Iterative Learning Control for Accurate Task-Space Tracking with Humanoid Robots

Pranav A. Bhounsule[1] and Katsu Yamane[2]

Abstract— **Precise task-space tracking with manipulator-type systems requires accurate kinematics models. In contrast to traditional manipulators, 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 solver 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 that learns to track the figure eight in 18 trials and to serve a drink without spilling in 9 trials.**

I. Introduction

Many applications of manipulator-type systems involve precise task-space control. For example, industrial manipulators that do welding or personal humanoid robots that folds clothes. Industrial manipulators rely on good CAD models and 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 light weight 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. Thus, small parameters errors in the CAD model can lead to big errors at the end-effectors.
4) When 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 can enable high fidelity position tracking. We use constrained optimization to solve the inverse kinematics using the imperfect kinematic model. The cost for the constrained optimization is the weighted squared sum of the deviation of the end-effector from the desired pose. The result of the constrained optimization is the joint motion as a function of time to produce the desired end-effector motion. However, when the motion is implemented on the robot, there is substantial tracking errors because of the imperfect kinematic model. So next, we use 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 to account for modeling errors. We try out two example motions on the humanoid: (i) drawing the figure eight, and (ii) serving a drink without spilling.

II. Background and Related Work

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 necessitates 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. These approaches directly build a inverse kinematics model experimental data [4], [5]. One of 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 a 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

[1] Dept. of Mechanical Engineering, University of Texas San Antonio, One UTSA Circle, San Antonio, TX 78249, USA. pranav.bhounsule@utsa.edu. [2] Disney Research, 4720 Forbes Avenue, Lower Level, Suite 110 Pittsburgh, PA 15213, USA. kyamane@disneyresearch.com

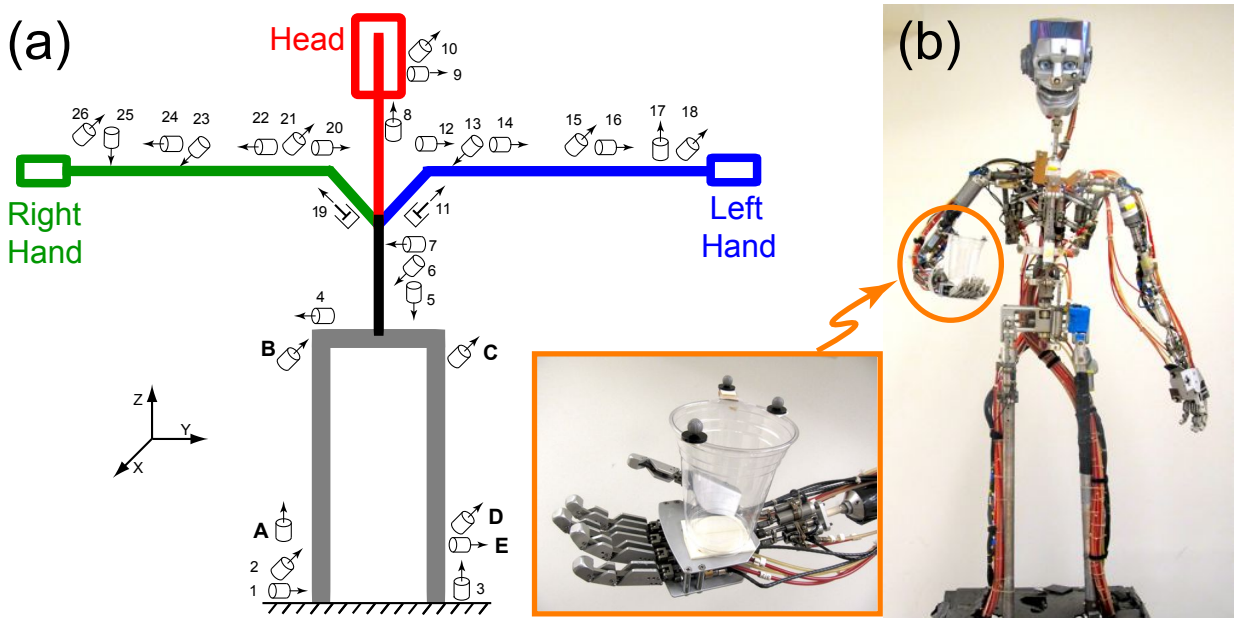

Fig. 1. (a) The kinematic model of the humanoid robot in the zero reference pose. In this pose all the joint angles zero. The robot has a total of 37 actuated joints. The figure shows only 26 actuated joints and which are numbered from 1 through 26. The actuated joints not shown here are the 10 finger joints and 1 eye-blink joint. The joints A through E are neither sensed nor actuated. In this paper, we only use the following 14 actuated joints for control; 1 through 7 in the body, and 19 through 26 in the right hand. (b) A photo of the humanoid robot that we used in this study. 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"). The above pose is our reference pose for the inverse kinematics (see Sec III-C). A plastic glass is glued to the right hand of the robot and is shown inset. We put three markers on the top of the glass which we track using Opti-track motion capture system.

with the above method is that the construction of the gating network becomes difficult in high dimensions.

Iterative Learning Control (ILC) can make up 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 [10]. 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. [11], [12] 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 non-linear map from task-space to joint space and show the efficacy of the algorithm experimentally on a humanoid. Specifically, we evaluate the non-linear map by using an inverse kinematics solver which find solutions within the joint limits. The main improvement over Arimoto's algorithm is that our algorithm is able to handle joint limits [9].

III. Methods

A. Robot hardware

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 differential transformer to sense 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 from differentiated and filtered position data from joint sensor and the force sensor in the valve. The gains on the lowest level controller are pre-set and cannot be changed. 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 position. We control the robot at the highest level in all experiments reported here, i.e. the input is a position command and the joint position is the measured variable, both of which occur at a bandwidth of 120 Hz.

B. Marker-based motion capture

We use an 8-camera motion capture system by OptiTrack [13]. The motion capture system outputs the position and the orientation of the end-effector by measuring the position of the three retro-reflective markers that we place on the end-effector.

We first use the motion capture system to develop a kinematic model of the robot. The robot joints are made to move in a random fashion and the joint position from the robot control system and the end-effector position and orientation from the motion capture system are simultaneously recorded. Then, a kinematic model is specified and parameters are fit using non-linear least squares. The model has limited accuracy due to unmodeled effects such as sensor noise

and link deflection due to load. We provide more details in Bhounsule and Yamane [14].

We also use the motion capture system to track the end-effector motion during the task-space experiments. We use the tracking errors in the position and the orientation in the iterative learning control algorithm which we describe in detail in Sec. III-D.

C. Inverse Kinematics

We need an inverse kinematics solver to map the desired end-effector motion to joint space for the low level position servo. The use of optimization based inverse kinematics solver provides a straight-forward and generalizable method of creating solutions for redundant robots, such as humanoid robots [15]. Specifically, by choosing a suitable cost function one can bias the solution to use certain joints over the other ones.

We use a constrained optimization software SNOPT [16] 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:

$$g(\boldsymbol{\theta}(t)) = \sum_{i=1}^{n_{\text{dof}}} w_i(\theta_i(t) - \theta_i^{\text{ref}}(t))^2, \tag{1}$$

$$\mathbf{h_1}(\boldsymbol{\theta}(t)) = \mathbf{X}_{\text{des}}(t) - \mathbf{f}(\boldsymbol{\theta}(t)) \leq |\epsilon|, \tag{2}$$

$$\mathbf{h_2}(\boldsymbol{\theta}(t)) = \boldsymbol{\theta}(t) - \boldsymbol{\theta}_{\text{min}} \geq 0, \tag{3}$$

$$\mathbf{h_3}(\boldsymbol{\theta}(t)) = \boldsymbol{\theta}(t) - \boldsymbol{\theta}_{\text{max}} \leq 0. \tag{4}$$

In the above scheme, we specify the desired motion of the end-effector, $\mathbf{X}_{\text{des}}(t)$ (e.g. trajectory of the pen to draw the figure eight). Further, we define n_{dof} to be the degrees of freedom used by the robot in the experiment. 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, $\theta_i^{\text{ref}}(t)$. We choose, $\theta_i^{\text{ref}}(t) = \theta_i^{\text{ref}}$ (constant), and is the joint angles corresponding to the robot pose shown in Fig. 1.
2) The joint weighting 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 (e.g., See Liu et al. [17])

D. Iterative Learning Control (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.

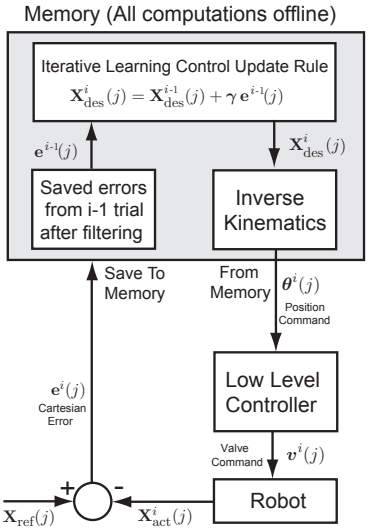

Fig. 2. Block diagram of our end-effector space iterative learning control algorithm. We filter the cartesian space errors $e^{i-1}(j)$ using a zero phase filter before applying the iterative learning update rule. The super-script i denotes the trial number and j denotes a time instance.

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 $\mathbf{X}_{\text{ref}}(j)$. Here we have concatenated the position and the orientation in the vector $\mathbf{X}$. The input to the inverse kinematics solver are the desired poses in end-effector space, which we denote by $\mathbf{X}^i_{\text{des}}(j)$. Let the measured position in task-space be defined by $\mathbf{X}^i_{\text{act}}(j)$. Let the error between actual end-effector position and reference position be denoted by $\mathbf{e}^i(j) = \mathbf{X}^i_{\text{ref}}(j) - \mathbf{X}_{\text{act}}(j)$. Our ILC algorithm is shown in Fig. 2 and described below:

1) Set the error $\mathbf{e^0}(j) = 0$ and initialize the desired position in the task-space $\mathbf{X}^0_{\text{des}}(j) = \mathbf{X}_{\text{ref}}(j)$.
2) For subsequent trials do:
   - Command modification in end-effector space: Update the feed-forward position command using the tracking error at trial i, $\mathbf{X}^i_{\text{des}}(j) = \mathbf{X}^{i-1}_{\text{des}}(j) + \boldsymbol{\Gamma e}^{i-1}(j)$. The manually tuned learning gain, Γ, is a 6x6 matrix of the form; $\Gamma = \text{diag}\{\gamma_1, \gamma_2, ...\gamma_6\}$. Further, for ILC to converge we need, $0 < \gamma_i \leq 1$.
   - Command initialization in joint-space: For the desired position in task-space $\mathbf{X}^i_{\text{des}}(j)$, use the inverse kinematics solver to find a desired joint command $\boldsymbol{\theta}^i(j)$.
   - Command execution on robot: Send the feed-forward commands $\boldsymbol{\theta}^i(j)$ $(j = 1, 2, ..., n_j)$ to the low level controller. Save the resulting tracking errors in the end-effector space, $\mathbf{e}^i(j)$ $(j = 1, 2, ..., n_j)$.
3) Stop when the error metric e^i_{norm} does not improve between trials. The learnt feed-forward command is then $\boldsymbol{\theta}^i(j)$ $(j = 1, 2, ..., n_j)$.

The error metric to check convergence is given as follows

$$\boldsymbol{e}^i_{\text{norm}} = \frac{1}{n_j}\sum_{j=1}^{n_j}\sum_{k=1}^{n_{\text{eff}}}(e^i_k(j))^2, \tag{5}$$

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 data points in the trial.

E. Performance of the algorithm

a) System equations: The ILC update equation is:

$$\mathbf{X}^i_{\text{des}}(j) = \mathbf{X}^{i-1}_{\text{des}}(j) + \boldsymbol{\Gamma}\boldsymbol{e}^{i-1}(j) \tag{6}$$

Next, we linearize the output equation which relates the actual joint position, $\mathbf{X}^i_{\text{act}}$ with the control equation $\mathbf{X}^i_{\text{des}}$.

$$\begin{aligned}\mathbf{X}^i_{\text{act}}(j) &= \mathbf{f}(\hat{\mathbf{f}}^{-1}(\mathbf{X}^i_{\text{des}}(j)))\\ &= \mathbf{F}(\mathbf{X}^i_{\text{des}}(j))\\ &\approx \mathbf{G}\mathbf{X}^i_{\text{des}}(j) \text{ where } \mathbf{G} = \frac{\partial \mathbf{F}}{\partial \mathbf{X}^i_{\text{des}}}\end{aligned} \tag{7}$$

where $\mathbf{f}$, $\hat{\mathbf{f}}$ are the true model and approximate model respectively, $\mathbf{F} = \mathbf{f}(\hat{\mathbf{f}}^{-1}(.))$. A first order approximation of $\mathbf{F}$ has to be linearly proportional to $\mathbf{X}^i_{\text{des}}$ for the equation to be dimensionally consistent.

b) Convergence analysis: To do convergence analysis, we need to express the error between successive trials. This is done as follows.

$$\begin{aligned}\boldsymbol{e}^i(j) &= \mathbf{X}_{\text{ref}}(j) - \mathbf{X}^i_{\text{act}}(j)\\ &= \mathbf{X}_{\text{ref}}(j) - \mathbf{G}\mathbf{X}^i_{\text{des}}(j)\\ &= \mathbf{X}_{\text{ref}}(j) - \mathbf{G}\mathbf{X}^{i-1}_{\text{des}}(j) - \mathbf{G}\boldsymbol{\Gamma}\boldsymbol{e}^{i-1}(j)\\ &= \mathbf{X}_{\text{ref}}(j) - \mathbf{X}^{i-1}_{\text{act}}(j) - \mathbf{G}\boldsymbol{\Gamma}\boldsymbol{e}^{i-1}(j)\\ &= \boldsymbol{e}^{i-1}(j) - \mathbf{G}\boldsymbol{\Gamma}\boldsymbol{e}^{i-1}(j)\\ &= (\mathbf{I} - \mathbf{G}\boldsymbol{\Gamma})\boldsymbol{e}^{i-1}(j)\end{aligned}$$

The condition for convergence is that the $|\boldsymbol{e}^{i-1}(j)| < |\boldsymbol{e}^i(j)|$. This condition is met when the eigenvalues of $(\mathbf{I} - \mathbf{G}\boldsymbol{\Gamma})$ are less than 1.

c) Stability analysis: We simplify the ILC update equation as follows:

$$\begin{aligned}\mathbf{X}^i_{\text{des}}(j) &= \mathbf{X}^{i-1}_{\text{des}}(j) + \boldsymbol{\Gamma}(\mathbf{X}_{\text{ref}}(j) - \mathbf{X}^{i-1}_{\text{act}}(j))\\ &= \mathbf{X}^{i-1}_{\text{des}}(j) + \boldsymbol{\Gamma}\mathbf{X}_{\text{ref}}(j) - \boldsymbol{\Gamma}\mathbf{G}\mathbf{X}^{i-1}_{\text{des}}(j)\\ &= (\mathbf{I} - \boldsymbol{\Gamma}\mathbf{G})\mathbf{X}^{i-1}_{\text{des}}(j) + \boldsymbol{\Gamma}\mathbf{X}_{\text{ref}}(j)\end{aligned}$$

The condition for stabile learning is that the control command, $\mathbf{X}^i_{\text{des}}(j)$ should be bounded. This happens when the when the eigenvalues of $(\mathbf{I} - \boldsymbol{\Gamma}\mathbf{G})$ are less than 1.

d) Tuning the learning gain: In the previous two sections, we saw that the learning gains, $\boldsymbol{\Gamma}$, affects the stability as well as convergence. These are the only parameter that needs to be manually tuned in the algorithm. We simplify this by choosing the same learning parameter for all six degrees of freedom. Thus $\boldsymbol{\Gamma} = \gamma I$, where I is 6x6 identity matrix and $0 < \gamma \leq 1$. The closer this value is to 1, the faster is the convergence. But the sensor noise limits the use of high gains. A straightforward way to enable high values for γ is to filter the noisy sensor data. This is discussed next.

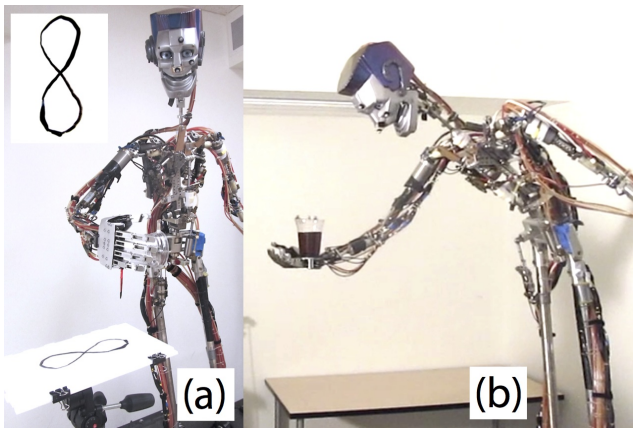

Fig. 3. (a) Robot after completing the writing task after learning. The figure "eight" drawn by the robot is shown inset, and (b) Robot doing the glass task after learning.

F. Zero phase filtering

We use a zero phase filter that removes sensor noise and provides sensor data with zero phase lag. 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 data points 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.

IV. Experimental Results

We show implementation of our algorithm on the humanoid robot shown in Fig. 1. In order to generate the results, we used the following 14 actuated joints shown in Fig. 1 (a); 1 through 7 in the body, and 19 through 26 in the right hand. The joints A through E are neither sensed or actuated but need to move in order to allow the loop joint in the lower body to move.

A. Task 1: Writing Task

The writing task consists 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 See III-C) to compute the joint position command $\boldsymbol{\theta}(t)$). 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 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 robots draws in the 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 Eqn. 5) 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. 4 (a).

Figure 5(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. Figure 6 (a) and (b) shows a plot of the drawing task in the x-y (horizontal plane) and the x-z plane (fore-aft plane). The solid black line is the reference. The blue dash-dotted line is the robot motion during trial 1. The red dashed lined shows the converged motion at trial 18.

After the robot has learnt the motion, we made the robot 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. 3 (a) and the actual drawing is shown inset.

We have also provided a video showing the learning trials and robot doing the drawing task. One thing to note in the video is that the robot uses some of the mid- and lower-body joints, in addition to the right hand joints to draw. This is because the plane of the drawing board is not completely reachable using only the hand degrees of freedom.

B. 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. 3 (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, $\gamma = 0.3$, 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.

e) Learning from identified model: We initialize the learning from the kinematics model we identified. Trial 1 produced an error of 2×10^{-2} (see Eqn. 5). 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 8 trials. This is about a two orders of magnitude improvement. A plot of the convergence rate is shown as red dots in Fig. 4 (b).

We also attach a video of the experiment. Note that the robot has to use joints on the mid-body and lower-body in addition to the joints on the hands to reach out to serve the drink.

f) Learning incorrect potentiometer calibration: We want to see if our learning algorithm can learn from incorrect potentiometer calibration. To simulate 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, but which 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 9 trials. This is almost two orders of magnitude improvement. A plot of the convergence rate is shown with unfilled green diamonds in Fig. 4(b). Also, Fig. 5 (b) shows the error in tracking for Trial 1 and for Trial 9.

V. Discussion

A. Better models give better convergence

Though ILC scheme is designed to correct for modeling errors, better models give better convergence [18], [19]. This is clear when we compare the two test cases in the glass task; (1) learnt from an identified model vs. (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 slightly 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 error in the converged trial 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 error in the z position between the writing task shown in Fig. 5 (a) (iii) and the glass task shown in Fig. 5 (b) (iii). It is seen that the error in the z-direction keep increasing as the robot stretches its hand to move the glass to serve the drink.

B. 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 re-calibrate the model.

C. 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. 2). 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 methods that work at the velocity level and use the Jacobian to map from task-space to joint space.

D. 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.

VI. Conclusions and future work

In this paper, we present an Iterative Learning Control algorithm for high-fidelity tracking of end-effector motions for high degree of freedom manipulator system in the presence of modeling errors. We create the given end-effector motion using an inverse kinematics solver based on constrained optimization. The cost function in the optimization provides flexibility to select a motion style by appropriate choice of reference pose and joint weights. To enable high-fidelity tracking in the presence of modeling errors, we iteratively modifying the end-effector reference motions using the tracking errors. These are then mapped back into joint space using the inverse kinematics solver. We demonstrate the efficacy of our algorithm on a high degree of freedom humanoid robot that iteratively learns writing the figure eight in 18 trials and learns to move a glass in a perfectly level manner in 9 trials.

Future work will be directed towards incorporating velocity limits (note that in the current work we only implement position limits), which will help to extend this work to fast motions.

References

[1] K.J. Astrom, and T. Hagglund, PID Controller: Theory, Design, and Tuning, *International Society for measurement and Control Seattle, WA*, 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. *Proceedings of the International Congress of Mechanical Engineering*, Gramado, Brazil. 2009.

[3] D. Sun, and J.K. Mills, High-accuracy trajectory tracking of industrial robot manipulator using adaptive-learning schemes, *American Control Conference*, 1999,

[4] A. Guez and Z. Ahmad. Solution to the inverse kinematics problem in robotics by neural networks. In *Neural Networks, IEEE International Conference on*, pages 617–624. 1988.

[5] R. Köker, C. Öz, T. Çakar and H. Ekiz. A study of neural network based inverse kinematics solution for a three-joint robot. *Robotics and Autonomous Systems*, 49(3):227–234, 2004.

[6] M. I. Jordan and D. E. Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive science*, 16(3):307–354, 1992.

[7] E. Oyama, N. Y. Chong, A. Agah, and T. Maeda. Inverse kinematics learning by modular architecture neural networks with performance prediction networks. In *Robotics and Automation, IEEE International Conference on*, volume 1, pages 1006–1012. IEEE, 2001.

[8] S.Arimoto, S.Kawamura, and F.Miyazaki. Bettering operation of robots by learning. *Journal of Robotic* 1(2):123-140, 1984.

[9] P. A. Bhounsule, K. Yamane and A. A. Bapat A task-level iterative learning control algorithm for redundant manipulators with joint position limits *IEEE–International Conference on Robotics and Automation (submitted)* , 2016

[10] D. Bristow, M. Tharayil, and A. G. Alleyne, A survey of iterative learning control *Control Systems, IEEE*, 26(3):96–114, 2006.

[11] S. Arimoto, M. Sekimoto and S. Kawamura. Iterative learning of specified motions in task-space for redundant multi-joint hand-arm robots. In *Robotics and Automation, IEEE International Conference on*, pages 2867–2873. IEEE, 2007.

[12] 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 Journal of Control, Measurement, and System Integration*, 1:312–319, 2008.

[13] OptiTrack - ARENA - Body motion capture software for 3D character animation and more. `http://www.naturalpoint.com/optitrack/products/arena/`, [Online; accessed 26 November 2013].

[14] P. A Bhounsule and K. Yamane. Full-Pose Calibration of a High Degree of Freedom Humanoid Robot using Marker-Based Motion Capture System. *Industrial Robot: An International Journal* 2015 (submitted).

[15] L. T. Wang and C. C. Chen, A combined optimization method for solving the inverse kinematics problems of mechanical manipulators *IEEE Transaction on Robotics*, vol=7, no=4, pp. 489–499, 1991.

[16] P. E. Gill, W. Murray and M. A. Saunders. Snopt: An SQP algorithm for large-scale constrained optimization. *SIAM journal on optimization*, 12(4):979–1006, 2002.

[17] K. C. Liu, and A. Hertzmann, and Z. Popović Learning physics-based motion style with nonlinear inverse optimization *ACM Transactions on Graphics (TOG)*, 24(3):1071–1081, 2005.

[18] P. A. Bhounsule, and K. Yamane. Iterative Learning Control for High-Fidelity Tracking of Fast Motions on Entertainment Humanoid Robots. *IEEE-RAS International Conference on Humanoid Robots*, Atlanta, GA, USA, 2013.

[19] C. G. Atkeson and J. McIntyre. Robot trajectory learning through practice. In *IEEE International Conference on Robotics and Automation*, volume 3, pp. 1737–1742, 1986.

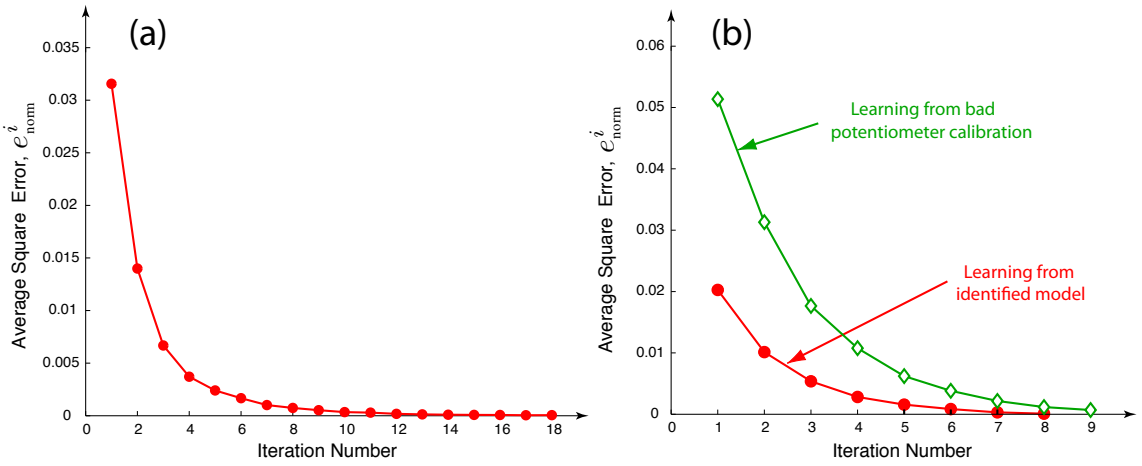

Fig. 4. Convergence of errors as a function of iteration number for (a) the writing task and (b) the glass task

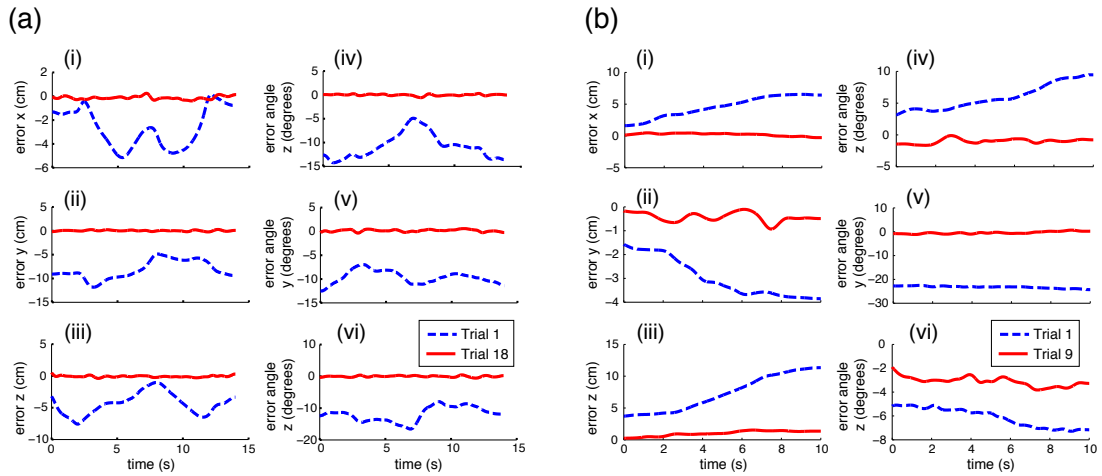

Fig. 5. Position and Orientation Errors for (a) the writing task and (b) the glass task (learning incorrect potentiometer calibration). (i,ii,iii) Errors in the position in the x, y, z directions in cm. and (iv,v,vi) Errors in the angle in the x, y, z direction in degrees. x is in the fore-aft direction, y is in the robot's left-right direction and z is in the up-down direction.

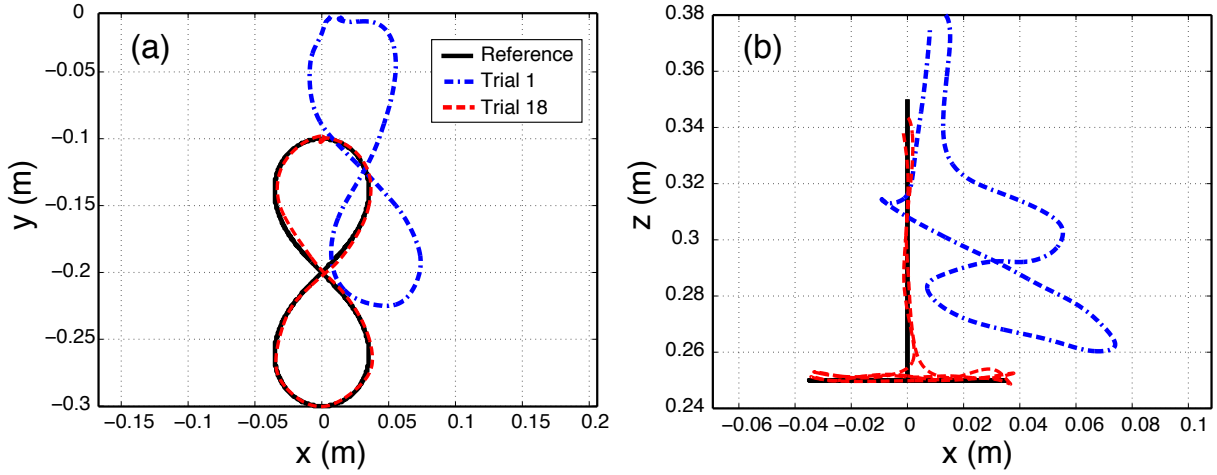

Fig. 6. (a) Plot in the x-y plane (z axis lines up with gravity). (b) Plot in the x-z plane (fore-aft plane of robot). The blue dash-dot lines and red dashed lines are the marker position measured by the motion capture system.