Combined Online and Offline Inverse Dynamics Learning for a Robot Manipulator

[1]Amrut Sekhar Panda, [2]Ravi Prakash, [2]Laxmidhar Behera, [1]Ashish Dutta
[1]Department of Mechanical Engineering, *Indian Institute of Technology Kanpur*, India
[2]Department of Electrical Engineering, *Indian Institute of Technology Kanpur*, India

***Abstract*—Due to the approximation errors in dynamic model, changing payloads and dynamic disturbances acting on the system, the model based tracking is not satisfactory. Hence the application of real-time machine learning techniques in inverse dynamics learning has gained prominence for collaborative human robot interaction. In this paper, we propose a novel combined online and offline Neural Network based learning technique in conjunction with an acceleration tracker for inverse dynamics learning of a robot manipulator. This eliminates the need for explicit reliance on the approximate analytical robot model while controlling the robotic systems. The proposed approach can even capture the system dynamics accurately at higher acceleration where non-linear forces such as non-linear friction and damping play a prominent role. The performance of the proposed inverse dynamic model has been verified using extensive simulations on control of a 6 DOF UR5 robot manipulator in an accurate physics based Pybullet Simulator. The efficacy of the proposed method has been validated by comparing the control performance using model based backstepping controller. The results show that the inverse dynamics learning based controller outperforms significantly its counterpart in real world scenarios where uncertainties are ubiquitous.**

I. Introduction

Robot manipulators are playing an important role in various industrial applications. Presently researchers are trying have manipulators and human work in one common environment. The manipulators has to be pretty safe for this purpose. To achieve that the dynamic model of the robot must be well known. The dynamic model of a manipulator can be modeled using Newton-Euler recursive algorithm or from Lagrangian method.But manipulators in the real world, experience various kinds of dynamic forces which are difficult to model. This renders the systems infeasible for compliant control. Learning control is one of the approach to address this issue.

Learning control means applying algorithms that can achieve accurate motion generation in an autonomous and safe way. The objective is to design algorithms that can facilitate the controllers to provide accurate tracking in multiple scenarios without requirement of excessive tuning.

Inverse dynamics learning [1] [2] [3] based control has become an active area of research towards designing of compliant motion. If we can have a model which can capture the system dynamics reasonably well, then the torque generation will only depend on the feedforward prediction of the model. As the model is never accurate, it brings in noises or perturbations. Generally these errors are rejected by employing Proportional Derivative Integral (PID) controller.

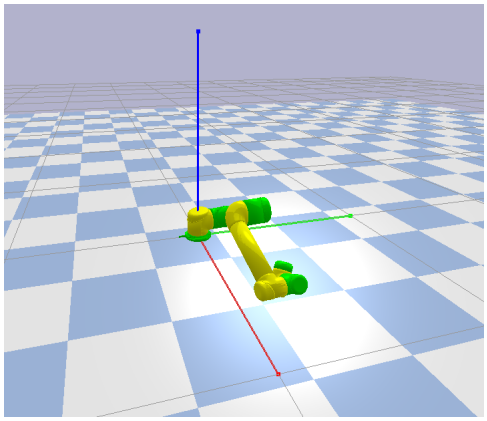

Fig. 1: UR5 Manipulator in Pybullet Simulator

The feedback gain for both proportional and derivative terms depend primarily on the accuracy of the model. So better model results in less feedback gains which in turn results in more compliant and reactive controller. But the problem lies in finding good estimates of the inverse dynamics model because of the reason stated above. As the degrees of freedom of the system increases the system becomes proportionately complex to model using traditional techniques.

The challenge in the field of learning control is to estimate a good inverse dynamics model. From the machine learning perspective the challenges are many. The inverse dynamics data distribution is non-stationary during execution. In order to collect data the workspace of the manipulator has to be well explored. This may not be as such as problem for system with lower degrees of freedom, but it can be infeasible for higher dimensional systems. So for higher dimensional system it is very difficult to obtain a global inverse dynamics model by just exploring the workspace of the system and learning it. So online learning becomes an attractive option. For a learning algorithm to be suitable for online scenario it has to both computationally efficient and robust. There has been recent work on creating efficient approximations of Gaussian Process Regression (GPR) [4] [5] [6]. Gaussian process Regression provide better estimates by associating uncertainty estimates with its predictions. Meier et al [7]have investigated drifting gaussian process regression for learning inverse dynamics where data is correlated and arriving during trajectory tracking.

Because of such difficulties from machine learning perspec-

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

tive, recent works have focused on designing algorithms which could be used in control loops. But having a algorithm that can make feedforward predictions fast enough does not solve the problem but creates new challenges. The learned model determines what next data point it gets to see. As our model is not perfect, the actual observed states will be different than the desired state. So the model will have to predict the next state from the current state which actually not a point in the trajectory. In order to compensate such errors PID controller is employed. The PID controller brings the system back to the trajectory and with time the algorithm learns from the observed states to generalize to the desired states. But this raises the question as how to design a PID controller. All the recent works address how to make online estimation of inverse dynamics modeling errors. This work focuses not only eliminating explicit requirement of PID controller to achieve tracking but also incorporating some knowledge of the system before deploying for feedforward prediction in online setting. The first one can be done by modeling the inverse dynamics error as a constant offset, which will be constantly adapted via online gradient descent to minimize the tracking error at the acceleration level. In other words it compensates the torque that creates difference in the value of actual and desired acceleration. The learning rate affects the magnitude of update at each step.This parameter is equivalent to the feedback gains of the PID control term.The offset term can be made variable, but assuming it a constant simplifies the problem. The second is done by incorporation of trained model which has been trained with some data of the system during its operation. So the trained model has captured some behaviour of the system [8] [9]. Deploying such models online will make the model to converge faster than the deploying a untrained model which in turn not only will reduce the gains of the constant offset term but also will require less data points to capture the inverse dynamics. The results obtained by adopting this approach was such that model was even able to reduce tracking error for wide variations in end-effector mass, joint frictions and damping.

II. Background

Accurate inverse dynamics model is the primary requirement for tracking desired accelerations. The dynamics of the a manipulator according to rigid body dynamics can be expressed as follows:

$$\tau = M(q)\ddot{q} + h(q, \dot{q}) \tag{1}$$

where $q, \dot{q}, \ddot{q}$ represent joint angles, joint velocities and joint accelerations respectively. M stands for the inertia matrix, h represent the torque produced due to modeled forces. The modeled forces are coriolis forces, centrifugal forces, gravitational forces, friction forces etc.Given accurate information of various rigid body dynamics (RBD) parameters the mass matrix M and h can be estimated.The estimated RBD model can be expressed as follows:

$$\hat{\tau}_{rbd} = \hat{M}(q^t)\ddot{q}_d^t + \hat{h}(q^t, \dot{q^t}) \tag{2}$$

where the $\hat{M}$ and $\hat{h}$ are the approximates for M and h. But the RBD model is incapable of encompassing all the non-linearities that arise in actual systems. So the above estimates are only rough approximations. While tracking the desired accelerations $\ddot{q}_d$,generally τ_{rbd} is applied on the actual system whose true dynamics is unknown, we obtain the actual acceleration $\ddot{q}_a$ which differs from $\ddot{q}_d$.We can express the actual acceleration as:

$${q_a}^t = M(q^t)^{-1}[(\hat{M}(q^t)\ddot{q}_d^t + \hat{h}(q^t, \dot{q^t})) - h(q, \dot{q})] \tag{3}$$

Now if our estimation of $\hat{M}$ and $\hat{h}$ are accurate then the actual acceleration of the system and the desire acceleration will be same. But this never occurs in actual systems. As a result typically a feedback term τ_{fb} is required. The feedback adds a compensating value from the measured error and ensures tracking. This task is generally achieved using PID controllers. The feedback gain for both proportional and derivative terms depend primarily on accuracy of the estimated model. So better model results in less feedback gains which in turn results in more compliant and reactive controller

A. Learning Inverse Dynamics Models

Many approaches to learn either full inverse dynamics or an error model has been proposed. While learning such models the total torque input applied on the system is combination of torque predicted by the existing approximate model ($\hat{\tau}_{rbd}$), torque predicted by an error model (f_{iderr}) and a feedback term (τ_{fb}).

$$\tau_{total} = \hat{\tau}_{rbd} + f_{iderr} + \tau_{fb} \tag{4}$$

Prediction of torque command using the learned models must be feasible in presence of real time system constraints. One of key challenges for a inverse dynamics model is computational efficiency.The model should be able to predict the torque command in least amount time, so that it can be applicable for real time applications.Recent advances in technology have rendered the availability of computational resources cheaper. But the inverse dynamics data distribution is non-stationary in the workspace of the system.So learning a global inverse dynamics model is very difficult. Hence, approaches that can learn the inverse dynamics incrementally and simultaneously computationally efficient have become a major direction of research involving learning inverse dynamics models. However learning inverse dynamics models that are globally valid in highly correlated data streams remains a challenge.

Some robustness can be achieved by using an analytical rigid body dynamics model and employing a learned error model on top of that.When the algorithm detects that prediction of the error model is poor, it can revert to the prior knowledge of the system.

Due to challenges inherent to learning globally valid inverse dynamics models, some attention has been directed towards learning task or context specific inverse dynamics learning models. In such cases, the learning process can be simplified by collecting task specific data for offline learning. But

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

challenge lies in deciding the correct context during run time such that the correct task can be chosen.

Finally the motivation for learning inverse dynamics models is to achieve the ability to employ low feedback gains. However, traditional inverse learning models require high learning rate in the beginning of training to ensure good tracking, such that relevant data is generated. Little amount of work has been done to automatically lower the feedback gain once a good model is learned.

All these work involve learning inverse dynamics from an approximate model which is typically derived using rigid body dynamics. But in this work we have estimated both the approximated model and error model together by learned model. The data can be collected by moving the manipulator in its workspace for certain number of times and recording state and torque data streaming from joints. An approximate model can be prepared by learning this data. Subsequently it can be deployed for further learning along with the feedback term while following the trajectory. This work shows that such models can be computationally efficient for feedforward prediction of torque command in real time.

B. Direct and Indirect Inverse Dynamics Learning Models

[9] [8]Recent attempts in the field of learning inverse dynamics can be classified in the category of indirect learning. The purpose is to optimise the following objective function:

$$L_{indirect}(w) = \sum_{(x_a,\tau_{total})\in D} \parallel \tau_{total} - f(x_a; w) \parallel \quad (5)$$

where the input data x_a = $(q, \dot{q}, \ddot{q}_a)$ consists of q and $\dot{q}$ which denotes joint angles and joint velocities respectively and $\ddot{q}_a$ which is the actual acceleration. τ_{total} is the applied torque on the system which produced acceleration $\ddot{q}_a$.The function $f(x; w)$ maps x to torque space where w stands for the parameter of the model.

In online setting,the robot tries to track $\ddot{q}_d$ from its states (q,$\dot{q}$).So it applies τ_{total} on its joint and observes the actual acceleration $\ddot{q}_a$. In the subsequent step x_a=$(q, \dot{q}, \ddot{q}_a)$ serve as the input instead of desired state x_d=$(q, \dot{q}, \ddot{q}_d)$. So it will lead to slow convergence since the data used is out of the trajectory data distribution which we want to optimise. This is especially difficult when $\ddot{q}_a$ = 0 because of static friction since many torque values can map to this value.

On the otherhand it has been shown that it is possible to measure the gradient of acceleration error of a system which can be classified as direct online learning algorithms [10]. These approaches tries to directly minimize the error between desired and actual accelerations by optimising the following loss function:

$$L_{direct}(w) = \|\ddot{q}_d - \ddot{q}_a\|^2_M \quad (6)$$

where the actual acceleration is a function of the error model. Traditional direct adaptive control methods have quite similar approaches- they use Lyapunov techniques to derive controllers that adjust the dynamics offset model with respect to some reference signal. However the feedback term cannot map the complete structure of the error model and requires relatively high learning rate for any change in payload. This can be taken care of by introducing an error model which has previously learned the model structure to some extent.

Our work is based on the above mentioned techniques. First a learning model can be trained with torque and state space data of the system. This learned model along with the acceleration tracker can both predict the torque command and learn from the new data obtained after application of torque on the system. Also it has been shown that such an approach can replace the necessity of analytical rigid body dynamics model based inertia, gravity and centrifugal matrices to determine part of the total torque command during runtime, thus simplifying the computation. It can even track different trajectories.Another advantage is, this approach is task independent and theoretically applicable anywhere throughout the state space. On the flip side, it does not retain the history of previously learned error models,thus cannot improve over time.

III. Combined Offline and Online Learning

To comply with the state-space dependence of the inverse dynamics modeling error, we assume that the error model is valid in the space of $x = (q, \dot{q}, \ddot{q})$. The model can be trained with data collected by moving the manipulator in its workspace. Furthermore,during task execution, we will treat the current state information as a new training dataset which can further improve and update our model. Thus our error models are indexed by k,which means kth iteration.For k=0 the error model that exist is the previously learned model at hand and we assume it to be $f^0_{iderr}(x; w^0) = 0$. Thus the total torque applied to the system at any given instant is combined output of both offline learned model and the feedback term τ_{fb}.

$$\tau_{total} = f^k_{iderr} + \tau_{fb} \quad (7)$$

Now we discuss the formulation of loss functions for both offline and online learning.

A. Learning Offline model

We discuss the details of learning the error model in offline setting. For any starting position and target position of the joint angles, we plan a motion trajectory. Then we compute the torque command based on the current state $(q, \dot{q})$ and desired accelerations $\ddot{q}_d$, apply this torque, and then measure actual accelerations $\ddot{q}_a$. Now we know what torque command achieves these measured accelerations and can use this data point to learn an inverse dynamics model. We collect all such data points over the course of one task execution, such that we have T data points to learn. Similarly we can execute many such task and collect data for the training.

In such indirect formulation our aim is to optimize the parameters w such that the difference between the applied torque τ_{total} and the inverse dynamics model f_{id} at x_a is minimized.

$$L_{indirect} = \sum_{(q,\dot{q},\ddot{q}_a,\tau_{total})\in D} \left\|\tau_{total} - f(q, \dot{q}, \ddot{q}_a; w)\right\|^2 \quad (8)$$

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

where $f(q, \dot{q}, \ddot{q}_a; w)$ is the learned error model.Note,optimising this loss function can learn unknown inverse dynamics model at the actual state and actual accelerations. Thus when online learning is in progress,feedback term is indispensable for good tracking of the desired acceleration -at least until the model has learned.

B. Direct Optimisation of the Inverse Dynamics Modeling Errors

In contrast to the above mentioned methods the above equation proposes to augment Equation (3) with a constant offset term w such that the actual acceleration can now be written as:

$$q_a^t = M(q^t)^{-1}[(\hat{M}(q^t)\ddot{q}_d^t + \hat{h}(q^t, \dot{q}^t) + w) - h(q, \dot{q})] \quad (9)$$

The offset term w has been designed to mend any any modeling error and perturbation which may result in inaccurate tracking. [11]So we also propose to perform gradient descent on inertia weighted acceleration error of the form

$$\begin{aligned} L_{Direct(w)} &= \frac{1}{2}\left\|\ddot{q}_d - \ddot{q}_a(w^t)\right\|_{M_t}^2 \\ &= \frac{1}{2}(\ddot{q}_d - \ddot{q}_a(w^t))^T M_t(\ddot{q}_d - \ddot{q}_a(w^t)) \end{aligned} \quad (10)$$

to minimize the error between desired and actual accelerations.Note that M_t is the $true$ inertia matrix.The gradient of the constant offset model can be expressed in the form of acceleration error.

$$\begin{aligned} \nabla_w L_{direct}(w) &= -\frac{\partial \ddot{q}_a(w^t)^T}{\partial w} M_t(\ddot{q}_d - \ddot{q}_a(w^t)) \\ &= -M_t^{-1} M_t(\ddot{q}_d - \ddot{q}_a(w^t)) \\ &= -(\ddot{q}_d - \ddot{q}_a(w^t)) \end{aligned} \quad (11)$$

Analytically the above objective function and its gradient cannot be determined because it requires prior knowledge of the unknown dynamics function within $q_a(w^t)$.But the actual accelerations can be evaluated by finite differencing of the state data directly obtained from the system.Given this,the feedback torque for the next step can be computed as

$$w^{t+1} I\& = w^t + \eta(\ddot{q}_d - \ddot{q}_a(w)) \quad (12)$$

which in turn will generate the corrected torque command

$$\tau_{t+1} = f(q_{t+1}.\dot{q}_{t+1}, \ddot{q}_d^{t+1}) + w^{t+1} \quad (13)$$

It can observed that there lies a connection between online gradient descent on the acceleration error and traditional PID term, which allows to analyse the influence of learning from feedback term perspective.For instance,the learning rate η has influence over the effect of current acceleration on the corrected torque command in the next time step. The higher learning rate will result in larger torque adjustment in similar way the higher gains of a PID controller. The consequence is poor compliancy. In this current form the offset cannot be used to make feedforward prediction about the inverse dynamics model error.

C. Online Learning Of Inverse Dynamics Error

Our task is to predict the required torque command from a locally valid inverse dynamics model as learning a globally valid model is difficult especially if the robot has more degrees of freedom. So we will utilise the data streaming from the robot during its operation for augmenting the earlier pre-trained offline model.

More elaborately,at time step t we want to use the current model $f(x_a, w)$ to predict the modeling error at current actual state and the desired accelerations. The current state data along with the desired acceleration are supplied to the model in order to predict the torque command. The applied torque command will result in the actual acceleration $\ddot{q}_a^t$,which is estimated from the measured position data using finite difference. It is to be noted that,the current observation $(x_a^t, \ddot{q}_a^t), \tau^t$ gives us the information about the inverse dynamics at the current actual state and actual information rather than current actual state and desired acceleration.Unless we are observing the desired accelerations well,we will loose useful information. Due to this reason,we always have the online learning on our inverse dynamics model running along with the feedback term on accelerations to make sure we are tracking them.

Our data points to be included in our data window is $[x^t = (x_a^t, \ddot{q}_a^t), y^t = \tau_{applied}]$.Once we get this data we can use this to update the parameters of our inverse dynamics model.In the subsequent step we will drop the old observation by including the new one which we get by repeating the procedure stated above.

IV. EXPERIMENT

We have evaluated the above approach in simulated experiment. We have taken a simulated universal robot's UR5 six degrees of freedom cobot for experimentation. We have compared the our proposed learned model to track trajectory with a backstepping controller.In this experiment we have tracked the actual trajectory of the end-effector versus the desired trajectory for any arbitrary random valid initial and target point for both controller under various operating conditions. The simulated experiment consists no.of steps such as building the simulated model,data generation,training the inverse dynamics model and deploying the model for control. Each step has been discussed below.

Number of Networks	5
Number of layers of each Network	4
Activation Function	tanh
Optimizer	Adams
Input Dimension	17
Output Dimension	1
Number of Hidden layer Neurons	200
Learning Rate	0.001
Number of Epochs	200
Number of Training samples	100,000

TABLE I: Neural network Architecture and Hyperparameter.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

A. Simulated Robot Model

Simulations can be performed using many state of the art simulators, e.g. Pybullet, MuJuCo [12], Robotics Toolbox [13], Gazebo,V-rep [14] or RoboAnalyzer [15]. We have used pybullet simulator [16] which is based on the Bullet physics engine, for simulation.The simulation window with the manipulator is shown in Fig. 1. The pybullet simulator has a very to use python API's for application of various controller. It is fast and does not require any additional dependencies. As our inverse dynamics model provides feedforward prediction of applied torque, so we need to have torque controllers at each joint.The simulator provides functionalities to imitate friction using velocity controllers at each joint.

B. Data Generation

We obtained the data required for training our inverse dynamics model by recording the joint states at all instant of its motion. The plan is the move the robot from one random point in the workspace to another random point following a certain trajectory. It is to be noted that,the workspace that we take should be away from singularities,so our workspace boundary must stay at some threshold distance from the boundaries of the actual workspace. We adopt polynomial functions to generate our trajectory. We take the polynomial of 5th degree for trajectory generation. From the boundary conditions we compute all the co-efficient of the polynomial. Then we use them to create trajectory between any two random points in the workspace.We do not take the orientation of the end-effector into account, although there is no such constraints that the algorithm demands. Then,at each step the values of joint angles,joint velocities and desired acceleration supplied to the analytical inverse dynamics model to calculate the torque command. When we apply the torque command the system will have attained some state. Because of the simulated environment the actual acceleration will be quite same as the desired acceleration. Subsequently these procedures are repeated at each time step until the end-effector reaches the desired position in the workspace. In the meantime joint state data streaming from robot along with the applied torque data at each time step are recorded. Similarly many such trajectories are executed and data collected. Repeating the above procedure we collected nearly 100,000 training samples.

C. Training the Offline Inverse dynamics model

We have employed a multi-layered neural network to learn our model. We have treated individual joint as a separate learning problem by providing inputs of all joints . This reduces the prediction of all joints as an extension of single joint prediction . In our case we need to train five models. The neural network consists of four layers with 200 neurons in each hidden layer. The activation function is taken to be $tanh$ activation.The input vector is 17 dimensional which consists of joint angles,joint velocities and joint accelerations of all joint angles together. The acceleration values of sixth joints in the training samples were mostly zero. So We have excluded the acceleration value as input. The output vector is one dimensional which consists of torque value of one out of five joint angles as the variation in value of sixth joint joint hardly matters for tracking cartesian positions.The learning rate is taken to be 0.001 and Adams optimiser for weight update. The network was trained upto 200 epochs.The input data was processed and normalized before feeding to the neural network.So five learning models are trained for five joints.

D. Deploying the model and Online learning

The next step is to deploy the model in control loop. As discussed in the previous section, the model along with the acceleration tracker has to be employed for simultaneous prediction and online training. During the motion of the manipulator state data was stored for evaluating the performance of the model.

We have compared the current scheme with the a backstepping controller for multiple test cases. The control law for backstepping controller is given below.

$$\tau = M(\ddot{\theta} + K_p\dot{e} + K_d(\dot{\theta} + K_p e - \theta_u) + e) + C \qquad (14)$$

The backstepping control requires inertia, gravity and centrifugal plus coriolis matrices for computing torque values, which in turn needs approximate analytical formulation of the system dynamics. We move the manipulator along a trajectory employing backstepping controller and our learned inverse dynamics model separately. In order to learn disturbances the offline model must be trained with as many data as possible for error minimisation.So it is essential to provide more steps in trajectory. For experimentation we take **1540** time steps with completion time of 5 seconds. Each network takes on average 0.001 seconds for both prediction and training. As all the five networks are independent of each other, parallel processing is obvious option for reduction in overall computational time.

Case	Payload mass	Payload status	Friction	Damping
1	0.2 Kg	Known	No	No
2	2 Kg	Unknown	No	No
3	0.4 Kg	Unknown	Yes	No
4	2 Kg	Unknown	Yes	Yes
5	0.2 Kg and 2 Kg	Unknown	Yes	Yes

TABLE II: Test Cases

Five separate operating scenarios have been considered for comparison.We have evaluated tracking performance for the following different scenarios (a)The end-effector payload same as the mass during the training and the payload mass is known a priori. (b) The end-effector payload different from the mass during the training (c) The end-effector mass is unknown and the joints have friction force enabled (d) The end-effector mass is unknown and the all the joints have both damping and friction enabled. For each case we have planned a polynomial trajectory with random initial and final points in the workspace.The payload mass in each case is given in the table above. Here the RMS error represents the rms error between the final target point and end-effector position at any

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

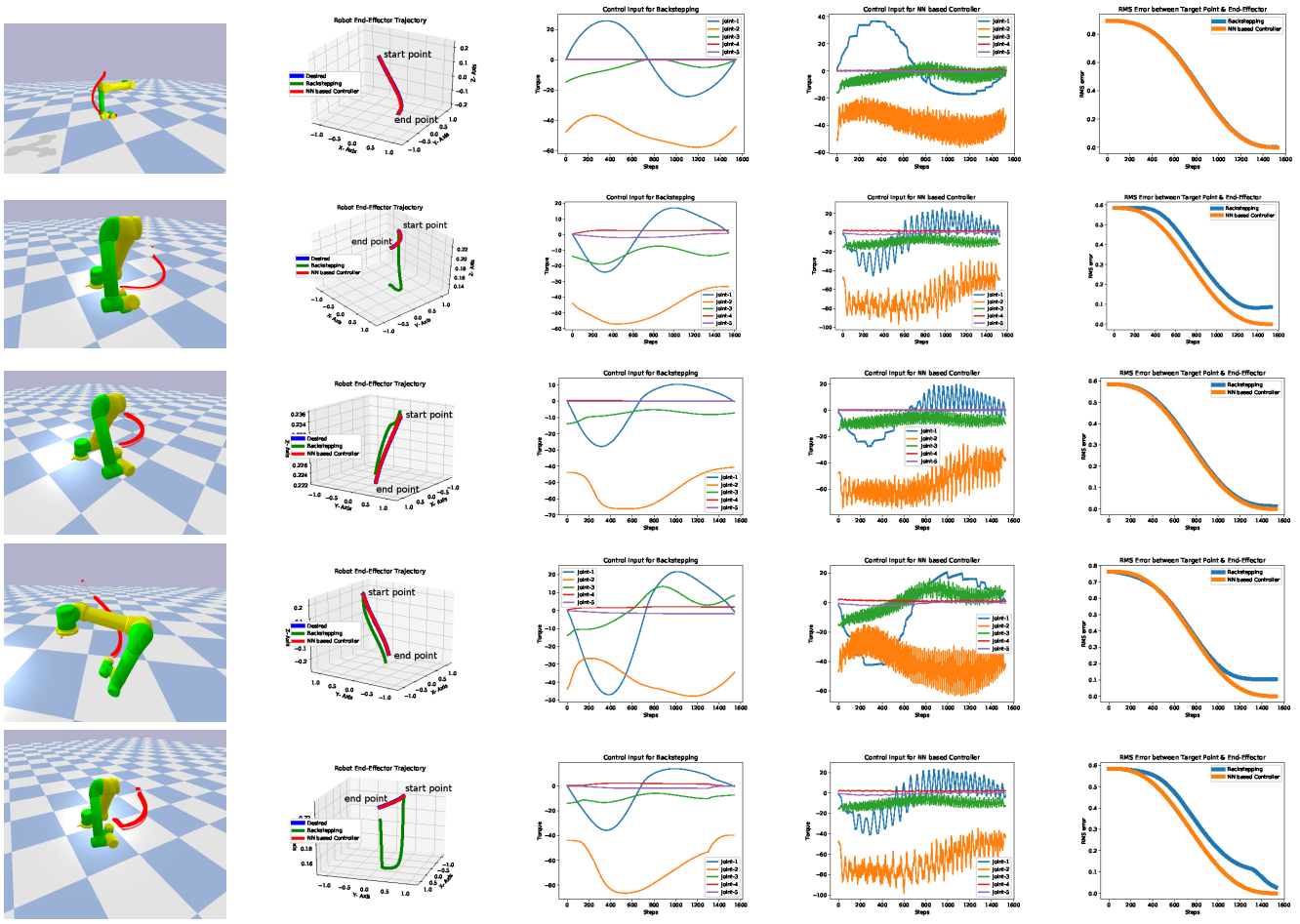

Fig. 2: The above figures show the trajectory and control inputs of backstepping and learned model for random initial and final point in the workspace :(i) First column of the figures contain the end-effector trajectories in Pybullet. (ii) Second column of the figures contain comparison of trajectories followed by both controllers. (iii) Third column of the figures contain control input of computed by backstepping controller.(iv) Fourth column of the figures contain the computed control employing learned inverse dynamics model controller.(v) The Fifth column of the figures contain the RMS error between end-effector current state and trajectory end point.

time step. In the first case we keep the payload mass 0.2 Kg. We generate the trajectory and employ both controllers separately to track the trajectory. As the mass is known, the backstepping controller perfectly tracks the desired trajectory with RMS error converging to zero. Because of online training the learned model outputs fluctuate but the mean follows the backstepping output. In the second case, the mass of payload is unknown. The backstepping controller by default takes the mass to be 0.2 Kg and generates control input. But the payload mass is actually 2 Kg. As a result backstepping controller miserably fails as the RMS error converges to a finite value other than zero. The learned inverse dynamics model closely follows the trajectory because of online learning and RMS error converges to zero. In the third case friction simulation is enabled in each joint. The friction can be modeled as the maximum value between a threshold value and $\frac{1}{3}$ of the applied torque [17]. As the payload mass is close to the backstepping default mass value, the controller is able to track the trajectory successfully and the RMS error value converges to zero. Both joint friction and damping are enabled in fourth case. As the payload mass along with friction and damping forces amplify disturbance in the system, the backstepping controller fails to converge to zero RMS error. In the last case we track a trajectory where the manipulator picks a unknown mass, carries it forward and releases it somewhere while following the trajectory. The manipulator picks the 2Kg mass at 200th time step and releases at 1300th time step. The backstepping controller completely fails to track the trajectory.But because of online learning the learned inverse dynamics model could track the trajectory reasonably well. The backstepping controller relies on inertia matrix and others for control input which in turn depends on mass value directly. For small change in mass value the controller does not require tuning of gains for successful trajectory tracking. But large change in values will require further tuning. The inverse dynamics learned model controller quickly learns the system dynamics and disturbance because of both offline and online training. The main advantages is the model can even

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

predict the torque command for high speed applications by accurately mapping all non-linear torques arising during the motion without any analytical inverse dynamics model.

E. Quantitative comparison

For each of the cases listed in the above table we have tested 50 trajectories for each case and computed the mean squared error between desired trajectory and actual trajectory produced by both backstepping controller and learned inverse dynamics model. The result have been shown in the table below. The entries of both second and third column in the table represent the mean and variance of RMS error.

Case No.	RMS Error Backstepping	RMS error Learned model
1	0.002 $\pm$0.001	0.0004 $\pm$0.0003
2	0.03 $\pm$0.035	0.0005 $\pm$0.0005
3	0.006 $\pm$0.002	0.0005 $\pm$0.0004
4	0.08 $\pm$0.04	0.0007 $\pm$0.0004
5	0.12 $\pm$0.03	0.0007 $\pm$0.0002

TABLE III: RMS Error Comparison

Note the percentage improvement in the RMSE of the proposed inverse dynamic control technique: For case 1: 60%, For case 2: 98%, For case 3: 91.6%, For case 4: 99.12%, For case 5: 99.41%. The results imply that our learned inverse dynamics model performs better than backstepping controller in presence of high system uncertainty.This is most suitable for real world robotic applications where system experiences many disturbances such as change in end-effector payload, mounting camera and other other equipments which are hard to model.Our proposed learned inverse kinematics model suits well in such scenarios.

V. Discussion And Future Work

We have presented the online learning control approach that combines both offline and online learning without any requirement of analytical inverse dynamics model. It is to be noted that in individual experiments, error models trained on both offline and online data have better performance for certain range of velocity and acceleration. Exploitation of structure of data sources for determining reliable sources can be an interesting direction for exploration. We have shown that the pre-neural networks can learn the larger inverse dynamics error model. Our learned model provides better prediction of model uncertainties compared to standard backstepping controller . For this model we do investigate how the system behaves when it undergoes strong perturbation.This could result in undesirable prediction because the input data excludes that input space.

Due to hard real time constraints we cannot increase the model size further. Adoption of better regression model can map inverse dynamics which remains an area of further investigation. However in our future work, we plan to learn the error model with better accuracy with less computational cost. Furthermore we would like compare our model with Gaussian process Regression models in evaluate the performance in presence of model uncertainties.

References

[1] Duy Nguyen-Tuong and Jan Peters. Model learning for robot control: a survey. *Cognitive processing*, 12(4):319–340, 2011.

[2] B. Bcsi, L. Csat, and J. Peters. Alignment-based transfer learning for robot models. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, Aug 2013.

[3] Niko Sünderhauf, Oliver Brock, Walter J. Scheirer, Raia Hadsell, Dieter Fox, Jürgen Leitner, Ben Upcroft, Pieter Abbeel, Wolfram Burgard, Michael Milford, and Peter Corke. The limits and potentials of deep learning for robotics. *CoRR*, abs/1804.06557, 2018.

[4] Arjan Gijsberts and Giorgio Metta. Real-time model learning using incremental sparse spectrum gaussian process regression. *Neural Networks*, 41:59 – 69, 2013. Special Issue on Autonomous Learning.

[5] Olivier Sigaud, Camille Salan, and Vincent Padois. On-line regression algorithms for learning mechanical models of robots: A survey. *Robotics and Autonomous Systems*, 59(12):1115 – 1129, 2011.

[6] Duy Nguyen-tuong, Jan R. Peters, and Matthias Seeger. Local gaussian process regression for real time online model learning. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1193–1200. Curran Associates, Inc., 2009.

[7] Franziska Meier and Stefan Schaal. Drifting gaussian processes with varying neighborhood sizes for online model learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 264–269. IEEE, 2016.

[8] F. Meier, D. Kappler, N. Ratliff, and S. Schaal. Towards robust online inverse dynamics learning. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4034–4039, Oct 2016.

[9] D. Kappler, F. Meier, N. Ratliff, and S. Schaal. A new data source for inverse dynamics learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4723–4730, Sep. 2017.

[10] Franziska Meier and Stefan Schaal. Drifting gaussian processes with varying neighborhood sizes for online model learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 2016*. IEEE, May 2016.

[11] Nathan Ratliff, Franziska Meier, Daniel Kappler, and Stefan Schaal. Doomed: Direct online optimization of modeling errors in dynamics. *arXiv preprint arXiv:1608.00309*, August 2016.

[12] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[13] Peter I. Corke. *Robotics, Vision & Control: Fundamental Algorithms in MATLAB*. Springer, second edition, 2017. ISBN 978-3-319-54413-7.

[14] M. Freese E. Rohmer, S. P. N. Singh. Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013. www.coppeliarobotics.com.

[15] Vaibhav Gupta, Rajeevlochana G. Chittawadigi, and Subir Kumar Saha. Roboanalyzer: Robot visualization software for robot technicians. In *Proceedings of the Advances in Robotics*, AIR 17, New York, NY, USA, 2017. Association for Computing Machinery.

[16] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019.

[17] 1955 Craig. *Introduction to robotics : mechanics & control / John J. Craig.* Addison-Wesley Pub. Co.,, Reading, Mass., 1986. Includes bibliographies and index.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE