Robotics and Autonomous Systems 124 (2020) 103386

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Learning inverse kinematics and dynamics of a robotic manipulator using generative adversarial networks

Hailin Ren, Pinhas Ben-Tzvi *

Department of Mechanical Engineering, Virginia Tech, Blacksburg, VA 24060, USA

ARTICLE INFO

Article history:
Received 1 May 2019
Received in revised form 22 October 2019
Accepted 22 November 2019
Available online 9 December 2019

Keywords:
Inverse kinematics
Inverse dynamics
Generative adversarial networks

ABSTRACT

Obtaining inverse kinematics and dynamics of a robotic manipulator is often crucial for robot control. Analytical models are typically used to approximate real robot systems, and various controllers have been designed on top of the analytical model to compensate for the approximation error. Recently, machine learning techniques have been developed for error compensation, resulting in better performance. Unfortunately, combining a learned compensator with an analytical model makes the designed controller redundant and computationally expensive. Also, general machine learning techniques require a lot of data to perform the training process and approximation, especially in solving high dimensional problems. As a result, state-of-the-art machine learning applications are either expensive in terms of computation and data collection, or limited to a local approximation for a specific task or routine. In order to address the high dimensionality problem in learning inverse kinematics and dynamics, as well as to make the training process more data efficient, this paper presents a novel approach using a series of modified Generative Adversarial Networks (GANs). Namely, we use Conditional GANs (CGANs), Least Squares GANs (LSGANs), Bidirectional GANs (BiGANs) and Dual GANs(DualGANs). We trained and tested the proposed methods using real-world data collected from two types of robotic manipulators, a MICO robotic manipulator and a Fetch robotic manipulator. The data input to the GANs was obtained using a sampling method applied to the real data. The proposed approach enables approximating the real model using limited data without compromising the performance and accuracy. The proposed methods were tested in real-world experiments using unseen trajectories to validate the "learned" approximate inverse kinematics and inverse dynamics as well as to demonstrate the capability and effectiveness of the proposed algorithm over existing analytical models.

1. Introduction

Identification of the Inverse Kinematics (IK) and the Inverse Dynamics (ID) plays an important role in precise robot control and trajectory tracking [1,2]. Existing literature details various approaches aimed at obtaining precise models of the system to lower feedback gain and improve adaptability in designing a stable controller [3,4]. These techniques can be broadly classified into two categories: analytical methods, and numerical methods.

Analytical methods involve deriving an explicit mathematical model of the system under consideration from first principles. However, these methods rely on simplifying assumptions, prior knowledge, and experimental parameter estimations using the real system. Imperfections in any of the above can cause the analytical model to differ from the real system. In most cases, deriving the underlying mathematical model is unnecessarily complicated, and could suffer from singularities and nonlinearities [5,6]. In contrast, numerical methods are data-driven and can provide approximate solutions within a desired tolerance [7]. With dedicated algorithms and sufficient data collected from real-world experiments, numerical methods can learn the uncertainty part in the real system that is difficult to model, and thereby provide better predictions of the system behavior [8].

Over the past few decades, the applicability of machine learning has improved greatly along with improvements in the computational capability of hardware. Many techniques have been developed to solve highly nonlinear problems, such as learning the sequences of motion primitives for robot manipulation [9], cleaning a table [10], and generating trajectories for biped robots to follow ZMP critics [11]. The majority of existing techniques have focused on solving high-level tasks or trajectory planning, while using a general model-based controller for the low-level actions, resulting in a hybrid control system. Reinforcement learning techniques became popular in the research community due to the applicability of physics engine simulations [12] and a replay buffer [13]. However, in many cases, relying on the analytical

* Corresponding author.
E-mail addresses: hailin@vt.edu (H. Ren), bentzvi@vt.edu (P. Ben-Tzvi).

https://doi.org/10.1016/j.robot.2019.103386
0921-8890/

model behind the physics engine instead of using real-world data builds a gap between the simplified analytical model and the complex real-world system.

Applying machine learning techniques to acquire the IK and ID of a given system has a history of almost two decades in the research community. Karlik et al. worked on finding the best Artificial Neural Network (ANN) configuration to solve the IK problem for a six Degree-of-Freedom (DOF) robotic arm [14]. Comparison of Radial Basis function network (RBF) and Multi-layer Perceptron Network (MLP) in solving IK of a 6-DOF arm was performed in [15]. A neural network architecture, combined with evolutionary techniques were used to solve the IK of a 6-DOF Stanford robotic manipulator in [16]. In addition to planar manipulators [17], the IK of a spatial 3-DOF structure was studied in [18]. Instead of using a single-agent neural network to solve the kinematic problem, Ansari et al. applied actor-critic architecture (two agents in one neural network architecture) to learn the IK of a 6-DOF robotic manipulator inside a reinforcement learning environment. However, this work explored only a discrete action space (joint space) instead of the continuous action space [19]. In addition to offline training techniques, an adaptive online strategy based on the Lyapunov stability theorem was presented to solve for the IK of redundant manipulators in [20]. Multiple soft computing algorithms for solving the IK of different robotic manipulators were compared in [7]. However, the majority of existing works used analytical models as ground truth or used analytic models embedded inside physics-based simulations, instead of using the dataset collected from the real world.

Compensation methods using reinforcement learning were developed for better trajectory following, and the learning process was demonstrated in real-world online conditions [1]. Even though the compensator was learned, they also used analytical models inside the controller.

Compared to the IK, learning the ID is more difficult due to the high dimensionality of the input. To address this issue, existing techniques in this domain have used analytical models along with learning approaches to handle the modeling error. To this extent, Meier et al. proposed a nonlinear function approximator to learn a constant error model in order to improve tracking performance on specific trajectories [21]. On the other hand, Rayyes et al. proposed learning the inverse statics model by taking advantage of the symmetry of the robot [22]. However, the improved efficiency offered by this method is limited to symmetric robot designs. Machine learning methods have also been used to learn rich dynamics as in the case of a soft robotic manipulator [8,23,24]. Deep learning networks along with physics-based simulators have also been used to study robot dynamics [25]. Reinforcement learning techniques have also been used to learn the closed-loop predictive controller for a real robot [8].

Similar to other numerical methods, the need for a large dataset plays an important role in training the neural network to approximate the target model. As such, data collection is the most time consuming and expensive part in the global estimation of the ID. The proposed approach in [8] requires real-world data collection lasting approximately two hours to develop a closed-loop controller from scratch. To overcome the problem of training a neural network with limited data, Generative Adversarial Networks (GANs) were proposed by the computer vision research community. The idea was to create additional "fake" data similar to the real world data, and thereby enlarge the total dataset available for training the target neural network [26–28]. GANs have also been used in inverse reinforcement learning to recover the reward functions embedded in training environments to perform specific tasks [29,30]. In a similar fashion, our work aims to approximate the real model globally using a limited real-world dataset, which is augmented with fake data generated using GANs. The main contributions of this paper are as follows:

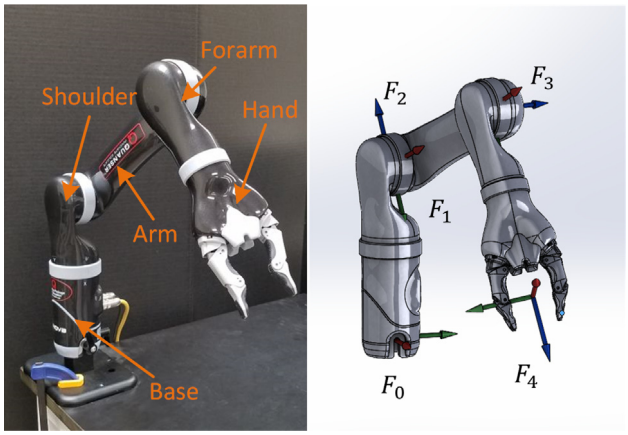


Fig. 1. 4-DOF MICO robotic manipulator with coordinate frame assignment.

- We extend the success of GANs used in the domain of computer vision towards learning the IK and the ID in cases where real-world data collection is expensive and highly nonlinear with high dimensional inputs.
- Four types of popular GANs, namely, CGANs [31], LSGANs [32], BiGANs [33], and DualGANs [34] were modified for applicability towards solving the IK and the ID problems. Performance of these methods was compared over the unseen real-world trajectories.
- Experimental evaluation of these methods was performed on a 3-DOF MICO robotic manipulator [35] and a 8-DOF Fetch robotic manipulator. To test the efficiency of the proposed modified GANs, all training processes were performed on a limited real-world dataset (collected over a period of 40 mins). The performance of the training process was also evaluated using different sizes of the partial dataset and different deviations for the generator in the GANs.

The rest of paper is organized as follows. Section 2 introduces the IK and ID of the robotic manipulators used in this paper. A brief introduction of generative adversarial networks (GANs) is also presented. Section 3 describes the modified GANs for learning the IK and ID. Details on the design of the neural network and sampling methods are presented. Section 4 discusses the simulation and experiment results. Finally, Section 5 concludes the work with directions for future research.

2. Preliminaries

2.1. Inverse kinematics and inverse dynamics

Kinematics describe the relationship between the coordinates in the joint space, q and the ones in the task space, x. The forward kinematics map the joint space to the task space, $FK : q \rightarrow x$ while the inverse kinematics presents the opposite mapping, $IK : x \rightarrow q$. Many methods have been developed to solve the kinematics problem, such as the geometric method and the Denavit–Hartenberg (DH) method. The closed loop equations have singularities and nonlinearities and thereby make the IK solving computationally expensive [5].

In this paper, we use a MICO robotic manipulator [35] and a Fetch robotic manipulator [36] as the experimental platforms for solving the IK and ID problems. As shown in Figs. 1 and 2, we used the standard Denavit–Hartenberg (DH) method for the coordinate frame assignments. With respect to the Fetch robotic manipulator, the head pan and tilt motions are trivial, and thus not considered in this paper. The transformation matrix from

Table 1
DH parameters and joint ranges of the MICO robotic manipulator.

Joint i	a_{i-1}	α_{i-1} (rad)	d_i	θ_i (rad)	θ_i limitations (rad)
1	0	0	$L_0 + L_1$	q_1	$[-\pi, \pi]$
2	0	$-\pi/2$	0	q_2	$[-11/18 * \pi, 1/9 * \pi]$
3	L_2	0	0	q_3	$[-23/36 * \pi, 5/36 * \pi]$
4	0	$-\pi/2$	$L_3 + L_4$	q_4	$[-\pi, \pi]$

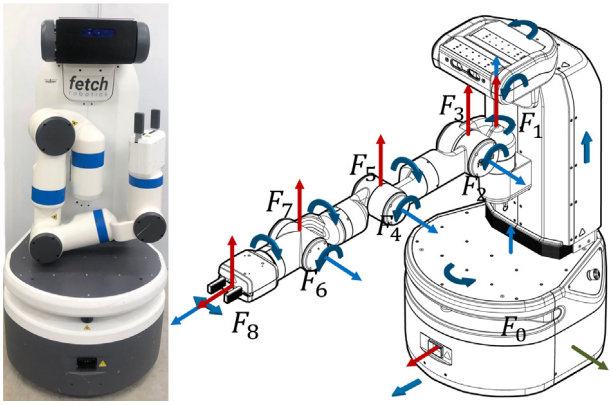


Fig. 2. 8-DOF Fetch robotic manipulator with coordinate frame assignment.

frame i, F_i, to frame j, F_j, is denoted as ${}^iT_j$. The transformation matrix from the base frame to the end-effector frame, ${}^0T_E$, can thus be obtained by multiplying the transformation matrices between intermediate frames, ${}^iT_{i+1}$, recursively,

$$ {}^0T_E = \prod_{i=0}^{n} {}^iT_{i+1} = \left[\begin{array}{c|c} {}^0R_E & {}^0P_E \\ \hline 0 & 1 \end{array}\right] \tag{1} $$

where ${}^0R_E$, ${}^0P_E$ are the rotation matrices from the base frame to the end-effector frame and the position vector of the end-effector in the base frame, n is the number of intermediate frames. The corresponding DH parameters are presented in Tables 1 and 2, respectively.

Dynamics correlates the torque and the force in each joint to the position, velocity and acceleration of the joint along with the external forces applied to the robot. The dynamics problem can be solved by various methods such as the Newton–Euler method, Lagrangian method, or Hamilton method. In this paper, we applied the Newton–Euler method along with the DH parameters to solve for the dynamic model of the robotic arm. The inverse dynamics of the robotic arm can be expressed as,

$$ \tau = M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) + J^\top F \tag{2} $$

where, M is the generalized inertial matrix, C is the generalized bias force including Coriolis, centrifugal forces, and friction terms, G is the gravitational force, J is the Jacobin matrix, and F is the external force applied to the end effector. Tables A.6 and A.7 in Appendix show the geometric and inertial properties of each link obtained from the Computer Aided Drawing (CAD) files for the MICO and the Fetch robotic manipulators, respectively. The moment of inertia is measured at the Center of Mass (CoM) of each link, aligned with its local coordinate system. $q, \dot{q}, \ddot{q}, \tau$ present the vectors of joint position, velocity, acceleration and generalized force variables, respectively. The inverse statics model can be obtained by setting the vectors of joint velocity and acceleration to zero, $\dot{q} = \ddot{q} = 0$. In this work, we do not apply any external force and torque to the end effector, which results in $F = 0$.

2.2. Generative adversarial networks (GANs) and variants

2.2.1. GANS and conditional GANs

Goodfellow et al. introduced Generative Adversarial Networks (GANs) for the first time in 2014 [37], followed by a series of GANs-family methods being developed for a wide variety of problems including generative tasks and discriminative tasks. Fig. 3 shows the original network structure of GANs. In the original formulation, GANs includes a generator G and a discriminator D. The generator is trained to learn a mapping from a low-dimension latent vector, $z \in \mathcal{Z}$ independent and identically distributed samples from a known prior p_z, to points in the space of natural data $\mathcal{X}$, while the discriminator, D is trained to learn a map $\mathcal{X} \mapsto [0, 1]$ that could determine if a sample $x \in \mathcal{X}$ is from the natural dataset, $x \sim p_{data}(x)$, or generated from the generator, $x \sim G; z \sim p_z$. Thus, the training process is to optimize the D to assign correct labels to both the natural dataset and the samples from the G, and optimize the G to minimize $\log(1 - D(G(z)))$ simultaneously. The minimax objective for the GANs is formulated as follows

$$ \min_G \max_D V(D, G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{3} $$

The Conditional Generative Adversarial Networks (CGANs) [26, 31] extended the original GANs to make both the Generator and the Discriminator be conditioned on additional information y, $G(z|y)$ and $D(x|y)$. The generator, G, takes the combined information of y and latent vector z as inputs while the discriminator, D, takes data x and y as inputs. The objective function of CGANs thus becomes,

$$ \begin{aligned} \min_G \max_D V(D, G) = {} & \mathbb{E}_{x\sim p_{data}(x)}[\log D(x|y)] \\ & + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z|y)))] \end{aligned} \tag{4} $$

2.2.2. Least square GANs (LSGANs)

The cross-entropy loss function proposed in the original formulation, Eq. (3), minimizes the Jensen–Shannon divergence between $p_{data}(x)$, the data distribution, and $p_g(z)$, the implicit distribution of the generator when $z \sim P_Z$ [37]. To overcome the difficulty and failure in training GANs using the original objective function, Mao et al. adopted a least squares loss function with $a-b$ coding scheme for the discriminator and showed that minimizing the objective function of LSGANs yields a minimization of the Pearson χ^2 divergence [32]. The proposed objective functions in LSGANs are defined as follows,

$$ \begin{aligned} \min_D V_{LSGAN}(D) = {} & \frac{1}{2}\mathbb{E}_{x\sim p_{data}(x)}[(D(x) - b)^2] \\ & + \frac{1}{2}\mathbb{E}_{z\sim p_z(z)}[(D(G(z)) - a)^2] \\ \min_G V_{LSGAN}(D) = {} & \frac{1}{2}\mathbb{E}_{z\sim p_z(z)}[(D(G(z)) - c)^2] \end{aligned} \tag{5} $$

where a and b denote the labels for fake data and real data, respectively. c denotes the value that G wants D to believe to be for fake data. Suggested by the parameters selection in [32], $0-1$ binary encoding scheme is used in this paper, where $a = 0, b = 1, c = 1$.

Table 2
DH parameters and joint ranges of the Fetch robotic manipulator.

Joint i	a_{i-1}	α_{i-1} (rad)	d_i	θ_i (rad)	θ_i, D_i limitations (rad, mm)
1	L_0	0	$L_1 + D_1$	0	[0, 400]
2	L_2	$-\pi/2$	L_3	θ_1	$[-23/45 * \pi, 23/45\pi]$
3	0	$-\pi/2$	0	$-\pi/2 + \theta_2$	$[-7/18 * \pi, 29/60 * \pi]$
4	0	$\pi/2$	$L_4 + L_5$	θ_3	Continuous
5	0	$-\pi/2$	0	θ_4	$[-43/60 * \pi, 43/60 * \pi]$
6	0	$\pi/2$	$L_6 + L_7$	θ_5	Continuous
7	0	$-\pi/2$	0	θ_6	$[-25/36 * \pi, 25/36 * \pi]$
8	0	0	$L_8 + L_9$	θ_7	Continuous

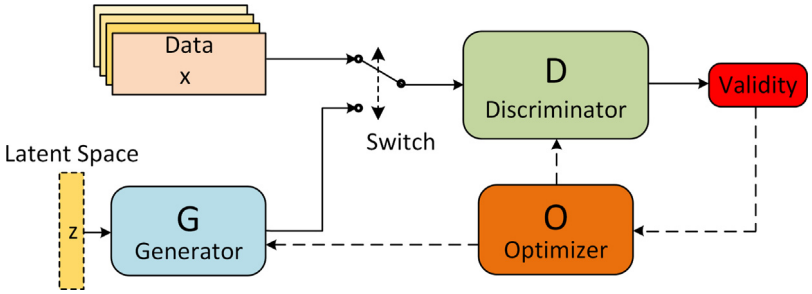


Fig. 3. Network structure of Generative Adversarial Networks (GANs).

Similarly, Arjovsky et al. suggested to minimize the smoother Wasserstein-1 distance between the generated and the natural data distributions, thereby proposing the Wasserstein GANs [38],

$$W(p_{data}, p_z) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim p_{data}(x)}[f(x)] - \mathbb{E}_{z\sim p_z(z)}[f(z)] \tag{6}$$

where $\|f\|_L \leq 1 : \mathcal{X} \mapsto \mathbb{R}$ is the family of functions that are 1-Lipschitz. To minimize the Wasserstein distance, a similar value function is used to optimize the training process,

$$V_{WGAN}(D_w, G) = \mathbb{E}_{x\sim p_{data}(x)}[D_w(x)] - \mathbb{E}_{z\sim p_z(z)}[D_w(G(z))] \tag{7}$$

With this formulation, $D_w : X \mapsto R$ is trained to serve as a function in computing the Wasserstein distance.

2.2.3. Bidirectional GANs (BiGANs) and DualGANs

Besides exploring the optimization in the objective functions, efforts have also been applied in designing a new architecture. Bidirectional Generative Adversarial Networks (BiGANs) provide Generative Adversarial Networks (GANs) with the means for learning the inverse mapping, projecting the data, $x \in \mathcal{X}$, back into its latent space, $z \in \mathcal{Z}$, [33]. An encoder, E, in BiGANs is trained to invert the generator G without direct communication with the generator, G. Similar to the BiGANs, DualGANs was proposed with two discriminators and two generators to learn the mappings between the two spaces, $\mathcal{X}$ and $\mathcal{Z}$. The primal GANs learns the translation from space $\mathcal{Z}$ to space $\mathcal{X}$, while the dual GANs learns to invert the task. With two discriminators and closed loop architecture, DualGANs perform better in translating between the two spaces [34].

3. Proposed algorithm

This section describes the proposed GAN architecture to approximate the IK and the ID of both MICO and Fetch robotic manipulators using real-world data.

3.1. Conditional generative adversarial networks and least squares generative adversarial networks

In the original GANs, the generator priors on latent vectors $z \in \mathcal{Z}$ as the inputs and produces the data $x \in \mathcal{X}$. To satisfy the mathematical model of the IK and the ID, each piece of data,

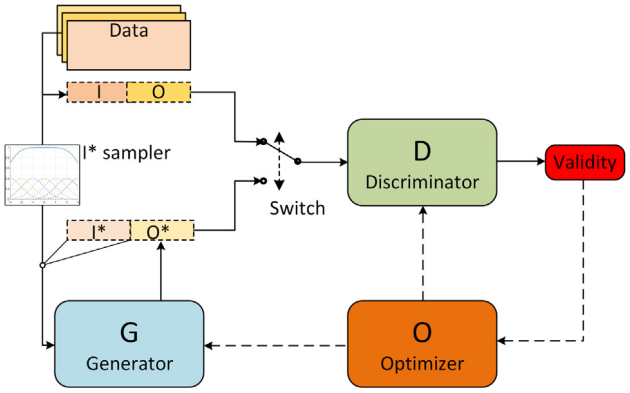


Fig. 4. Conditional Generative Adversarial Networks (CGANs) for the IK and the ID.

$[I \parallel O] \sim P_{data}$, consists of both the input I and the expected output O for IK and ID. $\parallel$ represents the concatenated form of data I and O in the dataset. In solving the IK, the position vector of the end effector in the workspace $x_e \in X_{end}$ is used as the input I, while the joint states $q_k \in Q_k$ that result in the corresponding position of the end effector form the output O. For the ID, the joint states along with their first and second derivatives, $q, \dot{q}, \ddot{q}$, are used as the input I, while the vectorized force, τ, presents the output O. As shown in Fig. 4, the discriminator, D, takes in one piece of data, $[I\|O]$ or $[I^*\|O^*]$ and outputs the validity of whether this piece of data is from the natural dataset or generated from the generator, G. We denoted $[I^*\|O^*]$ as the data generated from the generator. Based on the I part of the data sampled from the natural dataset with the predefined variance, σ^2, I^* is sampled using Gaussian Distribution $\mathcal{N}(I, \sigma^2)$ along with each input dimension. I^* is then fed into the generator, and it outputs O^*. I^* is then concatenated with O^* as a single instance of fake data to pit against the discriminator. The objective function of the modified CGANs is,

$$\begin{aligned}\min_G \max_D V(D, G) = {} & \mathbb{E}_{[I\|O]\sim p_{data}([I\|O])}[\log D(O|I)] \\ & + \mathbb{E}_{I^*\sim\mathcal{N}(I,\sigma^2)}[\log(1 - D(G(I^*)|I^*))]\end{aligned} \tag{8}$$

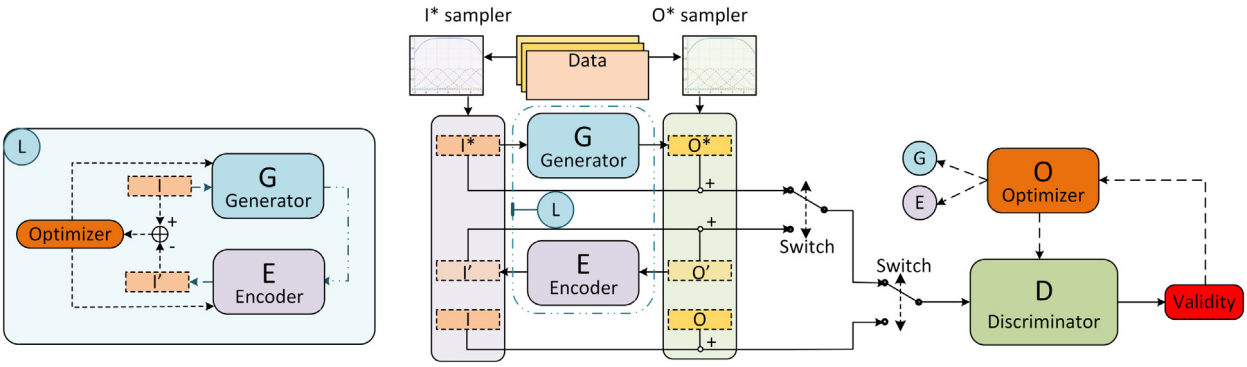


Fig. 5. Bidirectional Generative Adversarial Networks (BiGAN) for the IK and the ID.

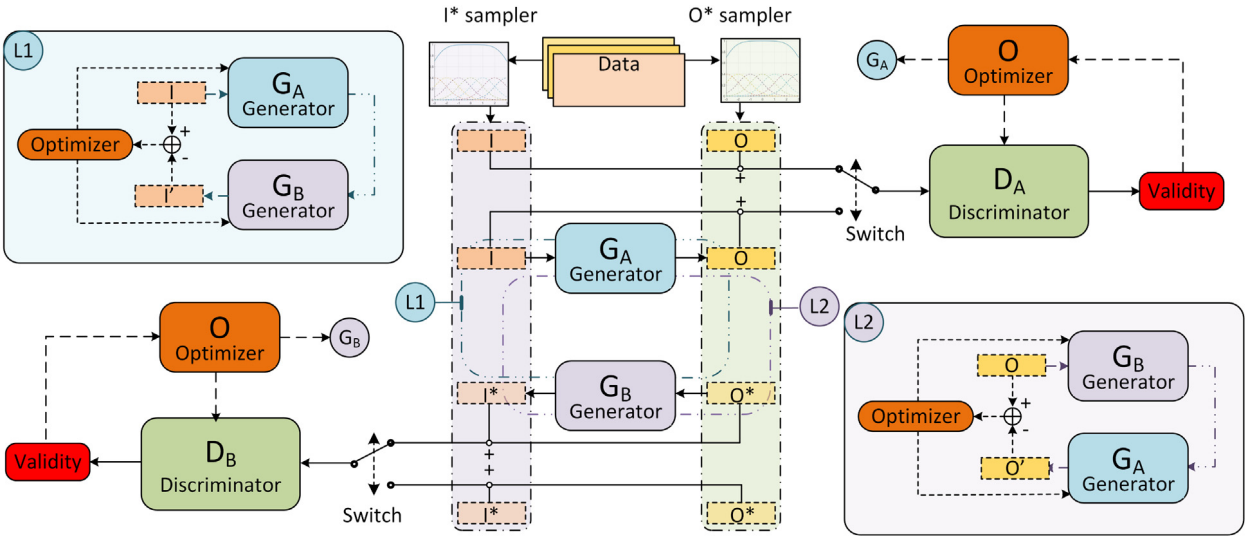


Fig. 6. Dual Generative Adversarial Networks (DualGAN) for the IK and the ID.

In accordance with the approach described in [32], the proposed LSGAN architecture replaces the cross-entropy loss term in the objective function Eq. (8) above, with the $0-1$ binary coding scheme in order to enforce the generator-discriminator architecture to produce realistic data,

$$\begin{aligned}\min_D V_{LSGAN}(D) &= \frac{1}{2}\mathbb{E}_{[I\|O]\sim p_{data}([I\|O])}[(D(O|I)-1)^2] \\ &+ \frac{1}{2}\mathbb{E}_{I^*\sim\mathcal{N}(I,\sigma^2)}[(D(G(I^*)|I^*))^2] \\ \min_G V_{LSGAN}(D) &= \frac{1}{2}\mathbb{E}_{I^*\sim\mathcal{N}(I,\sigma^2)}[(D(G(I^*)|I^*)-1)^2]\end{aligned} \tag{9}$$

Details in picking the standard deviation σ for training are presented in Section 3.4

3.2. Bidirectional generative adversarial networks

Bidirectional Generative Adversarial Networks (BiGANs) provide one extra component, encoder E, to the Generative Adversarial Networks. The encoder E in the original BiGANs aims to learn the hidden mapping from the data space to the latent noise space. With this architecture, BiGANs can learn both the forward and inverse mapping at the same time during the training process. With the same idea, the proposed network architecture, as shown in Fig. 5, embeds an encoder to provide the inverse mapping of the studied problem. An additional sampler is built to generate additional "fake" data, O', based on the sample data, O, from the natural dataset. These additional samples O' and the correspondingly generated I' are then used to train the encoder E and the discriminator for the generator G. In the original formula of BiGANs [33], the data pairs $[I'\|O']$ generated from the encoder E are believed to be natural data. To supervise the encoder's learning process of inverse mapping and thereby improve the stability of the optimization procedure, one additional optimization loop, L, is proposed in the network. In the added loop, the generator takes in data I and outputs data O. The data O is then mapped back to I' via the encoder. The corresponding loss function of the loop is defined as the L_2 distance between the original data I and the processed data I',

$$L = \|I - I'\|_2 = \|I - E(G(I))\|_2 \tag{10}$$

The following loop loss is added to the BiGANs objective function,

$$\begin{aligned}\min_{GE}\max_D V(D,G) &= \mathbb{E}_{[I\|O]\sim p_{data}([I\|O])}[\log D(O|I)] \\ &+ \mathbb{E}_{I^*\sim\mathcal{N}(I,\sigma_I^2)}[\log(1-D(G(I^*)|I^*))] \\ &+ \mathbb{E}_{O^*\sim\mathcal{N}(O,\sigma_O^2)}[\log(1-D(O^*|E(O^*)))]\end{aligned} \tag{11}$$

Details in picking the standard deviations σ_I and σ_O for training are presented in Section 3.4.

3.3. Dual-generative adversarial networks

DualGANs were proposed to create the mapping between two domains I and O. Compared to the derivatives of the GANs mentioned above, it consists of two GANs roughly with one GAN for

Table 3
Hyperparameter tuning for neural networks.

Parameters	Choices	Final selected parameters
Number of layers	[3, 4, 5, 6, 7, 8]	3 for IK 5 for ID
Number of neurons in hidden layers	[128, 256, 512, 1024]	256
Activation functions in hidden layers	['ReLU', 'Sigmoid', 'Leaky ReLU']	'Leaky ReLU'
Parameters for 'Leaky ReLU'	[0.1, 0.3, 0.4, 0.5, 0.7, 0.9]	0.1

forward mapping learning and one for inverse mapping learning. As shown in Fig. 6, the proposed DualGANs are composed of GAN-A and GAN-B. GAN-A (whose components are denoted with subscript A) is built to generate "fake" O prior on I, while GAN-B does the inverse job. Two additional optimization loops L_1 and L_2 are embedded in the network architecture. In optimization loop 1, samples I are transferred to O space via generator G_A then transferred back to I space via generator G_B as I^*. The L_2 distance between I and I^* are used as the objective function to optimize both G_A and G_B. A similar procedure is being performed in loop 2. As advocated in the original DualGANs [34], Wasserstein-1 distance is used to define the loss of both discriminators, D_A and D_B,

$$l_A^d(I, O) = D_A([I^* \parallel G_A(I^*)]) - D_A([I \parallel O]) \tag{12}$$

$$l_B^d(I, O) = D_B([G_B(O') \parallel O']) - D_B([I \parallel O]) \tag{13}$$

3.4. Data standardization and hyperparameter tuning

Feature scaling is an important process for data preparation in machine learning, especially when the range of values of raw data varies widely [39]. Considering the different ranges of the joint position as shown in Tables 1 and 2, the real-world data collected is standardized by removing the mean and scaling to unit variance before being processed to the training and validation dataset. With a unit variance for each data channel, standard deviation can be selected for the generator to generate input samples. A comparison of using different standard deviations for the generators is performed to evaluate their impact on the training performance in the Section 4.

To ensure optimal performance of each of the proposed methods, Hyperopt [40] was used for hyperparameters tuning. The tuned hyperparameters include the number of layers n_l, the number of neurons in each layer n_n, the activation functions used for the neurons in each layer, except the final output layer f_a, and the parameters for the activation, f_p, if needed. In order to ensure realistic comparison, tuning was performed to optimize the cumulative performance of all proposed methods, and the same set of tuned parameters were used across all proposed architectures. Details of the choices and tuned hyperparameters are presented in Table 3.

4. Training and experiment

4.1. Data collection for MICO robotic manipulator

To avoid overfitting of the training model in local paths, random trajectories, instead of predefined trajectories, were generated for the end effector of the MICO robotic manipulator to follow. Each trajectory consisted of multiple waypoints, distributed over the whole actuation space (joint space). In sampling the waypoints, two criteria were used to ensure the feasibility and safety in achieving the desired motion: (1) the joint positions should fall within the feasible configuration space and, (2) the z position of end effector, calculated from the forward kinematics, should be greater than 0.2 m to avoid the robotic manipulator from crashing into the table where the base was fixed. Cubic polynomials were used to generate the desired smooth motion between two consecutive waypoints in the trajectories, ensuring zero velocity constraints at each waypoint.

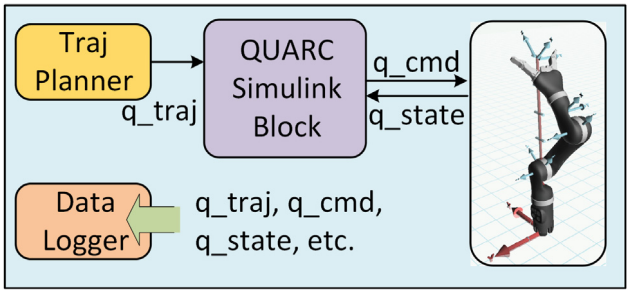


Fig. 7. The control and monitoring system was built in Simulink with QUARC™.

$$f(t) = a + bt + ct^2 + dt^3 \tag{14}$$

Fig. 1 shows the experimental setup. MATLAB Simulink and Quanser's QUARC™software were used to control and monitor the MICO robotic manipulator as shown in Fig. 7. The trajectory is planned in the joint space, q_traj, and then sent to a QUARC Simulink block to control the low-level joint actuators. The status data, such as the torque at each of the joints and the real-time joint positions were also collected using the QUARC Simulink block. Besides the joint status, the QUARC Simulink block also provides the position of the end effector using a predefined forward kinematics model. A data logger was built inside the system to collect all the data necessary for the neural network training.

Six trajectories were tracked, and each trajectory lasted for 800 s, with 100 intermediate waypoints. The interval between each consecutive waypoint was set to be 8 s taking into consideration the actuator specifications. Random seeding was used inside the trajectory generation program to avoid similar trajectories from being generated. The raw data was collected at 500 Hz, including the position, velocity, acceleration, and torque in the joint space, as well as the position of the end effector in the Cartesian coordinates. The data at the beginning and the end of the trajectories was removed, and the rest was downsampled to reduce the data size. A low-pass filter with a window size of 50 was applied to remove noise and then further downsampled. The post-processed data for each trajectory lasts for about 120 s with 12,000 data samples. Fig. 8 shows one of the trajectories without standardization as an example.

4.2. Data collection for fetch robotic manipulator

The Fetch robotic manipulators are controlled and system states can be monitored via Robot Operating System (ROS) [41]. With similar approach in generating the random trajectories of MICO robotics manipulator, waypoints are sampled over the actuation space except for the head pan and head tilt joints. As shown in Fig. 9, the waypoint generator sends *waypoint* samples to the MoveIt trajectory planner. The built-in MoveIt trajectory planner is in charge of determining whether the proposed waypoints are feasible or not by considering self-collision and collision with the ground plane. Only the feasible waypoints will

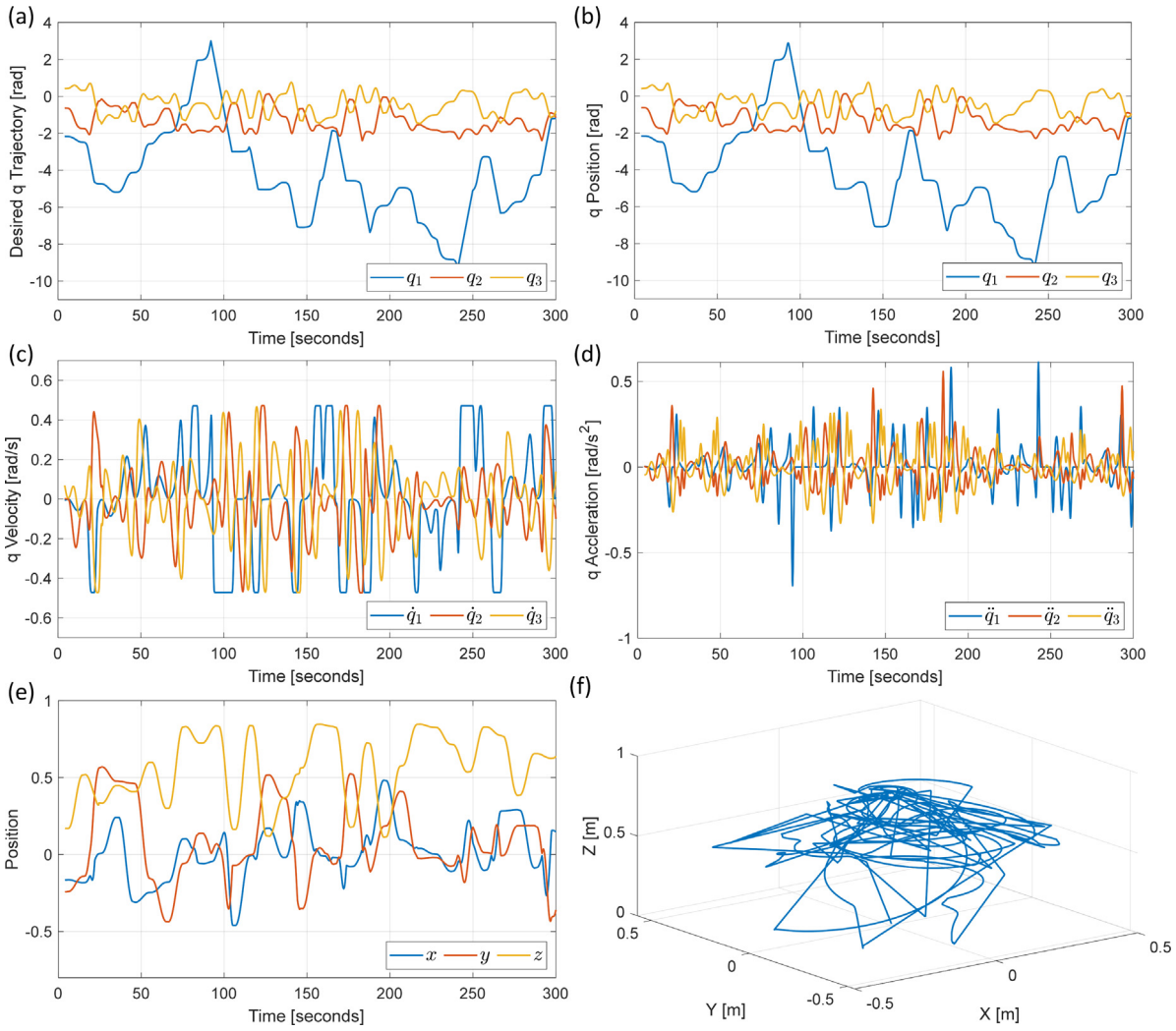


Fig. 8. One sample of the trajectories of MICO Robotic Manipulator in a real experiment.

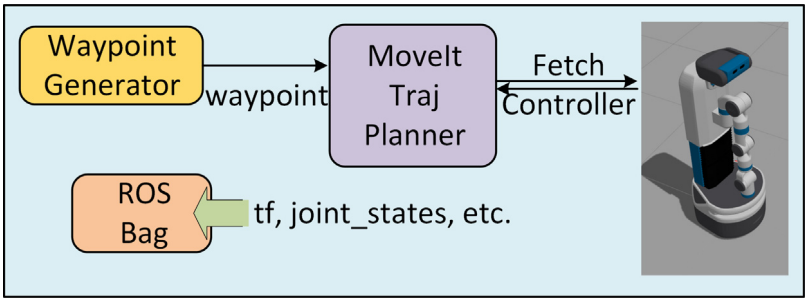


Fig. 9. The control and monitoring system was built in ROS.

be used to generate smooth trajectories by MoveIt and then performed by the real-world robotic manipulator. A ROS bag is created to record the system states, such as the frame coordinates, joint position and force/torque applied on the joints. Two trajectories were randomly generated and performed by the Fetch robotic manipulator. Each Trajectory lasts for 2400 s with around 600 feasible intermediate waypoints and raw data was collected at 100 Hz. Similar process was performed on the collected data to remove the noise and achieve downsampling, and the final dataset for each trajectory includes around 60,000 data samples. Fig. 10 shows one of the trajectories without standardization as an example.

4.3. Neural network design and training

The neural network architecture was built using Keras [42] with TensorFlow [43] running in the backend. To compare the performance between each network architecture, the same hyperparameters and optimizer were used. This includes the number of layers (three layers for the IK and five layers for the ID), number of neurons in each layer (256), activation function (Leaky ReLU with $\alpha = 0.1$) and the optimizer (Adam [44]).

For the MICO robotic manipulator, five of the performed trajectories were used to build the dataset for training purposes while the remaining trajectory was used for final performance

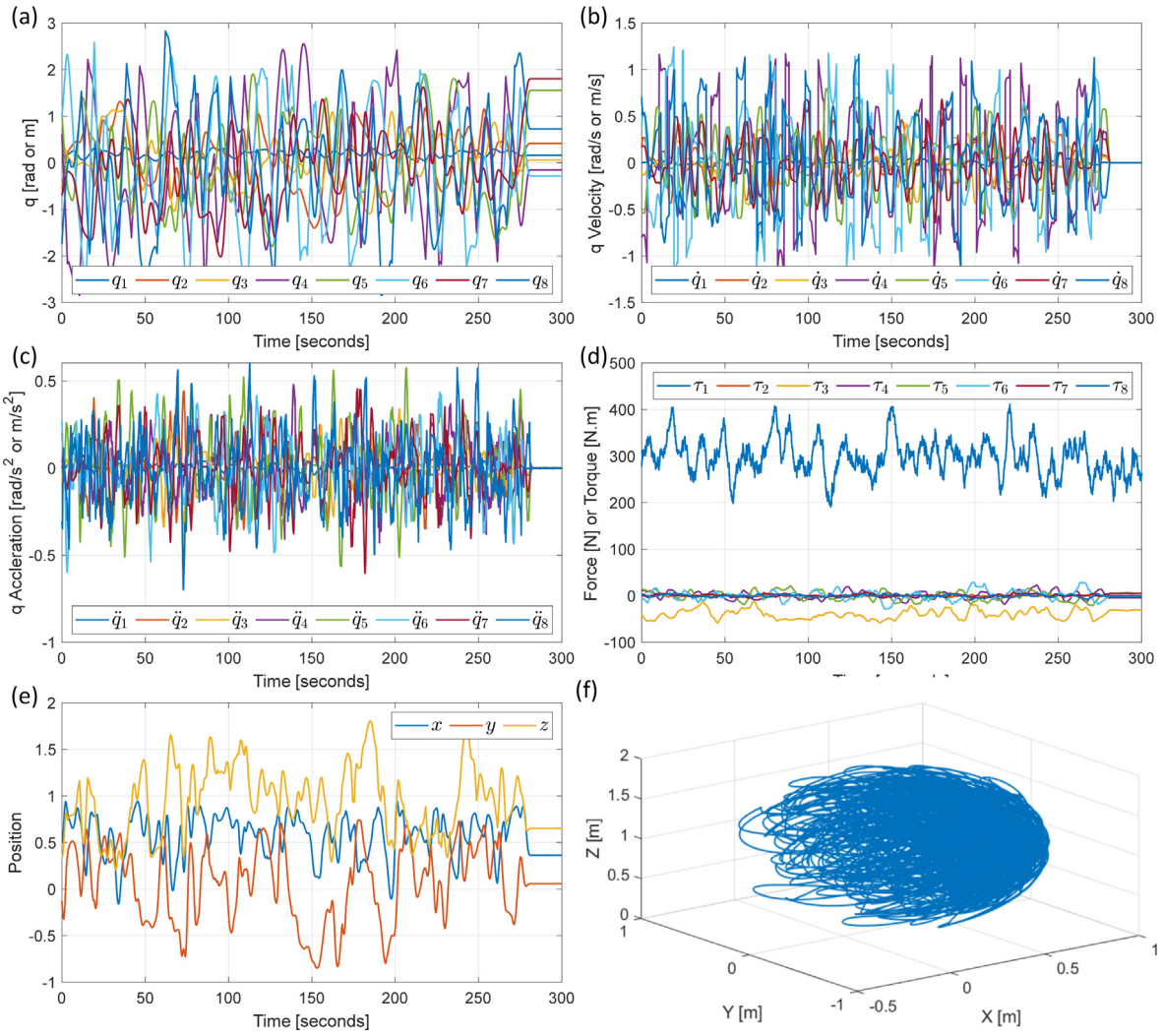


Fig. 10. One sample of the trajectories of fetch robotic manipulator in real experiment.

testing, which is defined as the *test dataset* in this paper. For the Fetch robotic manipulator, one trajectory was used for training purposes and the other one was used as the test dataset. The dataset was standardized in either case to remove the wide range difference among different data channels (feature) to gain faster convergence and more stable training process. Different standard deviation values ($\sigma \in \{0.0, 0.2, 0.4, 0.6\}$) were selected for the generator G in the GANs to evaluate the training performance of the proposed methods.

To evaluate the training performance of the proposed methods using limited data, a comparison of using different sizes of the overall data ($p \in \{1, 2, 3, 4, 5, 6\} \times 10,000$) for the training process was performed. In each training process, the dataset is further split into training and validation sets randomly. The training set consists of 80% of the whole data while the validation set covers the remaining 20%.

The training process was performed using an AMD Threadripper 2950x CPU and an NVIDIA Titan Xp GPU. Root Mean Squared Error (RMSE) was used for quantitative evaluation of the training, $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}$, where $\hat{Y}_i$ is the predicted values of Y_i. Each pair of dataset size and standard deviation was applied to the proposed methods and the baseline method to perform training for the IK and ID. Each training process includes 100 training epochs, and each training epoch covers all the training dataset. At the end of the each training epoch, validation dataset is used to evaluate the performance of the well-trained neural network on the unseen dataset selected for training, which is relatively smaller than the test dataset. To test the overall performance and the generalization of the well-trained neural network using the limited dataset, test dataset was used to estimate its overall performance over the whole actuation space. The training process of each proposed method and the baseline method takes around 23 mins to train the whole 60,000 dataset. The execution time of the generator neural network takes around 0.17 ms to perform a single prediction. For every IK and ID learning task, all of the proposed neural network algorithms converge.

Tables 4 and 5 detail the best performance of each proposed method using different dataset sizes and standard deviations for the generator for the IK and ID of the MICO and Fetch robotic manipulators, respectively. Each column of the method has two subcolumns, the first one is the loss function evaluated by the evaluation dataset and the second one is evaluated by the test dataset. For every task, 6 different sizes of dataset were used and defined by the first digit of Da#-Dev#. For example, Da4 indicates that $\mathbf{4} \times 10,000$ data points were used for the training. For every task and every size of the dataset, 4 different standard deviations were used for the generator, which is defined by the second digit of Da#-Dev#. For example, Dev2 indicates $\sigma = \mathbf{2} \times 0.2$ is applied in the generator. In the loss function evaluation of both validation dataset and training dataset, the predicted and ground-truth values were transformed back to the original data scale by

Table 4
Comparisons on the MICO robotic manipulator over the whole testing trajectory with different dataset sizes and standard deviations.

Algorithm	BiGANs[a]		DualGANs		GANs		LSGANs		Base[b]	
Inverse kinematics[c]										
Da1-Dev0[d]	0.12	0.4	0.75	0.73	0.11	0.4	0.92	1.04	0.04	0.54
Da1-Dev1	0.08	0.48	0.62	0.7	0.09	**0.39**[e]	0.87	1		
Da1-Dev2	0.09	0.44	0.78	0.76	0.11	0.46	0.81	0.98		
Da1-Dev3	0.5	0.53	0.8	0.8	0.11	0.41	0.97	1.02		
Da2-Dev0	0.1	0.34	0.52	0.64	0.09	**0.31**	0.19	0.38	0.03	0.36
Da2-Dev1	0.1	0.46	0.52	0.66	0.16	0.36	0.12	0.37		
Da2-Dev2	0.08	0.37	0.56	0.65	0.17	0.38	0.11	0.55		
Da2-Dev3	0.18	0.38	0.55	0.68	0.16	0.38	0.09	0.37		
Da3-Dev0	0.15	0.28	0.27	0.5	0.09	**0.26**	0.42	0.8	0.05	0.29
Da3-Dev1	0.13	0.32	0.32	0.51	0.08	0.29	0.08	0.33		
Da3-Dev2	0.22	0.38	0.29	0.5	0.1	0.3	0.09	0.54		
Da3-Dev3	0.14	0.33	0.32	0.48	0.15	0.34	0.17	0.75		
Da4-Dev0	0.11	**0.22**	0.33	0.47	0.1	0.25	0.16	0.29	0.06	0.26
Da4-Dev1	0.1	0.23	0.31	0.43	0.1	0.27	0.09	0.23		
Da4-Dev2	0.1	0.24	0.28	0.4	0.11	0.24	0.16	0.68		
Da4-Dev3	0.11	0.26	0.37	0.54	0.11	0.26	0.1	0.27		
Da5-Dev0	0.15	0.22	0.25	0.36	0.1	**0.2**	0.16	0.23	0.06	0.22
Da5-Dev1	0.11	0.22	0.2	0.33	0.11	0.23	0.14	0.24		
Da5-Dev2	0.1	0.23	0.49	0.6	0.11	0.24	0.15	0.26		
Da5-Dev3	0.12	0.25	0.29	0.43	0.17	0.24	0.15	0.25		
Da6-Dev0	0.15	0.18	0.14	0.17	0.13	0.18	0.14	0.18	0.09	**0.16**
Da6-Dev1	0.13	0.18	0.15	0.19	0.13	0.17	0.13	0.18		
Da6-Dev2	0.16	0.19	0.19	0.22	0.13	0.17	0.12	0.18		
Da6-Dev3	0.16	0.19	0.22	0.24	0.13	0.2	0.12	0.21		
Inverse dynamics[f]										
Da1-Dev0	0.07	0.45	0.48	0.72	0.1	0.47	0.12	0.51	0.01	**0.3**
Da1-Dev1	0.03	0.47	0.33	0.57	0.03	0.51	0.03	0.48		
Da1-Dev2	0.05	0.46	0.34	0.61	0.04	0.5	0.04	0.5		
Da1-Dev3	0.07	0.47	0.36	0.62	0.05	0.5	0.04	0.52		
Da2-Dev0	0.04	0.37	0.28	0.42	0.08	0.38	0.15	0.38	0.02	**0.26**
Da2-Dev1	0.03	0.39	0.45	0.55	0.03	0.37	0.04	0.36		
Da2-Dev2	0.06	0.38	0.31	0.46	0.04	0.39	0.05	0.41		
Da2-Dev3	0.08	0.38	0.29	0.5	0.06	0.41	0.07	0.39		
Da3-Dev0	0.04	0.36	0.25	0.46	0.06	0.36	0.08	0.36	0.02	**0.25**
Da3-Dev1	0.03	0.36	0.31	0.47	0.03	0.36	0.04	0.36		
Da3-Dev2	0.05	0.36	0.25	0.46	0.04	0.36	0.06	0.37		
Da3-Dev3	0.07	0.37	0.3	0.48	0.06	0.36	0.07	0.36		
Da4-Dev0	0.03	0.34	0.26	0.4	0.06	0.32	0.06	0.34	0.02	**0.21**
Da4-Dev1	0.03	0.33	0.27	0.4	0.03	0.32	0.04	0.32		
Da4-Dev2	0.05	0.34	0.28	0.38	0.04	0.32	0.05	0.33		
Da4-Dev3	0.07	0.33	0.34	0.43	0.07	0.33	0.07	0.34		
Da5-Dev0	0.04	0.33	0.31	0.38	0.05	0.3	0.05	0.3	0.02	**0.19**
Da5-Dev1	0.03	0.32	0.26	0.41	0.03	0.3	0.03	0.3		
Da5-Dev2	0.05	0.32	0.29	0.36	0.04	0.29	0.06	0.3		
Da5-Dev3	0.08	0.31	0.32	0.41	0.07	0.32	0.08	0.3		
Da6-Dev0	0.04	0.23	0.17	0.25	0.05	0.22	0.06	0.22	0.02	**0.14**
Da6-Dev1	0.04	0.23	0.18	0.29	0.04	0.22	0.04	0.22		
Da6-Dev2	0.07	0.24	0.31	0.35	0.06	0.23	0.06	0.23		
Da6-Dev3	0.09	0.24	0.39	0.43	0.09	0.23	0.1	0.24		

[a]The first subcolumn under each method is the validation loss and the second subcolumn is the test loss.
[b]A fully connected neural network is applied as the baseline.
[c]The loss was calculated using RMSE, each term of the loss function was inverse-transformed back to the original data scale by the standard scaler.
[d]Da#-Dev#, first digit indicates the size of data used for training , the second digit indicates the applied standard deviation for the generator. For example, Da**4**-Dev**2** indicates that $\mathbf{4} \times 10,000$ data is used for training process and $\sigma = \mathbf{2} \times 0.2$ is used for the generator.
[e]The bold value indicates the best performance among the methods using the same-size dataset.
[f]The loss was calculated using RMSE, each term of the loss function was inverse-transformed back to the original data scale by the standard scaler.

the standard scaler. Considering training on the same dataset size, the value of the best performance is highlighted in the table.

In solving the IK task of the MICO robotic manipulator, the proposed method, GANs, achieved the best performance when using the dataset in the least size. As the size of the dataset used for training increases, this advantage becomes smaller. All the proposed methods and the baseline achieved similar performance when using the whole dataset (including all five trajectories). In solving the ID task of the MICO robotic manipulator, the baseline methods slightly outperformed all the proposed methods considering the wide range of the generalized force executed at each joint. GANs and LSGANs achieve the best performance

Table 5
Comparisons of Proposed Methods on the Fetch Robotic Manipulator over the whole testing trajectory with different dataset sizes and standard deviations.

Algorithm	BiGANs[a]		DualGANs		GANs		LSGANs		Base[b]	
Inverse kinematics[c]										
Da1-Dev0[d]	0.94	0.96	0.93	0.95	1	1.01	2.14	2.14	0.61	**0.92**[e]
Da1-Dev1	0.95	0.96	0.93	0.95	0.97	0.98	1.26	1.26		
Da1-Dev2	0.92	0.96	0.93	0.95	0.95	0.97	1.17	1.17		
Da1-Dev3	0.94	0.95	0.93	0.95	0.95	0.97	2.5	2.5		
Da2-Dev0	0.97	0.98	0.9	0.91	1.01	1.02	1.06	1.06	0.71	**0.9**
Da2-Dev1	0.97	0.97	0.96	0.96	0.97	0.97	1.42	1.42		
Da2-Dev2	0.96	0.98	0.92	0.92	1	1.01	1.12	1.12		
Da2-Dev3	0.95	0.97	0.92	0.94	0.98	0.99	1.14	1.14		
Da3-Dev0	1.05	1.05	0.91	0.93	0.99	0.98	1.08	1.08	0.72	**0.89**
Da3-Dev1	1	1	0.9	0.92	1.02	1.02	1.06	1.06		
Da3-Dev2	1.02	1.03	0.91	0.93	1.04	1.02	1.02	1.02		
Da3-Dev3	1.02	1.05	0.9	0.92	0.98	1.02	1.1	1.1		
Da4-Dev0	1.04	1.05	0.9	0.91	0.99	0.99	1.19	1.19	0.74	**0.89**
Da4-Dev1	0.99	1	0.92	0.94	1	1.04	1.15	1.15		
Da4-Dev2	1	1.02	0.9	0.93	1.02	1.05	1.22	1.22		
Da4-Dev3	1	1.03	0.91	0.92	1.02	1.04	1.2	1.2		
Da5-Dev0	0.98	1	0.93	0.94	1.04	1.05	1.15	1.15	0.76	**0.89**
Da5-Dev1	1	1.04	0.91	0.92	1.01	1.03	1.13	1.13		
Da5-Dev2	1.03	1.05	0.92	0.94	1.07	1.1	1.08	1.08		
Da5-Dev3	0.99	1.01	0.9	0.92	1	1.03	1.18	1.18		
Da6-Dev0	1	1.04	0.91	0.93	1.01	1.07	1.15	1.15	0.73	**0.9**
Da6-Dev1	1.02	1.07	0.92	0.94	0.97	1	1.17	1.17		
Da6-Dev2	0.99	1.02	0.9	0.92	0.99	1.03	1.16	1.16		
Da6-Dev3	1.03	1.06	0.9	0.93	1.02	1.06	1.08	1.08		
Inverse dynamics[f]										
Da1-Dev0	2.77	**13.43**	15.46	14.32	2.88	13.65	3.51	13.66	1.4	14.22
Da1-Dev1	2.16	13.45	16.6	13.84	2.45	13.95	2.94	13.39		
Da1-Dev2	2.5	13.5	14.43	13.65	2.57	13.83	2.37	13.68		
Da1-Dev3	2.93	13.77	13.58	14.6	3.07	13.89	2.62	14.1		
Da2-Dev0	2.45	13.39	14.89	13.43	3.76	13.27	3.12	13.31	1.45	13.6
Da2-Dev1	2.06	**13.24**	12.87	13.78	2.61	13.5	2.53	13.27		
Da2-Dev2	2.54	13.26	14.58	13.61	2.65	13.55	2.43	13.36		
Da2-Dev3	3.02	13.49	14.31	13.7	3.06	13.54	2.76	13.4		
Da3-Dev0	2.49	13.18	14.25	13.44	2.89	12.88	3.21	12.97	1.5	13.23
Da3-Dev1	2.29	13.31	13.41	13.23	2.48	13	2.81	13.11		
Da3-Dev2	2.54	**12.9**	13.54	13.29	2.85	13.17	2.51	13.17		
Da3-Dev3	3.08	12.99	13.39	13.38	3.41	13.38	3.25	13.24		
Da4-Dev0	2.42	13.12	13.19	13.28	2.68	**12.79**	3.16	12.85	1.6	13.07
Da4-Dev1	2.3	12.99	13.17	13.28	2.59	13.13	2.71	12.83		
Da4-Dev2	2.61	12.89	13.3	13.43	3.2	13.25	2.66	13.13		
Da4-Dev3	3.28	12.87	12.29	13.43	3.67	13.18	3.19	12.96		
Da5-Dev0	2.47	12.83	13.38	13.33	2.93	12.88	3.19	12.81	1.63	12.92
Da5-Dev1	2.51	12.8	13.31	13.22	2.65	**12.67**	2.79	12.97		
Da5-Dev2	2.99	12.84	12.71	13.21	3.68	12.96	3.06	12.9		
Da5-Dev3	3.34	12.76	12.35	13.44	3.77	13.06	3.38	12.83		
Da6-Dev0	3.11	12.91	13.58	13.47	3.02	12.87	3.51	**12.7**	2.04	13.01
Da6-Dev1	2.92	12.89	13.57	13.6	3.35	12.91	3.46	12.94		
Da6-Dev2	2.99	12.94	12.85	13.37	4.14	13.16	3.31	12.92		
Da6-Dev3	3.79	13.01	13.19	13.33	4.31	13.04	3.94	12.84		

[a]The first subcolumn under each method is the validation loss and the second subcolumn is the test loss.
[b]A fully connected neural network is applied as the baseline.
[c]The loss was calculated using RMSE, each term of the loss function was inverse-transformed back to the original data scale by the standard scaler.
[d]Da#-Dev#, first digit indicates the size of data used for training , the second digit indicates the applied standard deviation for the generator. For example, Da**4**-Dev**2** indicates that $\mathbf{4} \times 10,000$ data is used for training process and $\sigma = \mathbf{2} \times 0.2$ is used for the generator.
[e]The bold value indicates the best performance among the methods using the same-size dataset.
[f]The loss was calculated using RMSE, each term of the loss function was inverse-transformed back to the original data scale by the standard scaler.

among all the proposed methods and achieved a loss around 0.22 Nm, while the baseline method achieved 0.14 Nm. For ID, the analytical model, which uses the mass properties estimated from CAD, performed the worst with an RMSE of 0.4082 Nm.

In solving the IK task of the Fetch robotic manipulator, the proposed methods achieved similar performance as the baseline. DualGANs outperformed all the other proposed methods and achieved a total loss of 0.91 while the baseline methods gained the least loss of 0.89. In solving the ID task of the Fetch robotic manipulator, almost all trials of the proposed methods performed better in the test evaluation than the baseline method. GANs gained the best performance with a loss of 12.67 while the baseline achieved 12.92 as the best value. The analytical model achieved an RMSE of 40.00.

It can also be noted that the baseline method has a much lower loss in the validation dataset than the proposed methods even if it performs similarly or worse in the test dataset. This "hidden" overfitting in a limited dataset may fail to generalize the neural network in the overall actuation space while the proposed methods are robust in such situations. Standard deviation also

plays an important role in the training performance, and it can be seen that in the ID task of the Fetch robotic manipulator, a "suitable" standard deviation gained the best performance compared to the same methods with either too large or too small standard deviation.

The results from the experimental validation show that the proposed algorithms can handle the estimation of the uncertainty components present in the system when compared to existing analytical methods. This could be attributed to the fact that unlike existing methods the proposed techniques were trained on real-world datasets instead of using a simplified analytical model such as physics-based simulations. In comparison to simplified analytical data, real-world datasets provide rich information including the effects of friction, damping, actuator properties, and mass-inertia effects of different links in the most realistic manner. This enables the trained neural network to perform precise predictions.

The experimental results also show that the proposed technique works well in cases of limited dataset compared to the widely used fully connected neural network, and even when the manipulator is used to track previously unseen trajectories. This validates the global approximation capabilities of the proposed techniques. The proposed technique was trained using trajectories generated randomly over the entire task space, making all points equally probable in the training dataset. This along with the sample generator and the generator-discriminator frameworks enables neural network training over the neighborhood of natural data while avoiding overfitting on the limited collected data. Together these features effectively extend the prediction capability of the proposed techniques over the entire task space.

5. Conclusion

In this paper, we have introduced a series of modified Generative Adversarial Networks for solving the inverse kinematics and dynamics of robots using real-world experimental data. Existing research has focused on learning the uncertainty along with a simplified analytical model or predicting the hindsight analytic model, which could then be used as ground truth. However, existing techniques do not allow for global estimation of the underlying model of the system. Moreover, they require extensive training datasets to perform accurate modeling, which is often time consuming.

The proposed neural network techniques apply generator and discriminator architectures to address the above mentioned limitations. Using the generator and discriminator frameworks, the proposed methods are able to perform better than state-of-the-art techniques in cases where there is insufficient data. We evaluated the proposed methods on a MICO robotic manipulator and a Fetch robotic manipulator with different-size training dataset. The experimental results show that the proposed method is able to solve the inverse kinematics and dynamics problems with the desired degree of accuracy. Also, the selection of different standard deviations for the generator was also studied in the training process. A "correctly" selected standard deviation will help the neural network achieve a lower loss in the performance and avoid overfitting.

The techniques proposed in this work open new ways for solving the inverse kinematic and the inverse dynamics problems for high dimensional nonlinear systems with a limited dataset. Future work in this domain aims at applying the proposed techniques to learn models for complicated robotic systems with different types of input data. This includes applications such as system identification techniques for animal locomotion such as that of lizards or bats based on joint tracking images. The proposed techniques also open up interesting venues for research in the domain of neural network design, such as the development of capsule based networks with dynamic routing. In comparison to existing architectures, this allows for handling local entry identification such as static or dynamic constraints.

Declaration of competing interest

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

Acknowledgments

The authors would like to thank Jiteng Yang, Bijo Sebastian and Daniel Budolak for their help in this work. We also gratefully acknowledge the support of NVIDIA, USA Corporation with the donation of the Titan Xp GPU used for this research.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.robot.2019.103386.

References

[1] Y.P. Pane, S.P. Nageshrao, J. Kober, R. Babuška, Reinforcement learning based compensation methods for robot manipulators, Eng. Appl. Artif. Intell. 78 (2019) 236–247, http://dx.doi.org/10.1016/j.engappai.2018.11.006, URL https://linkinghub.elsevier.com/retrieve/pii/S0952197618302446.

[2] Z. Shareef, P. Mohammadi, J. Steil, Improving the inverse dynamics model of the KUKA LWR IV+ using independent joint learning*, IFAC-PapersOnLine 49 (21) (2016) 507–512, http://dx.doi.org/10.1016/j.ifacol.2016.10.653, URL https://linkinghub.elsevier.com/retrieve/pii/S2405896316322650.

[3] K.M. Thu, A. Gavrilov, Designing and modeling of quadcopter control system using L1 adaptive control, Procedia Comput. Sci. 103 (2017) 528–535, http://dx.doi.org/10.1016/j.procs.2017.01.046, URL https://linkinghub.elsevier.com/retrieve/pii/S1877050917300479.

[4] L. Hawley, W. Suleiman, Control framework for cooperative object transportation by two humanoid robots, Robot. Auton. Syst. 115 (2019) 1–16, http://dx.doi.org/10.1016/j.robot.2019.02.003, URL https://linkinghub.elsevier.com/retrieve/pii/S0921889018303543.

[5] S. Kucuk, Z. Bingul, Inverse kinematics solutions for industrial robot manipulators with offset wrists, Appl. Math. Model. 38 (7–8) (2014) 1983–1999, http://dx.doi.org/10.1016/j.apm.2013.10.014, URL https://linkinghub.elsevier.com/retrieve/pii/S0307904X13006264.

[6] W.S. Rone, P. Ben-Tzvi, Continuum robot dynamics utilizing the principle of virtual power, IEEE Trans. Robot. 30 (1) (2014) 275–287, http://dx.doi.org/10.1109/TRO.2013.2281564, URL http://ieeexplore.ieee.org/document/6613525/.

[7] C. Lopez-Franco, J. Hernandez-Barragan, A.Y. Alanis, N. Arana-Daniel, A soft computing approach for inverse kinematics of robot manipulators, Eng. Appl. Artif. Intell. 74 (2018) 104–120, http://dx.doi.org/10.1016/j.engappai.2018.06.001, URL https://linkinghub.elsevier.com/retrieve/pii/S0952197618301325.

[8] T.G. Thuruthel, E. Falotico, F. Renda, C. Laschi, Model-based reinforcement learning for closed-loop dynamic control of soft robotic manipulators, IEEE Trans. Robot. 35 (1) (2019) 124–134, http://dx.doi.org/10.1109/TRO.2018.2878318, URL https://ieeexplore.ieee.org/document/8531756/.

[9] F. Stulp, E.A. Theodorou, S. Schaal, Reinforcement learning with sequences of motion primitives for robust manipulation, IEEE Trans. Robot. 28 (6) (2012) 1360–1370, http://dx.doi.org/10.1109/TRO.2012.2210294, URL http://ieeexplore.ieee.org/document/6295672/.

[10] D. Martínez, G. Alenyà, C. Torras, Planning robot manipulation to clean planar surfaces, Eng. Appl. Artif. Intell. 39 (2015) 23–32, http://dx.doi.org/10.1016/j.engappai.2014.11.004, URL https://linkinghub.elsevier.com/retrieve/pii/S0952197614002760.

[11] K. Hu, C. Ott, D. Lee, Learning and generalization of compensative zero-moment point trajectory for biped walking, IEEE Trans. Robot. 32 (3) (2016) 717–725, http://dx.doi.org/10.1109/TRO.2016.2553677, URL http://ieeexplore.ieee.org/document/7484896/.

[12] A. Gosavi, Simulation-based optimization, in: Operations Research/Computer Science Interfaces Series, vol. 55, Springer US, Boston, MA, 2015, http://dx.doi.org/10.1007/978-1-4899-7491-4, URL http://link.springer.com/10.1007/978-1-4899-7491-4.

[13] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, 2015, arXiv preprint, URL http://arxiv.org/abs/1509.02971.

[14] B. Karlik, S. Aydin, An improved approach to the solution of inverse kinematics problems for robot manipulators, Eng. Appl. Artif. Intell. 13 (2) (2000) 159–164, http://dx.doi.org/10.1016/S0952-1976(99)00050-0, URL http://linkinghub.elsevier.com/retrieve/pii/S0952197699000500.

[15] S.S. Chiddarwar, N. Ramesh Babu, Comparison of RBF and MLP neural networks to solve inverse kinematic problem for 6R serial robot by a fusion approach, Eng. Appl. Artif. Intell. 23 (7) (2010) 1083–1092, http://dx.doi.org/10.1016/J.ENGAPPAI.2010.01.028, URL https://www.sciencedirect.com/science/article/pii/S0952197610000692.

[16] R. Köker, A genetic algorithm approach to a neural-network-based inverse kinematics solution of robotic manipulators based on error minimization, Inform. Sci. 222 (2013) 528–543, http://dx.doi.org/10.1016/j.ins.2012.07.051, URL https://linkinghub.elsevier.com/retrieve/pii/S0020025512005233.

[17] A.-V. Duka, Neural network based inverse kinematics solution for trajectory tracking of a robotic arm, Proc. Technol. 12 (2014) 20–27, http://dx.doi.org/10.1016/j.protcy.2013.12.451, URL https://linkinghub.elsevier.com/retrieve/pii/S2212017313006361.

[18] A. Csiszar, J. Eilers, A. Verl, On solving the inverse kinematics problem using neural networks, in: 2017 24th International Conference on Mechatronics and Machine Vision in Practice, M2VIP, Vol. 2017-Decem, IEEE, 2017, pp. 1–6, http://dx.doi.org/10.1109/M2VIP.2017.8211457, URL http://ieeexplore.ieee.org/document/8211457/.

[19] Y. Ansari, E. Falotico, Y. Mollard, B. Busch, M. Cianchetti, C. Laschi, A multiagent reinforcement learning approach for inverse kinematics of high dimensional manipulators with precision positioning, in: 2016 6th IEEE International Conference on Biomedical Robotics and Biomechatronics, BioRob, Vol. 2016-July, IEEE, 2016, pp. 457–463, http://dx.doi.org/10.1109/BIOROB.2016.7523669, URL http://ieeexplore.ieee.org/document/7523669/.

[20] H. Toshani, M. Farrokhi, Real-time inverse kinematics of redundant manipulators using neural networks and quadratic programming: A Lyapunov-based approach, Robot. Auton. Syst. 62 (6) (2014) 766–781, http://dx.doi.org/10.1016/j.robot.2014.02.005, URL https://linkinghub.elsevier.com/retrieve/pii/S0921889014000360.

[21] F. Meier, D. Kappler, N. Ratliff, S. Schaal, Towards robust online inverse dynamics learning, in: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, Vol. 2016-Novem, IEEE, 2016, pp. 4034–4039, http://dx.doi.org/10.1109/IROS.2016.7759594, URL http://ieeexplore.ieee.org/document/7759594/.

[22] R. Rayyes, D. Kubus, J. Steil, Learning inverse statics models efficiently with symmetry-based exploration, Front. Neurorobotics 12 (2018) 68, http://dx.doi.org/10.3389/fnbot.2018.00068, URL https://www.frontiersin.org/article/10.3389/fnbot.2018.00068/full.

[23] R. Reinhart, Z. Shareef, J. Steil, Hybrid analytical and data-driven modeling for feed-forward robot control, Sensors 17 (2) (2017) 311, http://dx.doi.org/10.3390/s17020311, URL http://www.ncbi.nlm.nih.gov/pubmed/28208697.

[24] T.G. Thuruthel, E. Falotico, F. Renda, C. Laschi, Learning dynamic models for open loop predictive control of soft robotic manipulators, Bioinspir. Biomim. 12 (6) (2017) 066003, http://dx.doi.org/10.1088/1748-3190/aa839f, URL https://doi.org/10.1088/1748-3190/aa839f.

[25] B. Liang, T. Li, Z. Chen, Y. Wang, Y. Liao, Robot arm dynamics control based on deep learning and physical simulation, in: 2018 37th Chinese Control Conference, CCC, IEEE, 2018, pp. 2921–2925, http://dx.doi.org/10.23919/ChiCC.2018.8484058, URL https://ieeexplore.ieee.org/document/8484058/.

[26] A. Odena, C. Olah, J. Shlens, Conditional image synthesis with auxiliary classifier GANs, in: International Conference on Machine Learning, Sydney, Australia, 2016, URL http://arxiv.org/abs/1610.09585.

[27] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, P. Abbeel, InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets, in: International Conference on Neural Information Processing Systems, 2016, pp. 2180–2188, URL https://arxiv.org/pdf/1606.03657.pdf.

[28] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, W. Shi, Photo-realistic single image super-resolution using a generative adversarial network, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, IEEE, 2016, pp. 105–114, http://dx.doi.org/10.1109/CVPR.2017.19, URL http://ieeexplore.ieee.org/document/8099502/.

[29] J. Ho, S. Ermon, Generative adversarial imitation learning, in: 30th Conference on Neural Information Processing Systems, Barcelona, Spain, 2016, pp. 4565–4573, URL http://arxiv.org/abs/1606.03476.

[30] J. Fu, K. Luo, S. Levine, Learning robust rewards with adversarial inverse reinforcement learning, in: International Conference for Learning Representations, Vancouver, 2018, URL http://arxiv.org/abs/1710.11248.

[31] M. Mirza, S. Osindero, Conditional generative adversarial nets, 2014, arXiv preprint, URL http://arxiv.org/abs/1411.1784.

[32] X. Mao, Q. Li, H. Xie, R.Y. Lau, Z. Wang, S.P. Smolley, Least squares generative adversarial networks, in: 2017 IEEE International Conference on Computer Vision, ICCV, Vol. 2017-Octob, IEEE, 2017, pp. 2813–2821, http://dx.doi.org/10.1109/ICCV.2017.304, URL http://ieeexplore.ieee.org/document/8237566/.

[33] J. Donahue, P. Krähenbühl, T. Darrell, Adversarial feature learning, in: International Conference on Learning Representations, Toulon, France, 2017, URL http://arxiv.org/abs/1605.09782.

[34] Z. Yi, H. Zhang, P. Tan, M. Gong, DualGAN: Unsupervised dual learning for image-to-image translation, in: 2017 International Conference on Computer Vision, Venice, 2017, URL http://arxiv.org/abs/1704.02510.

[35] Kinova, MICO - Robotic arm, 2018, URL https://www.kinovarobotics.com/en/knowledge-hub/all-kinova-products.

[36] M. Wise, M. Ferguson, D. King, E. Diehr, D. Dymesich, Fetch & Freight: Standard Platforms for Service Robot Applications, Tech. rep..

[37] I.J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial networks, in: Neural Information Processing Systems, 2014, pp. 2672–2680, URL https://arxiv.org/abs/1406.2661.

[38] M. Arjovsky, S. Chintala, L. Bottou, Wasserstein Gan, 2017, arXiv preprint, URL http://arxiv.org/abs/1701.07875.

[39] F. Pedregosa FABIANPEDREGOSA, V. Michel, O. Grisel OLIVIERGRISEL, M. Blondel, P. Prettenhofer, R. Weiss, J. Vanderplas, D. Cournapeau, F. Pedregosa, G. Varoquaux, A. Gramfort, B. Thirion, O. Grisel, V. Dubourg, A. Passos, M. Brucher, M. Perrot andÉdouardand, A. Duchesnay, F. Duchesnay EDOUARDDUCHESNAY, Scikit-Learn: Machine Learning in Python, Tech. Rep., 2011, pp. 2825–2830, URL http://scikit-learn.sourceforge.net.

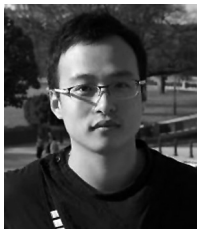
[40] J. Bergstra, D. Yamins, D. Cox, Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, in: 30 Th International Conference on Machine Learning, Vol. 28, tlanta, Georgia, 2013, URL http://proceedings.mlr.press/v28/bergstra13.pdf.

[41] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, A. Ng, ROS: An Open-Source Robot Operating System, Tech. rep., URL http://stair.stanford.edu.

[42] F. Chollet, et al., Keras, 2015, URL https://keras.io.

[43] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, M. Schuster, R. Monga, S. Moore, D. Murray, C. Olah, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, Tensorflow: Large-scale machine learning on heterogeneous systems, 2015, URL https://www.tensorflow.org/.

[44] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: International Conference for Learning Representations, San Diego, 2015, URL https://arxiv.org/pdf/1412.6980.pdf.



Hailin Ren (S'16) received the B.S Degree in Nanjing University of Science and Technology in 2013 and M.S. in Columbia University in 2014. He is currently pursuing the Ph.D. degree at the Virginia Polytechnic Institute and State University (Virginia Tech) under the supervision of Prof. P. Ben-Tzvi. His research interests include Artificial Intelligence, Computer Vision and Autonomous Robotics.

Pinhas Ben-Tzvi (S'02–M'08–SM'12) received the B.S. degree (summa cum laude) in mechanical engineering from the Technion—Israel Institute of Technology, and the M.S. and Ph.D. degrees in mechanical engineering from the University of Toronto. He is currently an Associate Professor of Mechanical Engineering at Virginia Tech. His current research interests include robotics and intelligent autonomous systems, human–robot interactions, robotic vision, machine learning, mechatronics design, systems dynamics and control,and novel sensing and actuation.

Dr. Ben-Tzvi is the recipient of the 2019 Virginia Tech Teaching Excellence Award and the 2018 Faculty Fellow Award. Dr. Ben-Tzvi is Technical Editor for the IEEE/ASME Transactions on Mechatronics, Associate Editor for ASME Journal of Mechanisms and Robotics, and an Associate Editor for IEEE Robotics and Automation Magazine, Automation and Systems and served as an Associate Editor for IEEE ICRA 2013-2018. He is a member of the American Society of Mechanical Engineers (ASME).