International Journal of Humanoid Robotics
Vol. 14, No. 3 (2017) 1750015 (24 pages)
© World Scientific Publishing Company
DOI: 10.1142/S0219843617500153

Accurate Task-Space Tracking for Humanoids with Modeling Errors Using Iterative Learning Control

Pranav A. Bhounsule*
Department of Mechanical Engineering,
University of Texas at San Antonio,
One UTSA Circle, San Antonio TX 78249, USA
pranav.bhounsule@utsa.edu

Katsu Yamane
Disney Research, 4720 Forbes Avenue,
Lower Level, Suite 110, Pittsburgh PA 15213, USA
kyamane@disneyresearch.com

Received 12 October 2016
Accepted 21 February 2017
Published 22 May 2017

Precise task-space tracking with manipulator-type systems requires an accurate kinematic model. In contrast to traditional manipulators, sometimes it is difficult to obtain an accurate kinematic model of humanoid robots due to complex structure and link flexibility. Also, prolonged use of the robot will lead to some parts wearing out or being replaced with a slightly different alignment, thus throwing off the initial calibration. Therefore, there is a need to develop a control algorithm that can compensate for the modeling errors and quickly retune itself, if needed, taking into account the controller bandwidth limitations and high dimensionality of the system. In this paper, we develop an iterative learning control algorithm that can work with existing inverse kinematics solvers to refine the joint-level control commands to enable precise tracking in the task space. We demonstrate the efficacy of the algorithm on a theme-park-type humanoid doing a drawing task, serving drink in a glass, and serving a drink placed on a tray without spilling. The iterative learning control algorithm is able to reduce the tracking error by at least two orders of magnitude in less than 20 trials.

Keywords: Iterative learning control; task-space tracking; humanoids; zero-reference model; constraint optimization.

1. Introduction

Many applications of manipulator-type systems involve precise task-space control. Examples include industrial manipulators that do welding and personal humanoid robots that fold clothes. Industrial manipulators rely on good CAD models and

*Corresponding author.

high-gain servo-controllers to accomplish precise end-effector motions. However, the techniques that work well for industrial manipulators may not transfer to humanoid robots for the following reasons:

(1) Humanoid robots are made lightweight due to weight and size constraints leading to flexible links. Thus, traditional CAD models which are based on rigid body assumptions are not valid. Additional modeling terms are needed to account for link flexibility, which can be hard.
(2) Humanoid robots tend to have small actuators for safety reasons and/or may have a low bandwidth controller. This makes it hard to implement a precise servo.
(3) Humanoid robots have large number of joints and long limbs, especially the human-size robots. Thus, small parameters errors in the CAD model can lead to big errors at the end-effectors.
(4) When the humanoid robots are used long-term (e.g., personal robots), some parts may wear out or be replaced with a slightly different alignment. Hence the original CAD model is not valid anymore.

Thus, for accurate task-space control of humanoid robots, one needs a method that can compensate for the modeling errors by modifying the joint-level control commands and to make up for part wear and/or replacement. We present an iterative learning control algorithm that can address the above issues.

In this paper, we show that a combination of constrained optimization and iterative learning control using data from motion capture (MoCap) system can enable high-fidelity task-space tracking even with strict joint limits. First, we create a rigid body-based kinematics model and fit the parameters by minimizing the error between model-predicted end-effector pose and measured pose using the MoCap system. When we invert the obtained kinematic model for task-space control, the accuracy is limited because of unmodeled effects, specifically joint flexibility and sensor noise, and position tracking errors due to limited control bandwidth. We therefore employ iterative learning control to improve the tracking performance. The iterative learning control algorithm modifies the desired end-effector motion based on end-effector tracking errors. We show that the learning algorithm is able to provide accurate tracking on three tasks on a theme-park-type humanoid: (i) drawing the figure eight, (ii) serving a drink without spilling, and (iii) serving a drink placed on a tray.

2. Background and Related Works

The main issue with task-space control is the lack of accurate kinematic models due to reasons mentioned earlier. Traditional feedback control methods such as proportional-integral-derivative control[1] are the preferred methods to correct for modeling errors because they have a simple structure and can be relatively easy to hand-tune. However, they are not preferred when the plant is subject to unexpected disturbances. In this case, it is common to have an adaptive controller that modifies the control parameters to make up for the varying loads.[2,3] However, most feedback control techniques rely on setting high gains which necessitate the use of high-bandwidth feedback control, typically 500 Hz or more. In our case, the control bandwidth of 120 Hz limits us to relatively low gains.

Learning-based method have also been used to do task-space control. One approach is to directly build an inverse kinematics model using the experimental data.[4,5] The biggest issue with this approach is that the inverse mapping is not unique.[6] To overcome the multiple solution nature of the inverse mapping, Oyama *et al.*[7] used multiple neural networks to represent the inverse kinematic solutions locally in different regions of the state space. These individual networks are called experts. Next, another neural network, called the gating network, is used to choose an expert to obtain the inverse kinematics solution. One of the problem with the above method is that the construction of the gating network becomes difficult in high dimensions.

Iterative learning control (ILC) can correct for modeling errors to enable high-fidelity tracking. In its simplest form, ILC modifies the control command in every iteration in the following way: the command at trial i is the sum of the command in trial $(i-1)$ and a control gain times the tracking error in trial $(i-1)$.[8] Because the tracking errors are reduced iteratively at every trial, the control gains can be kept small.

Traditionally, in ILC, the learning is at the joint level.[9] However, it is quite straightforward to extend ILC to task level using the appropriate mapping from the task space to the joint space. For example, Arimoto *et al.*[10,11] used the linearized mapping, i.e., the Jacobian, to map the incremental change in position from the task level to the joint level and showed the efficacy of their algorithm on a four-link manipulator in simulation. In our ILC algorithm, we use the nonlinear map from task space to joint space and show the efficacy of the algorithm experimentally on a humanoid robot. Specifically, we evaluate the nonlinear map by using an inverse kinematics solver which finds a solution within joint limits. The main improvement over Arimoto's algorithm is that our algorithm is able to handle joint limits (see Ref. 12 for more details). This paper is an extension of the paper we previously published[13] in the following ways: we give extensive details about the kinematic modeling including experimental results and provide an extensive evaluation of the iterative learning control algorithm on a number of tasks on the humanoid robot.

3. Hardware

3.1. *Humanoid robot*

We use a 37-degree-of-freedom, hydraulically-powered, fixed-base humanoid robot shown in Fig. 1. Each joint has either a rotary potentiometer or a linear variable

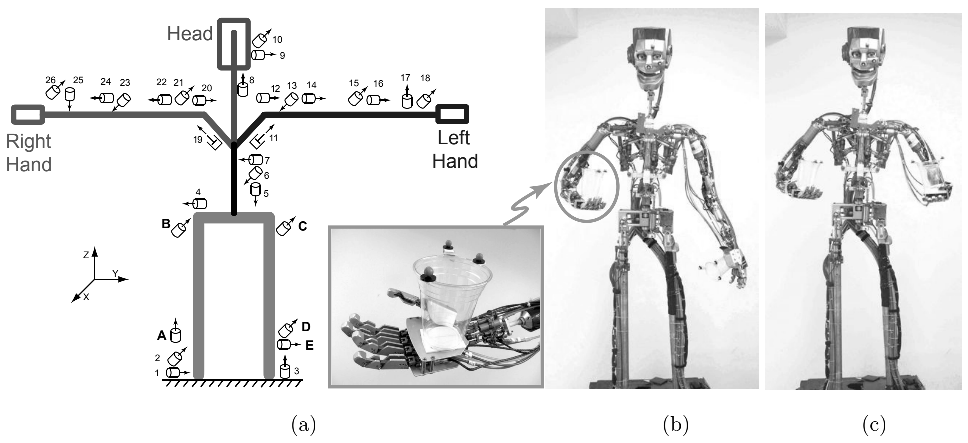


Fig. 1. (a) The kinematic model of the humanoid robot in the zero-reference pose has 26 degrees of freedom. In the reference pose all the joint angles are zero. The names of these joints are: (1) **B**ody **F**orebend (BF), (2) **B**ody **S**way (BS), (3) **B**ody **T**wist (BT), (4) **P**elvis (P), (5) **T**orso **T**wist (TT), (6) **T**orso **S**way (TS), (7) **T**orso **F**orebend (TF), (8) **H**ead **T**urn (HT), (9) **H**ead **N**od (HN), (10) **H**ead **TiLT** (HTLT), (11) **L**eft **S**houlder **S**hrug (LSS), (12) **L**eft **A**rm **F**orebend (LAF), (13) **L**eft **A**rm **O**ut (LAO), (14) **L**eft **A**rm **S**wing (LAS), (15) **L**eft **E**lbow (LE), (16) **L**eft **W**rist **T**wist (LWT), (17) **L**eft **W**rist **I**n–**O**ut (LWIO), (18) **L**eft **W**rist **F**orward–**B**ack (LWFB), (19) **R**ight **S**houlder **S**hrug (RSS), (20) **R**ight **A**rm **F**orebend (RAF), (21) **R**ight **A**rm **O**ut (RAO), (22) **R**ight **A**rm **S**wing (RAS), (23) **R**ight **E**lbow (RE), (24) **R**ight **W**rist **T**wist (RWT), (25) **R**ight **W**rist **I**n–**O**ut (RWIO), and (26) **R**ight **W**rist **F**orward–**B**ack (RWFB). The joints named **A**, **B**, **C**, **D**, and **E** are neither actuated nor sensed. Gross robot specifications are: height = 1.8 m (5′11″), body width = 0.28 m (9″), and length of hand = 0.67 m (2′2″). (b) A photo of the humanoid robot that we used in this study. The above pose is our reference pose for the inverse kinematics (see Sec. 5.1) for the writing and glass tasks. A plastic glass is glued to the right hand of the robot and is shown in the inset. We put three markers on the top of the glass which we track using our motion capture system. (c) The rest pose for the tray task. There are three markers on top of glasses attached to the hands. We use the glasses to elevate the markers so that they are in clear view of the motion capture system.

differential transformer to sense the joint position. There are two levels of control. At the lowest level, there is a single processor per joint. The processor runs a 1 kHz control loop that does high-gain position control using individual position data from joint sensor, velocity data from differentiated and filtered position data from joint sensor, and actuator force data from the pressure transducer in the valve. The gains on the lowest-level controller are preset and cannot be changed at runtime. At the highest level, there is a single computer that communicates with all the low-level processors at 120 Hz, sending desired joint position commands and receiving measured positions. The highest level serves as our interface to the robot, i.e., the input is a position command and the joint position is the measured variable, both of which occur at a rate of 120 Hz.

3.2. *Marker-based motion capture*

We use OptiTrack MoCap system along with their software suite ARENA.[14] We use eight cameras to track the three passive retro-reflective markers attached at each of the end-effectors: the head and the two hands. Using the position data from the three markers, the ARENA software computes the position and the orientation of each of the end-effectors.

The motion capture system is first used to create a kinematic model of the robot and subsequently used for iterative learning control to improve on the kinematic model for accurate task-space tracking. In all experiments, the position and the orientation of the end-effectors from the MoCap system and the robot's joint position in potentiometer tics are recorded simultaneously at 120 Hz.

4. Kinematic Model for Humanoid

4.1. *Joint model*

We assume a linear relation between the joint position in radians or meters and the measured joint position signal in counts from the potentiometers or linear variable differential transformers. Our model is

$$q_i = K_i(q_{i,\text{counts}} - q^0_{i,\text{counts}}), \tag{1}$$

where i denotes the joint number, K_i is the gain in radians per count for revolute joint or meters per count for prismatic joint, q_i is the angle in radians (denoted θ) or linear position in meters (denoted as s), $q_{i,\text{counts}}$ is the measured angle or linear position in counts, a digital signal from 0 to 4095, as we use a 12-bit encoder, and $q^0_{i,\text{counts}}$ is the bias for the angle or for the position.

4.2. *Kinematic model*

We use the zero-reference kinematic modeling.[15–18] In the zero-reference modeling approach the zero position (the reference configuration in which all joint positions have a value equal to zero) can be arbitrarily fixed,[19] unlike the more popular Denavit–Hartenberg modeling approach[20] where the pose depends on the robot geometry. Because we use the zero-reference kinematic modeling approach, we need to have a fixed zero-reference configuration, and it is chosen as shown in Fig. 1(a). Next, we choose the unit vector that defines the direction of motion of the joint axis. By defining a point through which the axis passes, we obtain the location of the joint axis.[19]

The homogeneous transformation $\boldsymbol{T}_i$ defines the motion of a point on the link attached to the joint i ($i = 1, 2, \ldots, 26, A, B, \ldots, E$) with respect to the previous joint. We define the homogeneous transformation, $\boldsymbol{T}_i$, for revolute and prismatic joints next.

Revolute joint. For this joint we have

$$\boldsymbol{T}_i^{\text{revolute}} = \begin{bmatrix} & \boldsymbol{R} & & (\boldsymbol{I} - \boldsymbol{R})\boldsymbol{r}^T \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2}$$

where $\boldsymbol{I}$ is the 3×3 identity matrix, $\boldsymbol{r} = [x_i, y_i, z_i]$ is a 3×1 vector that gives the location of the revolute joint with respect to previous joint in the local reference frame, and $\boldsymbol{R}$ is a 3×3 rotation matrix and is shown next:

$$\boldsymbol{R} = \begin{bmatrix} u_x^2 v\theta + c\theta & u_x u_y v\theta - u_z s\theta & u_x u_z v\theta + u_y s\theta \\ u_x u_y v\theta + u_z s\theta & u_y^2 v\theta + c\theta & u_y u_z v\theta - u_x s\theta \\ u_x u_z v\theta - u_y s\theta & u_y u_z v\theta + u_x s\theta & u_z^2 v\theta + c\theta \end{bmatrix},$$

where θ is the angle turned by the joint in radians, $\boldsymbol{u} = [u_x \quad u_y \quad u_z]$ defines the joint axis, and $s\theta = \sin\theta$, $c\theta = \cos\theta$ and $v\theta = 1 - \cos\theta$. To obtain the transformation given by Eq. (2) we do the following: first translate the point by a distance $-\boldsymbol{r}$ to pass through the center of the revolute joint; then rotate by an angle θ using the Rodrigues equation; and finally translate by $\boldsymbol{r}$ to restore it to the original position.[21]

The transform, $\boldsymbol{T}_i^{\text{revolute}}$, has five unknowns; two in $\boldsymbol{u}$, two in $\boldsymbol{r}$, and one in θ. We obtain the third component of $\boldsymbol{u}$ using the identity: $u_x^2 + u_y^2 + u_z^2 = 1$. By definition, the zero-reference model fixes one component of $\boldsymbol{r}$. Hence, there are just two unknowns in the position vector $\boldsymbol{r}$.

Prismatic joint. For this joint we have

$$\boldsymbol{T}_i^{\text{prismatic}} = \begin{bmatrix} 1 & 0 & 0 & su_x \\ 0 & 1 & 0 & su_y \\ 0 & 0 & 1 & su_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3}$$

where we define $\boldsymbol{u}$ as in the previous section and s defines the linear motion of the joint in meters.

The transform, $\boldsymbol{T}_i^{\text{prismatic}}$, has three unknowns; two in u and one in s. We obtain the third component of $\boldsymbol{u}$ using the identity: $u_x^2 + u_y^2 + u_z^2 = 1$.

The transformation $\boldsymbol{T}_e^i$ maps the ith-end-effector frame with the zero-reference pose. The zero-reference pose is shown in Fig. 1(b):

$$\boldsymbol{T}_e^i = \begin{bmatrix} & \boldsymbol{R}_e^i & & \boldsymbol{r}_e^i \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4}$$

Here, $\boldsymbol{r}_e^i = [x_e^i, y_e^i, z_e^i]$ is a 3×1 vector that defines the location of a marker with respect to the previous joint in the local reference frame, and we obtain $\boldsymbol{R}_e^i$ from Euler angles ϕ_x^i, ϕ_y^i, ϕ_z^i, using the x–y–z convention of Euler angles (also called Bryant angles):

$$\boldsymbol{R}_e^i = \boldsymbol{R}_x^i \boldsymbol{R}_y^i \boldsymbol{R}_z^i.$$

The transformation given by Eq. (4) has six parameters in all: three for the position (x_e^i, y_e^i, z_e^i) and three for the orientation $(\phi_x^i, \phi_y^i, \phi_z^i)$.

Like Eq. (4), the transformation T_r that maps the robot reference frame to the camera reference frame in the zero-reference pose is

$$\boldsymbol{T}_r = \begin{bmatrix} & \boldsymbol{R}_r & & \boldsymbol{r}_r \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{5}$$

From Fig. 1(b), we see that the humanoid has a closed-loop structure from the waist down (gray thick lines that constitute robot legs) and a tree structure from the waist up (thin multicolored lines that constitute the robot torso, the head, and the hands). We break the loop joint at joint 4. Now we have four serial chains: three from each of the end-effectors (the head, the left-hand, and the right-hand) to joint 1 and one from joint 4 to joint 3. Next, we show how to combine these serial chains to give the kinematic model and the end-effector pose.

If $\boldsymbol{T}_c^i$ is the transformation that maps the motion of the ith-end-effector with respect to camera frame then

$$\boldsymbol{T}_c^{\text{head}} = \boldsymbol{T}_r \boldsymbol{T}_1 \boldsymbol{T}_2 \boldsymbol{T}_A \boldsymbol{T}_B \boldsymbol{T}_4 \boldsymbol{T}_5 \boldsymbol{T}_6 \boldsymbol{T}_7 \boldsymbol{T}_8 \boldsymbol{T}_9 \boldsymbol{T}_{10} \boldsymbol{T}_e^{\text{head}}, \tag{6}$$

$$\boldsymbol{T}_c^{\text{left-hand}} = \boldsymbol{T}_r \boldsymbol{T}_1 \boldsymbol{T}_2 \boldsymbol{T}_A \boldsymbol{T}_B \boldsymbol{T}_4 \boldsymbol{T}_5 \boldsymbol{T}_6 \boldsymbol{T}_7 \boldsymbol{T}_{11} \boldsymbol{T}_{12} \boldsymbol{T}_{13} \boldsymbol{T}_{14} \boldsymbol{T}_{15} \boldsymbol{T}_{16} \boldsymbol{T}_{17} \boldsymbol{T}_{18} \boldsymbol{T}_e^{\text{left-hand}}, \tag{7}$$

$$\boldsymbol{T}_c^{\text{right-hand}} = \boldsymbol{T}_r \boldsymbol{T}_1 \boldsymbol{T}_2 \boldsymbol{T}_A \boldsymbol{T}_B \boldsymbol{T}_4 \boldsymbol{T}_5 \boldsymbol{T}_6 \boldsymbol{T}_7 \boldsymbol{T}_{19} \boldsymbol{T}_{20} \boldsymbol{T}_{21} \boldsymbol{T}_{22} \boldsymbol{T}_{23} \boldsymbol{T}_{24} \boldsymbol{T}_{25} \boldsymbol{T}_{26} \boldsymbol{T}_e^{\text{right-hand}}. \tag{8}$$

Finally, to ensure that the loop joint constraint closes at joint 3, we define the transformation $\boldsymbol{T}_r^{\text{loop}}$ as

$$\boldsymbol{T}_r^{\text{loop}} = \boldsymbol{T}_4 \boldsymbol{T}_C \boldsymbol{T}_D \boldsymbol{T}_E \boldsymbol{T}_3. \tag{9}$$

The transformation $\boldsymbol{T}_r^{\text{loop}}$ gives the position and the orientation of the end of the loop joint, i.e., at joint 3. For the loop to close we need to satisfy five constraints (one less than a maximum of six, because of the single degree of freedom at joint 3). The five constraints are enforced by choosing appropriate values for the five angles at joints **A**, **B**, **C**, **D**, and **E**. These five angles are neither sensed nor actuated on the robot.

We use the kinematics modeling utility in the rigid body dynamics simulator SDFast[22] to model the robot kinematics.

4.3. *Initial model parameters*

The initial model parameters estimation is done as follows. We first set the robot in the zero-reference pose as shown in the schematic diagram in Fig. 1(a). The positive direction for the joint axis corresponds to the direction that the joint should move for the raw joint position measurement system to give an increasing value. Setting up the joint axis direction this way ensures that the gains (K_is) are positive. We measure link lengths using a ruler. We obtain the bias terms, $q^0_{i,\text{counts}}$, from the measured position in counts in the reference pose. A rough estimate of the gain, K_i, for each joint is obtained as follows. We move the joint whose gain we want to estimate by a random amount. We measure the change in the angle in radians using a protractor, and also in tics using the potentiometer on the joint. We can estimate K_i by dividing the change in angle in radians to the change in angle in tics.

4.4. *Parameters identification*

For kinematic modeling, we need a rich dataset. We drive each joint using a sinusoidal trajectory that covers the full range of joint motion. The frequencies of individual joints are chosen to be nonmultiples of each other. This ensures that we were not getting repeating motion. The total data collection time is about 30 min. However, for analysis purposes, we down-sample the data by a factor of 240 to get about 750 unique poses. A kinematic model is specified and parameters are fit using nonlinear least squares. The model has limited accuracy due to unmodeled effects such as sensor noise and link deflection due to load. Thus, when the kinematic model is inverted for task-space control there are errors. We therefore use an iterative learning control algorithm to modify the control command based on error between the predicted end-effector position and orientation using the inverse kinematics model and the measurements from the motion capture system to enable accurate task-space tracking (see Sec. 5.2).

The parameters identification step improves on the initial estimates obtained earlier using the data collected from MoCap. This is done by updating the model parameters based on minimization of the squared prediction error using the parameters optimization software called SNOPT,[23] a nonlinear constraint optimization software based on sequential quadratic programming.

Our parameters optimization problem is as follows: for a given set of poses specified by the joint position data in counts, we find parameters for the model given by Eqs. (1) and (6)–(9) that best explain the measured end-effector position and orientation data from all three end-effectors, and across all the poses.

We use a single cost function consisting of sum of end-effector position and orientation error:

$$\text{Cost} = \underbrace{\frac{\sum_{i=1}^{N_{\text{head}}}(\Delta r_i^2 + w_i \Delta\theta_i^2)}{N_{\text{head}}}}_{\text{Cost-head}} + \underbrace{\frac{\sum_{i=1}^{N_{\text{left-hand}}}(\Delta r_i^2 + w_i \Delta\theta_i^2)}{N_{\text{left-hand}}}}_{\text{Cost-left-hand}} + \cdots$$
$$+ \underbrace{\frac{\sum_{i=1}^{N_{\text{right-hand}}}(\Delta r_i^2 + w_i \Delta\theta_i^2)}{N_{\text{right-hand}}}}_{\text{Cost-right-hand}}, \quad (10)$$

$$(\Delta r_i)^2 = (\Delta x_i)^2 + (\Delta y_i)^2 + (\Delta z_i)^2, \quad (11)$$

$$(\Delta \theta_i)^2 = (\Delta \theta_i^x)^2 + (\Delta \theta_i^y)^2 + (\Delta \theta_i^z)^2, \quad (12)$$

where $\Delta x_i = x_i^{\text{model}} - x_i^{\text{expt}}$ denotes the difference between values from the kinematic model [obtained from appropriate transform in Eq. (6), (7) or (8)] and the measurement for the x-coordinate of one of the end-effector for the ith-pose and so on. Similarly, $\Delta\theta_i^x = \theta_i^{x,\text{model}} - \theta_i^{x,\text{expt}}$ denotes the difference between values from the kinematic model and the measurement for the Euler angle corresponding to rotation along x-axis for the ith-pose and so on. N_{head}, $N_{\text{left-hand}}$ and $N_{\text{right-hand}}$ denote the numbers of observations for the head, the left-hand, and the right-hand, respectively. The weight, w_i, denotes a weight on the angular position with respect to the linear position and was set to 1 for the parameters identification.

We now describe the optimization parameters. We have a total of 178 parameters given in the list below. Item 1 gives the parameters for the joint model [see Eq. (1)] and the rest of the items are for the kinematic model [see Eqs. (2)–(9)].

(1) We have two parameters, K_i and $q^0_{i,\text{counts}}$, for each joint. As there are 26 joints, we have a total of 52 joint-level parameters.
(2) We specify the position and the orientation of the ith-rigid body formed by the markers with respect to the end-effector using the transformation $\boldsymbol{T}_e^i$ and given by Eq. (4). This transformation has six parameters; three for the position and three for the orientation. For the three end-effectors, there are a total of 18 parameters.
(3) We specify the position and the orientation of the robot's reference frame with respect to the camera frame using the transformation $\boldsymbol{T}_r$ and given by Eq. (5). This transformation has six parameters; three for position and three for orientation.
(4) The tree structure from joint 5 or Torso Twist and up [thin multicolored lines in Fig. 1(b)] has 20 revolute joints and two sliding joints. Each revolute joint adds four parameters [one less than the usual five for revolute joints because we know the joint position in radians from the joint model, see Eq. (2)]; and each sliding joint adds two parameters [one less than the normal three for prismatic joints because we know the joint position in meters from joint model, see Eq. (3)]. Thus we have a total of 92 $(20 \times 4 + 2 \times 6)$ parameters for the tree structure.
(5) The closed-loop structure in the lower body (shown as thick gray line in Fig. 1) has nine revolute joints but only four revolute joints are sensed and actuated. In each actuated joint, there are two numbers that describe the joint axis. Since there are four actuated joints, there are a total of eight parameters. The only length parameters in the loop structure are the height and width of the loop. Thus, the loop structure has a total of 10 parameters.

4.5. *Results for kinematic modeling*

Next, we present results for the parameters identification problem described in Sec. 4.4. Table 1 gives the root-mean-square errors for the positions and the orientations of the head, the left hand, and the right hand. The average position accuracy

Table 1. Fit accuracy: root-mean-square errors for the positions and the orientations of the head, the left hand, and the right hand. See Sec. 4.4 for details.

Name	Head	Left hand	Right hand	Average root-mean-square error
Δx (cm)	0.47	0.71	0.74	0.64
Δy (cm)	0.28	0.42	0.56	0.42
Δz (cm)	0.57	0.76	0.64	0.66
Δr (cm)	**0.80**	**1.12**	**1.12**	**1.00**
$\Delta \theta_x$ (degrees)	0.67	0.97	1.15	0.93
$\Delta \theta_y$ (degrees)	0.8	1.22	1.56	1.19
$\Delta \theta_z$ (degrees)	1.03	1.07	0.97	1.02
$\Delta \theta$ (degrees)	**1.47**	**1.89**	**2.17**	**1.84**

is 1 cm and the average orientation accuracy is 1.84°. For a 1.8 m tall robot, an accuracy of 1 cm corresponds to a percentage error of only 0.6%, which is reasonably small, but not small enough for accurate task-space tracking, as we will see in the next section.

The residual errors for the head are lower than the left hand and the right hand, indicating a better fit for the head. Both hands have similar residual errors. We believe that the higher residual error in the hands as compared to the head is because of two reasons: link deflection due to gravity loading and sensor noise. Since the hands have a bigger mass and larger moment arm than the head, the hands would have a bigger end-effector deflection than the head. Also, since the hands have more degrees of freedom and therefore more joint sensors, the effect of sensor noise is more dominant for the hand than the head. We could have improved the fit further by adding a model that accounts for link deflection due to gravity loading,[24] and a model accounting for sensor accuracy.[25] However, we decided to forego this because we can easily improve the tracking accuracy for joint and task-space control by using learning or feedback control algorithms.

Figure 2 shows the histograms of the position and the orientation errors. The errors are normally distributed with a mean of zero, except for the position error in gravity direction, Δz_i, which has a mean around 0.5 cm. We speculate that this error bias is due to gravity loading which we have not accounted for in our model.

Consistency check. We do three more nonlinear least squares calculations for individual end-effectors separately using the individual cost terms defined in Eq. (13). We compare these fits with each other and with the least squares fit for all the end-effectors taken together described by all terms in Eq. (13). To check the consistency we compare some of the common parameters in the four optimizations, namely the pin alignment and the link lengths in the lower body. The maximum variation between the four optimizations for the pin axis is 4° and for the link length is 4 cm, primarily because of parameter redundancy (e.g., the height of the closed loop and the height of the mid-body). However, since the variations we obtain across different optimizations are within 2%, we conclude that the solutions are consistent.

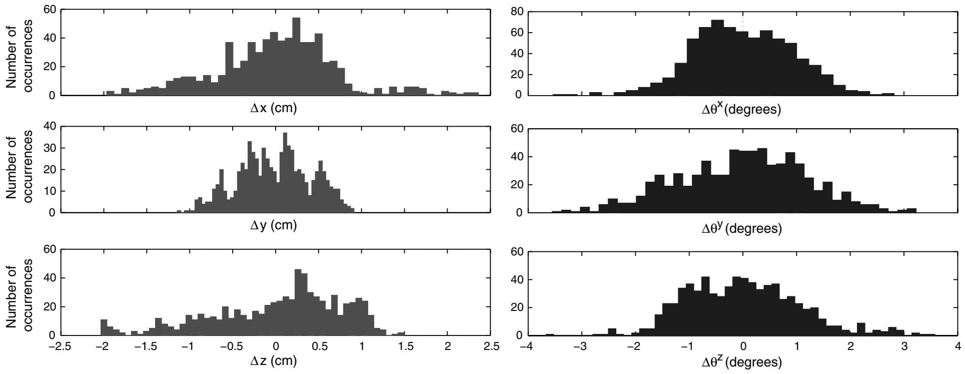


Fig. 2. Distribution for the fit accuracy of the left hand: histograms of the position errors are on the left-hand side and the orientation errors are on the right-hand side. There are about 750 datapoints per histogram.

5. Task-Level Control

We use the inverse of the identified kinematic model for task-level control but found it not accurate enough for precise tracking. We therefore used an iterative learning control algorithm to improve the tracking performance and described it in this section.

5.1. *Inverse kinematics*

We need an inverse kinematics solver to map the desired end-effector motion to joint space for the low-level position servo. Sometimes it is possible to find a closed-form inverse kinematics solution (see, e.g., Ref. 26), but this is not always the case. In such instances, a numerical optimization provides a straightforward and generalizable method of finding an inverse kinematics solution, especially for redundant robots such as humanoids.[27] Specifically, by choosing a suitable cost function one can bias the solution to use certain joints over the other ones.

We use the optimization software SNOPT[23] again, to develop an inverse kinematics solver. The problem here is to find the joint angles as a function of time, $\boldsymbol{\theta}(t)$, to minimize the cost function g, subject to end-effector constraints $\mathbf{h_1}$, the lower joint limits $\mathbf{h_2}$, and upper joint limits $\mathbf{h_3}$. We define these functions next:

$$g(\boldsymbol{\theta}(t)) = \sum_{i=1}^{n_{\text{dof}}} w_i(\theta_i(t) - \theta_i^{\text{ref}}(t))^2 + \sum_{j=1}^{\hat{n}_{\text{eff}}} W_j(X_{j,\text{des}} - f_j(\boldsymbol{\theta}(t)))^2, \tag{13}$$

$$\mathbf{h_1}(\boldsymbol{\theta}(t)) = \mathbf{X}_{i,\text{des}}(t) - \mathbf{f}_i(\boldsymbol{\theta}(t)) \leq |\epsilon|, \quad \text{where } i = 1, 2, \ldots, n_{\text{eff}}, \tag{14}$$

$$\mathbf{h_2}(\boldsymbol{\theta}(t)) = \boldsymbol{\theta}(\boldsymbol{t}) - \boldsymbol{\theta}_{\min} \geq 0, \tag{15}$$

$$\mathbf{h_3}(\boldsymbol{\theta}(t)) = \boldsymbol{\theta}(\boldsymbol{t}) - \boldsymbol{\theta}_{\max} \leq 0. \tag{16}$$

Here $\mathbf{f}$ is the identified kinematic model, $\mathbf{X}_{i,\mathrm{des}}$ is the ith-desired pose element for the inverse kinematics, n_{dof} is the degrees of freedom that are used for the end-effector control, $\hat{n}_{\mathrm{eff}}$ are the end-effector pose elements with a soft constraint, the n_{eff} are the end-effector pose elements with hard constraints. We assume $\epsilon = 10^{-3}$.

The following are free parameters which the motion designer can tune in order to bias the motion.

(1) The reference angles for the joints, $\theta_i^{\mathrm{ref}}(t)$. We choose $\theta_i^{\mathrm{ref}}(t) = \theta_i^{\mathrm{ref}}$ (constant values), which are the joint angles corresponding to the robot pose shown in Fig. 1.
(2) The joint weights, w_i. We intuitively chose a weight of 1 for the joints for the hand degrees of freedom and choose a weight of 10 for the joints in the mid- and lower-body. This choice of this particular weight distribution has the effect of finding solutions that involve bigger excursions of the hand degrees of freedom than the body degrees of freedom, similar to what humans would do when doing tasks using their hands. Alternately, the weights can be chosen using inverse optimization using motion capture data (see, e.g., Ref. 28).
(3) The end-effector weights, W_j. These weights are only used in the tray task for the orientation of left hand. They were tuned to 10.

5.2. *ILC algorithm*

The joint angle solution $\boldsymbol{\theta}(t)$ leads to poor performance when we implement it on the robot because of the imperfect kinematic model. So, we implement an iterative learning control to improve the tracking performance of the inverse kinematics solution. We describe the algorithm next.

Let i represent the trial number and j the time index that goes from 1 to n_j (end time). Let the reference motion in task space be defined by $\mathbf{X}_{\mathrm{ref}}(j)$. Here we have concatenated the position and the orientation in the vector $\mathbf{X}$. The inputs to the inverse kinematics solver are the desired poses in end-effector space, which we denote by $\mathbf{X}^i_{\mathrm{des}}(j)$. Let the measured position in task space be defined by $\mathbf{X}^i_{\mathrm{act}}(j)$. Let the error between actual end-effector position and reference position be denoted by $\mathbf{e}^i(j) = \mathbf{X}^i_{\mathrm{ref}}(j) - \mathbf{X}_{\mathrm{act}}(j)$. Our ILC algorithm is shown in Fig. 3 and described below:

(1) Set the error $\mathbf{e}^{\mathbf{0}}(j) = 0$ and initialize the desired position in the task space $\mathbf{X}^0_{\mathrm{des}}(j) = \mathbf{X}_{\mathrm{ref}}(j)$.
(2) For subsequent trials do:
    - *Command modification in end-effector space*: Update the feed-forward position command using the tracking error at trial i, $\mathbf{X}^i_{\mathrm{des}}(j) = \mathbf{X}^{i-1}_{\mathrm{des}}(j) + \boldsymbol{\Gamma}\boldsymbol{e}^{i-1}(j)$. The manually-tuned learning gain, Γ, is a 6×6 matrix of the form: $\Gamma = \mathrm{diag}\{\gamma_1, \gamma_2, \ldots, \gamma_6\}$. Further, for ILC to converge we need $0 < \gamma_i \leq 1$.
    - *Command initialization in joint space*: For the desired position in task space $\mathbf{X}^i_{\mathrm{des}}(j)$, use the inverse kinematics solver to find a desired joint command $\boldsymbol{\theta}^i(j)$.

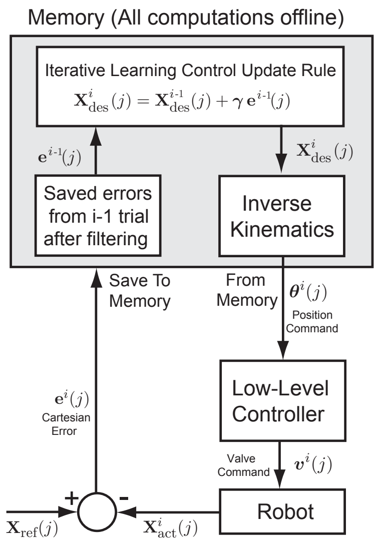

Memory (All computations offline)
Iterative Learning Control Update Rule
$\mathbf{X}^i_{\text{des}}(j) = \mathbf{X}^{i\text{-}1}_{\text{des}}(j) + \gamma\,\mathbf{e}^{i\text{-}1}(j)$
$\mathbf{e}^{i\text{-}1}(j)$
$\mathbf{X}^i_{\text{des}}(j)$
Saved errors from i-1 trial after filtering
Inverse Kinematics
Save To Memory
From Memory
$\boldsymbol{\theta}^i(j)$
Position Command
Low-Level Controller
$\mathbf{e}^i(j)$
Cartesian Error
Valve Command
$v^i(j)$
Robot
+
-
$\mathbf{X}_{\text{ref}}(j)$
$\mathbf{X}^i_{\text{act}}(j)$

Fig. 3. Block diagram of our end-effector space iterative learning control algorithm. We filter the Cartesian space errors $e^{i-1}(j)$ using a zero-phase filter before applying the iterative learning update rule. The superscript i denotes the trial number and j denotes a time instance.

- *Command execution on robot*: Send the feed-forward commands $\boldsymbol{\theta}^i(j)$ $(j = 1, 2, \ldots, n_j)$ to the low-level controller. Save the resulting tracking errors in the end-effector space, $\mathbf{e}^i(j)$ $(j = 1, 2, \ldots, n_j)$.

(3) Stop when the error metric e^i_{norm} does not improve between trials. The learnt feed-forward command is then $\boldsymbol{\theta}^i(j)$ $(j = 1, 2, \ldots, n_j)$.

The error metric to check convergence is given as follows:

$$\boldsymbol{e}^i_{\text{norm}} = \frac{1}{n_j} \sum_{j=1}^{n_j} \sum_{k=1}^{n_{\text{eff}}} (e^i_k(j))^2, \tag{17}$$

where $e^i_k(j)$ is the tracking error in the pose element k, at iteration i and at time j, $n_{\text{eff}} = 6$, and n_j is the total datapoints in the trial.

Tuning the learning gain. The free parameters in the ILC algorithm are the learning gains, $\boldsymbol{\Gamma}$. The choice of these parameters affects the learning rate, convergence, and stability (see Ref. 12 for conditions on the learning gain). We simplify the choice, by using the same learning parameter for all six degrees of freedom. Thus $\boldsymbol{\Gamma} = \gamma I$, where I is 6×6 identity matrix and $0 < \gamma \leq 1$.

Zero-phase filtering. We use a zero-phase filter to remove sensor noise and to provide a zero-phase lag.[29] The zero-phase filtering is done as follows. We first filter in the forward direction using a second-order Butterworth filter with cutoff frequency of 1 Hz. Next, we pad the forward-filtered signal with about 120 reflected datapoints

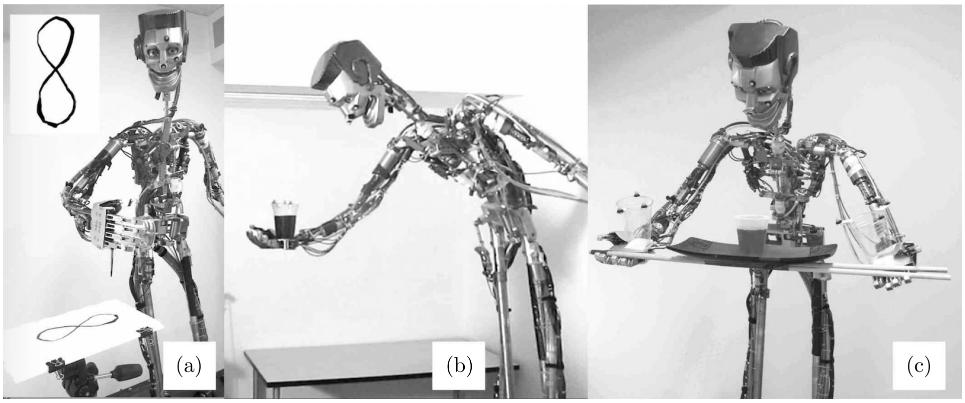


Fig. 4. (a) Robot doing the writing task. The figure "eight" drawn by the robot is shown in the inset. (b) Robot doing the glass task. The robot is serving the drink without spilling. The learning was done with no liquid in the glass. (c) The robot doing the tray task with a drink placed on the tray. The learning was done without the tray.

at the beginning and at the end. Then, we reverse the concatenated signal and filter again with the same Butterworth filter. This process of forward filtering followed by reverse filtering produces a signal with zero delay. The padding of data removes unnecessary transients in the beginning and the end of the filtered signal. Note that the zero-phase filtering is anti-causal and it needs the sensor values for the entire trajectory and is done offline.

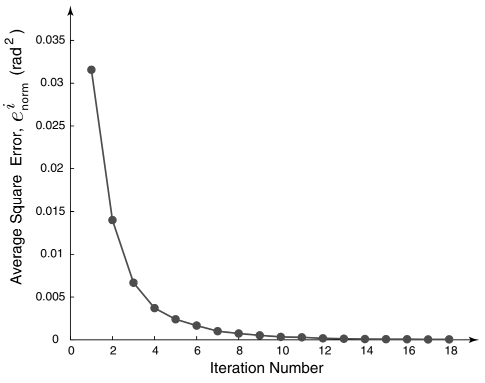
5.3. *Results for task-level iterative learning control*

We demonstrate the implementation of our algorithm on the humanoid robot on three control tasks (see Fig. 4): (1) drawing the figure eight (writing task), (2) using one hand to serve a glass of drink (glass task), and (3) serving a drink atop a tray using both hands (tray task). For the writing and glass tasks, the joints 1–7 in the body and joints 19–26 in the right hand were used. For the tray tasks, all joints except the joints controlling the head (joints 8–10) were used. Note that the joints A–E are neither sensed or actuated but need to move in order to allow the loop joint in the lower body to move. Also see Figs. 1(b) and 1(c) for the experimental setup and placement of markers for learning.

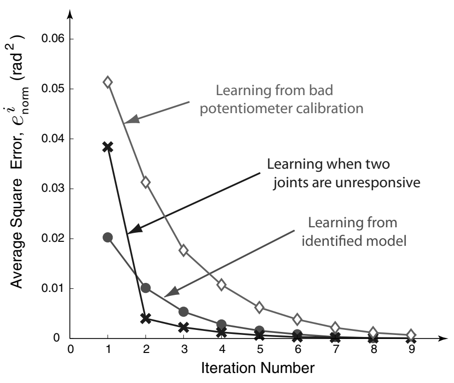
***Task* 1 (*Writing task*).** The writing task consists in drawing the lemniscate, which is the figure eight or the ∞ symbol. We specify the lemniscate equation in $\mathbf{X}_{\text{des}}(t)$. We use the inverse kinematic solver (see Sec. 5.1) to compute the joint position command $\boldsymbol{\theta}(t)$. Here $n_{\text{eff}} = 6$ corresponding to the position and orientation of the right hand and $\hat{n}_{\text{eff}} = 0$. The joint command is then played back on the robot. The motion of the end-effector is tracked by three markers placed on the right hand. The kinematic model does not take into account the actuator dynamics or the link deflection, and does not produce error-free tracking. Finally, we use the inverse ILC algorithm to improve the tracking performance.

The learning parameter γ was manually tuned to 0.3 by running a few trials on the robot. While the robot is learning, the robot draws in air and the motion capture system helps to measure the motion of the end-effector. The ILC algorithm uses the tracking error to improve the performance. The error metric [see Eq. (17)] for trial 1 is 3×10^{-2}, and it decreases to 2×10^{-5} in 18 trials. This is almost a three orders of magnitude improvement. The rate of convergence is shown in Fig. 5(a).

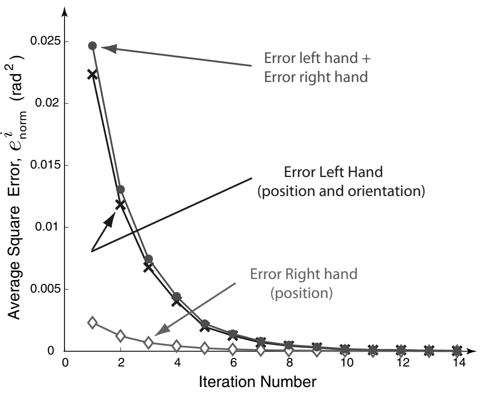
Figure 6(a) shows errors in the position and the orientation as a function of time for the first trial and the converged trial. We can see that the error is reduced to almost zero by the learning algorithm. Figures 7(a) and 7(b) show the plots of the drawing tasks in the x–y-plane (horizontal plane) and the x–z-plane (fore–aft plane).



(a)



(b)



(c)

Fig. 5. Convergence of errors as a function of iteration number for: (a) the writing task, (b) the glass task, and (c) the tray task.

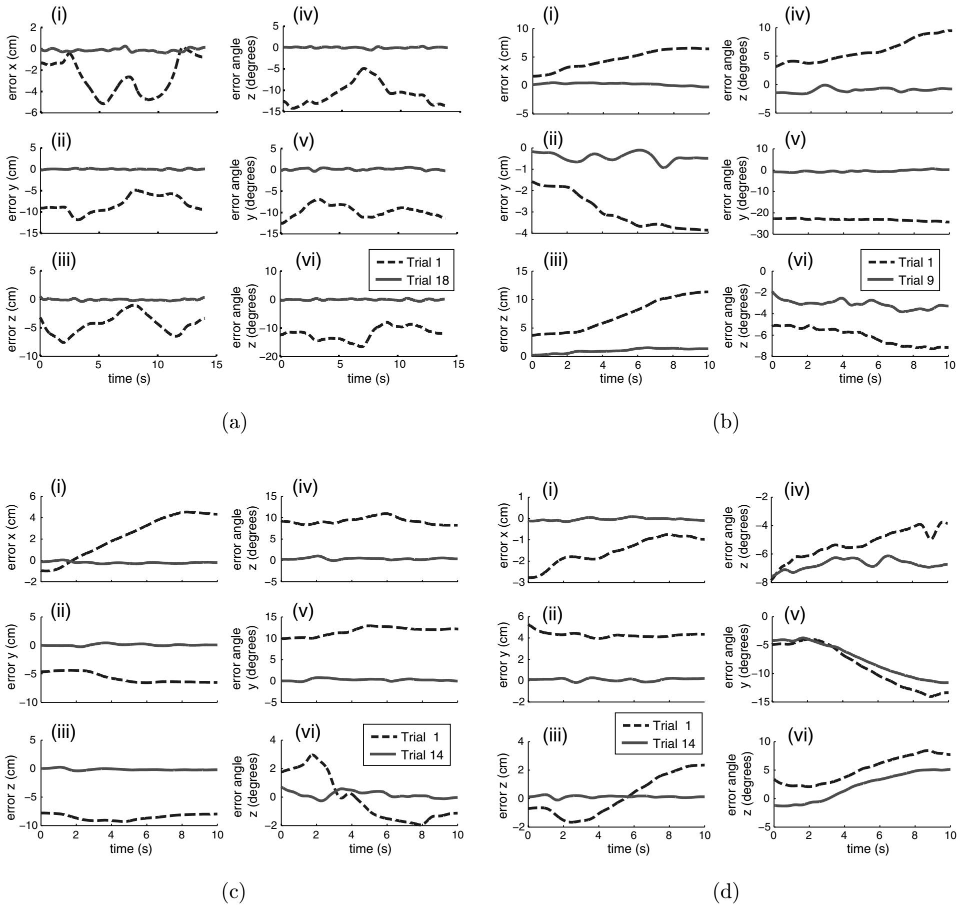


Fig. 6. Position and orientation errors for: (a) the writing task, (b) the glass task (learning incorrect potentiometer calibration), (c) tray task (left hand), and (d) tray task (right hand). (i)–(iii) Errors in the position in the x-, y-, and z-directions in cm and (iv)–(vi) errors in the angle in degrees in the x-, y-, and z-directions, respectively. x is in the fore–aft direction, y is in the robot left–right direction, and z is in the up–down direction. We do not learn orientation error in the left hand. This is enforced as a soft constraint in the inverse kinematics.

The solid black line is the reference. The dash-dotted line is the robot motion during trial 1. The dashed lined shows the converged motion at trial 18.

After the robot has learnt the motion, we made the robot to draw the lemniscate on a piece of paper using a brush dipped in black paint. A snapshot of the robot after completing the task is shown in Fig. 4(a) and the actual drawing is shown in the inset.

***Task* 2 (*Glass task*).** The glass task consists of moving glass in a straight line in the fore–aft direction while maintaining a constant height and constant orientation throughout the motion, as if the robot is serving a drink to a person in front of it.

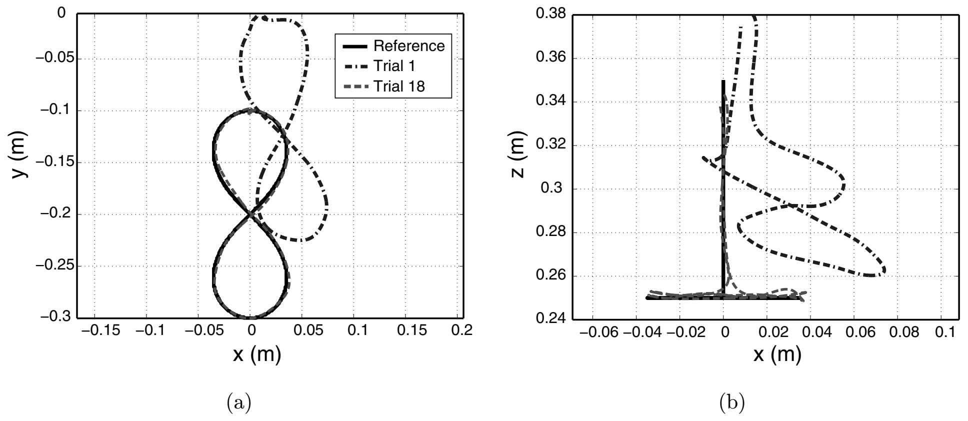


Fig. 7. (a) Plot in the $x{-}y$-plane (z-axis lines up with gravity). (b) Plot in the $x{-}z$-plane (fore–aft plane of robot). The dash-doted lines and dashed lines are the marker positions measured by the motion capture system.

Figure 4(b) shows the task executed with the converged trial and with the glass filled with a liquid.

We use the same value for the learning parameter, $\gamma = 0.3$. Also, $n_{\text{eff}} = 6$ corresponding to the position and orientation of the right hand and $\hat{n}_{\text{eff}} = 0$. We did learning in three scenarios and we describe them next. For all the learning scenarios, the motion capture system tracked a set of three markers placed on top of the glass. The three markers were used by the motion capture software to give the glass position and orientation for the learning algorithm.

Learning from identified model. We initialize the learning from the kinematics model we identified. Trial 1 produced an error of 2×10^{-2} [see Eq. (17)]. This is understandable because our forward model ignores the actuator dynamics and the link deflection. However, after using iterative learning control the error norm decreased to 2×10^{-4} in eight trials. This is about a two orders of magnitude improvement. A plot of the convergence rate is shown with filled circle in Fig. 5(b).

Learning incorrect potentiometer calibration. In order to test if our learning algorithm can learn from incorrect potentiometer calibration, we change the gain and the bias by 10% on joint 7 (bends the torso in the fore–aft direction), joint 20 (moves the right shoulder in the front–back direction), and joint 23 (right elbow joint) (see Fig. 1). We use the earlier kinematic model that does not account for the incorrect potentiometer calibration. The error in trial 1 was 5×10^{-2} but it decreased to 7×10^{-4} in nine trials. This is almost two orders of magnitude improvement. A plot of the convergence rate is shown with diamond shape in Fig. 5(b). Also, Fig. 6(b) shows the errors in tracking for trial 1 and trial 9.

Learning when some joints are unresponsive. One situation that can come when such robots are deployed is that one or more joints might be unresponsive in

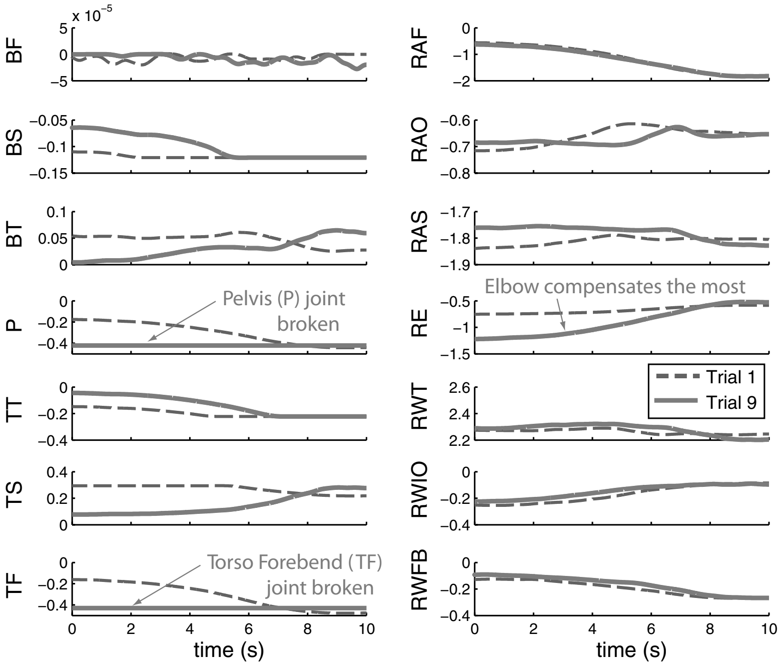


Fig. 8. Joint command versus time for normal operations (dashed lines) and for learnt motion after accommodating for unresponsive joints (continuous lines). The P joint and the TF both are frozen and will not move. The RE joint compensates the most for task-space control.

the event of a controller failure. In such cases, it is desirable to be able to temporarily accommodate the failure. The ILC framework can be used for temporary fault accommodation by exploiting the system kinematic redundancy.

To simulate a fault, we made the controller on two joints unresponsive. The two joints we made unresponsive were the Pelvis joint (bends the mid-body in the fore–aft direction) and Torso Forebend (bends the upper body in the fore–aft direction) (see Fig. 1). The system first runs the converged command from the first case presented in this section [trial 1 shown by the cross in Fig. 5(b)]. This gives a rather large error (about 4×10^{-2}) in the task space. We set new joint limits in the inverse kinematics solver and use the ILC algorithm to improve the tracking errors in task space. In nine trials, the ILC algorithm converges and produces the net error of 6×10^{-5}. This is almost two orders of magnitude improvement. The crosses in Fig. 5(b) show the convergence rate.

Figure 8 shows the joint command sent to the robot for all the 14 joints for this experiment. The dashed lines indicate the initial command that assumes all joints are working fine. The solid lines show the learnt command. Note that the Elbow joint mostly compensates for the unresponsive Pelvis and Torso Forebend joints.

***Task* 3 (*Tray task*).** The tray task consists of the robot moving a tray with both hands in a straight line in the fore–aft direction while maintaining a constant height and constant orientation of the *tray* throughout the motion. Figure 4(c) shows the task executed with the learnt command and with a filled glass placed on the tray.

There were practical issues in getting the tray task to work. The robot is designed such that the robot hands can get within about 75 cm from each other. Thus the robot needs a suitably wide tray. The ideal way to move an object with both hands is to maintain the position of both hands with respect to each other and a constant orientation of the hands. This ensures that the tray moves level with respect to the world frame. However, the robot joints limits are quite severe that we could not find inverse kinematics solution that will give a big range of motion for the tray while respecting the position and orientation for both hands. We modified the tray supports in such a way that the robot makes line contact with the tray on the left hand and point contact with the tray on the right hand [see Fig. 4(c)]. This way we do not need to do orientation control on the right hand. This eases the hard constraint on the right hand letting us do a large range of motion.

For this task, we are interested in maintaining the orientation and position of the left hand and the position of the right hand. Thus $n_{\mathrm{eff}} = 9$ which denotes the hard constraints in Eq. (14). We try to maintain a suitable orientation of the right hand using soft constraint, thus $\hat{n}_{\mathrm{eff}} = 3$. We tuned the weight $W_j = 10$ in Eq. (13). The rest pose given by θ_i^{rest} is shown in Fig. 1(c). The learning parameter γ was set at 0.3 for all end-effector elements. The positions were measured in meters and the orientations were measured in terms of quaternions. While the robot is learning, the tray is not placed on the hands. The tracking errors are used by the ILC algorithm to improve the performance.

The error metric [see Eq. (17)] for trial 1 was 2×10^{-2} and was 2×10^{-3} for the left hand and the right hand, respectively. This decreased to 3×10^{-4} and 3×10^{-5} for the left hand and right, respectively, in 14 trials. This is almost three orders of magnitude improvement for each hand individually. The rate of convergence is shown in Fig. 5(c).

Figure 6(c) shows errors in the position and orientation as a function of time for the first trial and the converged trial for the right and left hands. We can see that the error is reduced to almost zero by the learning algorithm. Note that the orientation error for the right is nonzero because we did not learn this error. We enforced the right hand orientation as a soft constraint in the inverse kinematics.

After the robot has learnt the motion, we checked if the ILC algorithm indeed works by placing a tray with a glass filled with liquid. A snapshot of the robot after completing the task is shown in Fig. 4(c).

6. Discussion

We have provided a comprehensive technique for accurate task-space tracking for a humanoid robot with serial and parallel linkages using a MoCap system. We started with an approximate kinematic model and identified its parameters using nonlinear least squares using a constraint optimization software. The average errors in the position and orientation were 1 cm and 1.84°, respectively. The limited accuracy was because of joint flexibility and sensor noise which was not factored in our rigid body kinematics model. Thus, using the kinematics model, we were unable to get accurate task-space tracking.

We then improved on the tracking by incorporating an iterative learning control algorithm that corrected the joint command based on errors in end-effector tracking. We tested the algorithms on a drawing task and serving a drink task, both involving one hand and serving a drink involves using a tray with both hands. In each of these cases, the task-space error was reduced by at least two–three orders of magnitude in a maximum of 18 trials.

Practical kinematic calibration for humanoid robots. Humanoid robots, unlike traditional industrial manipulators, might need periodic calibration for the following reasons: (1) some humanoids (e.g., the robot used in this research) are retrofitted with replacement parts during their life cycle; (2) to keep the cost low, humanoids are often fitted with cheap sensors (e.g., potentiometers) which need frequent calibration; (3) some humanoids have special transmissions such as pulleys, cables, and tendons which need to be frequently calibrated; and (4) sometimes CAD models of humanoids may not be available. Hence a quick and accurate technique for calibration is needed.

The marker-based motion capture system allows us to collect rich data from multiple end-effectors simultaneously. In our study, the motion capture time lasted for 30 min for a 26-degree-of-freedom humanoid. Starting from a crude model of the robot which was estimated using a protractor and a ruler, our nonlinear least squares optimization program was able to fit 178 parameters in about 1 h on a standard desktop computer (circa 2010). Our fit accuracy was 1 cm for the position, which is within 0.6% when compared with the robot height, and 1.84° for the orientation, averaged across all the end-effectors. Thus the kinematic calibration procedure we propose is fast, accurate, and can be automated if desired.

Better models give better convergence. Though ILC scheme is designed to correct for modeling errors, better models give better convergence.[30] This is clear when we compare the two test cases in the glass task: (1) learnt from an identified model versus (2) learnt from identified model but with incorrect pot calibration. In the former case, the error in the converged trial is 2×10^{-4} which is smaller than the error in the converged trial of 7×10^{-4} in the latter case.

Another place where better models lead to better convergence is evidenced is when we compare the writing task with the glass task. The errors in the converged trials for the writing task and glass task are 2×10^{-5} and 2×10^{-4}, respectively. Thus in the writing task, the final convergence is an order of magnitude better than in the glass task. This is explained as follows: in the glass task, the robot has to stretch its hands and body outward to reach out. The robot is flexible and the structural loading causes link deflection which is not accounted in our model. On the other hand, in the writing task, the robot does not have to reach out and the link deflection is much smaller. In other words, our kinematic model for the writing task is much more accurate than in the glass task. The evidence for the above explanation can be seen by comparing the errors in the z-position between the writing task shown in Fig. 6(a)(iii) and the glass task shown in Fig. 6(b)(iii). It is seen that the error in the z-direction keeps increasing as the robot stretches its hand to move the glass to serve the drink.

Use of existing robot model. Our method treats the inverse kinematics solver as a black-box during the learning process. This is advantageous for two reasons: (1) There is no need to rewrite the inverse kinematics solver. This is specially advantageous for humanoid robots that have an inverse kinematics solver already available. (2) Even if the robot model changes a bit, for example due to wear and tear or part replacement, there is no need to recalibrate the model.

Handling joint limits. From our experience, we know that humanoid robots often operate close to the position limits. In our method, the joint limits are handled by the constrained optimization at the inverse kinematics solver (see Fig. 3). In all the experiments reported here, we had multiple joints at their position limits, but the learning proceeded seamlessly, converging to a small tracking error. We believe that this is a significant advantage of our method over others that work at the velocity level and use the Jacobian to map from task space to joint space.

Joint limits. Another way of doing the learning is to do the update in the joint space[10,11]:

$$\theta^{i}_{\mathrm{des}}(j) = \theta^{i-1}_{\mathrm{des}}(j) + \boldsymbol{\gamma}\widehat{J}^{-1}\boldsymbol{e}^{i-1}(j). \tag{18}$$

The disadvantage of the above update rule is that if one is close to the joint limit that the projection via the Jacobian $\widehat{J}^{-1}$ may not give an improvement at all if one is close to the joint limit. This might cause the ILC algorithm to not converge or, even worst, to become unstable. Please see Bhounsule *et al.*[12] showing comparison of our method with that of Arimoto *et al.*[10,11] Our method does not suffer from this because we use a nonlinear inverse that find solutions within the joint limits.

Fault accommodation. Fault diagnosis and isolation is critical to reduce downtime and also for timely intervention. Once the fault is detected, the ILC algorithm we present here can be used to accommodate failures by exploiting the robot kinematic redundancy. We have presented one such scenario in this paper: the robot serves the glass without spilling even though two of its joints are unresponsive. A future direction would be to exploit the structural redundancy (redundant hardware, e.g., two motors on one joint) and the functional redundancy (hardware that performs different functions, e.g., estimates position using inertial measurement unit if the potentiometer breaks)[31] to develop workarounds when joints on the robot become unresponsive.

Limitations of our method. Our method has all the limitations of ILC: it is an offline method; it needs manual tuning to work well; and it can only improve a single trajectory at a time. In addition, our method needs an inverse kinematics solver that is able to find solutions within joint limits. This can be computationally expensive and can lead to issues if the manipulator is in singular configuration.

Appendix A. Multimedia Extension

A video of the robot doing the three tasks is available as a multimedia extension on the following url: https://www.disneyresearch.com/publication/accurate-task-space-tracking/.

References

1. K. J. Astrom, *PID Controllers: Theory, Design and Tuning* (Instrument Society of America, 1995).
2. T. B. Cunha, C. Semini, E. Guglielmino, V. J. De Negri, Y. Yang and D. G. Caldwell, Gain scheduling control for the hydraulic actuation of the HyQ robot leg, in *Proc. COBEM*, Vol. 4 (ABCM, 2009), pp. 673–682.
3. D. Sun and J. K. Mills, High-accuracy trajectory tracking of industrial robot manipulator using adaptive-learning scheme, in *American Control Conf.*, Vol. 3 (IEEE, 1999), pp. 1935–1939.
4. A. Guez and Z. Ahmad, Solution to the inverse kinematics problem in robotics by neural networks, in *IEEE Int. Conf. Neural Networks* (IEEE, 1988), pp. 617–624.
5. R. Köker, C. Öz, T. Çakar and H. Ekiz, A study of neural network-based inverse kinematics solution for a three-joint robot, *Robot. Auton. Syst.* **49**(3) (2004) 227–234.
6. M. I. Jordan and D. E. Rumelhart, Forward models: Supervised learning with a distal teacher, *Cogn. Sci.* **16**(3) (1992) 307–354.
7. E. Oyama, N. Y. Chong, A. Agah and T. Maeda, Inverse kinematics learning by modular architecture neural networks with performance prediction networks, in *Proc. IEEE Int. Conf. Robotics and Automation*, Vol. 1 (IEEE, 2001), pp. 1006–1012.
8. S. Arimoto, S. Kawamura and F. Miyazaki, Bettering operation of robots by learning, *J. Robot. Syst.* **1**(2) (1984) 123–140.
9. D. A. Bristow, M. Tharayil and A. G. Alleyne, A survey of iterative learning control, *IEEE Control Syst.* **26**(3) (2006) 96–114.
10. S. Arimoto, M. Sekimoto and S. Kawamura, Iterative learning of specified motions in task-space for redundant multi-joint hand–arm robots, in *Proc. IEEE Int. Conf. Robotics and Automation* (IEEE, 2007), pp. 2867–2873.
11. S. Arimoto, M. Sekimoto and S. Kawamura, Task-space iterative learning for redundant robotic systems: Existence of a task-space control and convergence of learning, *SICE J. Control Meas. Syst. Integr.* **1**(4) (2008) 312–319.
12. P. Bhounsule, K. Yamane and A. Bapat, A task-level iterative learning control algorithm for accurate tracking in manipulators with modeling errors and stringent joint position limits, in *Proc. ASME Dynamics Systems and Controls Conf.* (Minneapolis, USA, 2016).
13. P. Bhounsule and K. Yamane, Iterative learning control for accurate task-space tracking with humanoid robots, in *Proc. Int. Conf. Humanoid Robots* (Seoul, South Korea, 2015).
14. NaturalPoint, OptiTrack ARENA: Body motion capture software for 3d character animation and more (2013), http://www.naturalpoint.com/optitrack/products/arena/.

15. B. Mooring, The effect of joint axis misalignment on robot positioning accuracy, in *Proc. ASME Int. Computers in Engineering Conf.* (ASME, 1983), pp. 151–156.
16. B. Mooring and G. Tang, An improved method for identifying the kinematic parameters in a six axis robot, in *Proc. ASME Int. Conf. Computers in Engineering* (ASME, 1984).
17. K. Gupta, Kinematic analysis of manipulators using the zero reference position description, *Int. J. Robot. Res.* **5**(2) (1986) 5–13.
18. S. L. Delp, F. C. Anderson, A. S. Arnold, P. Loan, A. Habib, C. T. John, E. Guendelman and D. G. Thelen, Opensim: Open-source software to create and analyze dynamic simulations of movement, *IEEE Trans. Biomed. Engrg.* **54**(11) (2007) 1940–1950.
19. B. W. Mooring, Z. S. Roth and M. R. Driels, *Fundamentals of Manipulator Calibration* (Wiley-Interscience, 1991).
20. J. Denavit, A kinematic notation for lower-pair mechanisms based on matrices, *Trans. ASME, J. Appl. Mech.* **22** (1955) 215–221.
21. A. Broun, C. Beck, T. Pipe, M. Mirmehdi and C. Melhuish, Building a kinematic model of a robots arm with a depth camera, in *Advances in Autonomous Robotics* (Springer, 2012), pp. 105–116.
22. M. G. Hollars, D. E. Rosenthal and M. A. Sherman, *SD/FAST Users Manual* (Symbolic Dynamics Inc., 1991).
23. P. E. Gill, W. Murray and M. A. Saunders, SNOPT: An SQP algorithm for large-scale constrained optimization, *SIAM J. Optim.* **12**(4) (2002) 979–1006.
24. R. P. Judd and A. B. Knasinski, A technique to calibrate industrial robots with experimental verification, *IEEE Trans. Robot. Autom.* **6**(1) (1990) 20–30.
25. J.-M. Renders, E. Rossignol, M. Becquet and R. Hanus, Kinematic calibration and geometrical parameter identification for robots, *IEEE Trans. Robot. Autom.* **7**(6) (1991) 721–732.
26. H. A. Park, M. A. Ali and C. G. Lee, Closed-form inverse kinematic position solution for humanoid robots, *Int. J. Hum. Robot.* **9**(3) (2012) 1250022.
27. L. C. Wang and C.-C. Chen, A combined optimization method for solving the inverse kinematics problems of mechanical manipulators, *IEEE Trans. Robot. Autom.* **7**(4) (1991) 489–499.
28. C. K. Liu, A. Hertzmann and Z. Popović, Learning physics-based motion style with nonlinear inverse optimization, *ACM Trans. Graph.* **24**(3) (2005) 1071–1081.
29. H. Elci, R. W. Longman, M. Q. Phan, J.-N. Juang and R. Ugoletti, Simple learning control made practical by zero-phase filtering: Applications to robotics, *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.* **49**(6) (2002) 753–767.
30. C. Atkeson and J. McIntyre, Robot trajectory learning through practice, in *Proc. IEEE Int. Conf. Robotics and Automation*, Vol. 3 (IEEE, 1986), pp. 1737–1742.
31. M. L. Visinsky, J. R. Cavallaro and I. D. Walker, Robotic fault detection and fault tolerance: A survey, *Reliab. Engrg. Syst. Safety* **46**(2) (1994) 139–158.

Pranav A. Bhounsule is an Assistant Professor in the Department of Mechanical Engineering at the University of Texas at San Antonio, USA. He received his B.E. degree from Goa University, India, in 2004, the M.Tech. degree from Indian Institute of Technology Madras, India, in 2006, and the Ph.D. degree from Cornell University, USA, in 2012. From 2012–2014, he was a Post-Doctoral Research Fellow at Disney Research, Pittsburgh, USA. He received the award for best paper in Biologically Inspired Robotics at the *Climbing and Walking Robots Conference*, 2012. He also received a US National Science Foundation Research Initiation Grant in 2016. His research interests include model-based and learning-based control methods for legged and humanoid robots.

Katsu Yamane is a Senior Research Scientist at Disney Research, Pittsburgh, USA, and an Adjunct Associate Professor at the Robotics Institute, Carnegie Mellon University, USA. He received his B.Sc., M.Sc., and Ph.D. degrees in Mechanical Engineering in 1997, 1999, and 2002, respectively, from the University of Tokyo, Japan. Prior to joining Disney in 2008, he was an Associate Professor at the University of Tokyo and a Post-Doctoral Fellow at Carnegie Mellon University. Dr. Yamane is the recipient of King-Sun Fu Best Transactions Paper Award and Early Academic Career Award from IEEE Robotics and Automation Society and the Young Scientist Award from the Ministry of Education, Japan. His research interests include humanoid robot control and motion synthesis, physical human–robot interaction, character animation, and human motion simulation.