Augmenting Loss Functions of Feedforward Neural Networks with Differential Relationships for Robot Kinematic Modelling

Francesco Cursi*[1,2], Digby Chappell[2], Petar Kormushev[2]

***Abstract*— Model learning is a crucial aspect of robotics as it enables the use of traditional and consolidated model-based controllers to perform desired motion tasks. However, due to the increasing complexity of robotic structures, modelling robots is becoming more and more challenging, and analytical models are very difficult to build, particularly for redundant robots. Machine learning approaches have shown great capabilities in learning complex mapping and have widely been used in robot model learning and control. Generally, inverse kinematics is learned, directly obtaining the desired control commands given a desired task. However, learning forward kinematics is simpler and allows the computation of the robot Jacobian and enables the exploitation of the optimality of controllers. Nevertheless, typical learning methods have no knowledge about the differential relationship between the position and velocity mappings. In this work, we present two novel loss functions to train feedforward Artificial Neural network (ANN) which incorporate this information in learning the forward kinematic model of robotic structures, and carry out a comparison with standard ANN training using position data only. Simulation results show that incorporating the knowledge of the velocity mapping improves the suitability of the learnt model for control tasks.**

I. Introduction

Kinematic modelling is an important aspect of robotics, as it produces a mapping from joint space control variables to task space variables, and vice versa. However, with advancing technology and requirements, robot models are becoming more and more complex, thus obtaining accurate kinematic models is becoming increasingly challenging.

Machine learning techniques have become very popular and effective in dealing with complex physical models, and they have been widely used in robotics for modelling and control [1]. Machine learning approaches for control can be mainly divided into model-free and model-based. Model-free approaches, such as those used in Reinforcement Learning, do not require any knowledge about the robot model, instead learning policies based on some reward function [2], [3]. Reinforcement learning, however, is less data-efficient than supervised learning, requires an appropriate reward function definition, and may need to be retrained for new tasks [4].

On the other hand, model-based techniques are used to create an approximation of the robot model, without the need of analytical mapping, which may be hard to obtain due to the complexity of the system. 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. Different works have focused on model learning for robotics, such as in [6], where a Gaussian mixture model was used to build the kinematic model of a robot catheter, or [7], where neural networks were employed to learn the inverse kinematics of a soft robot directly.

The vast majority of research has been focused on learning the inverse kinematics of robotic structures, as in [8]–[10], computing the control variables directly from task space variables. However, the technique of learning inverse kinematics has a number of drawbacks:

- it represents an anticausal relationship, with usually a non unique solution [11], as it could be for redundant robots, and it makes this mapping complicated to obtain;
- it does not allow the exploitation of redundancy in highly articulated robots and finding optimal control variables [12];
- it requires information about the robot pose, which it might be unavailable during use due to the lack of sensors (as it could be in robotic surgery).

Learning the forward kinematics of the robot, instead, can be beneficial to overcome these limitations. In [13], [14] feedforward Artificial Neural Networks (ANNs) were used to learn the forward kinematics of a robotic system and analytical derivation is carried out to compute the robot Jacobian for control purposes. A similar approach is used in [15], [16] where the derivation of the Jacobian is exploited to solve the redundancy problem by means of Null Space Projection [17].

The main disadvantage of learning the forward kinematics is that the velocity mapping, which is generally used for control, is obtained by differentiating the learned forward model. This may lead to inaccurate approximations, as the learning techniques have no knowledge about the differential relationship between position and velocity mappings. Recently, Lagrangian Neural Networks [18], [19] have been presented as architectures that are capable of learning the Lagrangian dynamics of a system by incorporating derivatives of the neural network model as part of the loss function used during learning. This technique embeds physical differential relationships in the model, and could be an appealing research direction for kinematic modelling.

Since neural networks have shown great capabilities in modelling complex functions [20] and in robot control [21]–[23], the contribution of this work is two-fold:

- proposing two novel loss functions to learn the forward

*Corresponding author
[1] Hamlyn Centre, Imperial College London, 25 Exhibition Road, London, UK
[2] Robot Intelligence Lab, Imperial College London, 25 Exhibition Road, London, UK
Email: [f.cursi17, d.chappell19, p.kormushev]@imperial.ac.uk

kinematics model of robot manipulators, which incorporate physical differential relationships during training to additionally learn velocity mapping;
- comparing the capabilities of the proposed training strategies and of standard training in modelling the forward kinematics of robot manipulators and evaluate their performances in robot kinematic control.

Even though our focus is on feedforward neural networks, a similar approach can be tested on other machine learning techniques, as long as the derivative of the model can be computed (e.g. in Gaussian Process Regression).

It is worth mentioning that the focus of this work is only on learning robots' forward kinematics, yet generalization to the inverse kinematics is also being investigated. Additionally, the proposed strategies are here validated on serial-link robotic manipulators because it is possible to easily simulate a ground truth model, and then better estimate the performances of the proposed approach.

The paper is structured as follows:

Section II describes robot kinematic modelling, control, and neural networks. Section III presents the proposed training strategies, and the experiments performed to test them; Section IV shows the modelling and control results using the proposed training strategies, and, finally conclusions are drawn in Section V.

II. Problem Formulation

In this section, a brief introduction of robot kinematic modelling and control, as well as an introduction to neural networks is presented.

A. Robot Kinematic Modelling

Robot kinematic modelling is typically described in the case of rigid-body robots, where forward kinematics refers to the mapping from joint space configuration $\boldsymbol{q} \in \mathbb{R}^{n_j}$, where the robot has n_j actuated joints, to task space [24]. However, in general, robot kinematic modelling finds the relationship between positional control variables $\boldsymbol{\theta} \in \mathbb{R}^{n_j}$ (e.g. motor positions), to the task space pose of the end-effector $\boldsymbol{T} \in \mathbb{R}^{4\times4}$, which is generally expressed with respect to a fixed reference frame. The end-effector's pose is a function of the positional control variables, $\boldsymbol{T}(\boldsymbol{\theta}) = \begin{bmatrix} \boldsymbol{R}(\boldsymbol{\theta}) & \boldsymbol{P}(\boldsymbol{\theta}) \\ \boldsymbol{0} & 1 \end{bmatrix}$, and describes the end-effector's position $\boldsymbol{P}(\boldsymbol{\theta}) \in \mathbb{R}^3$ and orientation $\boldsymbol{R}(\boldsymbol{\theta}) \in \mathbb{R}^{3\times3}$.

In this work, we consider task space position only, defined as:

$$\boldsymbol{P}(\boldsymbol{\theta}) = FK(\boldsymbol{\theta}). \tag{1}$$

The derivative of equation (1) with respect to time yields the forward kinematic mapping for velocity:

$$\dot{\boldsymbol{P}}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) = \boldsymbol{J}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}}, \tag{2}$$

where $\boldsymbol{J}(\boldsymbol{\theta}) = \frac{\partial \boldsymbol{P}}{\partial \boldsymbol{\theta}}$ is the Jacobian from $\boldsymbol{\theta}$ to $\boldsymbol{P}$, and is an $3 \times n_j$ matrix.

B. Robot Kinematic Control

For control purposes, having a model of the robotic system allows properties of traditional optimal controllers to be exploited, and thus guarantee stability of the system. In the majority of control tasks, the robot must find the optimal control variable space trajectory to achieve a desired end effector trajectory - something that can be achieved using inverse kinematics. The goal of the inverse kinematics is to compute the required control variables for the end-effector to follow a desired Cartesian trajectory, defined by $[\tilde{\boldsymbol{P}}(t), \dot{\tilde{\boldsymbol{P}}}(t)]$.

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

$$\boldsymbol{\theta}_t^* = \arg\min_{\boldsymbol{\theta}} \frac{1}{2}||\tilde{\boldsymbol{P}}_t - \boldsymbol{P}_t||^2 \ , \tag{3}$$

where $\tilde{\boldsymbol{P}}_t$ is the desired position at the instant t, and $\boldsymbol{P}_t = FK(\boldsymbol{\theta}_t)$ is the end-effector position computed with the forward kinematic model of the robot.

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

$$\begin{aligned} \dot{\boldsymbol{\theta}}_t^* &= \arg\min_{\dot{\boldsymbol{\theta}}} \frac{1}{2}||\dot{\tilde{\boldsymbol{P}}}_t - \boldsymbol{J}\dot{\boldsymbol{\theta}}||^2 \\ \text{and} \quad \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \dot{\boldsymbol{\theta}}_t^* \Delta t \ , \end{aligned} \tag{4}$$

where $\boldsymbol{J}$ is the Jacobian of the forward kinematic model, and Δt is the motion sampling time. For redundant manipulators, the minimum norm solution is usually chosen, which can be computed as:

$$\begin{aligned} \dot{\boldsymbol{\theta}}_t^* &= \arg\min_{\dot{\boldsymbol{\theta}}} \frac{1}{2}||\dot{\boldsymbol{\theta}}||^2 \\ \text{s.t} \quad \dot{\tilde{\boldsymbol{P}}}_t &= \hat{\boldsymbol{J}}\dot{\boldsymbol{\theta}} \ . \end{aligned} \tag{5}$$

The solution of both (4) and (5) can be directly computed by means of the Jacobian pseudoinverse $\boldsymbol{J}^\dagger$ as:

$$\dot{\boldsymbol{\theta}}^* = \boldsymbol{J}^\dagger \dot{\boldsymbol{P}} \ . \tag{6}$$

In case of redundant robots, however, the pseudoinverse only allows one particular solution to be found. The general solution can be expressed as a combination of these minimum control variable velocities $\dot{\boldsymbol{\theta}}^*$ and the component of the control variable velocities that are in the null-space of the robot $\boldsymbol{N}\dot{\boldsymbol{\theta}}_0$:

$$\dot{\boldsymbol{\theta}} = \dot{\boldsymbol{\theta}}^* + \boldsymbol{N}\dot{\boldsymbol{\theta}}_0, \tag{7}$$

where $\boldsymbol{N} \in \mathbb{R}^{n_j \times n_j}$ is the null space projector. $\boldsymbol{N}$ can be calculated from the Jacobian and its pseudo-inverse:

$$\boldsymbol{N} = \boldsymbol{I} - \boldsymbol{J}^\dagger \boldsymbol{J}. \tag{8}$$

C. Feedforward Artificial Neural Networks

Given a dataset of input points $\boldsymbol{x} \in \mathbb{R}^{n_{in}}$ and output points $\boldsymbol{y} \in \mathbb{R}^{n_{out}}$, the goal of regression is to find best the relationship between the two, meaning

$$\boldsymbol{y} \sim \boldsymbol{f}(\boldsymbol{x}) \tag{9}$$

where $\boldsymbol{f}(\cdot)$ can be any linear or nonlinear function. A very popular approach for nonlinear regression is the use of feedforward artificial neural networks (ANNs). It has been shown that neural networks can be regarded to as universal approximators, meaning they can model any suitably smooth function, given enough hidden units, to any desired level of accuracy [25], [26]. They are thus capable of representing complicated behaviours, without the need of any mathematical or physical prior model.

Feedforward networks consist of different layers of neurons. The first layer is the input layer, the last one is the output layer, and all the others in between are called hidden layers. Each layer has several neurons, each one receiving inputs from the neurons of the previous layer and sending an output to the neurons of the following layer. The final output of the neural network is then a nonlinear function of the input values, controlled by the nodes' weights. These weights can be retrieved by minimizing a desired cost function [27].

A great advantage of ANN compared to other machine learning approaches is 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.

For each network layer, with input $\boldsymbol{x}_i$ and output $\boldsymbol{y}_i$, the derivative of the output with respect to the input can be computed as

$$\frac{\partial \boldsymbol{y}_i}{\partial \boldsymbol{x}_i} = \frac{\partial \boldsymbol{y}_i}{\partial \boldsymbol{h}_i}\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{z}_i}\frac{\partial \boldsymbol{z}_i}{\partial \boldsymbol{x}_i}\ , \tag{10}$$

where $\boldsymbol{z}_i = \boldsymbol{W}_i\boldsymbol{x}_i + \boldsymbol{W}_{i,0}$, with $\boldsymbol{W}_i$ being the matrix of weights of the layer and $\boldsymbol{W}_{i,0}$ the biases, and $\boldsymbol{h}_i$ the activation function. The first two partial derivatives can be easily computed analytically once the activation function is chosen (for instance sigmoid), and $\frac{\partial \boldsymbol{z}_i}{\partial \boldsymbol{x}_i} = \boldsymbol{W}_i$. Because the total network is a cascade of layers, the final derivative of the network output with respect to the network inputs can be calculated analytically and iteratively by applying the chain rule to the derivatives of each layer.

III. Method

In this section, we present the proposed training strategies to learn a forward kinematic model of a robot and the experiments carried out to test each strategy.

A. Robot Kinematic Modelling with Neural Networks

In order to optimally control a robotic system using the techniques described in section II-B, both the forward kinematics of a robot should be computed, obtaining both a mapping from the control variables to the tip position, and information about the robot Jacobian. These are then used to perform inverse kinematics at velocity level, which can be numerically integrated to produce optimal control variable trajectories for the control problem. Since the control strategy is generally formulated at the velocity level as in (4) and (5), one possibility would be to use one single network to learn the Jacobian. However, information about the expected end-effector's position is also needed for control, especially in the case where no external sensors can be used for feedback compensation. Another approach would be to use two different networks [28]; one learning the mapping to the end-effector's position and one to the Jacobian, but this solution would be time consuming and suboptimal, as the two networks would not share any information about the dependencies between the two mappings. Furthermore, if an ANN is used to learn the inverse mapping directly, the robot end-effector's position needs to be used as input to the ANN. In many robotics applications, such as in surgery, measuring the end-effector's position is not possible, thus learning the inverse mapping would not be applicable.

Given a dataset of control variable positions and velocities and task space positions and velocities, $\mathcal{D} = \{\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}, \boldsymbol{P}, \dot{\boldsymbol{P}}\}$, a neural network with weights $\boldsymbol{W}$, can be used to approximate the forward kinematic model of a robot:

$$\hat{\boldsymbol{P}} = NN(\boldsymbol{\theta}, \boldsymbol{W}). \tag{11}$$

In the traditional, simplest case, this network is trained to minimise a mean squared error loss between predicted task space position and the actual task space position of the robot:

$$L_1 = \frac{1}{N}\sum_{i=1}^{N} ||\boldsymbol{P}_i - \hat{\boldsymbol{P}}_i||_2^2. \tag{12}$$

However, the neural network is differentiable with respect to its input, $\boldsymbol{\theta}$, so an estimate of the Jacobian of the robot, $\hat{\boldsymbol{J}}(\boldsymbol{\theta}, \boldsymbol{W})$ can also be obtained, and used to estimate the task space velocity of the robot $\hat{\dot{\boldsymbol{P}}} = \hat{\boldsymbol{J}}\dot{\boldsymbol{\theta}}$. A second loss term, mean squared error between predicted task space velocity and the actual task space velocity of the robot can then be calculated:

$$L_2 = \frac{1}{N}\sum_{i=1}^{N} ||\dot{\boldsymbol{P}}_i - \hat{\dot{\boldsymbol{P}}}_i||_2^2. \tag{13}$$

This additional task space velocity loss term allows the neural network to inherently learn the differential relationship of the forward kinematic model it is approximating.

Further information to the loss function can be obtained by exploiting the inverse kinematics mapping (6) and the ground measured control variables velocities $\dot{\boldsymbol{\theta}}$. The minimum norm expected control velocity can be computed as $\hat{\dot{\boldsymbol{\theta}}} = \hat{\boldsymbol{J}}^\dagger\dot{\boldsymbol{P}}$. This allows the formulation of the error between the minimum control variable velocities calculated from the predicted task space velocities and the minimum control variable velocities calculated from the actual task space velocities as:

$$\boldsymbol{e}_{\dot{\boldsymbol{\theta}}^*} = \hat{\dot{\boldsymbol{\theta}}} - \hat{\boldsymbol{J}}^\dagger\hat{\boldsymbol{J}}\dot{\boldsymbol{\theta}} = \hat{\boldsymbol{J}}^\dagger\dot{\boldsymbol{P}} - \hat{\boldsymbol{J}}^\dagger\hat{\dot{\boldsymbol{P}}}. \tag{14}$$

This is the projection of the predicted task space velocity error into the control variable space, $\hat{\boldsymbol{J}}^\dagger(\dot{\boldsymbol{P}} - \hat{\dot{\boldsymbol{P}}})$, and gives a third loss term:

$$L_3 = \frac{1}{N}\sum_{i=1}^{N} ||\hat{\boldsymbol{J}}^\dagger(\dot{\boldsymbol{P}}_i - \hat{\dot{\boldsymbol{P}}}_i)||_2^2. \tag{15}$$

We hypothesise that this projected velocity loss term provides the forward kinematic model with knowledge of the inverse kinematic velocity relationship of the pseudo-inverse of the Jacobian.

The effects of the three loss terms presented in (12), (13) and (15) are investigated in this paper. It is worth clarifying that the information on the Jacobian in (13) and (15) is directly added during training by analytical derivation of the network model.

B. Model Training Strategies

The loss function has direct effects on the modelling capabilities of the ANN, as it is responsible of the optimal tuning of the network wights. Generally, ANNs for robot modelling are trained only considering data on the control variables' positions and on the end-effector's position to directly learn either the forward or the inverse position mapping. This strategy, however, neglects useful information about the velocity mapping, which is instead highly beneficial for robot kinematic control.

For this purpose, we compare three strategies to train an ANN for forward kinematic robot model, using combinations of the three loss terms presented in Section II. The three training strategies are as follows:

1) Position loss (P Loss): The loss used in this training strategy is traditional position loss only,

$$L_P = L_1. \tag{16}$$

Important to note is that although the model is trained on position data only, the Jacobian of the neural network can still be calculated to predict task space velocity during evaluation. Yet, the Jacobian is only used for control purposes and not exploited in the training.

2) Position and velocity loss (PV Loss): This training strategy uses a weighted combination of the position and velocity loss terms, allowing the neural network to learn both the forward kinematic positional mapping and the velocity mapping:

$$L_{PV} = w_1 L_1 + w_2 L_2. \tag{17}$$

For simplicity, in this work we use $w_1 = w_2 = 0.5$.

3) Position, velocity, and inverse velocity loss (PVI Loss): This strategy combines the previous loss terms with inverse velocity loss to allow the neural network to inherently learn the robot's inverse kinematic velocity relationship:

$$L_{PVI} = w_3 L_1 + w_4 L_2 + w_5 L_3. \tag{18}$$

In this work, $w_3 = w_4 = 0.45$, $w_5 = 0.1$ from empirical trials.

IV. Results and Discussion

Two sets of experiments were completed in order to test the performance of each training strategy: robot model learning and robot control.

A. Robot Model Learning

In this work we tested the three proposed cost functions to learn the kinematic models of three different robotic structures (Figure 1): a 2DOF (RR) planar robot, a 4DOF (PRRR) robot, a 7DOF robot with all revolute joints. For the 2DOF and the 4DOF each link is 0.5 m long, and for the 7DOF all links are 0.5 m long, except for the last one being 0.125 m long.

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 controlling the robot at the joint level, commanding desired joint motions. In this work, the control variables are excited sinusoidally, with different combinations of frequencies, and such that their motion spans the whole joint range. The ground truth simulated robot models are used to collect the data of the corresponding end-effector position. In total 24000 data points were collected for the 2DOF robot, 48000 for the 4DOF, and 111486 for the 7DOF. It is worth noticing that the data are collected such to not explore the whole workspace, so as to also evaluate each strategy's capability of generalizing to unseen data points.

In order to train the networks with $PV\ Loss$ and $PVI\ Loss$, data for the control variable and end-effector velocity need to be collected too. These data can be retrieved by taking the time derivative of the data on the control variable and end-effector positions. In a real scenario, noise in the measurements can be accentuated by the derivation. However, both $PV\ Loss$ and $PVI\ Loss$ need the velocity data only for the offline training an not during the real time control, therefore noise impact can be effectively reduced with filtering. For this reason, the data for the training are here considered to be noise free.

For learning each robot model, the same datasets and the same optimization algorithm are used for each method. In all cases, Pytorch and ADAM optimizer [29] were used, with a learning rate of 10^{-3} and a maximum number of epochs set to $20 \cdot 10^3$ for the 2DOF and 4DOF, and $100 \cdot 10^3$ for the 7DOF. Moreover, for the 2DOF and the 4DOF the ANN has one single hidden layer with 30 neurons, whereas for the 7DOF it has 4 hidden layers with 60 neurons each. In all cases, however, $sigmoid$ activation function is used to ensure continuity of the derivatives.

Even though the data are collected with continuous motion, for the offline model learning the data points are randomly shuffled and split in a training and test set (80%-20%). Table I reports the Root Mean Squared Errors (RMSE) in the training and test sets for each robot model and for each proposed cost function.

For both the 2 and 4 DOF robot models, it can be noticed that including information about the velocity mapping results in lower errors at the velocity level ($RMSE_{\dot{P}}$), without much deterioration of the learning at position level ($RMSE_P$). For the 7 DOF, instead, all learning strategies perform similarly at both position and velocity level.

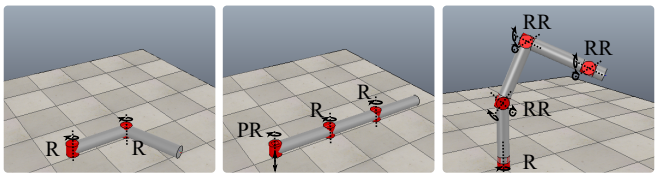


Fig. 1: The 2DOF, 4DOF, 7DOF robot models used for testing the three training strategies. P and R stand for prismatic and revolute joint respectively.

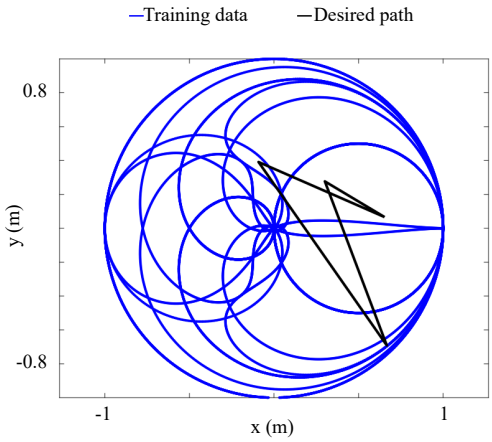


Fig. 2: An exemplary comparison for the 2DOF robot between the data points used to train the network and the desired paths for the control tests. The path covers both explored and unexplored datapoints.

B. Robot Control Tests

The models learnt through the proposed learning strategies are here tested to evaluate their performances in controlling the three robots to follow various desired paths. For each robot 5 tests are conducted to follow polygons with 3, 4, 5, 6, and 10 vertices, randomly selected in the 3D Cartesian space. Each side of the polygon is linearly discretized in 100 points and the control law defined in II-B is employed to compute the control variable commands to reach each desired point. Figure 2 shows an exemplary path for the 2DOF robot compared to the data used for training the network. As described in IV-A, the desired paths cover both explored and unexplored regions in the space, which is needed to evaluate the capabilities of generalizing of the three training strategies.

Moreover, in these tests we are assuming an open-loop control, namely no information about the actual end-effector position is exploited in the control. In addition, bounds on the control variable limits or self collisions are not considered in this work, but they could be easily included in the controller as constraints.

Figure 3 shows the tracking results for the different paths and for the different robots with the three proposed learning strategies, whereas Table II reports the RMSE. We evaluated the error between the desired $\tilde{\boldsymbol{P}}$ and the actual $\boldsymbol{P}$ end-effector position, obtained from the ground truth simulated model, $\epsilon_{\boldsymbol{P}} = \frac{1}{N_p}\sum_{n=1}^{N_p}||\tilde{\boldsymbol{P}}_n - \boldsymbol{P}_n||$, with N_p the total number of points in the path.

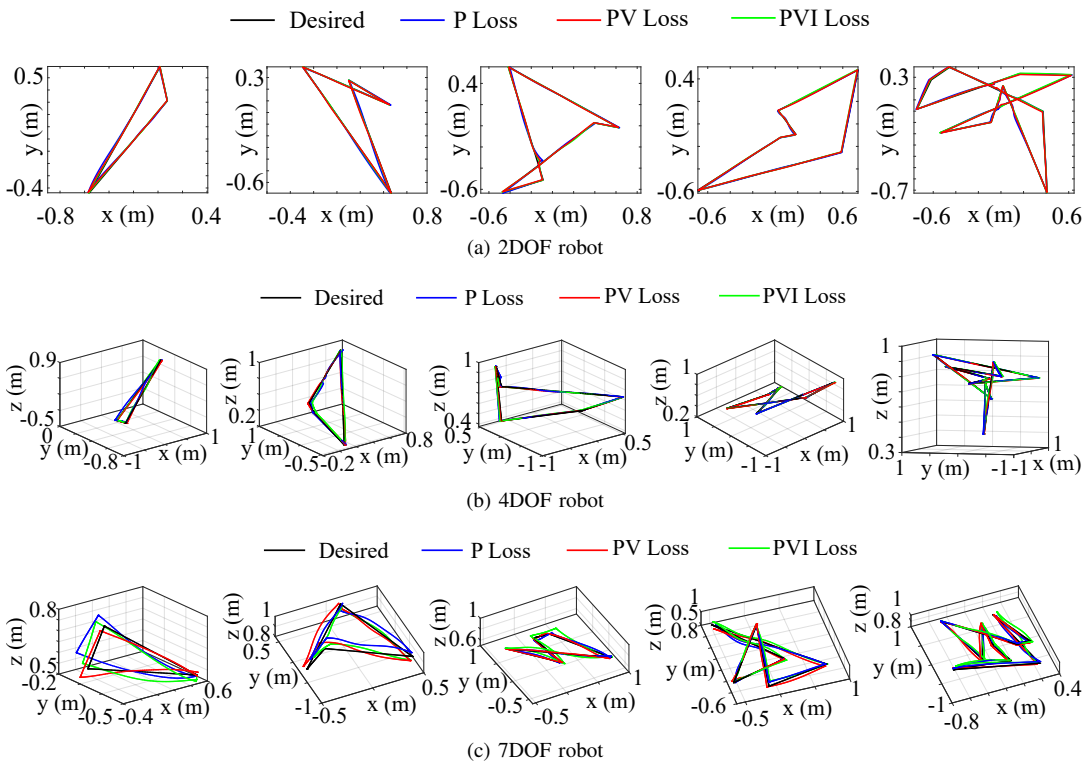
Even though the controller achieves small errors between the desired end-effector position and the expected one $\hat{\boldsymbol{P}}$, both from Figure 4 and Table II it can be seen that the models learnt with the two proposed $PV\ Loss$ and $PVI\ Loss$ generally yield better tracking results. The standard $P\ Loss$ achieves better results only in few cases for the 7 DOF arm. Overall $PV\ Loss$ is the one with best performances, with median position error values of $11.44,\ 17.98,\ 39.58$ mm for the 2 DOF, 4 DOF, 7 DOF robot respectively, compared to $13.51,\ 25.35,\ 52.25$ mm for the standard $P\ Loss$ and $10.780,\ 21.30,\ 49.64$ for the $PVI\ Loss$.

These results show that incorporating information about the velocity mapping leads to better control performances and $PV\ Loss$ seems to be sufficient to improve the tracking accuracy. $PV\ Loss$ resulted in a $15.33\%,\ 29.09\%,\ 24.25\%$ overall reduction of the end-effector position error for the 2 DOF, 4 DOF and 7 DOF robots compared to the standard $P\ Loss$. $PVI\ Loss$, instead, yielded $20.24\%,\ 15.96\%,\ 5.00\%$ improvement for each robot. One possible reason for $PV\ Loss$ outperforming the others is that including the velocity mapping in the loss means specifying information about the slope of the kinematic model at each datapoint. This, in turn, would result in smoother and more accurate kinematic model.

To assess the statistical significance of the results, we conducted a two-tailed Student t-test on the control tests for the 2, 4, and 7 DOF robots on the path tracking tasks. Figure 4 reports the p-values when comparing $P\ Loss$ against $PV\ Loss$, $P\ Loss$ against $PVI\ Loss$, and $PV\ Loss$ against $PVI\ Loss$, given the null hypothesis that the test results come from normal distributions with equal means with a significance level of 5%. For both the 2 and 4 DOF, results from $P\ Loss$ and $PV\ Loss$ are statistically significant and the null hypothesis can be rejected; for the 7 DOF, instead,

TABLE I: RMSE in the training and testing datasets for each training loss: $\mathbf{RMSE_P}$ is the error on the end-effector position in mm, and $\mathbf{RMSE_{\dot{P}}}$ on the velocity in mm/s.

	2DOF				**4DOF**				**7DOF**			
	$\mathbf{RMSE_P}$		$\mathbf{RMSE_{\dot{P}}}$		$\mathbf{RMSE_P}$		$\mathbf{RMSE_{\dot{P}}}$		$\mathbf{MSE_P}$		$\mathbf{RMSE_{\dot{P}}}$	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**	**Tr + ain**	**Test**
P Loss	3.5637	3.5214	0.1647	0.1665	**5.6152**	**5.6143**	1.9013	1.9222	**16.8671**	**17.6352**	**25.0870**	**25.3425**
PV Loss	**1.9494**	**1.9647**	**0.0571**	**0.0585**	9.3696	9.4218	1.2083	1.2124	17.2337	17.8885	26.0768	26.6458
PVI Loss	2.4174	2.4286	0.0750	0.0730	6.7020	6.7656	**1.0109**	**1.0398**	18.5578	19.2329	25.2916	25.6733



(a) 2DOF robot

(b) 4DOF robot

(c) 7DOF robot

Fig. 3: Actual and desired robots' end-effector positions for tracking the desired paths for: 3a) 2DOF robot; 3b) 4DOF robot; 3c) 7DOF robot.

TABLE II: RMSE between the desired and the actual tip position ϵ_P (in mm) for the control tasks on the different paths defined by the number of vertices for the different robots.

	2DOF			4DOF			7DOF		
#Vertices	P Loss	PV Loss	PVI Loss	P Loss	PV Loss	PVI Loss	P Loss	PV Loss	PVI Loss
3	14.6531	11.8384	**10.7767**	25.3530	**16.3475**	17.6197	52.2532	**36.5668**	38.5655
4	13.5108	**11.4391**	11.9525	20.6080	**17.2653**	24.7878	75.3516	**63.0311**	88.8702
5	13.1353	11.9717	**11.4862**	31.5394	**21.6719**	24.8936	**32.7040**	39.5821	58.4742
6	10.9540	**9.0664**	9.9368	28.2653	**19.3334**	19.9761	56.1324	46.2807	**43.9248**
10	13.8671	10.6835	**8.9751**	22.6860	**17.9754**	21.3022	**31.8849**	33.5757	49.6399

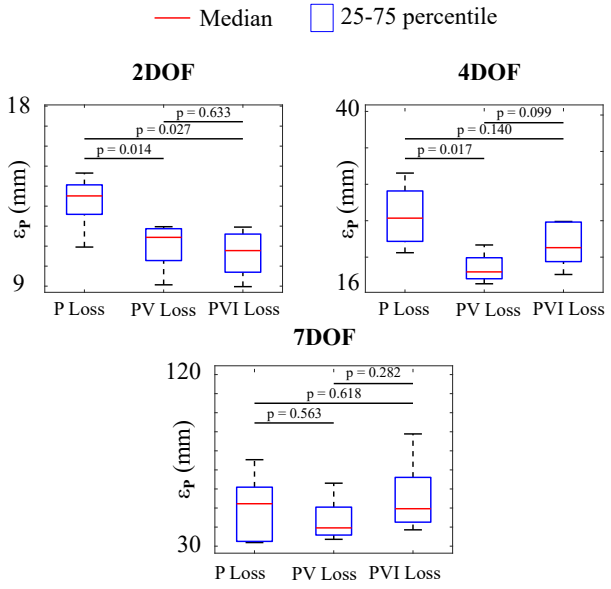


Fig. 4: Overall results of the end-effector positioning errors from the three learning strategies for the three different robots on the given paths.

there is not a large statistical significance.

All the learning strategies show worse performances in the tracking task for the 7DOF robot, due to larger modelling inaccuracies compared to the other models.

V. Conclusions and Future Work

In conclusion, in this work we propose and compare three different learning startegies to train ANNs for robot kinematic model learning. The focus has been on learning the forward kinematics as this, opposed to learning the inverse kinematics directly, has different advantages such as:

- does not necessarily need information about the tip position for control purposes, which might not be available;
- allows the exploitation of the Jacobian matrix and optimal control techniques.

The three strategies consist in using different loss functions during the training of the ANN an we compared the standard $P\ Loss$ with the two novel losses $PV\ Loss$ and $PVI\ Loss$, that additionally include information about the froward velocity mapping and inverse velocity mapping. Simulation results on different robots and the control tests to track

various paths show that including the information about the velocity mapping leads to better performances and smaller tracking errors, with both $PV\ Loss$ and $PVI\ Loss$ having higher tracking accuracy than $P\ Loss$.

However, further investigation is needed to better understand the effect of the inverse mapping in the training. In fact, $PV\ Loss$ is the one that performs the best, showing that just including the forward kinematic velocity mapping is sufficient to achieve better performance.

Even though the networks were trained in a simulation environment on noise-free data, this is not a limitation as the position and velocity data is needed only for the offline training, so the effect of noise can be reduced with filtering.

We believe this work could improve the kinematic modelling of a range of robots, particularly those that are too complex to model analytically. This is especially appealing for applications where control is of huge importance, but robot models are complicated, such as in surgical or soft robotics. Future work will focus on investigating more thoroughly the effects of the proposed loss functions, including more complex techniques for weighting each loss term, such as adaptive weighting during training. Furthermore, additional tests on more complex real robots, where modelling is challenging, also including the end-effector orientation, will be performed.

REFERENCES

[1] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: A survey," pp. 319–340, 11 2011.

[2] Z. Guo, W. Ren, J. Huang, and C. Wang, "A reinforcement learning approach for inverse kinematics of arm robot," in *ACM International Conference Proceeding Series*. New York, NY, USA: Association for Computing Machinery, 7 2019, pp. 95–99.

[3] A. Perrusquía, W. Yu, and X. Li, "Multi-agent reinforcement learning for redundant robot control in task-space," *International Journal of Machine Learning and Cybernetics*, vol. 12, no. 3, pp. 231–241, 1 2020.

[4] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 9 2013.

[5] J. Peters and S. Schaal, "Learning to Control in Operational Space," *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 197–212, 2 2008.

[6] B. Yu, J. d. G. Fernández, and T. Tan, "Probabilistic Kinematic Model of a Robotic Catheter for 3D Position Control," *Soft Robotics*, vol. 6, no. 2, pp. 184–194, 4 2019.

[7] T. G. Thuruthel, E. Falotico, M. Cianchetti, and C. Laschi, "Learning Global Inverse Kinematics Solutions for a Continuum Robot," in *CISM International Centre for Mechanical Sciences, Courses and Lectures*. Springer International Publishing, 2016, vol. 569, pp. 47–54.

[8] A. D'Souza, S. Vijayakumar, and S. Schaal, "Learning inverse kinematics," in *IEEE International Conference on Intelligent Robots and Systems*, vol. 1, 2001, pp. 298–303.

[9] 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 *IEEE International Conference on Fuzzy Systems*, vol. 2019-June. Institute of Electrical and Electronics Engineers Inc., 6 2019.

[10] B. Bocsi, D. Nguyen-Tuong, L. Csato, B. Scholkopf, and J. Peters, "Learning inverse kinematics with structured prediction." Institute of Electrical and Electronics Engineers (IEEE), 12 2011, pp. 698–703.

[11] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: a survey," *Cognitive Processing*, vol. 12, no. 4, pp. 319–340, 11 2011.

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

[13] R. F. Reinhart and J. J. Steil, "Hybrid Mechanical and Data-driven Modeling Improves Inverse Kinematic Control of a Soft Robot," *Procedia Technology*, vol. 26, pp. 12–19, 2016.

[14] J. M. Bern, Y. Schnider, P. Banzet, N. Kumar, and S. Coros, "Soft Robot Control with a Learned Differentiable Model," in *2020 3rd IEEE International Conference on Soft Robotics, RoboSoft 2020*. Institute of Electrical and Electronics Engineers Inc., 5 2020, pp. 417–423.

[15] 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, p. 68, 8 2020.

[16] C. Salaün, V. Padois, and O. Sigaud, "Control of redundant robots using learned models: An operational space control approach," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009*, 12 2009, pp. 878–885.

[17] S. Chiaverini, G. Oriolo, and I. D. Walker, "Kinematically Redundant Manipulators," in *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 245–268.

[18] M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, and S. Ho, "Lagrangian Neural Networks," *arXiv*, 3 2020.

[19] M. Lutter, C. Ritter, and J. Peters, "Deep Lagrangian Networks: Using Physics as Model Prior for Deep Learning," *7th International Conference on Learning Representations, ICLR 2019*, 7 2019.

[20] F. Cursi and G.-Z. Yang, "A Novel Approach for Outlier Detection and Robust Sensory Data Model Learning," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 11 2019, pp. 4250–4257.

[21] F. Cursi, V. Modugno, and P. Kormushev, "Model predictive control for a tendon-driven surgical robot with safety constraints in kinematics and dynamics," *International Conference on Intelligence Robots and Systems (IROS)*, 7 2020.

[22] N. Kumar, V. Panwar, N. Sukavanam, S. P. Sharma, and J. H. Borm, "Neural network-based nonlinear tracking control of kinematically redundant robot manipulators," *Mathematical and Computer Modelling*, vol. 53, no. 9-10, pp. 1889–1901, 5 2011.

[23] H. Sadjadian, H. D. T. Member, and A. Fatehi, "Neural Networks Approaches for Computing the Forward Kinematics of a Redundant Parallel Manipulator," *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 2008.

[24] T. G. Thuruthel, Y. Ansari, E. Falotico, and C. Laschi, "Control Strategies for Soft Robotic Manipulators: A Survey."

[25] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1 1991.

[26] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303–314, 12 1989.

[27] C. Bishop, *Neural Networks for Pattern Recognition*. Clarendon Press, 1995.

[28] 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 *IEEE International Conference on Intelligent Robots and Systems*. Institute of Electrical and Electronics Engineers Inc., 12 2018, pp. 5125–5132.

[29] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 12 2015.