Augmented Neural Network for Full Robot Kinematic Modelling in SE(3)

Francesco Cursi, *Member, IEEE*, Weibang Bai, Weiyi Li, Eric M. Yeatman, *Fellow, IEEE*, and Petar Kormushev

***Abstract*—Due to the increasing complexity of robotic structures, modelling robots is becoming more and more challenging, and analytical models are very difficult to build. Machine learning approaches have shown great capabilities in learning complex mapping and have widely been used in robot model learning and control. Generally, the inverse kinematics is directly learned, yet, learning the forward kinematics is simpler and allows computing exploiting the optimality of the controllers. Nevertheless, the learning method has no knowledge about the differential relationship between the position and velocity mappings. Currently, few works have targeted learning full robot poses considering both position and orientation. In this letter, we present a novel feedforward Artificial Neural network (ANN) architecture to learn full robot pose in SE(3) incorporating differential relationships in the learning process. Simulation and real world experiments show the capabilities of the proposed network to properly model the robot pose and its advantages over standard ANN.**

***Index Terms*—Model learning for control, machine learning for robot control, kinematics.**

I. Introduction

MODELLING the kinematics is an important aspect in robotics, as it allows mapping the control variables to the task space and viceversa [1]. However, with the advancement of technology and needs, robot models are becoming more and more complex, thus obtaining accurate models is becoming very challenging.

Machine learning techniques such as Artificial Neural Networks (ANN) have become very popular and efficient in dealing with complex physical models, and they have been widely used in robotics for modelling and control [2]. Learning approaches like Reinforcement learning or trail-and-error-based [3], [4] need a lot of data, appropriate definition of the reward function, and might not be applicable in scenarios where safety is required (e.g. robotic surgery).

On the other hand, black-box model-based techniques are used to approximate models of robots given some input and output data. The model can then be employed to formulate the control as an optimization problem [5] and exploit the efficacy of standard and consolidated control techniques.

Most state-of-the-art works only learn the mapping to robots' tip position, without considering the orientation. Different works have focused on implementing ANN to learn quaternions [6] or to extract an object pose from camera images [7] by using recurrent or convolutional networks, resulting in high network complexity.

With regards to control, the vast majority of research works have been focusing on learning the inverse kinematics of robotic structures, as in [8]–[12], directly finding the control variables, given a desired task. Learning the inverse kinematics is however challenging as the mapping is generally not univocal and it does not allow exploiting redundancies in the control, since the output depends only on the training data [13]. The forward kinematics is generally used only for prediction, as in [14].

Having access to the forward kinematic model, conversely, allows combining robot models (as in micro-macro manipulators for surgical applications [15], [16]), and employing optimal control techniques to better control the robotic system, by exploiting redundancies as in [17], [18] or in [19] where the learnt model is used in a model predictive controller. This requires differentiating the kinematic mapping to compute the robot's Jacobian, which is performed only after the model is learnt. ANN with differential relationships have recently been used for learning partial differential equations [20], but their use in robotics is still limited. Yet, as shown in [21], augmenting the training loss function with differential relationships for the velocity mapping allows the network to learn a better model and then be more accurate for control purposes. A similar approach has been presented in [22], [23], but it requires ground truth values for each entry of the Jacobian matrix.

In this work we aim to advance our recent work in robot kinematic model learning [21] by:

- proposing a feasible representation for the orientation in terms of Roll, Pitch, Yaw (RPY) angles;
- proposing a neural network architecture to learn full robots' pose in SE(3) for the position and orientation, incorporating differential relationships during training.

Manuscript received January 31, 2022; accepted May 18, 2022. Date of publication June 9, 2022; date of current version June 15, 2022. This letter was recommended for publication by Associate Editor C. Della Santina and Editor C. Gosselin upon evaluation of the reviewers' comments. This work was supported by Engineering and Physical Sciences Research Council (EPSRC) programme grant Micro-Robotics for Surgery under Grant EP/P012779/1. *(Corresponding author: Weibang Bai.)*

Francesco Cursi is with the Hamlyn Centre, Imperial College London, SW7 2BX London, U.K., and also with Robot Intelligence Lab, Imperial College London, SW7 2BX London, U.K. (e-mail: cursifrancesco@gmail.com).

Weibang Bai, Weiyi Li, and Eric M. Yeatman are with the Hamlyn Centre, Imperial College London, SW7 2BX London, U.K. (e-mail: wbbai@imperial.ac.uk; li.wy@outlook.com; e.yeatman@imperial.ac.uk).

Petar Kormushev is with the Robot Intelligence Lab, Imperial College London, SW7 2BX London, U.K. (e-mail: p.kormushev@imperial.ac.uk).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2022.3180428, provided by the authors.

Digital Object Identifier 10.1109/LRA.2022.3180428

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

The proposed model will also be made available online at https://github.com/cursi36/AugNet_RobotKinematics. RPY angles have been chosen due to their simpler representation. Additionally, our proposed representation allows overcoming discontinuities in model learning, but the use of RPY still requires proper motion planning to avoid singularities (e.g. Gimbal lock). We here compare our model to a standard feedforward ANN learning and test it for controlling two different robotic structures.

The letter is thus structured as follows.

Section II briefly describes feedforward ANN, along with robot kinematic modelling and control. Section III presents the proposed architecture; Section IV shows the modelling and control results using the proposed architectures, and, finally conclusions are drawn in Section V.

II. Problem Formulation

In this section a brief introduction of ANN, along with robot kinematic modelling and control, is presented.

A. Feedforward Artificial Neural Networks

Given a dataset of input and output points and output, feedforward ANN have been shown to be universal approximators capable of learning the complex mapping between them [24]. Thanks to their parametric and layered structure, it is easily possible to compute the derivatives of the network output with respect to the network weights through back-propagation. Additionally, it is also possible to compute in a similar way the derivatives of the output with respect to the inputs. This will be very useful for solving the inverse kinematics of the robot as shown in the following sections.

B. Robot Kinematic Modelling

Robot kinematics finds the relationship between the control variables $\boldsymbol{\theta} \in \mathbb{R}^{n_m}$ and the tip pose, generally expressed with respect to a fixed reference frame as $\boldsymbol{T} \in \mathbb{R}^{4\times4}$.

The control variables are the inputs to the actuation systems, which maps the control values onto the joint values $\boldsymbol{q}$ in the configuration space. With the knowledge of the joint values, the forward kinematics then allows mapping the joint values onto the Cartesian space and obtain the tip pose $\boldsymbol{T}(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{R}(\boldsymbol{q}) & \boldsymbol{P}(\boldsymbol{q}) \\ \boldsymbol{0} & 1 \end{bmatrix}$, with $\boldsymbol{R}(\boldsymbol{q}) \in \mathbb{R}^{3\times3}$ describing the tip orientation, and $\boldsymbol{P}(\boldsymbol{q}) \in \mathbb{R}^3$ the tip position.

In the simplest case, such as for serial-link manipulators, the mapping from the control values $\boldsymbol{\theta}$, which could be the motor positions, to the joint positions $\boldsymbol{q}$ is straightforward, as it is just a linear relationship, depending on the gear ratio. In more complex robots, such as continuum robots or flexible robots, the actuation system is more complex, as it could be pneumatic, hydraulic, or tendon-based [25]–[27]. This makes the mapping from actuation to joint space nonlinear $\boldsymbol{q} = \boldsymbol{q}(\boldsymbol{\theta})$. Therefore, in general the mapping from the control variables to the tip pose can be expressed as $\boldsymbol{T} = \boldsymbol{T}(\boldsymbol{q}) = \boldsymbol{T}(\boldsymbol{q}(\boldsymbol{\theta}))$.

Due to such complexities in certain robotic structures [9], ANN result useful to learn the whole mapping from actuation space to Cartesian space (tip pose) $\boldsymbol{\theta} \rightarrow \hat{\boldsymbol{\mathcal{P}}}$. $\hat{\boldsymbol{\mathcal{P}}} = \begin{bmatrix} \hat{\boldsymbol{P}} & \hat{\boldsymbol{\psi}} \end{bmatrix}^T$ is used in this work to represent the ANN expected output, with $\hat{\boldsymbol{P}} \in \mathbb{R}^3$ the expected tip position and $\hat{\boldsymbol{\psi}}$ a vector representing the tip orientation. As discussed in the following subsection, different representations can be used. The layered and parametric structure of ANN allows computing the robot expected Cartesian Jacobian $\hat{\boldsymbol{J}} = \frac{\partial \hat{\boldsymbol{\mathcal{P}}}}{\partial \boldsymbol{\theta}}$.

C. Representations for Orientations

A rotation matrix $\boldsymbol{R} \in \mathbb{R}^{3\times3}$ can be generally obtained from three different representations [28]:

1) Angle-Axis: A unit-norm axis specifies the rotation direction and an angle defines the rotation amount.

2) Quaternions: Unit quaternions are 4-dimensional vectors that, similarly to the angle-axis representation, allow obtaining a rotation matrix by considering the rotation about a certain axis by a given angle. The advantage of using quaternions is that their derivation is singularity-free.

3) Roll, Pitch, Yaw: Euler angles like Roll, Pitch, and Yaw define a rotation matrix by considering consecutive rotations about defined axis. In Roll, Pitch, Yaw representation a rotation matrix is obtained as $\boldsymbol{R}(\alpha, \beta, \gamma) = \boldsymbol{R}_x(\alpha)\boldsymbol{R}_y(\beta)\boldsymbol{R}_z(\gamma)$, where $\boldsymbol{\psi} = \begin{bmatrix} \alpha & \beta & \gamma \end{bmatrix}^T$ define the rotations about the x, y, z axes of a fixed reference frame.

As shown in [29], all these representations are discontinuous and difficult for ANN to learn. Furthermore, the whole quaternion and the axis in the Angle-Axis representation need to be unitary, thus requiring additional manipulation such as an appropriate representation (for instance in spherical coordinates or in a hyper-space) or adjustments to the cost function [29], [30].

Due to fewer requirements in the representation, we therefore focus on learning the orientation by means of RPY angles, expressing the pose as $\hat{\boldsymbol{\mathcal{P}}} = \begin{bmatrix} \hat{\boldsymbol{P}} & \hat{\psi} \end{bmatrix}^T$. This choice comes with the drawback of properly planning the robot motion to avoid singularities (e.g. Gimbal lock).

D. Robot Kinematic Control

For control purposes, having a model of the robotic system allows exploiting traditional optimal controllers, and thus guarantee system's stability. The goal of the inverse kinematics is to compute the control variables, in order to follow a desired Cartesian trajectory (in terms of both position and orientation), defined by $\tilde{\boldsymbol{\mathcal{P}}}(t), \dot{\tilde{\boldsymbol{\mathcal{P}}}}(t)$, with $\dot{\tilde{\boldsymbol{\mathcal{P}}}}(t) = \begin{bmatrix} \dot{\tilde{\boldsymbol{P}}} & \tilde{\boldsymbol{\omega}} \end{bmatrix}^T$, with $\dot{\tilde{\boldsymbol{P}}}$ the tip linear velocity and $\tilde{\boldsymbol{\omega}}$ the tip generalized angular velocity. When RPY are used, the time derivatives of the angles do not directly map to the angular velocity, but need to be multiplied by a transformation matrix. However, since for our control tests a sequence of positions and RPY angles are specified, the generalized angular velocity is here expressed as the rate of change of the RPY angles as $\tilde{\boldsymbol{\omega}} = \dot{\tilde{\boldsymbol{\psi}}}$. This also allows by-passing the transformation matrix to obtain the angular velocity from RPY representation, which might lead to additional mathematical singularities in the control. $\tilde{}$ is here used to indicate desired quantities and $\hat{}$ the modelled ones.

At each timestep t, the optimal control variables can be computed as:

$$\boldsymbol{\theta}_t^* = \arg\min_{\boldsymbol{\theta}} \frac{1}{2}||\tilde{\boldsymbol{\mathcal{P}}}_t - \hat{\boldsymbol{\mathcal{P}}}_t||^2 \tag{1}$$

where $\tilde{\boldsymbol{\mathcal{P}}}_t$ is the desired pose at the instant t, and $\hat{\boldsymbol{\mathcal{P}}}_t = \hat{\boldsymbol{\mathcal{P}}}(\boldsymbol{\theta}_t)$ is the end-effector expected tip pose.

Generally, a simplification of (1) is to solve the control problem at the velocity level, carrying out a local linearization of the kinematics, as:

$$\dot{\boldsymbol{\theta}}_t^* = \arg\min_{\dot{\boldsymbol{\theta}}} \frac{1}{2}||\dot{\tilde{\boldsymbol{\mathcal{P}}}}_t - \hat{\boldsymbol{J}}\dot{\boldsymbol{\theta}}||^2$$

$$\text{and} \quad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \dot{\boldsymbol{\theta}}_t^* dt \tag{2}$$

where dt is the motion sampling time. For redundant manipulators, the minimum norm solution is usually chosen, which can be computed as:

$$\dot{\boldsymbol{\theta}}_t^* = \arg\min_{\dot{\boldsymbol{\theta}}} \frac{1}{2}||\dot{\boldsymbol{\theta}}||^2$$

$$\text{s.t} \quad \dot{\tilde{\boldsymbol{\mathcal{P}}}}_t = \hat{\boldsymbol{J}}\dot{\boldsymbol{\theta}}. \tag{3}$$

Both for (2) and (3), in absence of additional motion constraints, the optimal solution is $\dot{\boldsymbol{\theta}}_t^* = \hat{\boldsymbol{J}}^{\dagger}\dot{\tilde{\boldsymbol{\mathcal{P}}}}_t$, with † representing the pseudoinverse operator and $\hat{\boldsymbol{J}} = [\hat{\boldsymbol{J}}_p \quad \hat{\boldsymbol{J}}_\omega]^T \in \mathbb{R}^{6\times n_m}$ being the tip pose Jacobain mapping the velocities of the control variables to the linear and generalized angular velocities.

III. Network Architecture

In this section we present the proposed $AugNet$ network to learn robots' pose which directly incorporates information about differential relationships in the model.

A. Position Learning in SE(3)

Learning the mapping from actuation space to the 3D tip position is a straightforward process. A standard ANN can be used with the inputs being the actuation space values $\boldsymbol{\theta}$ and the tip positions as outputs $\hat{\boldsymbol{y}} = \hat{\boldsymbol{P}}$. The network derivatives can then be computed as in Appendix V-A.

B. Orientation Learning in SO(3)

To overcome the discontinuity limitation in learning orientations, in this work we use ANN to learn the mapping from the actuation space to the trigonometric representation of each RPY angle as $\boldsymbol{\theta} \rightarrow \hat{\boldsymbol{y}}$ with $\hat{\boldsymbol{y}} = [\hat{\sin}\,\alpha \quad \hat{\cos}\,\alpha]^T$, where α is the Roll angle, for instance. The same applies to β, γ. This allows training the network on data values that are more continuous, even at higher-order derivatives, given that the sine and cosine are both infinitely differentiable.

The RPY angles can then be computed as $\hat{\alpha} = \text{atan2}(\frac{\hat{\sin}\,\alpha}{\hat{\cos}\,\alpha})$ (same for β, γ) and their derivatives can be easily computed as in Appendix V-B. Employing RPY can however still pose challenges in the motion planning, due to possible singularities (e.g. Gimbal lock).

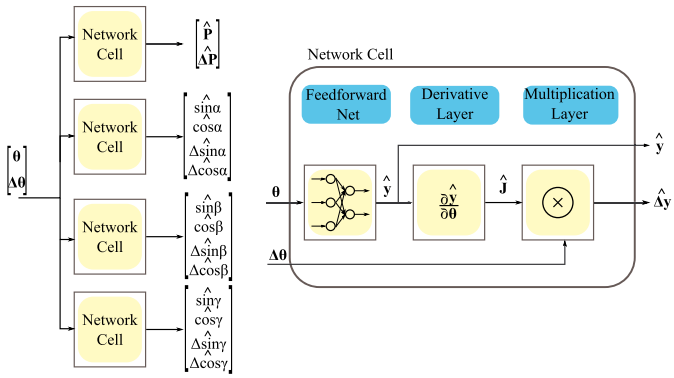


Fig. 1. The augmented network architecture to model the robot forward kinematics in SE(3) for the tip position $\hat{\boldsymbol{P}}$ and the orientation in terms of RPY angles α, β, γ. Each network cell learns each mapping independently. The structure of the network cells is shown on the right-hand side, where $\hat{\boldsymbol{y}}$ is a generic output of the cell.

C. AugNet Model

In order to learn the full robot pose and include information about the differential relationships between the pose itself and the tip rates of changes, $AugNet$ is employed. It consists of 4 different network cells, with each cell learning each mapping independently: one for the mapping from actuation space to 3D tip position; one for the mapping to the Roll angle α, one for Pitch angle β, and one for the Yaw angle γ in trigonometric representation (Fig. 1).

Each network cell takes as input the actuation values $\boldsymbol{\theta}$ and their rate of change per unit of time $\boldsymbol{\Delta\theta}$. The actuation values are then fed into a Feedforward Network Layer (which can consist of different hidden layers with various nodes), outputting the predicted value $\hat{y}(\theta, w)$, where w are the network's weights in the feedforward layer. With the given mapping, its derivatives are computed obtaining $\hat{J}_y(\theta, w) = \frac{\partial \hat{y}(\theta, w)}{\partial \theta}$ from the Derivative Layer. This layer does not add any new weights in the model, but just computes the network derivatives as shown in Appendix A.

Given the computed output derivatives and the inputs rate of changes, the final output rate of change can be computed in the multiplication layer as $\hat{\Delta}y = \hat{J}_y \Delta\theta$. The current implementation may suffer from scalability issues due to the explicit computation of the Jacobian matrix.

It is worth noticing that the input $\Delta\theta$ is only used in the training phase to compute the output rate of change. For control purposes only the Feedforward Layer is retained, which allows computing the predicted output and the Jacobian.

D. Network Training Loss

As mentioned, the proposed $AugNet$ consists of 4 independent cells to learn the full pose, therefore each cell is independently trained.

In order to take into account the differential mapping each cell's weights are optimized as:

$$w^* = \arg\min_{w} \; l$$

$$\text{with} \quad l = \frac{w_1}{D}\sum_{d=0}^{D} ||y_{m,d} - \hat{y}(\theta_d, w)||^2 + \frac{w_2}{D}\sum_{d=0}^{D} ||\Delta y_{m,d} - \hat{J}_y(\theta_d, w)\Delta\theta_d||^2, \quad (4)$$

where $\theta_d, \Delta\theta_d$ are the measured control variable values and their rate of changes, $y_{m,d}, \Delta y_{m,d}$ the measured output and its rate of change in the dataset consisting of D samples, and $w_1 = 0.1, w_2 = 1$ are heuristically user-defined parameters. Setting $w_2 >> w_1$ allows favouring the differential mapping. If $w_1 = 1, w_2 = 0$ the learning would correspond to the standard techniques used for training ANN for forward robot kinematic modelling. Due to the possible different scales between the measured tip positions and velocities, the outputs might need normalization.

1) Position Loss: For learning the tip position, (4) can be directly used with $y_m = P_m$, $\Delta y_m = \Delta P_m$ being the measured tip positions and their rate of change, $\hat{y} = \hat{P}$ the predicted output, $\hat{J}_y = \hat{J}_p \in \mathbb{R}^{3\times n_m}$ the Cartesian position Jacobian.

2) Orientation Loss: For each RPY angle the values in (4) correspond to $y_m = \begin{bmatrix}\sin\alpha_m & \cos\alpha_m\end{bmatrix}^T$, $\Delta y_m = \begin{bmatrix}\Delta\sin\alpha_m & \Delta\cos\alpha_m\end{bmatrix}^T$, where α_m is the measured angle in the dataset. In this case the network Jacobian will correspond to $\hat{J}_y = \begin{bmatrix}\hat{J}_{sin} & \hat{J}_{cos}\end{bmatrix}^T \in \mathbb{R}^{2\times n_m}$ as described in Appendix V-B.

Additionally, due to the trigonometric relationship between sines and cosines a further term is added in the cost to minmize the weights, namely:

$$l_\alpha = l + \frac{w_3}{D}\sum_{d=0}^{D}\left((\hat{\sin}\,\alpha_d)^2 + (\hat{\cos}\,\alpha_d)^2 - 1\right) \quad (5)$$

where $w_3 = 10^{-2}$ in order to penalize the model when the trigonometric relationship is not satisfied. The same applies to the Pitch and Yaw angles β, γ.

E. Full Pose Network Model

Once $AugNet$ is trained with the trigonometric representation for the orientation, only the Feedforward Layer of each cell will be retained for control purposes. Therefore the input is only the control variables θ and the predicted pose is obtained by stacking together each output as $\hat{\mathcal{P}} = \begin{bmatrix}\hat{P} & \hat{\alpha} & \hat{\beta} & \hat{\gamma}\end{bmatrix}^T$, where the RPY angles are computed from the learnt trigonometric mapping as $\hat{\alpha} = \text{atan2}(\frac{\hat{\sin}\,\alpha}{\hat{\cos}\,\alpha})$, $\hat{\beta} = \text{atan2}(\frac{\hat{\sin}\,\beta}{\hat{\cos}\,\beta})$, $\hat{\gamma} = \text{atan2}(\frac{\hat{\sin}\,\gamma}{\hat{\cos}\,\gamma})$. Additionally, the predicted pose Jacobian is also computed as:

$$\hat{J} = \begin{bmatrix}\hat{J}_p \\ \hat{J}_\omega\end{bmatrix} = \begin{bmatrix}\hat{J}_p \\ \hat{J}_\alpha \\ \hat{J}_\beta \\ \hat{J}_\gamma\end{bmatrix} \in \mathbb{R}^{6\times n_m} \quad (6)$$

where $\hat{J}_p \in \mathbb{R}^{3\times n_m}$ is the tip position Jacobian directly computed from the network structure, whereas $\hat{J}_\alpha, \hat{J}_\beta, \hat{J}_\gamma \in \mathbb{R}^{1\times n_m}$ are the Jacobians of the RPY angles that need to be computed with some manipulation of the trigonometric derivatives as in Appendix V-B.

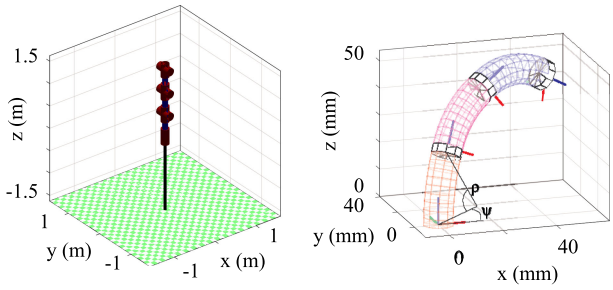


Fig. 2. The 7-DOF and soft robot models.

IV. Results

The proposed network architectures is here employed to model and control two different robotic systems, comparing its performance to a standard network learning approach. Matlab Robotics Toolbox is used for the simulation tests.

A. Robot Model Learning

We tested the proposed architectures to learn the kinematic models of: an articulated 7-DOF arm, of total length of 1.625 m; a 3-segment soft continuum robot, with length of 80 mm (Fig. 2). For the 7-DOF arm the control variables are the 7-dimensional joint position vector. For the soft robot, each segment is controlled by specifying the bending (ρ) and twist (Ψ) angles in the configuration space, for a total of 6 control variables. Its simulated model relies on a constant-curvature model.

We here compare the modelling and control results on simulation tests comparing the proposed $AugNet$ model to a more standard ANN network ($FFNet$) that doesn't include the differential relationships in the training (meaning $w_1 = 1, w_2 = 0$ in the loss function (4)).

1) Data Collection: In many robotic scenarios, the model of the robot is completely unknown. Therefore, the only feasible way to collect data for the learning is by commanding directly desired values for the control variables. In order to explore as much as possible of the reachable workspace, the control variable are excited with a set of random motions such as to span the whole control variable range. Fig. 3 shows an example with 3 randomly generated motions for the 7-DOF robot. For the soft robot, each control variable was excited with 20 different random motions, each one sampled in 443 points with a sampling time of 100 ms, for a total of 8860 datapoints. For the 7-DOF, 30 random motions were used, sampled in 443 points, resulting in 13290 datapoints collected. The ground truth simulated robot models are used to collect the data of the corresponding tip position and RPY angles. The RPY angles are then used to collect their trigonometric representations in terms of sines and cosines.

In order to train the proposed $AugNet$, data for the rates of change of the control variables and for the tip pose need to be collected too. Since the commanded motions are smooth, the

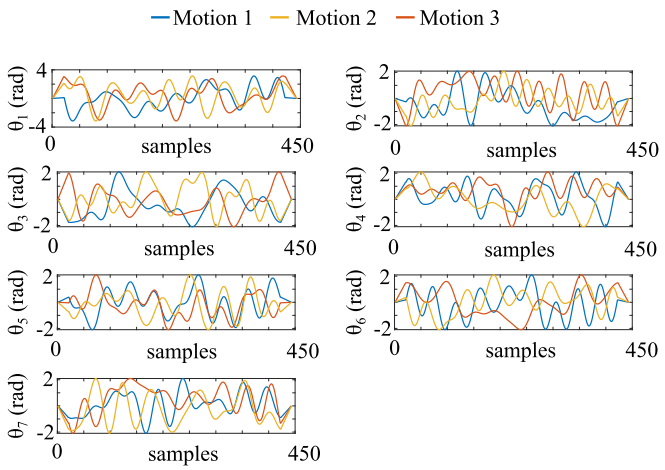


Fig. 3. Examples of three randomly generated motions for the data collection of the 7-DOF robot.

rates of changes can be computed by subtracting the current values to the ones at the previous time step.

In a real scenario, noise would be present in the measurements and its effects would be accentuated in the differentiation. However, the data on the rate of change is only needed for the offline training, therefore noise impact can be effectively reduced with filtering the data.

2) Network Architecture: For learning each robot model, the same datasets and the same optimization algorithm are used for both $FFNet$ and $AugNet$ models. A $70\% - 30\%$ random split between training and test data is chosen for the offline model learning. For the soft robot, the network cell for learning the tip position consists of two hidden layers with 100 and 50 neurons each, while each cell for the RPY angle has one hidden layer with 100 neurons. For the 7-DOF robot the network cell for the tip position consists of three hidden layers with 100, 50, 50 neurons each, and each RPY cell angle has two hidden layers with 100 and 50 neurons each. In all cases $sigmoid$ activation function is used to ensure continuity of the derivatives. The network structures were obtained heuristically and by trial-and-error, choosing the one with smallest cost on the training and test sets.

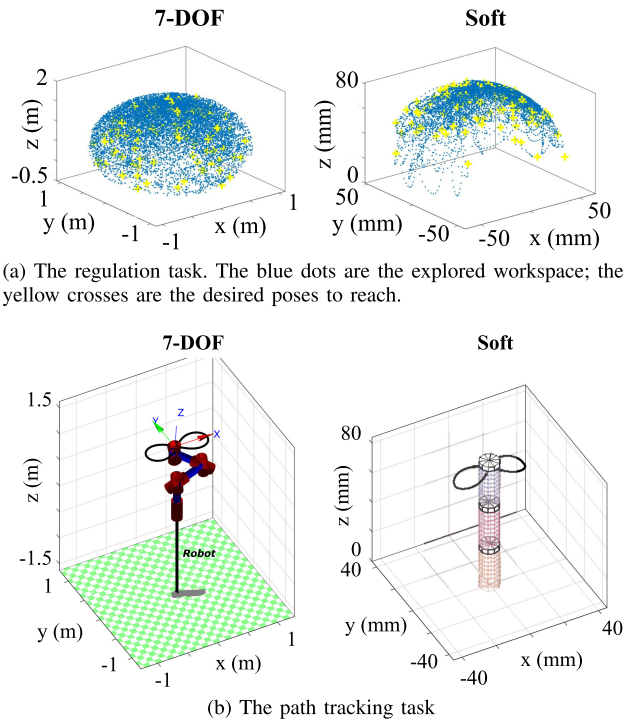
To better compare the capabilities of $AugNet$ over $FFNet$ we trained each model 5 different times with a varying number of epochs. In each training, the network architecture, the datasets, and the optimizer remained the same. We used Pytorch and selected ADAM [31] as optimizer, with a learning rate of 10^{-3} to train each model.

B. Robot Control Tests

We tested the learnt models in two different control tests, conducted in pure open-loop. Therefore the performance only relies on the accuracy of the learnt models.

1) Regulation Task: We randomly specified 100 poses within the robot workspace and numerically solved the inverse kinematics to reach each one. The robots always start from their home configuration.

2) Path Tracking Task: We specified an infinity-shaped path within the robots' workspace and used the control strategy



(a) The regulation task. The blue dots are the explored workspace; the yellow crosses are the desired poses to reach.

(b) The path tracking task

Fig. 4. Representation of the control tasks for the 7-DOF and soft robot: (a) regulation task; (b) the path tracking task.

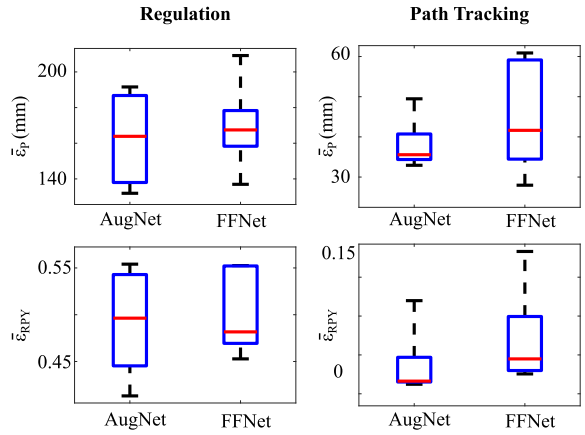
described in II-D to track the desired path. For the soft robot the path is sampled in 300 points, whereas for the 7-DOF robot 100 samples are used.

Fig. 4 shows the desired tasks for each robot. Due to the design of the soft robot and its thin and not convex workspace, it is not possible to control independently the motion along each Cartesian component and the tip orientation. For this reason, only x, y components of the tip position are controlled in each test. For the 7-DOF robot, instead, desired tip positions and orientations are specified.

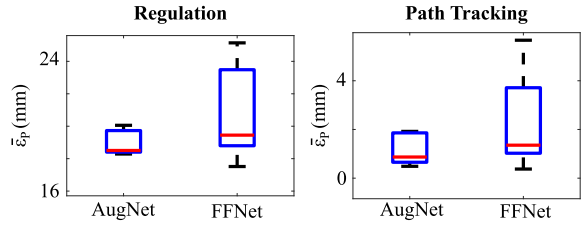
Fig. 5 and 6 report the control results for each robot and for each trained network. We compared the $AugNet$ networks and $FFNet$ networks by considering the mean tip position error norm $\bar{\epsilon}_P = \frac{1}{N}\sum_n \epsilon_{p,n} = \frac{1}{N}\sum_n ||\tilde{P}_n - P_n||$ and the mean tip orientation error norm $\bar{\epsilon}_{RPY} = \frac{1}{N}\sum_n \epsilon_{RPY,n} = \frac{1}{N}\sum_n ||[1\,1\,1]^T - \cos(\tilde{\psi}_n - \psi_n)||$, where $n = 1 \dots N$ is the number of samples in the task, $\tilde{P}, \tilde{\psi}$ are the desired tip position and orientation, and P, ψ the actual tip position and orientation obtained from the ground truth simulated models.

For the path tracking task of the 7-DOF robot, the median error for $AugNet$ results to be 35.56 mm for the tip position tracking and 0.02 for the RPY; for $FFNet$ the median tip position error is 41.62 mm and 0.04 for the orientation. For $AugNet$ the worst performing network is the 5-th one, both on tracking the tip position and the RPY angles with a maximum error of 90 mm on the tip position and 0.59 on the RPY. For $FFNet$ both the second and fourth networks lead to the worst performances.

With regards to the regulation task the median error for $AugNet$ results to be 163.88 mm and 0.49 for the orientation, compared to 167.47 mm and 0.48 for $FFNet$. The larger errors



(a) Box plot for the 7-DOF robot



(b) Box plot for the soft robot

Fig. 5. Overall box plots for the simulated control tasks with each of the 5 trained $AugNet$ and $FFNet$ networks for: (a) 7-DOF robot; (b) soft robot. $\bar{\epsilon}_P$ and $\bar{\epsilon}_{RPY}$ are the mean norm of the tip position and orientation errors. The red lines are the median of the mean errors and the box area corresponds to the 75-th percentile.

in the regulation tasks for both models are due to the fact that some desired poses are in very little explored regions of the workspace, so the inverse kinematics with the learned models does not manage to converge to an accurate solution.

Regarding the soft robot path tracking task, it is again $AugNet$ to show the best performance with a median tip positioning error of 0.86 mm compared to a median error of 1.35 mm for $FFNet$. In one case when using the 4-th $FFNet$ network, the task cannot even be completed stably and the robot diverges from the reference. In terms of the regulation task, the median tracking error for $AugNet$ is 18.5 mm, compared to 19.45 mm of the trained $FFNet$ networks. Additionally the trained $AugNet$ networks show very similar results, leading to much smaller variance in the path tracking control tests.

C. Real World Experiments

Because of the soft robot still being under design, and given the better performance of $AugNet$, we further tested the control with our proposed $AugNet$ learned model from the simulation on a real 7-DOF KUKA IIWA LBR 14 arm. The robot is required to follow in open-loop an infinity shape path with constant orientation and a square path with varying yaw (Fig. 7). The mean error norms for the infinity shape path result to be $\bar{\epsilon}_P = 16.61$ mm and $\bar{\epsilon}_{RPY} = 0.001$, whereas for the square path the errors are $\bar{\epsilon}_P = 44.10$ mm and $\bar{\epsilon}_{RPY} = 0.03$.

V. Conclusion

In this letter we propose $AugNet$, a neural network architecture to include differential relationships in the training of forward kinematics mapping. Additionally, we propose a trigonometric representation for the orientation in terms of RPY angles in order to overcome discontinuties in model training and properly learn full robots poses in SE(3).

Simulation and real world experiments show that the proposed $AugNet$ outperforms a standard feedforward ANN learning approach. The main drawback of the current approach is that RPY angles have been used for control and it must be ensured that the desired commanded values are continuous to avoid instability in the control. Further investigation is however needed in order to properly propose a representation for quaternions or Angle-Axis that would be suitable for learning with ANN, as in [29].

Additionally, our tests show that there are still generalization issues to consider. In fact, the errors both in the tracking and regulation task are mainly due to the fact that some desired poses might be in little explored areas where the models were not accurately learnt.

Appendix A
Artificial Neural Network Derivatives

Consider a feedforward Neural Network (ANN) with L layers, where $y^l = a(z^l) \in \mathbb{R}^{n_o}$ is the output of layer l, a is the activation function, $z^l = W^l x^l$, with $W^l \in \mathbb{R}^{n_o \times n_i}$ contains both the weights and biases of the layer. The gradient of the network with respect to all the weights $w^{[0:l]}$ up to layer l can be computed as:

$$g_w^l = \frac{\partial y^l}{\partial w^{[0:l]}} = \frac{\partial y^l}{\partial z^l}\left[\frac{\partial z^l}{\partial w^{[0:l-1]}} \; \frac{\partial z^l}{\partial w^l}\right]$$
$$= \frac{\partial y^l}{\partial z^l}\left[W^l \frac{\partial z^{l-1}}{\partial w^{[0:l-1]}} \quad \frac{\partial z^l}{\partial w^l}\right]. \tag{7}$$

The derivative $\frac{\partial y^l}{\partial z^l} \in \mathbb{R}^{n_o \times n_o}$ is a diagonal matrix whose elements depend on the chosen activation function and can be analytically computed. For a $sigmoid$ function it results that $\frac{\partial y_j^l}{\partial z_j^l} = \frac{e^{-z_j^l}}{(1+e^{-z_j^l})^2}$, whereas for a $swish$ function $\frac{\partial y_j^l}{\partial z_j^l} = \frac{z_j^l e^{-z_j^l}}{(1+e^{-z_j^l})^2} + \frac{1}{(1+e^{-z_j^l})}$ for each element $j = 1 \dots n_o$.

The derivative $\frac{\partial z^l}{\partial w^l} \in \mathbb{R}^{n_o \times n_o n_i}$ is such that $\frac{\partial z_j^l}{\partial w_{h,k}^l} = \begin{cases} x_k^l, \text{if } h = j \\ 0, \text{ otherwise} \end{cases}$.

By iterating through all the network layers $l = 0 \dots L$ is then possible to analytically compute the gradient of the full network with respect to all the network weights.

The derivatives of the network with respect to the inputs x^0 can be similarly computed analytically and iteratively considering that, for each layer l:

$$g_x^l = \frac{\partial y^l}{\partial x^0} = \frac{\partial y^l}{\partial z^l}\frac{\partial z^l}{\partial x^0} = \frac{\partial y^l}{\partial z^l} W^l \frac{\partial z^{l-1}}{\partial x^0}. \tag{8}$$

Starting again from the first layer $l = 0$, which is such that $\frac{\partial z^0}{\partial x^0} = W^0$, the final network derivative can be obtained.

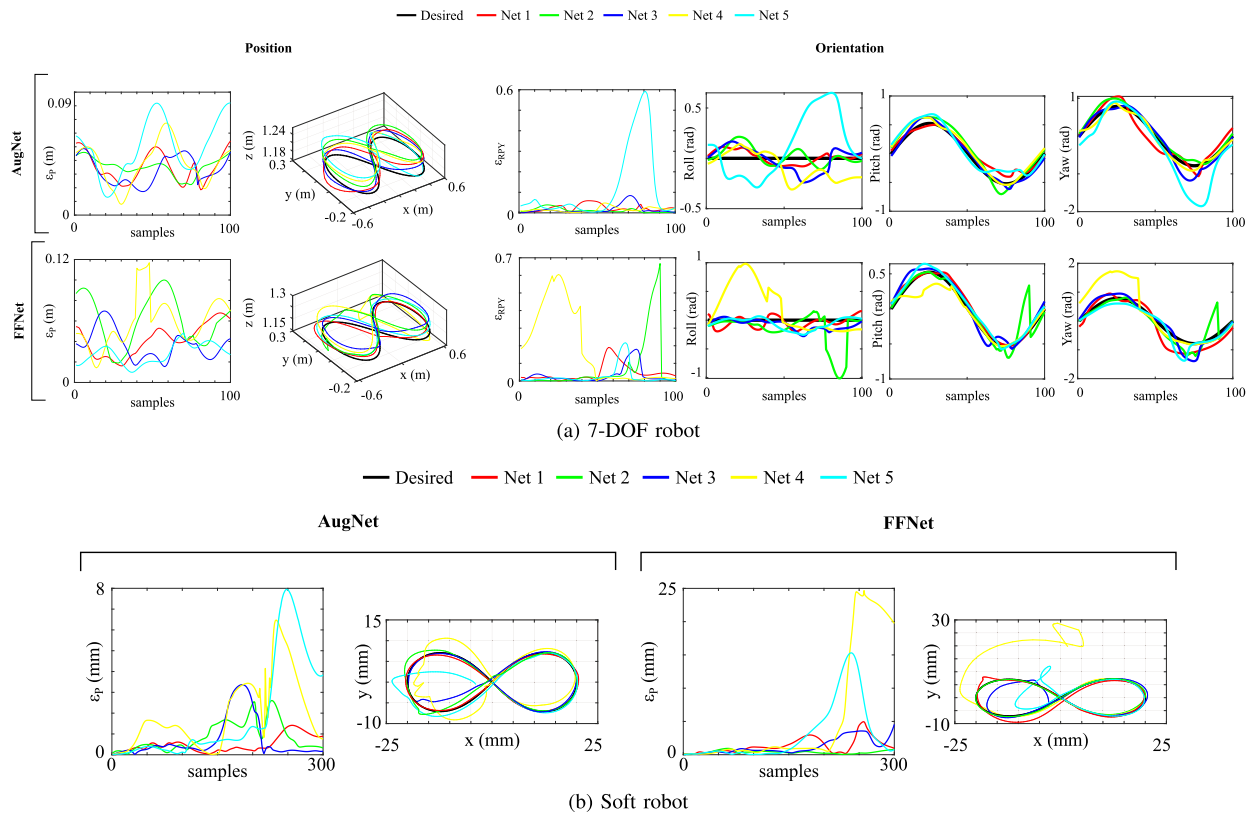


Fig. 6. Simulation path tracking results with both $AugNet$ and $FFNet$ models, for each trained network for: (a) 7-DOF robot; (b) soft robot. $\epsilon_P = ||\tilde{P}_n - P_n||$ is the norm of the error between the desired and the actual tip position; $\epsilon_{RPY} = ||\ [1\ 1\ 1]^T - \cos(\tilde{\psi}_n\ -\ \psi_n)||$ is the norm of the error on the Roll, Pitch, Yaw angles.

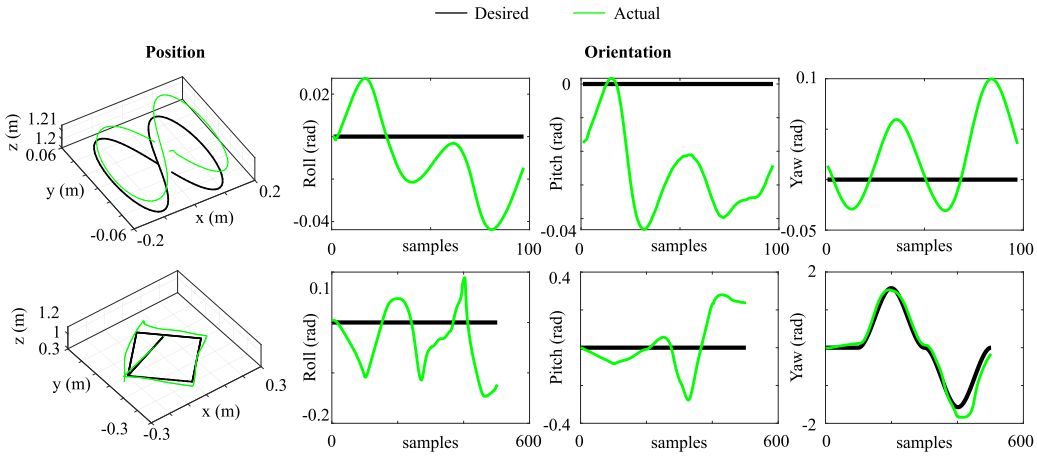


Fig. 7. Path tracking results on the real KUKA IIWA LBR 14 robotic arm.

A. Position Derivatives

Given the ANN to map the motor values $\theta \in \mathbb{R}^{n_m}$ to the 3D tip position $\hat{P}$ in III, the robot Jacobian for the linear velocity can be directly computed as $\hat{J}_p = g_x^L$ from (8).

B. RPY Derivatives

Considering the chosen representation for the orientation, the ANN outputs a two dimensional vector for each angle consisting of the sine and cosine of the angle itself. As an example, for the Roll angle α the will map the the motor values θ to $\hat{y}_{trig} = \begin{bmatrix}\hat{\sin}\,\alpha & \hat{\cos}\,\alpha\end{bmatrix}^T$. The network derivative would then be the matrix $\hat{J}_{trig} = \frac{\partial \hat{y}_{trig}}{\partial \theta} = \begin{bmatrix}\hat{J}_{sin} \\ \hat{J}_{cos}\end{bmatrix} \in \mathbb{R}^{2\times n_m}$ computed from (8).

The angle is computed as $\hat{\alpha} = \text{atan}(\frac{\hat{\sin}\,\alpha}{\hat{\cos}\,\alpha})$, where atan$(\cdot)$ is the arctangent function.

Considering the arctangent function $\xi = \text{atan}(\psi)$, its derivative is $\frac{\partial \xi}{\partial \psi} = \frac{1}{1+\psi^2}$, where, in our case, $\xi = \hat{\alpha}, \psi = \frac{\hat{\sin}\,\alpha}{\hat{\cos}\,\alpha}$. The

angle Jacobian can thus be computed as:

$$\begin{aligned}\hat{J}_\alpha &= \frac{\partial \alpha}{\partial \theta} = \frac{\partial}{\partial \theta}\left(\text{atan}\left(\frac{\hat{\sin}\,\alpha}{\hat{\cos}\,\alpha}\right)\right) = \frac{1}{1+(\frac{\hat{\sin}\,\alpha}{\hat{\cos}\,\alpha})^2}\frac{\partial}{\partial \theta}\left(\frac{\hat{\sin}\,\alpha}{\hat{\cos}\,\alpha}\right) \\ &= \frac{1}{1+(\frac{\hat{\sin}\,\alpha}{\hat{\cos}\,\alpha})^2}\left(\frac{1}{\hat{\cos}\,\alpha}\hat{J}_{sin} - \frac{\hat{\sin}\,\alpha}{(\hat{\cos}\,\alpha)^2}\hat{J}_{cos}\right) \\ &= \hat{\cos}\,\alpha\ \hat{J}_{sin} - \hat{\sin}\,\alpha\ \hat{J}_{cos}.\end{aligned} \tag{9}$$

This can be repeated for each RPY angle.

ACKNOWLEDGMENT

The authors would like to thank Shen Treratanakulchai for the modelling of the soft robot. Data supporting this study are openly available on the Robot Intelligence Lab website https://www.imperial.ac. UK/robot-intelligence/. For the purpose of open access, the author has applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising.

REFERENCES

[1] F. Cursi, W. Bai, and P. Kormushev, "Kalibrot: A. simple-to-use matlab package for robot kinematic calibration," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2021, pp. 8852–8859.

[2] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: A survey," *Cognitive Process.*, vol. 12, no. 4, pp. 319–340, Nov. 2011.

[3] Z. Guo, W. Ren, J. Huang, and C. Wang, "A reinforcement learning approach for inverse kinematics of arm robot," in *Proc. ACM Int. Conf. Proc. Ser.*, New York, NY, USA, 2019, pp. 95–99.

[4] A. Alattar, F. Cursi, and P. Kormushev, "Kinematic-model-Free redundancy resolution using multi-point tracking and control for robot manipulation," *Appl. Sci.* vol. 11, no. 11, 2021, Art. no. 4746.

[5] J. Peters and S. Schaal, "Learning to control in operational space," *Int. J. Robot. Res.*, vol. 27, no. 2, pp. 197–212, 2008.

[6] D. García-Retuerta, R. Casado-Vara, A. Martin-del Rey, F. De la Prieta, J. Prieto, and J. M. Corchado, "Quaternion neural networks: State-of-the-art and research challenges," in *Proc. Int. Conf. Intell. Data Eng. Autom. Learn.*, 2020, pp. 456–467.

[7] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes," in *Proc. Robot. Sci. Syst. Found.*, Aug. 2018.

[8] W. Xu, J. Chen, H. Y. Lau, and H. Ren, "Data-driven methods towards learning the highly nonlinear inverse kinematics of tendon-driven surgical manipulators," *Int. J. Med. Robot. Comput. Assist. Surg.*, vol. 13, no. 3, 2017, Art. no. e1774.

[9] W. Bai *et al.*, "Task-based LSTM kinematic modelling for a tendon-driven flexible surgical robot," *IEEE Trans. Med. Robot. Bionics*, vol. 4, no. 2, pp. 339–342, May 2022.

[10] A. D'Souza, S. Vijayakumar, and S. Schaal, "Learning inverse kinematics," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2001, pp. 298–303.

[11] J. Demby'S, Y. Gao, and G. N. Desouza, "A study on solving the inverse kinematics of serial robots using artificial neural network and fuzzy neural network," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, 2019, pp. 1–6.

[12] B. Bocsi, D. Nguyen-Tuong, L. Csato, B. Scholkopf, and J. Peters, "Learning inverse kinematics with structured prediction," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2011, pp. 698–703.

[13] O. Sigaud, C. Salaün, and V. Padois, "On-line regression algorithms for learning mechanical models of robots: A survey," *Robot. Auton. Syst.*, vol. 59, no. 12, pp. 1115–1129, 2011.

[14] R. Grassmann, V. Modes, and J. Burgner-Kahrs, "Learning the forward and inverse kinematics of a 6-DOF concentric tube continuum Robot in SE(3)," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2018, pp. 5125–5132.

[15] F. Cursi and P. Kormushev, "Pre-operative offline optimization of insertion point location for safe and accurate surgical task execution," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2021, pp. 4040–4047.

[16] F. Cursi, W. Bai, E. M. Yeatman, and P. Kormushev, "Optimization of surgical robotic instrument mounting in a macro-micro manipulator setup for improving task execution," *IEEE Trans. Robot.*, to be published, doi: 10.1109/TRO.2022.3171097.

[17] F. Cursi, G. P. Mylonas, and P. Kormushev, "Adaptive kinematic modelling for multiobjective control of a redundant surgical robotic tool," *Robotics*, vol. 9, no. 3, Aug. 2020, Art. no. 68.

[18] C. Salaün, V. Padois, and O. Sigaud, "Control of redundant robots using learned models: An operational space control approach," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2009, pp. 878–885.

[19] F. Cursi, V. Modugno, L. Lanari, G. Oriolo, and P. Kormushev, "Bayesian neural network modeling and hierarchical MPC for a tendon-driven surgical robot with uncertainty minimization," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 2642–2649, Apr. 2021.

[20] M. Raissi, A. Yazdani, and G. E. Karniadakis, "Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations," *Science*, vol. 367, no. 6481, pp. 1026–1030, Feb. 2020.

[21] F. Cursi, D. Chappell, and P. Kormushev, "Augmenting loss functions of feedforward neural networks with differential relationships for robot kinematic modelling," in *Proc. 20th Int. Conf. Adv. Robot.*, 2021, pp. 201–207.

[22] J. M. Bern, Y. Schnider, P. Banzet, N. Kumar, and S. Coros, "Soft robot control with a learned differentiable model," in *Proc. 3rd IEEE Int. Conf. Soft Robot.*, 2020, pp. 417–423.

[23] G. Fang, Y. Tian, Z.-X. Yang, J. M. P. Geraedts, and C. C. L. Wang, "Efficient Jacobian-based inverse kinematics of soft robots by learning," *IEEE/ASME Trans. Mechatronics*, to be published, doi: 10.48550/arXiv.2012.13965.

[24] C. Bishop, *Neural Networks for Pattern Recognition*. Oxford, U.K.: Clarendon Press, 1995.

[25] W. Hong, A. Schmitz, W. Bai, P. Berthet-Rayne, L. Xie, and G. Z. Yang, "Design and compensation control of a flexible instrument for endoscopic surgery," in *Proc. - IEEE Int. Conf. Robot. Automat.*, 2020, pp. 1860–1866.

[26] W. Bai, Q. Cao, C. Leng, Y. Cao, M. G. Fujie, and T. Pan, "A novel optimal coordinated control strategy for the updated robot system for single port surgery," *Int. J. Med. Robot. Comput. Assist. Surg.*, vol. 13, no. 3, Sep. 2017, Art. no.e1844.

[27] W. Bai *et al.*, "Anthropomorphic dual-arm coordinated control for a single-port surgical robot based on dual-step optimization," *IEEE Trans. Med. Robot. Bionics*, vol. 4, no. 1, pp. 72–84, Feb. 2022.

[28] B. Siciliano and O. Khatib, Eds., *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer, 2008.

[29] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 5738–5746.

[30] D. Pavllo, C. Feichtenhofer, M. Auli, and D. Grangier, "Modeling human motion with quaternion-based neural networks," *Int. J. Comput. Vis.*, vol. 128, no. 4, pp. 855–872, 2020.

[31] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Representations*, Dec. 2015.