IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 8 November 2022, accepted 25 December 2022, date of publication 4 January 2023, date of current version 9 January 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3234104

RESEARCH ARTICLE

A Deep Learning Approach to Navigating the Joint Solution Space of Redundant Inverse Kinematics and Its Applications to Numerical IK Computations

CHI-KAI HO, LI-WEI CHAN, CHUNG-TA KING, (Senior Member, IEEE), AND TING-YU YEN
Department of Computer Science, National Tsing Hua University, Hsinchu 30013, Taiwan

Corresponding author: Chi-Kai Ho (chikaiho@gmail.com)

This work was supported in part by the Ministry of Science and Technology, Taiwan, under Grant MOST 109-2218-E-007-023; and in part by the Information and Communications Research Laboratories of Industrial Technology Research Institute, Taiwan.

ABSTRACT As an increasing number of robotic manipulators possess seven or more degrees-of-freedom (DoF), solving inverse kinematic (IK) for kinematically redundant manipulators is becoming critical. Numerical optimizations are commonly used to solve the problem due to their generality and accuracy. Unfortunately, they typically only generate one joint solution at a time, despite the multiple joint configurations that redundant manipulators can provide to move the end-effector to a target position. The long iterative optimization process is also a concern, particularly if extra constraints such as obstacle avoidance have to be evaluated. In this paper, we show that numerical methods may be complemented by deep learning to overcome these limitations. Through deep learning, the solution space of redundant IK may be learned with neural networks (NNs), which allows multiple distinct joint solutions corresponding to a given target position to be obtained by navigating the solution space. The main challenge is to overcome the one-to-one functional mapping of NNs. This paper solves this problem with a novel probabilistic encoding of manipulator poses and their corresponding infinite number of joint solutions. Two examples are presented to demonstrate the application of the proposed method to facilitate numerical IK computations: (1) finding a good initial joint solution to bootstrap the numerical IK calculation, and (2) evaluating extra constraints, such as obstacle avoidance, off the optimization iterations. Experiments show that the proposed method can accelerate the execution of different numerical IK modules in the popular IKpy package up to 50% for a 7-DoF manipulator, depending on the accuracy required.

INDEX TERMS Feature encoding, inverse kinematics, redundant robotic manipulators, unsupervised learning.

I. INTRODUCTION

A robotic manipulator is kinematically redundant if it possesses more degrees of freedom (DoF) than that required to execute a given task. Redundancy gives the manipulators more flexibility in executing the same task at the end-effector level in different ways at the joint level [7]. Let x denote the m-dimensional coordinate of the end-effector in the workspace required by the task and θ the n-dimensional joint configuration. A manipulator is redundant if $m < n$. Mathematically, this means that the *inverse kinematic* (IK) function, which calculates the joint configuration θ by given a workspace coordinate x,

$$\theta = f^{-1}(x) \tag{1}$$

has an infinite number of solutions for the given x, where $f()$ is the *forward kinematic* (FK) function calculating $x = f(\theta)$.

The associate editor coordinating the review of this manuscript and approving it for publication was Amjad Ali.

Solving the IK function is a key computation in robot control. If the accuracy of the solutions is a major concern, then the IK problem can be solved by two general approaches. Analytical methods compute the joint solutions in closed-form expressions based on the structure of the manipulator [3], [23], [31], [46]. They can obtain joint solutions accurately and fast. However, for redundant manipulators, the closed-form expressions are difficult to derive and often rely on special measures such as parameterization of redundant dimensions to obtain solutions, which limit the applicability of analytic methods.

Numerical methods, on the other hand, solve the IK function through an iterative optimization process, e.g., by gradient descent or Newton-like root-finding methods, to obtain one joint solution [5], [7], [30], [39], [42]. They can be applied to arbitrary manipulators with high accuracy. For example, the popular IKpy package [28], which includes many numerical IK solvers, can achieve a precise up to 7 digits. However, the iterative optimization process often takes a considerable amount of time to compute. Besides, numerical methods were traditionally targeted at obtaining one joint solution at a time. A common practice for redundant manipulators to resolve the infinite number of joint solutions of a pose to just one is to impose extra constraints or optimization directions [7], [39]. It not only increases optimization complexity and prolongs the computation time, but also limits the full flexibility inherent in redundant manipulators.

While numerical methods strive for single solutions, there are increasing demands for exploiting the IK solution space to discover multiple joint solutions corresponding to the given pose of the end-effector. In other words, instead of constraining the optimization process towards one solution, it is often desirable to gather all or many joint solutions first and then choose the ones that satisfy the requirements. The latter is normally faster and easier than formulating the constraints into the numerical formulations. Unfortunately, as far as we know, there is no efficient and systematic way of finding multiple distinct joint solutions for a given pose with numerical methods. One direction is to traverse the null space of the pseudo inverse Jocobian matrix [39] and another is to start the numerical calculations with different starting poses. In either case, extra computations are incurred without guaranteed computation time.

In this paper, we propose to leverage the modeling capability of neural networks (NNs) to assist numerical methods for flexible control of kinematically redundant manipulators. Neural networks are parameterized functions that can approximate any continuous function. Thus, by precomputing a sufficient number of pose and joint solution pairs, i.e., (x, θ), it is possible to learn the solution space of Eq. (1) with a NN. Ideally, given a target pose, the NN can generate multiple distinct joint solutions to the IK function. By filtering out those that do not satisfy the task requirements, one or a few solutions may be selected to go through the numerical methods to obtain the final joint solutions that are accurate to the required precision. In this way, the complex computation of task requirements can be moved out of the optimization iterations of the numerical methods and carried out once after joint solutions to the target pose are generated with the NN. The computation time of the NN should be quite short, because the NN learns from precomputed solutions.

However, there are two challenges if the above ideas are to be realized. First, although NNs can learn one-to-one mapping to approximate any continuous function, they cannot approximate the ill-posed redundant IK function in Eq. (1) well. To train the NNs to perform one-to-many mapping is very difficult, because the multiple outputs may compete for updating the weights of the NN, causing the training to fail. Although there is a large body of data-driven research that leveraged neural networks (NNs) to learn the IK function [1], [10], [11], [12], [13], [29], [36], [41], [44], most of them avoided addressing the one-to-many mapping problem of redundant IK by focusing instead on generating only one joint solution. Some papers employed customized training datasets that admitted only one joint solution for one desired pose [1], [10], [17], while others imposed extra constraints to reduce the infinite number of joint solutions of a given pose to one [14], [37], [44]. As far as we know, there is no NN-based work that addresses this challenge with a single neural network.

The second challenge is to identify and track the ''distinct'' joint solutions to a given pose of the end-effector, even if we can navigate the solution space. Note that there are potentially an infinite number of joint solutions corresponding to a target pose. Many solutions are very similar and there is no need to duplicate the computation to process them. A systematic way of calculating distinct joint solutions of a given pose is needed.

This paper proposes a novel deep learning approach, called *Probabilistic Selective Inverse Kinematics* (PSIK), to modeling the redundant IK solution space. To address the one-to-many mapping in redundant IK, an extra index, called *posture index*, is introduced to map probabilistically to a joint solution of the desired pose. By varying the index, it is possible to navigate the solution space to discover different joint solutions. The posture indices are actually feature vectors characterizing the poses and the corresponding joint solutions, and they can be learned via *Variational Auto-Encoder* (VAE) [20], [40]. To track the distinct joint solutions, the solutions to a pose are first clustered and the posture indices corresponding to the different clusters are extracted and stored into an auxiliary dictionary to work together with the trained NN.

We consider 7-DoF robotic manipulators in this paper. Note that 6-DoF manipulators can theoretically move the end-effector to any position in the Cartesian workspace with any orientation. They are thus considered as ''general-purpose'' nonredundant manipulators [39]. In practice, however, joint range limitations, workspace obstacles, and kinematic singularities may cause barrier regions that the manipulator cannot reach. To increase dexterity and

robustness, many commercial manipulators were purposely made redundant with at least 7 DoFs, e.g., Franka Emika Panda, Kuka LBR iiwa, Rethink Robotics Sawye, etc. We therefore focus on ''general-purpose'' redundant manipulators with 7-DoF. We also assume that the manipulator has only one end-effector, e.g., a gripper, which is located at the end of the link chain.

Note also that PSIK can generate all valid solutions to the IK function. Unfortunately, their accuracy cannot match that of the numerical methods at this stage. This is because the cost is prohibitively high to sample a training dataset that is sufficiently dense to reach that level of accuracy, even for 7-DoF manipulators. Nevertheless, the PSIK models can extend their capability of modeling the IK solution space to assist numerical IK computations. Two example applications are presented to demonstrate their effectiveness: (1) finding a good initial joint solution to bootstrap the numerical IK calculations, and (2) evaluating secondary tasks, such as obstacle avoidance, off the optimization iterations of numerical IK.

The main contributions of the paper are as follows:

- A novel deep learning approach is proposed that can model the ill-posed redundant IK solution space to find multiple distinct joint solutions of a given pose. To the best of our knowledge, this is the first work to use a single NN to generate multiple joint solutions for 7-DoF manipulators. This opens up a new direction in researching NN-based IK methods.
- An automated procedure is presented to collect the training dataset. Only sparse data are needed to train a NN that can closely model the solution space of redundant IK. Training of the NN model only needs to be done once after the manipulator is developed.
- Effectiveness of PSIK in facilitating numerical IK computations is demonstrated with two example applications. Experiments based on the popular IKpy package show that PSIK can accelerate the execution of the different numerical IK modules in IKpy up to 50% for a 7-DoF manipulator, depending on the accuracy required.

The remainder of the paper is organized as follows. Sec. II introduces related works, followed by Sec. III on background of this paper. Sec. IV describes the proposed PSIK method, whose applications are discussed in Sec. V. Experiments and results are shown in Sec. VI. Conclusions are drawn in Sec. VII.

II. RELATED WORK

In this section, we review briefly the different approaches to solving the IK function, with emphasis on handling the redundant IK and exploiting the multiple joint solutions to a given pose of the end-effector.

Analytic methods derive closed-form expressions for solving the IK function with high accuracy [3], [23], [31], [46]. For redundant IK, the derived expressions can readily be used for obtaining multiple joint solutions for a given pose. Particularly, most analytic methods for redundant manipulators relied on parameterization of the redundant dimensions to resolve the ill-posed IK function. Hence, by applying different values to the introduced parameters, it is possible to obtain different joint solutions corresponding to the given pose. The primary shortcoming of analytical methods is that the closed-form expressions are very complex and difficult to derive, requiring full knowledge of the kinematic structure of the manipulators. Besides, it is unclear how to systematically change the parameters to generate distinct joint solutions of the same pose.

Numerical methods, on the other hand, were traditionally targeted at obtaining one joint solution of the IK function [5], [7], [30], [39], [42]. The most popular numerical approach is based on the Jacobian matrix that finds a linear approximation to the movement of the end-effector relative to instantaneous changes in the joints. The IK function may be solved by the inverse of the Jacobian matrix through gradient descent or Newton-like root-finding methods following an iterative optimization process. They can be applied to arbitrary manipulators with high accuracy. For example, the popular IKpy package [28] includes many numerical IK solvers to achieve a precise up to 7 digits. There are also heuristics that move each joint of the manipulator iteratively to minimize the deviation of the end-effector [2], [43]

For redundant manipulators, numerical methods often relied on redundancy resolution techniques [7], [39] that imposed extra constraints in the optimization iterations to reduce the infinite number of joint solutions of a pose to just one. Since they were not designed to exploit the joint solution space, as far as we know, there is no efficient and systematic way of finding multiple distinct joint solutions for a given pose. One direction is to traverse the null space of the pseudo inverse Jocobian matrix [39] or to start the numerical calculations with different seeds. In either case, extra computations are incurred without guaranteed computation time.

Data-driven methods have been applied to study the joint solution space and various properties of the IK function [33], [35]. Using the sampled data to train neural networks, we can obtain more concise modeling of the joint solution space and approximation of the IK function, which are useful for facilitating manipulator task executions. Such efforts can be dated back to the 1990s [12], [13], [32]. For example, in [12], a set of NNs were trained to identify the finite set of solutions to the IK problem for a non-redundant manipulator. The purpose is to provide the ability to choose a particular solution at run time. In [13], the techniques were extended to redundant manipulators to obtain neural networks to approximate the IK function. Our work shares similar goals, but aims to train a single neural network for more complex 7-DoF manipulators.

Most later NN-based research mainly focused on the functional approximation characteristic of NNs and tried to train NNs to approximate the IK function [1], [10], [11], [14], [15], [17], [29], [36], [37], [44]. Unfortunately,

as an approximation, the trained NNs can hardly achieve an accuracy comparable to that of the numerical methods. Furthermore, for redundant manipulators, the one-to-one functional mapping nature of NNs also has difficulty in approximating the ill-posed redundant IK function in Eq. (1). For example, the work in [11] evaluated different NN architectures in modeling the joint solution space for manipulators with different DoFs, and concluded that the pattern-free input data make it quite difficult for NNs to learn the high-dimension mapping function between end-effector poses and required joint values.

To work around, some papers employed customized training datasets that admitted only one angle solution for one desired pose [1], [10], [17], and some imposed extra constraints to reduce the infinite number of joint solutions of a given pose to one [9], [14], [37], [44]. It turns out that the NNs developed in these works can at best provide a crude model of the joint solution space and produce only single solutions. This also makes it difficult to traverse the solution space to obtain different joint solutions for the same target pose. Works such as redundancy circle [9] fixed one joint, e.g., the elbow, and allowed it to rotate freely, while using the remaining 6 joints to arrive at a unique joint solution to satisfy the target pose. Unfortunately, the solution space that they can exploit is limited, within one self-motion manifold of certain poses. In this paper, we consider modeling and navigating the entire unconstrained, task-agnostic joint solution space and address directly the one-to-many mapping in IK for redundant manipulators.

In [29], a new method for learning a mapping between redundant states and low-dimensional postures was proposed. The work considered a high-DoF, complex musculoskeletal robot and attempted to find sets of internal pressures of pneumatic artificial muscles to meet a target position. An auto-encoder architecture was employed to handle the training data and supervised learning was adopted to learn known low-dimensional corresponding vectors. To collect the training data, they moved the arm first to the designated positions and randomly gave different pressures to obtain multiple solutions. Although the proposed method can generate different solutions to the same positions, the positions still need to be on the designated trajectories. In contrast, we focus more on the poses in the entire workspace and address the multiple-solution issue in the whole workspace.

III. BACKGROUND

In this section, some background information related to the proposed method is discussed. Recall that PSIK adopts a deep learning approach to training a neural network with *Variational Auto-encoder* (VAE) [20] that can model the ill-posed redundant IK solution space and navigate the space to obtain multiple distinct joint solutions to a given pose. Such a neural model is shown to be useful for facilitating the numerical IK computations.

A. REDUNDANT IK SOLUTION SPACE

Although the IK function of a kinematically redundant manipulator is ill-posed, it has very important application in practice, allowing the manipulator to move without displacing the end-effector. Such a property is referred to as *self-motion* [4], [7], which gives the manipulators more flexibility in executing the same task at the end-effector level in different ways at the joint level or in maintaining operation robustness.

Joint configurations corresponding to a given end-effector pose can be represented as a finite set of disjoint continuous manifolds in the configuration space, called the *self-motion manifolds* [4]. The end-effector remains motionless as the manipulator moves along these manifolds. Various properties of the joint solution space for kinematically redundant manipulators have been investigated extensively in the past, including the number of self-motion manifolds, their geometry, and the relationship to kinematic singularities [4], [27], [35].

B. DEEP GENERATIVE MODELS

The architecture of PSIK is based on a deep generative model, *Variational Auto-encoder* (VAE). In principle, one can use any generative model, e.g., GANs [18] and normalizing flow models [21], to model the joint solution space for navigating. However, other methods may have difficulties in making the training process to converge [34], so we chose VAE for its simplicity just to examine our idea. Below, we first give a brief introduction to deep generative models and then VAE.

Deep generative models view any phenomenon in the world under a certain probabilistic distribution. The goal is to approximate the hidden distribution of the desired target by using a finite dataset to train a parametric model. With the learned model, it is then possible to perform various downstream tasks based on the learn model of the phenomenon. For example, it is hard to identify a human from raw pixels. However, we know that the image of a human image is comprised of some basic artifacts, such as eyes, hair, skin color, etc. Deep generative models extract and describe these basic artifacts with known distributions so that we can exploit these statistical artifacts to estimate an instance of a human [26], [45] and even generate a new instance [38].

Under the context of modeling the redundant IK solution space, our insight is that it is possible to decompose the joint solution space into several control artifacts, and the redundancy of the manipulator is just one or several of the control artifact. An control artifact could refer to the angle between the plane of the elbow joint and the vertical plane, and another control artifact could refer to the orientation of the end-effector. In other words, to deal with redundant manipulators and multiple distinct solutions, we can directly apply deep generative models with a sufficient number of control artifacts to describe the hidden distribution of the manipulator's IK solution space. Ideally, if we know

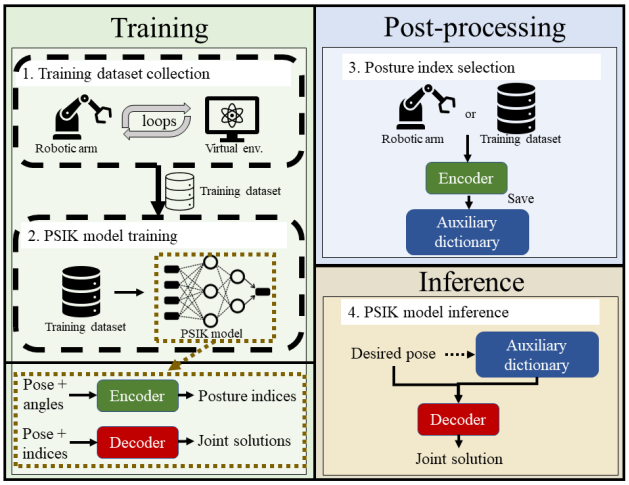


FIGURE 1. An overview of the proposed PSIK method.

the role of every learned artifact, we can even obtain a specific joint solution with respect to the desired pose of the end-effector by giving appropriate values in each artifact. The artifacts mentioned here are exactly the *posture index* introduced in Sec. I. In the following discussions, the latter will be used, because it is more straightforward to associate posture indices with different appearances of the manipulator.

C. VARIATIONAL AUTO-ENCODER

Variational auto-encoder [20] (VAE) is a generative model with stochastic latent variables. The objective of VAE is to approximate the distribution of the observations $p(x)$. To do so, VAE introduces a latent variable $z \sim g(z)$, namely artifacts, and a likelihood $p(x|z)$ to model $p(x)$. VAE assumes the latent variable z obeys a known prior distribution $g(z)$, e.g., standard Gaussian distribution. However, directly evaluating the integral of $p(x|z)g(z)$ is intractable, so a proposal distribution $q(z|x)$ is also introduced. In deep learning, we commonly regard $p(x|z)$ as a decoder and the proposal distribution as an encoder. All together, the final loss function is shown below.

$$Loss = -\int_z q(z|x) \cdot \log p(x|z)\, dz + KL(q(z|x)\|Nor(0, 1)), \tag{2}$$

where x is the observation, and z is the latent vector.

IV. APPROACH

In this section, the proposed PSIK (Probabilistic Selective Inverse Kinematics) architecture is introduced. The overall goal is to develop a novel architecture that can navigate the joint solution space of the IK function for kinematically redundant manipulators to supply multiple joint solutions at any pose in the whole workspace. Such a capability is very useful for assisting numerical IK calculations, which will be discussed in the next section.

A. OVERVIEW OF PSIK

Fig. 1 gives an overview of the proposed PSIK system. There are three parts. In the training stage, we first collect an unbiased training dataset across the entire workspace of the manipulator. This step is straightforward and can be done with a few lines of code as described in Sec. IV-B. The collected dataset is a multi-modal dataset, in which a pose may have multiple joint solutions. Such a dataset cannot be used to train NNs, which in essence are one-to-one mapping functions. PSIK overcomes this limitation with a novel probabilistic NN architecture, which will be presented in Sec. IV-C. The key idea is to extract characterizing features of poses and their corresponding joint solutions, and identify signature features, called *posture indices*, that can characterize the self-motion manifolds of the poses.

In the second stage, an *auxiliary dictionary* is introduced to help track signature posture indices and important properties of the solution space. Finally, in the inference stage, given a desired pose, the associated posture indices in the auxiliary dictionary are examined and one is selected (see Sec. IV-D). Next, the desired pose and the selected posture index are fed into the trained neural model to produce the final joint solution, as introduced in Sec. IV-E. Note that the different posture indices of a given pose will generate different postures for the manipulator. We thus have a systematic way of generate distinct postures for a given pose. Also, by altering the posture index, it is now possible to navigate the joint solution space.

B. UNBIASED TRAINING DATASET COLLECTION

PSIK adopts deep generative models and thus needs a training dataset for offline NN training. As mentioned earlier, most existing NN-based methods employed a biased training dataset permitting only one joint solution for each desired pose. In this way, the NN models can be trained successively as one-to-one mapping functions. The problem is that the training dataset has to be specially and manually designed, which hinders the automation of the data collection process. Furthermore, if the applications or requirements are changed, new dataset has to be collected and the process has to be repeated.

On the other hand, this paper aims to solve the one-to-many mapping from a desired pose to the multiple joint solutions for kinematically redundant manipulators. Therefore, we only collect an unbiased multi-model dataset across the entire workspace of the manipulator. The idea is straightforward by moving each joint of the manipulator in turn for a fixed displacement, say x_i degrees for a rotating joint i, across the working range of that joint. For each movement of a joint, the configuration of all the joints is recorded and the corresponding pose of the end-effector is obtained by forward kinematics. After all joints are turned across their working range in sequence, an unbiased dataset that covers the entire workspace of the manipulator can be collected. Note that in the dataset, a pose may have multiple

joint solutions to reach it. The size of the dataset and the density of the known poses depends on the displacement x_i to collect the data. Hence, the displacements are important design parameters in PSIK.

C. PSIK MODEL TRAINING

A typical NN model is a parameterized one-to-one mapping function. However, the training dataset collected in the previous subsection is multi-modal, in which a pose has multiple joint solutions to map to. It is not possible to learn a set of parameters for the NN model to output all solutions at once, causing the training to fail [11]. In this paper, we propose to add an extra index vector to the input layer so that the NN function maps from a given desired pose and an index to an joint solution. This solves the one-to-many mapping problem.

Since this index vector, called *posture index*, helps to walk through the infinite number of joint solutions for a given pose, it must be continuous. Also, all poses of the manipulator must have their own posture index, and adjacent poses must have similar posture index values. Therefore, it is critical how posture index should be represented and learned from the collected unbiased dataset. In this paper, we propose to encode the posture index with probability distributions. The posture index now covers a range of values instead of discrete points.

To implement PSIK, we adopt *Variational Auto-encoder* (VAE) [20], [40]. The VAE has the important advantage of approximating posterior with continuous latent variables. Fig. 2 shows the architecture of PSIK. The encoder is trained to extract the features of poses and their corresponding joint solutions into a latent vector characterizing by probability distributions, e.g., normal distributions. Let the dimensionality of the latent vector be k. Then, the encoder will output k sets of means μ and variances σ. Next, the decoder samples the k sets of means and variances to obtain a posture index and converts the posture index with the target pose back to the joint solution. It should be noted that we do not really sample latent variables, because backpropagation cannot handle sampling. Instead, the reparameterization technique is used to implement the idea. The original equation in VAE [20] is modified slightly by adding extra conditions. The loss function is as follows:

$$Loss = -\int_z q(z|g,x)\cdot \log p(x|g,z)\,dz + KL(q(z|g,x)\|Nor(0,1)), \quad (3)$$

where g is the target pose, x is the corresponding joint solution, and z is the posture index. The objective is to minimize the loss.

D. POSTURE INDEX SELECTION

In Sec. IV-C, we introduce the proposed PSIK architecture and how it associates a desired pose with multiple joint solutions. The architecture has an encoder for generating posture indices and a decoder for obtaining different joint

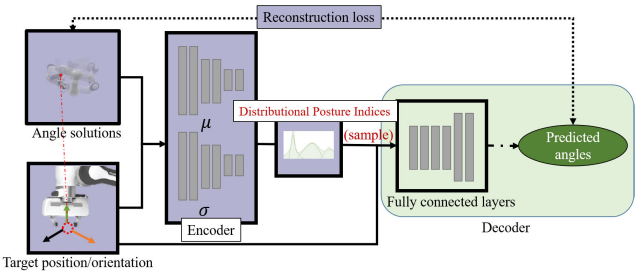


FIGURE 2. Implementation of the PSIK neural model.

solutions. During inference, the decoder receives a pose and a posture index to generate a joint solution. The problem is that the values of the posture index cannot be set arbitrarily. Not any posture index would generate a meaningful posture for the manipulator. Being a means to walk through the self-motion manifolds of a given pose while being consistent across adjacent poses, the posture index must have some *signature* values, which allow distinct postures of a given pose to be identified and represented. In addition, different poses will have a different set of signature posture indices.

To track the signature posture indices for different poses, we propose in PSIK to maintain an auxiliary dictionary to store poses and their corresponding signature posture indices. Although poses and posture indices are continuous across the workspace of the manipulators, the continuity in PSIK NN models ensure that adjacent poses have similar joint solutions and posture indices. Therefore, a sparse and discrete dictionary can serve the purpose. The question is how to learn and obtain the signature posture indices. There are various ways that this can be done.

In PSIK, we adopt the strategy that leverages the training dataset. After the encoder is trained, we feed all the poses collected in the training dataset to the encoder once more to obtain the corresponding posture indices. Then, a simple clustering is applied to identify those posture indices that are sufficiently apart as the signature indices. Next, a KD-tree and a dictionary data structure are used to store the signature posture indices of each pose that appears in the training dataset, as shown in Fig. 3. Additional information may be stored in the dictionary, for example, the available ranges of the posture index for a given pose, the distance to singularities, the self-motion manifold id, and so on [13], [33], [35]. In the next subsection, we will discuss how this auxiliary dictionary can be used. Note that this procedure does not exclude at all the possibility that new entries are added into the dictionary dynamically as we gather more data during the operations of the manipulator.

Note also that it is possible to skip querying the auxiliary dictionary for posture indices and instead to leverage the current or initial state of the manipulator to find the posture index for the target pose. The idea is to feed the current joint configuration and the desired pose into the encoder to obtain a posture index, which is then used to obtain the joint solution to reach the target pose via the decoder. The rationale is that

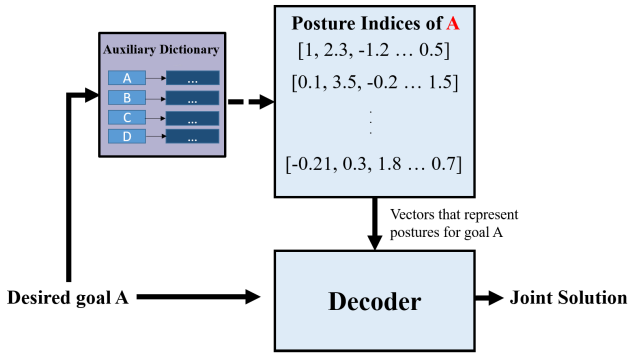


FIGURE 3. The KD-tree and the dictionary to track positions in the dataset and their posture indices.

the manipulator may move to the target pose with a posture similar to that of the initial state of the manipulator.

E. PSIK MODEL INFERENCE

After the PSIK neural models are trained and the auxiliary dictionary is set up, they can be used to infer joint solutions. Given a desired pose, a typical workflow of the inference in PSIK is as follows. First, the auxiliary dictionary is queried to retrieve posture indices corresponding to the pose. If the posture indices are fed into the decoder together with the desired pose, multiple joint solutions can be obtained. All of them can move the robotic manipulator to the desired pose. Second, the retrieved posture indices can optionally be adjusted before they are fed into the decode to obtain even more postures. Third, one joint solution is selected as the final output. Note that the first two steps in effect navigate the self-motion manifolds corresponding to the desired pose via the posture index to obtain multiple joint solutions for that pose. Depending on the application, they can be skipped or simplified, as discussed below. The last step corresponds to the redundancy resolution optimizations [7], [39] in typical numerical IK methods for redundant manipulators.

A couple of details in the above workflow need to be discussed further. In the first step, when the auxiliary dictionary is queried, the given desired pose may not be in the dictionary. Therefore, in PSIK, we actually search for the closest poses in the dictionary and take their posture indices as the indices for the desired pose. The range to search in the dictionary is a design parameter. In the second step, how to alter the posture indices to obtain more joint solutions is also a design problem. Since the mapping from the workspace to the joint configuration space is nonlinear, there seems no easy answer and we will leave it for future research. In the following, we will only consider the simple strategy that increments or decrements each component of the posture index with a small amount and sees how the posture of the manipulator is changed.

In the third step, the most important decision is to select one posture index as the final output. The goodness evaluation of the posture indices really depends on the application requirements, which are often expressed as constraints or utilities. As noted above, this roughly corresponds to the optimization operations in redundancy resolution in numerical IK methods that reduces the infinite number of joint solutions of a pose to just one. An important difference however is that in numerical methods the optimizations are usually formulated as part of the mathematical formulations and solved during the numerical iterations, while in PSIK the optimizations are evaluated after the joint solutions are obtained. This difference will be further elaborated when we discuss the applications in the next section.

F. DISCUSSIONS

Although kinematic redundancy is task-dependant, i.e., the dimensionality of the joint configuration space is larger than that of the task workspace, the training of the PSIK neural models is essentially task-agnostic, except for the dimensionality of the workspace. In other words, as soon as the dimensionality of the tasks is determined, e.g., 3-dimensional position only, 6-dimensional position plus orientation, or any other values, an unbiased training dataset independent of the tasks can be collected and used for training the neural models. Furthermore, during inference, any task-dependant requirements and optimizations are performed after the joint solutions corresponding to the given target pose are produced by the PSIK models, not during the process in calculating the solutions. This means that the PSIK models only need to be trained offline once for 3- as well as 6-dimensional workspace and then can be used throughout its lifetime. Of course, this does not exclude the possibility of improving the models dynamically for their specific working environments.

The primary goal of PSIK is to navigate the joint solution space of kinematically redundant manipulators to obtain multiple solutions for a given pose. We propose in this paper the posture index as the means for navigating the space. On the other hand, as mentioned in I, the joint configurations corresponding to a given end-effector pose can be represented as a finite set of disjoint continuous self-motion manifolds [4]. To model the configuration space of redundant manipulators more closely to the level of self-motion manifolds with PSIK models, we can leverage the analytic techniques proposed in previous works [4], [33], [35] on the training dataset to identify the self-motion manifolds and then annotate the information in the auxiliary dictionary. Alternatively, the signature posture indices mentioned in Sec. IV-D, which are resulted from a clustering of all the posture indices of a pose, may correspond closely to the self-motion manifolds. Further studies are needed to establish the link.

V. APPLICATIONS

In this section, two applications are introduced to demonstrate how the offline-trained PSIK models can facilitate online IK computations. The first application is to find a good initial joint solution with PSIK to bootstrap the numerical IK calculations, as suggested in [13]. The second application is to leverage PSIK's ability of generating multiple joint

TABLE 1. The average number of iterations to reach the target position by numerical IK.

Distance to Target (cm)	>5.0	<5.0	<1.0	<0.1	<0.02	<0.01
Iteration Count	23.07	14.39	9.94	6.64	3.19	0.19

TABLE 2. The percentage improvement in distance to reach the target position for the first five iterations by numerical IK.

Iteration	1	2	3	4	5
Distance Improvement (%)	18.18	32.70	10.13	8.02	6.67
Accumulated Distance Improvement (%)	18.18	50.88	61.01	69.03	75.70

solutions so that requirements of secondary tasks, such as obstacle avoidance, can be evaluated off the numerical IK iterations and at the stage after the multiple solutions are produced by PSIK. In this way, numerical IK methods do not need to calculate extra constraints during the optimization iterations, speeding up the computations.

A. BOOTSTRAPPING NUMERICAL IK

As mentioned earlier, numerical IK methods typically rely on iterative approximations, such as Newton's method or gradient descent, to solve for the joint variables to reach the given target pose. The number of iterations correlates to the distance between the initial position of the end-effector and the target position. Table 1 shows the number of iterations that the numerical IK method needs to run to reach the required position precision (0.01 cm), even when the end-effector has already reached a certain distance from the target position. In this experiment, the 7-DoF Franka Emika Panda robotic manipulator was the target, and the IK function was solved by the L-BFGS-B IK solver [6] in the IKpy package [28]. The manipulator was initially positioned with the joint configuration of (0,0, …,0), and 10,000 positions in the workspace were randomly sampled as the target position. The average values are reported.

It can be seen from the table that the numerical IK method needs quite a few iterations to reach the target position. If the precision requirement is higher, say to 7 digits, then the number of iterations will dramatically increase. We can also see that even when the remaining distance from the current position to the target position is smaller than 0.02 cm, the numerical IK method still takes a few iterations to reach the target.

Table 2 examines the effects of iterative numerical optimizations from another perspective, i.e., the percentage distance improvements for the first few iterations. We can see that the numerical method moves the manipulator gradually towards the target position. In average it takes two iterations to bring the end-effector to about half way between the starting location and the target location, and the first five iterations only take it to about 75% of the way. Therefore, if numerical IK can start from a pose that is close to the target pose, the number of iterations can be significantly reduced.

Using PSIK, this can be done easily with several operational options. Given a target pose, if there is no special requirements or constraints, we can query the auxiliary dictionary for a pose that is closest to the target and then randomly choose one associated posture index. The posture index and the target pose is then fed into the PSIK decoder to obtain the joint solution. The resultant joint solution can serve as the starting pose of the numerical IK method. Alternatively, we can leverage the current state of the manipulator by taking the current pose and joint configuration as inputs to the PSIK encoder to generate a posture index. The obtained posture index is then sent to the decoder together with the target pose to generate the joint solution to bootstrap the numerical IK computations.

The problem with the above two schemes is that they ignore the manipulator transition from the current pose to the target pose. One workaround is to combine the two. We can take the posture index generated in the second scheme from the PSIK encoder to compare with the posture indices obtained from the dictionary in the first scheme. The posture index from the latter that is the most "similar" to the former is chosen to feed into the PSIK decoder to produce the joint solution. The rationale is to allow the manipulator to transit from the current configuration to the target pose in a most smooth way. Again, there are many details in evaluating the "similarity" of two posture indices, and we will leave the topic for future study.

Extra requirements such as posture similarity discussed above, and singularity or obstacle avoidance may be added as tasks demand. In PSIK, these requirements may be evaluated after we obtain multiple joint solutions to the target pose, a point to be further elaborated in the next subsection. One final note is that the time of querying the auxiliary dictionary and executing the PSIK neural models must be significantly lower than that of iterative approximations by numerical IK from a random initial solution. In the next section, we will evaluate the effectiveness of using PSIK to bootstrap numerical IK methods.

B. CONSTRAINTS OFF NUMERICAL ITERATIONS

If the task imposes extra requirements, such as obstacle avoidance, numerical IK methods normally will formulate them as optimization goals or constraints and compute them as part of the approximation iterations. If the application imposes more constraints, the approximation formulations will become more complex and harder to converge, resulting in more iteration count and computation time. With PSIK, the extra requirements need not be formulated into the numerical formulations. They can be checked after the PSIK models produce multiple joint solutions for the given target pose. The joint solution that best satisfies the constraints is then returned as the initial guess for the numerical IK solver. In this way, the numerical IK solver needs not calculate

these constraints during the iterative optimization process. This simplifies the computation complexity of each iteration and speeds up the total execution. Note that since the joint solutions produced by PSIK can move the end-effector close to the target position, the pose of the manipulator will remain similar when moving from those positions to the target position. This will be further verified in the next section.

Algorithm 1 shows a simple framework for leveraging the multiple joint solutions generated by PSIK to evaluate extra requirements, including adjusting the posture indices to get better solutions. The framework contains an evaluation function that incorporates extra requirements of the task to evaluate how well a joint solution satisfies the requirements. The evaluation functions may include information such as positions of the obstacles and consult the auxiliary dictionary for information such as singularities of the joint solution space. The framework also has an adjust function to determine how to adjust the posture index. In Algorithm 1, we show an example evaluation function, $F_{req}(\theta, I) = \rho$, and demonstrate how each element of the posture index is adjusted in turn to find another posture index that can meet the application requirements better.

Algorithm 1 A Framework for Extended Task Execution With PSIK

Require:
Desired pose: *pose*;
Posture indices: $I_1, \ldots, I_k$;
Joint solution: θ;
Auxiliary dictionary: *Dict(pose)*;
PSIK decoder: $PSIK\_decoder(pose, I_i) = \theta$;
Task requirements: I;
Evaluation function: $F_{req}(\theta, I) = \rho$, where ρ is the degree that θ satisfies I; $\rho = 0$ if not satisfy;
Adjustment function: $F_{adj}(\theta) = x$, where x is the amount to adjust for the posture index;

Output: Joint solution θ_o that best satisfies the requirements

$\{I_1, \ldots, I_k\} \leftarrow Dict(pose)$
for $i = 1$ to k **do**
  $\theta \leftarrow PSIK\_decoder(pose, I_i)$
  $\rho \leftarrow F_{req}(\theta, I_i)$
  while $\rho = 0$ **do**
    $I_i \leftarrow I_i + F_{adj}(\theta)$
    $\theta \leftarrow PSIK\_decoder(pose, I_i)$
    $\rho \leftarrow F_{req}(\theta, I_i)$
  end while
end for
$\theta_o \leftarrow$ the joint solution that has the maximum ρ

VI. EXPERIMENTS

In this section, details of the experimental setup are given in Sec. VI-A. Since the posture index is the most important element in PSIK, its design is first studied in Sec. VI-B.

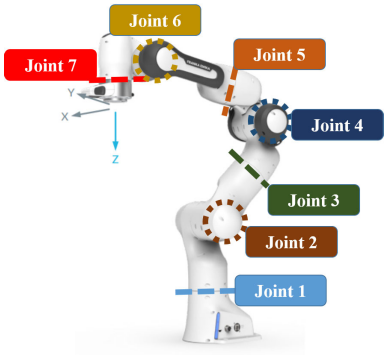


FIGURE 4. The 7-DoF Franka Emika Panda.

The capability of PSIK to model the joint solution space of redundant IK is then evaluated in Sec. VI-C. Sec. VI-D focuses on using PSIK to navigate the redundant solution space to find multiple distinct joint solutions to reach a given target position is examined. Finally, in Sec. VI-E, applications of PSIK to facilitate numerical IK computations are evaluated.

A. EXPERIMENTAL SETUP

We evaluate the proposed methods using a 7-DoF robotic manipulator, Franka Emika Panda.[1] Its structure is shown in Fig. 4. Joint 1, which is the axis nearest the base, controls the orientation of the whole manipulator. Joint 4 is an elbow joint, which gives more versatility to the end-effector. The first three axes (Joints 1, 2, and 3) determine the position of the elbow joint, and the remaining axes determine the pose of the end-effector. The rotation limits of the joints are as follows: Min/Max (degree) = Joint 1: -166/166, Joint 2: -101/101, Joint 3: -166/166, Joint 4: -176/-4, Joint 5: -166/166, Joint 6: -1/215, Joint 7: -166/166. The robot has a maximum stretch of 855 mm. The experiments were conducted in a virtual environment, the PyBullet [8] physics simulator, for avoiding unexpected collisions and damage. In the virtual environment, the physical parameters of the manipulator are the same as those of the real robot, except the self-collision mechanism was turned off. All experiments were run on a computer with an Intel i7-8700 CPU and 64 GB of memory.

The training dataset consists of the motor angles of the arm, Cartesian coordinate (x, y, z) of the end-effector, and the orientation of the end-effector. The dataset was collected every 30 degrees of each motor. The dataset contains 6,967,296 data points with a total size of 589.5 MB. From the collected data, we identified 580,608 different end positions, resulting in about 12 data points per position on average. Note that it does not mean there are only 12 postures per position. Since the dataset is sparse, it is not possible to obtain all the joint solutions to the same end position. To cope with the sparsity and to generate a sufficient number of joint solutions to a target position for selection, we actually included all the

[1] https://www.franka.de/

data points that could be reached within 1 cm from the given target position as the joint solutions to that position. Hence, in Sec. VI-D, we also consider the posture indices from the neighboring nodes that are close to the target position for selection.

In addition to this dataset, we also collected a second dataset called the *uni-position dataset* by dropping the data points in the first dataset that have the same target position of the end-effector. The second dataset was prepared for especially training the two baseline methods for comparison. As stated previously, the data points with the same target positions (input) are considered noise while training a classic fully-connected network, because the wights are in competition. Hence, we attempted to alleviate the impact of the ''duplicated'' data by removing those data with the same positions. The uni-position dataset contains 523,267 data points with a total size of 61.2 MB.

To implement PSIK, the encoder and decoder each had five fully connected layers. The encoder had 2048, 2048, 1024, 512, and 4 neutrons in the five layers respectively, whereas the decoder had 512, 1024, 1024, 512, and 7 neutrons. Each layer was followed by an ELU activation function, except for the output layer. The output layer of the encoder generates two outputs: one stands for the mean, and the other stands for the variance. We have tried different parameters to train the neural networks of PSIK. It is found that the learning rate should be less than or equal to 0.0005, decreased after every 1000 training epochs, and the batch size should be 65536. Stable and satisfactory results can be obtained for the entire workspace after 3300 epochs.

B. DIMENSIONALITY OF POSTURE INDICES

Posture index is the most critical element in PSIK, which allows us to navigate the joint solution space, or more precisely the self-motion manifolds corresponding to a pose. The dimensionality of posture index is thus an important design parameter, which will be studied in this subsection.

Since posture index can be used to navigate the self-motion manifolds of a pose, its dimensionality must be closely related to the dimension of the redundant joint space of the manipulator. In this experiment, we first consider the case that the tasks only require the end-effector to reach the target position, i.e., the desired pose consists only of the position. Therefore, the workspace of the tasks is a 3-dimensional Cartesian space and the redundant joint space has a dimension of four. We tried different dimensionalities of the posture index, from one to seven, to train the PSIK models. The experiments show that the models with a dimension smaller than three failed to fit the training data. The loss values of these models stopped decreasing in an early stage. On the other hand, the models with a dimension greater than or equal to three can successfully achieve a good comparable performance. The main difference among them is the speed to converge. The longer the indices are, the faster the loss value decreases.

This result confirms that the dimensionality of posture index is closely related to the dimension of the redundant joint space. However, it also indicates that its dimensionality needs to be exactly the same as the redundant joint space, perhaps due to the nature of probabilistic distribution of posture index. Since the models with 3-dimensional indices need more hyperparameter adjustments and training time, we therefore use 4-dimensional posture indices in most experiments discussed in this section, if the tasks only require the end-effector to reach a target position. On the other hand, the dimensionality can be reduced to one if the tasks require both position and orientation of the end-effector.

C. MODELING JOINT SOLUTION SPACE

In this subsection, we evaluate how closely the proposed PSIK can model the joint solution space of redundant IK. Although accuracy is not of primary concern in this paper, because we focus on using PSIK to assist numerical IK computations, accuracy nevertheless is a good indicator of the closeness of PSIK in modeling the solution space. If the tasks only require the end-effector to reach the target position, i.e., the desired pose consists only of the position, then the accuracy can be measured in terms of *distance error*, which is the Euclidean distance from the given target position to the center of the end-effector by setting the joints according to the produced joint solution. If the desired poses include target position as well as orientation, then the accuracy can be measured by the distance error and the *cosine similarity* of the orientation of the end-effector.

1) POSITION ONLY

We first consider the case in which the desired poses only contain the position. The modeling capability of PSIK is compared against two NN-based methods: an adaptive neuro-fuzzy inference system (ANFIS) [11] and a deep fully-connected neural network (FCN) [11], [41]. PSIK used a 4-dimensional posture index. The parameters of the ANFIS were as follows: the number of premise functions of each feature was 2, the range of allowed values of the exponent in the premise functions was from 0.2 to 0.5, the range of allowed values of the exponent in the premise functions was from 1 to 3, and the range of allowed values of the coefficients in the consequent functions was from -10 to 10. The solver of the ANFIS was a particle swarm optimizer (PSO), in which the number of populations was 100 and the number of iterations was 200. Limited by the computing resources available to us, ten premise functions for each feature is the largest configuration we can compute. FCN served as our baseline and had 3, 512, 1024, 1024, 512, and 7 neutrons in the five layers, which were identical to the PSIK decoder except for the input layer.

In this experiment, we noted that the training dataset had different densities in different regions of the workspace due to the characteristics of the different joints and their links. Therefore, we divided the workspace with different sizes of circles centered around the base of the manipulator and

TABLE 3. Modeling of joint solution space in terms of average distance error (cm) for tasks requiring position only.

Method	Dataset	Radius (cm)																		Ave.
		5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	
PSIK	Original	1.31	2.16	1.22	1.21	2.36	1.15	1.35	1.34	1.31	1.78	1.90	1.61	1.6	2.18	2.16	1.86	2.45	2.11	**1.73**
ANFIS	Uni-position	91.1	88.6	88.2	85.8	82.2	82.5	75.9	76.5	74.7	70.1	70.3	71.3	68.3	66.2	66.5	62.9	60.5	68.4	**75.0**
FCN	Original	26.2	23.9	28.6	27.4	29.0	29.9	30.5	37.1	40.6	35.3	41.8	50.9	59.5	56.4	66.2	70.0	76.3	80	**45.0**
FCN	Uni-position	18.6	18.8	18.6	19.2	22.2	22.8	26.4	32.1	31.3	40.1	40.6	41.7	57.9	59.6	67.9	66.5	76.9	80.5	**41.2**

sampled 100 random target positions on each of the circles. The PSIK model was then used to find the joint solution. Since the tasks did not require the end-effector orientation, the first posture index in the auxiliary dictionary for the target position was chosen.

Table 3 shows the results. For PSIK, the average distance errors are relatively uniform across different regions, because it treats the dataset as distributions during training and considers nearby data points together. However, the average distance error is about 1.73 cm. One reason for the limited accuracy is the KL divergence that prevented PSIK from fitting the training dataset, which is similar to a regularization term in training. If high accuracy is required, the joint solution provided by PSIK can serve as an initial solution to bootstrap numerical IK, as will be discussed in Sec. VI-D.

In contrast, FCN can only achieve a 45 cm average distance error. This is not surprising, because the multiple solutions corresponding to the same pose prevent the neural network from finding an effective association. Even if we use the uni-position dataset, which keeps only one solution for each pose, the FCN still suffers from poor performance. The average distance error is improved by only about 5 cm. We speculate that this is because the dataset kept inconsistent joint solutions across adjacent poses. Thus, the neural network might receive similar inputs but have quite different joint solutions. Both FCN models perform better when the radius of target positions is under 40 cm, similar to PSIK. Finally, the average distance error of the ANFIS model is 70 cm, which is the worst among the three models compared.

2) POSITION AND ORIENTATION

We next examine PSIK when the desired poses include target position as well as orientation. Again, the workspace was divided by circles of different sizes centered around the base of the manipulator, and 100 random target positions on each of the circles were sampled with a random orientation. The PSIK model was then used to find the joint solution. Specifically, 50 data points closest to the target position in the KD-tree were retrieved, their associated 1-dimensional posture indices were used to find the corresponding joint configurations, and the joint configuration that best matches the target orientation was returned.

From Table 4, we can see that the average distance error is 3.2 cm and the average cosine similarity is 0.99, which means the deviation of the orientation is about 3 degrees. Compared with Table 3, the average distance error increases significantly. This is because the data points in the KD-tree closest to the target position did not have matching orientations. We had to search data points that were further away from the target position to find matching orientations. This problem can be solved by adjusting the elements of the posture indices that correspond to the closest data points to align the pose, as will be discussed in Sec. VI-D.

TABLE 4. Modeling of joint solution space for tasks requiring position and orientation.

	Radius (cm)								Ave.
	10	20	30	40	50	60	70	80	
Position (Dist. Error)	3.42	3.43	2.78	2.85	3.01	3.16	3.4	3.58	**3.20**
Orientation (Cos Similarity)	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	**0.99**

D. MODELING REDUNDANCY

The ability of PSIK to navigate the solution space to find distinct joint solutions is evaluated in this subsection. This is done by randomly sampling one target position from each of the four quadrants of the workspace and then selecting ten different posture indices in each position from the auxiliary dictionary. As mentioned in Sec. IV-D, the posture indices stored in the auxiliary dictionary are selected to be as distinct as possible. The four randomly sampled target positions had the coordinates (x, y, z) in the Cartesian workspace as follows (all numbers in meter):

- **a**: (0.1357, -0.321, 0.222)
- **b**: (-0.333, 0.246, 0.1999)
- **c**: (-0.188, -0.168, 0.321)
- **d**: (0.2020, 0.2021, 0.2022)

1) POSITION ONLY

Again, the case in which the tasks only ask for end-effector position is considered first. The posture indices are 4-dimensional vectors. To examine the diversity of the ten postures for each of the four positions, we show the variance of their joints in Table 5. As shown in the table, Joint 1 has a high variance in every position, which means Joint 1 has quite different values in the ten joint solutions generated by the ten posture indices in all four positions. Since Joint 1 is nearest to the base of the manipulator, this implies that the robot starts by facing very different directions and then moves its

TABLE 5. The variance of each joint in the four target positions based on ten different posture indices.

Target Position	Joint 1	2	3	4	5	6	7
a	138.53	70.26	176.80	5.69	179.89	54.97	0.01
b	205.18	56.01	159.10	17.50	249.05	121.48	71.94
c	164.97	98.49	178.18	3.03	142.63	109.57	0.01
d	172.87	55.37	249.41	1.28	341.17	131.19	1.50

end-effector to reach the same target position. If we examine Joints 1, 2, and 3 together, which determine the location of the elbow (Joint 4), we can also see that they have high variances. It means that Joint 4 can have very different positions while still taking the end-effector to the same target position.

Table 6 illustrates the ten postures generated from the ten posture indices to reach each of the four target positions using PSIK. From the table we can see that the manipulator can approach the target position (red dot) from different directions with different heights of the elbow joint and even different orientations of the end-effector. The postures are quite distinct if examined visually.

Table 7 shows the accuracy of the ten joint solutions of the four target positions. It can be seen that the average distance error is between 1.1 cm to 1.5 cm. The smallest distance error is 0.27 cm, occurred at position **d** using joint solution 2. The worst error is 3.24 cm using solution 7 at position **a**.

2) POSITION AND ORIENTATION

We next examine the case in which the desired poses contain both position and orientation. Now, the posture indices are only 1-dimensional scalars. Table 8 illustrates the five postures generated from the five posture indices to reach the four target positions while maintaining a given orientation. The orientation was chosen randomly in the experiment. It is interesting to see that the postures corresponding to a pose look similar. This is not surprising because the posture indices are scalars and conceptually there is only one way to varying the postures. We also trained PSIK using 3- and 4-dimensional posture indices, and it turned out that the extra elements cause the same changing trend, i.e., slope adjustment, but not orientation.

3) NAVIGATION WITH POSTURE INDEX

Posture index is a vector. Intuitively, every element in the posture index should affect some aspects of the final posture of the manipulator. Unfortunately, the unsupervised learning to train the PSIK models lacks semantic explanation of the elements of posture index, though it allows our approach to generalize to any DoF robotic manipulator. In this set of experiments, we study the effects of changing different elements in the posture indices and how they help us to navigate the self-motion manifolds of a pose.

Let us start with 4-dimensional posture indices, i.e., the desired poses contain position only. Table 9 shows the changes of the postures when different elements in a posture index are changed for the target position (-0.25, -0.25, 0.25). The changes of each element was 0.01 per step. There are two different trends in the postures when the manipulator tries to reach the given target position. One affects the orientation of the end-effector (elements 1 and 4), and the other affects the body of the manipulator but keeps both the orientation and the location fixed (elements 2 and 3). More specifically, element 1 makes the end-effector rotate around the vertical line, whereas element 4 rotates it around the horizontal line. In other words, we can change the two elements to control the direction that the end-effector approaches the target position. Fig. 5 shows the results when we adjust both elements 1 and 4. As can be seen, the end-effector moves along slant lines. On the other hand, elements 2 and 3 of the posture index change the slope of the end-effector. They cause similar effects except for the side.

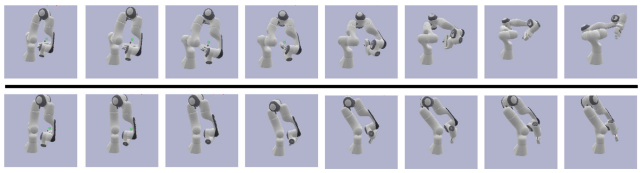


FIGURE 5. Effects of changing elements 1 (above) and 4 (below) in posture index, which adjust the end-effector along a slant line while keeping the target position unchanged.

We next study how the roles of the elements in the posture index change when the dimensionality of the index is changed. We examine 3-dimensional posture index first. It is found that two of the elements affect the orientation. This implies that they are essential so that the end-effector can approach the target position from any direction in a 3-dimensional workspace. The remaining element alters the slope, which means that the two elements that do the same in the 4-dimensional posture index are now merged into one due to their similarity. This also explains why the dimensionality of the posture index has to be at least three. When the dimensionality of the posture index increases, PSIK models can have more room to contain the two types of changing trends, which eases the difficulty of training. However, different elements in the posture index may cause similar effects.

E. APPLICATIONS OF PSIK

In this subsection, we evaluate the effectiveness of PSIK in facilitating the numerical IK computations as discussed in Sec. V. The experiments were conducted based on the IKpy package [28]. The main numerical IK method used was the L-BFGS-B method [6]. It estimates the inverse Hessian matrix to steer its search through the variable space to reduce the deviation to the target position. By maintaining the history of the past m ($m < 10$) updates of the position and the gradient instead of the whole inverse Hessian matrix, L-BFGS-B converges fast and requires less memory. In this set of experiments, a computer with an AMD Ryzen 5 3600 6-core processor and 16 GB memory was used.

TABLE 6. Illustrations of the ten different joint solutions in the four target positions considering position only.

Target Position	Joint Solution 1	2	3	4	5	6	7	8	9	10
a										
b										
c										
d										

TABLE 7. The distance errors (cm) of the ten joint solutions for each of the four target positions.

Target Position	Joint Solution 1	2	3	4	5	6	7	8	9	10	**Avg.**
a	0.98	0.65	1.03	0.82	1.70	1.28	3.24	0.63	1.20	2.72	1.43
b	0.93	1.51	1.46	2.44	1.32	0.40	0.74	1.48	1.13	1.83	1.32
c	0.94	1.45	3.35	1.26	0.51	1.64	0.77	1.09	0.29	0.63	1.19
d	1.03	0.27	1.04	2.18	2.05	1.31	1.18	1.39	2.94	1.17	1.46

1) BOOTSTRAPPING NUMERICAL IK

We first examine PSIK in finding a good initial joint solution to bootstrap the numerical IK calculations. Suppose the tasks require solving IK for position only. Given a target position, the numerical IK solver finds a joint solution for the end-effector to reach that position. PSIK can provide a nearby joint configuration for the solver to start the optimization iterations.

In the experiments, 10,000 positions in the workspace of the 7-DoF Franka Emika Panda were randomly selected as the target position for IK calculations and the average of these 10,000 results are reported. The accuracy was set to 0.01 cm on each dimension. In other words, the solution provided by the numerical IK method must be able to move the end-effector to within 0.01 cm from the given target position. According to the default starting configuration of IPky, the original IK solver started the optimization iterations from the joint configuration (1.55,1.35,1.55,-1.35,1.55,1.35,1.55), which was as close to the origin of the joints as possible. In PSIK, the data point in the KD-tree closest to the target position was returned as the initial solution to the numerical method.

Performance of the numerical IK method with and without using PSIK to bootstrap is shown in Table 10. From the table, we can see that PSIK improves the total execution time and the iteration count by 52.8% and 56.8% respectively over the original IK method, if the tasks require only the end-effector to reach a target position.

Our experiments also show that PSIK can provide an initial configuration to the IK solver that is in average 1.48 cm away from the target position. This is much better than the original IK solver without using PSIK, which starts with an average distance of 86.22 cm away from the target position.

We next consider solving IK for position as well as orientation. As discussed in Sec. V-B, when the application requires the end-effector of the robotic arm to move to a desired position with a specific orientation, the orientation evaluation may be moved outside of the optimization formulations with PSIK. This is done by the following steps:

1) PSIK generates multiple joint solutions with the given target position.
2) The joint solution that best match the target orientation is chosen.
3) The chosen joint solution serves as the initial solution to start the numerical IK optimization until an optimized joint solution that satisfies the position accuracy is obtained.
4) The optimized joint solution is adjusted to align to the target orientation.

The last step may be performed by adjusting the corresponding posture index as discussed in the last subsection or by heuristics similar to CCD or FABRIK [2], [43]. Note that since the joint solution obtained from PSIK can move the end-effector very close to the target position, the pose of the manipulator will remain similar when moving from that position to the target position. It is thus sufficient to examine the candidate joint solutions generated from PSIK for conformity to the imposed orientation.

TABLE 8. Illustrations of the five different joint solutions in the four target positions considering position and orientation.

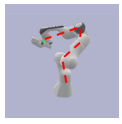
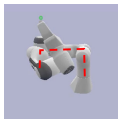
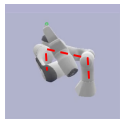
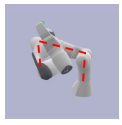
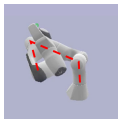
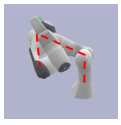
Target Position	Joint Solution 1	2	3	4	5
a					
b					
c					
d					

TABLE 9. Effects of different elements in the vector of posture index for the target position (-0.25, -0.25, 0.25).

Element 1						
Error (cm)	0.08	0.24	0.6	0.81	0.65	0.51
Element 2						
Error (cm)	0.06	0.15	0.18	0.08	0.49	0.6
Element 3						
Error (cm)	0.03	0.58	0.66	0.38	0.37	0.38
Element 4						
Error (cm)	0.03	0.11	0.18	0.19	0.42	0.86

TABLE 10. Bootstrapping numerical IK for position only with PSIK.

	Avg. Execution Time (sec)			Avg. # of	Initial
	PSIK	Numerical	Total	Iterations	Distance (cm)
Numerical Only (a)	-	0.0576	0.0576	40.37	86.22
PSIK + Numerical (b)	0.0009	0.0263	0.0272	17.44	1.48
Improvement of (b) over (a)			52.8%	56.8%	

The overall results are shown in Table 11. In this experiment, PSIK queried 50 data points closest to the target position in the KD-tree to find the joint configuration with the highest cosine similarity. The returned configuration was then served as the initial solution to start numerical IK iterations to approximate the target position. The IK iterations did not

TABLE 11. Bootstrapping numerical IK for position and orientation with PSIK.

	Avg. Execution Time (sec)				Avg. # of Iterations	Cos Similarity
	PSIK	Ori. Eval.	Numerical	Total		
Numerical Only (a)	-	-	0.2504	0.2504	156.78	1.00
PSIK + Numerical (b)	0.0164	0.0006	0.1089	0.1259	65.22	0.99
Improvement of (b) over (a)				49.72%	58.40%	

TABLE 12. Effects of PSIK on bootstrapping different numerical IK solvers.

	Numerical Solver	Numerical Only (a)		PSIK+Numerical (b)		Improvement of (b) over (a)	
		Iter.	Time (sec)	Iter.	Time (sec)	Iter. (%)	Time (%)
Position Only	L-BFGS-B	35.70	0.0717	27.1	0.0578	24.09	19.39
	BFGS	27.41	0.0877	19.26	0.0708	29.73	19.27
	SLSQP	25.99	0.0300	16.93	0.0215	34.86	28.33
	CG	34.50	0.1227	26.13	0.1051	24.26	14.34
Position + Orientation	L-BFGS-B	72.37	0.1662	29.53	0.0859	59.20	48.32
	BFGS	77.23	0.1942	21.72	0.1005	71.88	48.25
	SLSQP	52.91	0.0739	19.39	0.0472	63.35	36.13

evaluate orientation, because the joint configuration returned by PSIK had already matched the target orientation. From the table, we can see that the improvement of total execution time and IK iteration count over the original IK method are nearly 50% and 60% respectively. The poses generated by PSIK also match the required orientation closely.

Finally, we evaluate the generality of PSIK in bootstrapping numerical IK methods. The performance of PSIK in couple with other numerical IK methods in IKpy, including BFGS method [16], Sequential Least Squares method (SLSQP) [22], and Conjugate Gradient method (CG) [25] is evaluated. Table 12 compares the average execution time and iteration count by these numerical IK methods with and without PSIK to bootstrap.

The table shows that PSIK can speed up all the numerical IK methods considered. For position only, the improvements in execution time range from 14% to 28%. For position and orientation, the improvements are from 36% to 48%. This is not surprising, because PSIK only provides initial solutions to numerical IK approximation and does not affect internal operations of the solvers. Thus it can work well with all numerical IK methods.

2) CONSTRAINTS OFF NUMERICAL ITERATIONS

We have shown above the effectiveness of PSIK in moving the orientation requirement out of the numerical IK formulation and providing initial solutions that satisfy the orientation requirement. In fact, PSIK can assist in evaluating other constraints during IK computations. We examine one application and see how PSIK may help.

Consider the problem of obstacle avoidance in robotic manipulators. Most current methods incorporate obstacles as additional constraints into the numerical IK formations to solve the problem [7], [39]. Using PSIK, we can collect all the extra constraints and requirements into a separate procedure and evaluate the joint solutions generated from PSIK with the procedure for their conformity to the constraints. The procedure can be organized as a hierarchical pruning model, with each layer pruning out joint configurations that do not satisfy one of the constraints. By placing stricter constraints on the top layers, we can prune more joint configurations early, leading to better performance.

In this experiment, we assume that the manipulator needs to move the end-effector to a target position with a given orientation. There are a number of static obstacles in the workspace to avoid and certain task requirements for the postures. The goal is to determine a joint solution to satisfy all the constraints and allow the end-effector to reach the target pose. Note that for simplicity we only concern the final posture of the manipulator here, not the motion path. Nevertheless, the final joint configuration may well serve as a goal in motion planning [19], [24].

Four different tasks with different requirements and thus computing complexities were designed:

- Task A: position+orientation
- Task B: position+orientation, 1 obstacle
- Task C: position+orientation, 2 obstacles
- Task D: position+orientation, 2 obstacles, 1 requirement on posture

With the increasing task requirements, the original numerical IK solver needs to calculate more constraints during optimization iterations. Therefore, the computation time per iteration will increase. On the other hand, with PSIK, the numerical IK solver only needs to perform basic IK computations for target position and leaves the constraint evaluations to the separate procedure mentioned above. Hence, the computation time will remain almost the same for increasingly complex tasks.

Table 13 shows the effectiveness of PSIK in evaluating the constraints. In the table, the "Constraint Time" is the time to evaluate the extra constraints, and the "Total Time" of "PSIK+Numerical" includes the time of query, clustering,

TABLE 13. Effects of PSIK on evaluating extra constraints.

Task Name	PSIK+Numerical		Numerical		
	Constraint Time (sec)	Total Time (sec)	Total Time (sec)	No. of Iter.	Time/Iter. (msec)
Task A	0.017	0.1055	0.1651	73.69	2.24
Task B	0.026	0.1096	0.3190	75.24	4.24
Task C	0.025	0.1008	0.4612	75.62	6.10
Task D	0.027	0.1143	0.7095	81.19	8.66

and iterative IK approximation. From the table, we can see that for the original numerical IK, the total execution time and time per iteration both increase when the complexity of the constraints increase. In contrast, the time to process the constraints with PSIK and thus the total execution time of "PSIK+Numerical" remain nearly unchanged with the increase in constraint complexity.

VII. CONCLUSION AND FUTURE WORKS

In this paper, a novel deep learning approach is proposed that can model the ill-posed redundant IK solution space to find multiple distinct joint solutions of a given pose. The key idea of the proposed PSIK is to introduce an extra probabilistic vector, the posture index, to allow the navigation in the redundant joint solution space. Such a capability is very useful for speeding up numerical IK computations. To the best of our knowledge, this is the first work that can use a single NN to generate multiple joint solutions for 7-DoF robotic manipulators. Furthermore, the process to collect the training dataset can be fully automated with a few lines of code, and it does not impose any special constraints on training data. Thus, the proposed PSIK method can be applied to any robotic manipulators.

This work can find a diverse set of applications in robotic manipulators. Two example applications are presented in the paper to demonstrate the effectiveness of PSIK in facilitating numerical IK computations. Experiments based on the popular IKpy package show that PSIK can accelerate the execution of the different numerical IK modules in IKpy up to 50% for a 7-DoF manipulator. Other applications are possible. For example, if there are two different high-DoF robotic manipulators, it is possible to use the posture indices to make one manipulator imitate the other. The two manipulators not only follow the same trajectories by the end-effector but also have similar postures. This could be very useful when one wants to transfer the skills from one robot to another. Another possibility is to choose the best posture index that can achieve certain optimization goals, e.g., reaching a goal with minimal power or avoiding dynamic obstacles.

This paper opens up a new direction for NN-based IK methods. The current paper focuses mainly on the ability of PSIK to model the joint solution space of redundant IK and its applications to assist numerical IK computations. Although PSIK can obtain solutions to redundant IK, the accuracy still can not match that by numerical IK at this point. It is thus interesting to exploit the accuracy of PSIK models in the future. This work uses a sparse training dataset (by every 30° per joint) and has a subcentimeter distance error (∼0.5 cm). It is interesting to see whether denser training dataset can improve accuracy and if there is a limit. Furthermore, a systematic way of adjusting the elements of the posture index is needed to change the posture of the manipulator to a desired one. Another possible future work is to research how to model the distinct self-motion manifolds with PSIK and the use of PSIK to study the various properties of joint solution space of redundant IK.

REFERENCES

[1] A. R. J. Almusawi, L. C. Dülger, and S. Kapucu, "A new artificial neural network approach in solving inverse kinematics of robotic arm (Denso VP6242)," *Comput. Intell. Neurosci.*, vol. 2016, pp. 1–10, Aug. 2016.

[2] A. Aristidou and J. Lasenby, "FABRIK: A fast, iterative solver for the Inverse Kinematics problem," *Graph. Models*, vol. 73, no. 5, pp. 243–260, Sep. 2011.

[3] T. Asfour and R. Dillmann, "Human-like motion of a humanoid robot arm based on a closed-form solution of the inverse kinematics problem," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2003, pp. 1407–1412.

[4] J. W. Burdick, "On the inverse kinematics of redundant manipulators: Characterization of the self-motion manifolds," in *Proc. Int. Conf. Robot. Autom.*, May 1989, pp. 264–270.

[5] S. R. Buss and J.-S. Kim, "Selectively damped least squares for inverse kinematics," *J. Graph. Tools*, vol. 10, no. 3, pp. 37–49, Jan. 2005.

[6] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM J. Sci. Comput.*, vol. 16, no. 5, pp. 1190–1208, 1995.

[7] S. Chiaverini, G. Oriolo, and A. A. Maciejewski, "Redundant robots," in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Cham, Switzerland: Springer, 2016, pp. 221–242.

[8] E. Coumans and Y. Bai, "Pybullet, a Python module for physics simulation for games, robotics and machine learning," Tech. Rep., 2016. [Online]. Available: http://pybullet.org/

[9] M. Crenganis, R. Breaz, G. Racz, and O. Bologa, "Inverse kinematics of a 7 DOF manipulator using adaptive neuro-fuzzy inference systems," in *Proc. 12th Int. Conf. Control Autom. Robot. Vis. (ICARCV)*, Dec. 2012, pp. 1232–1237.

[10] A. Csiszar, J. Eilers, and A. Verl, "On solving the inverse kinematics problem using neural networks," in *Proc. 24th Int. Conf. Mechatronics Mach. Vis. Pract. (M2VIP)*, pp. 1–6. IEEE, 2017.

[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. (FUZZ-IEEE)*, Jun. 2019, pp. 1–6.

[12] D. DeMers and K. Kreutz-Delgado, "Learning global direct inverse kinematics," in *Proc. 4th Int. Conf. Neural Inf. Process. Syst.*, vol. 4, 1991, pp. 589–594.

[13] D. DeMers and K. Kreutz-Delgado, "Issues in learning global properties of the robot kinematic mapping," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 1993, pp. 205–212.

[14] A. D'Souza, S. Vijayakumar, and S. Schaal, "Learning inverse kinematics," in *Proc. Int. Conf. Intell. Robots Syst.*, vol. 1, Oct. 2001, pp. 298–303.

[15] A.-V. Duka, "Neural network based inverse kinematics solution for trajectory tracking of a robotic arm," *Proc. Technol.*, vol. 12, pp. 20–27, Jan. 2014.

[16] R. Fletcher, *Practical Methods of Optimization*. Hoboken, NJ, USA: Wiley, 2000.

[17] J. Ghasemi, R. Moradinezhad, and M. A. Hosseini, "Kinematic synthesis of parallel manipulator via neural network approach," 2019, *arXiv:1904.04668*.

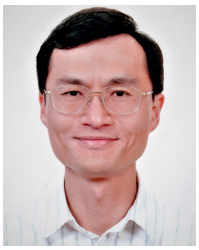
[18] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Commun. ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[19] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[20] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," 2013, *arXiv:1312.6114*.
[21] I. Kobyzev, S. Prince, and M. Brubaker, "Normalizing flows: An introduction and review of current methods," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 11, pp. 3964–3979, May 2020.
[22] D. Kraft, *Software Package for Sequential Quadratic Programming*. Cologne (Köln), Germany: DFVLR, 1988.
[23] K. Kreutz-Delgado, M. Long, and H. Seraji, "Kinematic analysis of 7 DOF anthropomorphic arms," in *Proc. IEEE Int. Conf. Robot. Automat.*, May 1990, pp. 824–830 vol. 2.
[24] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Iowa State Univ., Ames, IA, USA, Tech. Rep. TR-98-11, 1998.
[25] J. Lenarčič, "An efficient numerical approach for calculating the inverse kinematics for robot manipulators," *Robotica*, vol. 3, no. 1, pp. 21–26, Jan. 1985.
[26] Y. Lu and P. Xu, "Anomaly detection for skin disease images using variational autoencoder," 2018, *arXiv:1807.01349*.
[27] C. L. Luck and S. Lee, "Self-motion topology for redundant manipulators with joint limits," in *Proc. IEEE Int. Conf. Robot. Automat.*, May 1993, pp. 626–631.
[28] P. Manceron. (2022). *IKPy: An Inverse Kinematics Library Aiming Performance and Modularity (V3.3.3)* [Online]. Available: https://github.com/Phylliade/ikpy
[29] H. Masuda, A. Hitzmann, K. Hosoda, and S. Ikemoto, "Common dimensional autoencoder for learning redundant muscle-posture mappings of complex musculoskeletal robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Nov. 2019, pp. 2545–2550.
[30] Y. Nakamura and H. Hanafusa, "Inverse kinematic solutions with singularity robustness for robot manipulator control," *J. Dyn. Syst., Meas., Control*, vol. 108, no. 3, pp. 163–171, Sep. 1986.
[31] S. Neppalli, M. A. Csencsits, B. A. Jones, and I. D. Walker, "Closed-form inverse kinematics for continuum manipulators," *Adv. Robot.*, vol. 23, no. 15, pp. 2077–2091, 2009.
[32] E. Oyama, N. Young Chong, A. Agah, and T. Maeda, "Inverse kinematics learning by modular architecture neural networks with performance prediction networks," in *Proc. ICRA. IEEE Int. Conf. Robot. Autom.*, May 2001, pp. 1006–1012.
[33] J. Pan and D. Manocha, "Efficient configuration space construction and optimization for motion planning," *Engineering*, vol. 1, no. 1, pp. 46–57, Mar. 2015.
[34] Z. Pan, "Loss functions of generative adversarial networks (GANs): Opportunities and challenges," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 4, no. 4, pp. 500–522, Aug. 2020.
[35] A. Peidró, Ó. Reinoso, A. Gil, J. M. Marín, and L. Payá, "A method based on the vanishing of self-motion manifolds to determine the collision-free workspace of redundant robots," *Mechanism Mach. Theory*, vol. 128, pp. 84–109, Oct. 2018.
[36] K. D. Polyzos, P. P. Groumpos, and E. Dermatas, "Solving the inverse kinematics of robotic arm using autoencoders," in *Proc. Conf. Creativity Intell. Technol. Data Sci.* Cham, Switzerland: Springer, 2019, pp. 288–298.
[37] A. Ramdane-Cherif, B. Daachi, A. Benallegue, and N. Levy, "Kinematic inversion," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sep. 2002, pp. 1904–1909.
[38] A. Razavi, A. V. D. Oord, and O. Vinyals, "Generating diverse high-fidelity images with VQ-VAE-2," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–11.
[39] B. Siciliano, "Kinematic control of redundant robot manipulators: A tutorial," *J. Intell. Robot. Syst.*, vol. 3, no. 3, pp. 201–212, Sep. 1990.
[40] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 3483–3491.
[41] J. S. Toquica, P. S. Oliveira, W. S. R. Souza, J. M. S. T. Motta, and D. L. Borges, "An analytical and a deep learning model for solving the inverse kinematic problem of an industrial parallel robot," *Comput. Ind. Eng.*, vol. 151, Jan. 2021, Art. no. 106682.
[42] C. W. Wampler, "Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-16, no. 1, pp. 93–101, Jan. 1986.
[43] L.-C.-T. Wang and C. C. Chen, "A combined optimization method for solving the inverse kinematics problems of mechanical manipulators," *IEEE Trans. Robot. Autom.*, vol. 7, no. 4, pp. 489–499, Aug. 1991.
[44] Y. Xia and J. Wang, "A dual neural network for kinematic control of redundant robot manipulators," *IEEE Trans. Syst., Man, Cybern., B, Cybern.*, vol. 31, no. 1, pp. 147–154, Feb. 2001.
[45] W. Xu and Y. Tan, "Semisupervised text classification by variational autoencoder," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 1, pp. 295–308, Jan. 2019.
[46] I. Zaplana and L. Basanez, "A novel closed-form solution for the inverse kinematics of redundant manipulators through workspace analysis," *Mechanism Mach. Theory*, vol. 121, pp. 829–843, Mar. 2018.

CHI-KAI HO received the bachelor's degree from the Department of Computer Science, National University of Kaohsiung, Taiwan. He is currently pursuing the Ph.D. degree with the Department of Computer Science, National Tsing Hua University, Taiwan. His current research interests include reinforcement learning and robotic manipulation.

LI-WEI CHAN received the bachelor's degree from the Department of Computer Science, Tonghai University, Taiwan, and the master's degree from the Department of Computer Science, National Tsing Hua University, Taiwan. His current research interests include reinforcement learning and robotic manipulation.



CHUNG-TA KING (Senior Member, IEEE) received the B.S. degree in electrical engineering from the National Taiwan University, Taiwan, in 1980, and the M.S. and Ph.D. degrees in computer science from Michigan State University, East Lansing, MI, USA, in 1985 and 1988, respectively. From 1988 to 1990, he was an Assistant Professor in computer and information science at the New Jersey Institute of Technology, NJ, USA. In 1990, he joined as the Faculty Member of the Department of Computer Science, National Tsing Hua University, Taiwan, where he is currently a Professor. He served as the Chair of the Department, from 2009 to 2012. His research interests include parallel and distributed processing and networked embedded systems.

TING-YU YEN received the bachelor's degree from the Department of Computer Science, National Tsing Hua University, Taiwan, where he is currently pursuing the master's degree with the Department of Computer Science. His current research interests include reinforcement learning and robotic manipulation.