55 (2022) 331–336.
- [21] J. Liu, P. Borja, C. Della Santina, et al., Physics-informed neural networks to model and control robots: a theoretical and experimental investigation, *arXiv preprint, arXiv:2305.05375*, 2023.
- [22] H. Liu, K. Wu, P. Meusel, et al., Multisensory five-finger dexterous hand: the DLR/HIT Hand II, in: 2008 IEEE/RSJ Int. Conf. Intell. Robots Syst., IEEE, 2008, pp. 3692–3697.
- [23] C. Li, A. Fahmy, J. Sienz, et al., Development of a neural network-based control system for the DLR-HIT II robot hand using leap motion, *IEEE Access* 7 (2019) 136914–136923.
- [24] Y. Gao, L. Fang, X. Jiang, et al., Research on tolerance optimal allocation method for a 6-DOF series manipulator based on DH-parameters, *Proc. Inst. Mech. Eng., Part C, J. Mech. Eng. Sci.* 237 (2023) 2291–2305.
- [25] T.M. Wani, T.S. Gunawan, S.A.A. Qadri, et al., A comprehensive review of speech emotion recognition systems, *IEEE Access* 9 (2021) 47795–47814.
- [26] J.J. Craig, *Introduction to Robotics: Mechanics and Control*, 3rd ed., Pearson, 2014.
- [27] E. Arruda, eaa3/dlr_hit_hand_ii, https://github.com/eaa3/dlr_hit_hand_ii, 2019. (Accessed 25 April 2024).
- [28] J. Butterfaß, M. Grebenstein, H. Liu, et al., DLR-Hand II: next generation of a dextrous robot hand, in: Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164), vol. 1, IEEE, 2001, pp. 109–114.
- [29] I. Goodfellow, Y. Bengio, A. Courville, Regularization for deep learning, in: *Deep Learning*, The MIT Press/Massachusetts, Cambridge/London, England, 2016, pp. 224–233, <http://www.deeplearningbook.org>.
- [30] E.O. Göbel, U. Siegner, *The New International System of Units (SI): Quantum Metrology and Quantum Standards*, John Wiley & Sons, 2019.