Towards Robust Online Inverse Dynamics Learning

Franziska Meier[1,2], Daniel Kappler[2,3], Nathan Ratliff[3] and Stefan Schaal[1,2]

Abstract— **Learning of inverse dynamics modeling errors is key for compliant or force control when analytical models are only rough approximations. Thus, designing real time capable function approximation algorithms has been a necessary focus towards the goal of online model learning. However, because these approaches learn a mapping from actual state and acceleration to torque, good tracking is required to observe data points on the desired path. Recently it has been shown how online gradient descent on a simple modeling error offset term to minimize tracking at acceleration level can address this issue. However, to adapt to larger errors a high learning rate of the online learner is required, resulting in reduced compliancy. Thus, here we propose to combine both approaches: The online adapted offset term ensures good tracking such that a nonlinear function approximator is able to learn an error model on the desired trajectory. This, in turn, reduces the load on the adaptive feedback, enabling it to use a lower learning rate. Combined this creates a controller with variable feedback and low gains, and a feedforward model that can account for larger modeling errors. We demonstrate the effectiveness of this framework, in simulation and on a real system.**

I. INTRODUCTION

Learning control brings the promise of algorithms that achieve accurate motion generation in an autonomous and safe manner. The goal is to design learning algorithms that can augment controllers such that they provide accurate tracking in diverse task settings without the need for excessive tuning. At the same time, applications of robotic systems increasingly involve human interaction, creating the need for robust and compliant control algorithms.

Inverse dynamics control has created a promising avenue towards the design of compliant movement generation. With a good model of a systems dynamics, torque generation can mainly rely on the feedforward prediction of the dynamics model. Modeling errors or perturbations are typically rejected by a Proportional Derivative Integral (PID) controller [1]. The required feedback gains depend on the accuracy of the inverse dynamics model. The better the model, the smaller the feedback gains can be, which in turn results in a more compliant and reactive controller. Unfortunately, estimating good inverse dynamics models is hard.

Thus, developing machine learning algorithms to estimate inverse dynamics models has become an active research area within the field of learning control. Just from a pure machine learning perspective, challenges in this setting are plentiful. The inverse dynamics data distribution is non-stationary during execution and thus offline learning of this model would require data of a well explored state space. For high-dimensional systems this is infeasible. Thus, online estimation of the inverse dynamics is required. For a learning algorithm to be successful in the online learning scenario, it needs to be computationally efficient and a robust parameter estimation framework is required, such that the models prediction can be trusted. To this end, there has been recent progress on creating efficient approximations of Gaussian process regression (GPR)[2], [3], [4], [5]. Gaussian process regression provides a robust learning framework and allows to associate uncertainty estimates with its predictions, both of which we believe to be important components of a controller with learning in the loop.

Because of the difficulties from a machine learning perspective alone, research and progress has been focused on creating algorithms that could eventually be used within a control loop. However, assuming we have a fast enough learning algorithm, actually employing it comes with its own challenges. For instance, the model, when used to create feedforward predictions, influences its next observation. This creates a learning feedback loop - the learned model essentially determines what next data point it gets to see. Unless we start out with a perfect model, the actual observed state will be different than the desired state. While we can still incorporate that observed data point, it provides an observation of the inverse dynamics in the current actual state, instead of the desired. As a result, the model has just incorporated a data point that is not necessarily on the trajectory we are trying to track. Of course, we can employ a PID controller to generate feedback torques that brings us back to our desired trajectory, and with time the learning algorithm should observe enough data points to allow it to generalize to the desired states. However, this begs the question of how to design the PID controller. The goal of inverse dynamics learning is to be able to reduce feedback gains, yet we need them – at least when starting learning – to make sure we track and generate useful data.

It is this problem that recent work [6] on online estimation of inverse dynamics modeling errors addresses. In contrast to the above mentioned methods, this work does not explicitly require a PID controller to achieve tracking. The key to this approach is the modeling of the inverse dynamics error as a constant offset, that is continuously adapted via online gradient descent to minimize the tracking error at acceleration level. This can be understood as computing a variable feedback term, generating the missing torques to achieve desired accelerations. The learning rate of the

[1] CLMC Lab, University of Southern California, Los Angeles, USA [2] Autonomous Motion Department, MPI for Intelligent Systems, Tübingen, Germany [3] Lula Robotics Inc. fmeier@usc.edu

This research was supported in part by National Science Foundation grants IIS-1205249, IIS-1017134, EECS-0926052, the Office of Naval Research, the Okawa Foundation, and the Max-Planck-Society.

gradient descent directly affects how much we update the offset to minimize the acceleration error at each time step. Thus this parameter corresponds to the feedback gains of a more traditional PID control term. The restrictive assumption of a constant model error makes online learning relatively simple, however it also means no structure of the modeling error is captured, and thus generalization is not possible.

In this work, we show how to extend this new framework to use a drifting Gaussian process (GP) [7] to learn a local model of the inverse dynamics error online. We use the constant offsets at each time step as data points for our GP, such that it can learn to predict the modeling errors in a feedforward fashion. As the GPs predictions improve, the constant offsets work is reduced to reject noise and perturbations, which opens up the possibility of reducing the learning rate – the feedback gain. Furthermore, we discuss how to robustify the online hyper parameter learning and predictions of the constantly changing function approximator.

In the following, we first review the inverse dynamics learning problem and the framework of [6] in Section II. Then, in Section III, we show how to introduce a drifting Gaussian process model within this framework and how to make robust feedforward predictions. Finally, we evaluate our approach in Section IV and conclude with a discussion of the proposed approach in Section V.

II. Background

When employing inverse dynamics control the torque τ to control the system is computed as $\tau = \tau_{\text{ff}} + \tau_{\text{fb}}$, where the feedforward command τ_{ff} is computed from the assumed inverse dynamics model and the feedback term τ_{fb} is traditionally a PID control term. Current inverse dynamics approaches leverage the general Lagrangian mechanical form of the equations of motion:

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

where $q, \dot{q}, \ddot{q}$ denote the joint position, velocities and accelerations, M is the inertia matrix and h collects all the modeled forces such as gravitional, Coriolis, centrifugal forces, and friction terms. Given sufficiently rich data and Rigid Body Dynamics (RBD) assumptions, best fit RBD parameters found using linear regression techniques [8], resulting in approximate $\hat{M}$ and $\hat{h}$. Unfortunately, the RBD model typically is not flexible enough to capture all non-linearities of the actual systems dynamics. As a result, the estimated RBD model is generally only a rough approximation. Thus, when attempting to track desired accelerations $\ddot{q}_d$ the torque estimated through the approximate inverse dynamics model

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

results in actual accelerations $\ddot{q}_a$

$$\begin{aligned} \ddot{q}^t_a &= M(q^t)^{-1}[\hat{\tau}_{\text{rbd}} - h(q^t, \dot{q}^t)] && (3)\\ &= M(q^t)^{-1}[\big(\hat{M}(q^t)\ddot{q}^t_d + \hat{h}(q^t, \dot{q}^t)\big) - h(q^t, \dot{q}^t)] && (4)\end{aligned}$$

which differ from the desired accelerations $\ddot{q}^t_d$, resulting in inferior tracking.

To address this, various approaches to learning inverse dynamics (error) models have been proposed, such that the total feedforward torque command is a combination of any existing approximate RBD model and a learned error model

$$\tau_{\text{ff}}(q, \dot{q}, \ddot{q}; \boldsymbol{w}^t) = \hat{\tau}_{\text{rbd}}(q, \dot{q}, \ddot{q}) + \tau_{\text{rbdErr}}(q, \dot{q}, \ddot{q}; \boldsymbol{w}). \tag{5}$$

Most recent work on online inverse dynamics learning is concerned with creating computationally efficient methods to learn a globally valid model of the inverse dynamics. To this end, computationally efficient learning algorithms have been developed [9], [10], [2], [3], [4], [5] These methods optimize the following loss

$$\mathcal{L} = \sum_{(q, \dot{q}, \ddot{q}_a, \tau_{\text{applied}}) \in \mathcal{D}} \|\tau_{\text{applied}} - f(q, \dot{q}, \ddot{q}_a; \boldsymbol{w})\|^2 \tag{6}$$

where $f(q, \dot{q}, \ddot{q}; \boldsymbol{w})$ is the function learned. Note, with this loss function, these methods can only consider information about the unknown inverse dynamics model at the actual state and actual accelerations. Thus, when learning is performed online, the feedback term is paramount to achieve good tracking of the desired accelerations – at least until the model has reached a certain level of accuracy.

A. Direct Optimization of Inv. Dynamics Modeling Errors

In contrast to the above mentioned methods [6] proposes to augment Equation (3) with a constant offset term $\boldsymbol{w}$ such that the actual acceleration can now be written as:

$$\ddot{q}^t_a = M(q^t)^{-1}[\big(\hat{M}(q^t)\ddot{q}^t_d + \hat{h}(q^t, \dot{q}^t) + \boldsymbol{w}\big) - h(q^t, \dot{q}^t)].$$

This offset $\boldsymbol{w}$ is designed to correct any modeling error or perturbation that would lead to inaccurate tracking. Thus, this approach proposes to perform gradient descent on the inertia weighted acceleration error of the form

$$\begin{aligned} \mathcal{L}_{\text{direct}}(\boldsymbol{w}) &= \frac{1}{2}||\ddot{q}^t_d - \ddot{q}^t_a(\boldsymbol{w}^t)||^2_{M_t} && (7)\\ &= \frac{1}{2}\left(\ddot{q}^t_d - \ddot{q}^t_a(\boldsymbol{w}^t)\right)^T M_t \left(\ddot{q}^t_d - \ddot{q}^t_a(\boldsymbol{w}^t)\right) \end{aligned}$$

to minimize the error between desired and actual accelerations. Note that $M_t = M(q^t)$ is the *true* unknown inertia matrix. When using a constant offset model, its gradient is just given through the raw acceleration error:

$$\begin{aligned} \nabla_{\boldsymbol{w}} \mathcal{L}_{\text{direct}}(\boldsymbol{w}) &= -\frac{\partial \ddot{q}^t_a(\boldsymbol{w}^t)}{\partial \boldsymbol{w}^t}^T \left(M_t(\ddot{q}^t_d - \ddot{q}^t_a(\boldsymbol{w}^t))\right) \\ &= -M_t^{-1} M_t \big(\ddot{q}^t_d - \ddot{q}^t_a(\boldsymbol{w}^t)\big) \\ &= -\big(\ddot{q}^t_d - \ddot{q}^t_a(\boldsymbol{w}^t)\big). \end{aligned}$$

We cannot analytically evaluate either the objective or its derivative because it involves the true (unknown) dynamics function within $\ddot{q}^t_a(\boldsymbol{w}^t)$. But, in practice, this final expressions can be estimated (using finite differencing for the accelerations) from the measured state information streaming from the robot. Given this, the torque offset for the next time step is computed as

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t + \eta(\ddot{q}^t_d - \ddot{q}^t_a(\boldsymbol{w})) \tag{8}$$

which results in the adjusted torque command

$$\tau^{t+1} = \hat{\tau}_{\text{rbd}}(q^{t+1}, \dot{q}^{t+1}, \ddot{q}_d^{t+1}) + \boldsymbol{w}^{t+1} \tag{9}$$

To reduce the effects of noise, exponential smoothing is performed on the $\boldsymbol{w}$, for more details on achieving a robust online gradient descent algorithm see [6].

Note, that [6] also draws a connection between online gradient descent [11] on the acceleration error and the more traditional PID term, which allows us to analyze the effect of the learning terms from a feedback term perspective. For instance, the learning rate η influences the effect of the current acceleration error on the torque adjustment of the next time step. Thus, a higher learning rate will create a larger adjustment, just as higher gains of a PID controller will. Because of this, increasing the learning rate also results in reduced compliancy. Furthermore, in its current form the offset is simply a constant offset and as such cannot be used to make feedforward predictions about the inverse dynamics model error.

While [6] proposes the more general setting of using a nonlinear model f_{on} instead of a constant model, we opt to learn an independent function approximator in parallel to the adaptive feedback for two reasons: First, while the derivation in [6] is valid for general function approximators, it is not clear yet how to generalize it to models such as Gaussian processes. For Gaussian processes the gradient update would involve optimizing the log likelihood function, a combination of the loss function and prior over functions. This update would involve both the true targets and the true inertia matrix, quantities that are unknown. Second, we believe that it is advantageous to use a combination of the constant error offset and a nonlinear function approximator that can feedforward predict larger modeling errors. This enables the use of the constant offset $\boldsymbol{w}$ as a variable feedback term with low gains to reject disurbances while the feedforward term is augmented with a nonlinear error model τ_{rbdErr}. Thus, in this work, we take this framework, and show how to utilize the online optimized offsets $\boldsymbol{w}$ to learn a locally valid inverse dynamics error model τ_{rbdErr}. With the feedforward predictions of τ_{rbdErr} the errors that $\boldsymbol{w}$ has to account for should become smaller, such that we can run the online adaptation of $\boldsymbol{w}$ at a lower learning rate while retaining tracking accuracy.

III. Online Learning of Inverse Dynamics Modeling Error

The recent advances in online learning adaptive control can be viewed as online learning of feedback terms, which are optimized to counteract the modeling errors of the approximate inverse dynamics model and any other perturbations. In this light, we can write the total torque at time step t, as

$$\tau^t = \tau_{\text{ff}}^t + \tau_{\text{fb}}^t = \hat{\tau}_{\text{rbd}}(x_{\text{a}}^t, \ddot{q}_d^t) + f_{\text{on}}(\ddot{q}_d^{t-1}, \ddot{q}_a^{t-1}, \boldsymbol{w}^t)$$

where $x_{\text{a}}^t = (q_{\text{a}}, \dot{q}_{\text{a}})$ is the current actual state and where the feedback term has been replaced by the online estimated offset $f_{\text{on}}(\ddot{q}_d^{t-1}, \ddot{q}_a^{t-1}, \boldsymbol{w}^t) = \boldsymbol{w}^t$. In this work, we aim at reducing the work that these online torque offsets $\boldsymbol{w}^t$ have to do. We achieve this by incorporating a nonlinear function approximator that estimates a model of the inverse dynamics error, and can thus feedforward predict the torque offsets needed to achieve desired accelerations $\ddot{q}_d^t$.

With this additional feedforward term the total amount of torque at time step t is then computed as

$$\tau^t = \hat{\tau}_{\text{rbd}}(x_{\text{a}}^t, \ddot{q}_d^t) + f_{\text{rbdErr}}(x_{\text{a}}^t, \ddot{q}_d^t) + f_{\text{on}}^t(\ddot{q}_d^{t-1}, \ddot{q}_a^{t-1}, \boldsymbol{w}^t) \tag{10}$$

where f_{rbdErr} denotes the function approximating the error made by the rigid body dynamics model.

A. Online Learning of Inverse Dynamics Error

As discussed earlier, online learning of a global inverse dynamics model or error is a difficult problem. Because our goal is to simply feedforward predict the current error offset, we only need a locally valid model. Thus, we use a fixed size window of recent observations, to approximate the landscape of the current modeling error. In this work, we use Gaussian process regression with a localizing kernel to perform this task, an approach that we have recently shown to work well for online inverse dynamics learning [7]. The key difference to our previous work is that here we explicitely use the online learned model in the control loop, such that the GP predictions influences the torque commands.

More concretely, at time step t, we want to use the current model f_{rbdErr}^t to predict the modeling error at the current actual state and the desired acceleration, $\tau_{\text{rbdErr}}^t = f_{\text{rbdErr}}^t(x_{\text{a}}^t, \ddot{q}_d^t)$. This torque estimate is combined with the torque $\hat{\tau}_{\text{rbd}}$ estimated from the approximate RBD model and the online learned feedback term f_{on}^t to compute the total torque command τ^t (Equation 10) that is send to the robot. Applying this torque command will result in actual accelerations $\ddot{q}_a^t$, which are estimated from measured position data via finite differencing. Note, that this observation $\{(x_{\text{a}}^t, \ddot{q}_a^t), \tau^t\}$ gives us information about the inverse dynamics at the current actual state with actual measured accelerations, instead of the current actual state and the desired accelerations. Thus, if we are not tracking the desired accelerations well, we do not necessarily receive useful information. Because of this, we always have the online learning of f_{on} running – the feedback term on accelerations – to make sure we are tracking them.

The estimate of the torque error τ_{rbdErr}^t that was made by the approximate inverse dynamics model is given as

$$\tau_{\text{rbdErr}}^t = \tau^t - \hat{\tau}_{\text{rbd}}(x_{\text{a}}^t, \ddot{q}_a^t) \tag{11}$$

As a result, the data point that needs to be included in our data window is then given as $\{\boldsymbol{x}^t = (x_{\text{a}}^t, \ddot{q}_a^t), y^t = \tau_{\text{rbdErr}}^t\}$.

Once we obtained this new data point, it is time to update the parameters of the Gaussian process model. Specifically, we update the data window by dropping the oldest observation and including the new one, to obtain $\mathcal{D}^t = \{\boldsymbol{x}^n, y^n\}_{n=t-N}^t$, where N denotes the window size. Then, we update the GPR model on $\mathcal{D}^t$ to obtain updated hyper parameters $\theta^t = \{\sigma_y^t, \sigma_f^t, \lambda^t\}$, where σ_y^2 is the noise variance and σ_f^2 and λ denote the signal variance and length scale of the squared exponential kernel. Furthermore, instead of optimizing the

Algorithm 1 Control Loop with Online Learning

```
Require: $\epsilon$, $\kappa$, $\theta^t_{\text{smoothed}}$, $\Psi^t_{\text{smoothed}}$ , $\mathcal{D}^t$, $\boldsymbol{w}^t$, $\pi(q^t, \dot{q}^t)$,
    system($\tau^t$), $q^t$,$\dot{q}^t$
1: $\ddot{q}^t_d = \pi(q^t, \dot{q}^t)$
2: $\Psi^{t+1}_{\text{smoothed}} = (1-\kappa)\Psi^t_{\text{smoothed}} + \kappa f^t_{\text{rbdErr}}(q^t, \dot{q}^t, \ddot{q}^t_d, \theta^t_{\text{smoothed}})$
3: $\tau^t = \hat{\tau}_{\text{rbd}}(x^t) + \boldsymbol{w}^t + \Psi^{t+1}$
4: $q^{t+1}$ = system($\tau^t$)
5: $\ddot{q}^{t+1}, \ddot{q}^t_a$ = finiteDiff($q^{t+1}$,$q^t$)
6: $\mathcal{D}^{t+1} = \{q^n, \dot{q}^n, \ddot{q}^n_a, \tau^n - \hat{\tau}_{\text{rbd}}(q^n, \dot{q}^n, \ddot{q}^t_a)\}^t_{n=t-N}$
7: $\theta^{t+1}_{\text{smoothed}} = (1-\epsilon)\theta^t_{\text{smoothed}} + \epsilon$ optimize($\theta^t_{\text{smoothed}}$,$\mathcal{D}^{t+1}$)
8: $\boldsymbol{w}^{t+1}$ =optimize($\ddot{q}^t_a, \ddot{q}^t_d, \boldsymbol{w}^t$)
9: return $\theta^{t+1}_{\text{smoothed}}$, $\Psi^{t+1}_{\text{smoothed}}$ , $\mathcal{D}^{t+1}$, $\boldsymbol{w}^{t+1}$
```

hyper parameters at each time step from scratch we can initialize θ^t with the parameters from the previous time step θ^{t-1}. An overview of this process is given in Algorithm 1, which demonstrates the computations of one control loop instance.

B. Robust Learning and Prediction

Robustly optimizing and predicting feedforward terms is a key concern when moving towards applying online learning on a real system. Here we take a couple of measures to robustify the hyper parameter optimization and predictions of the Gaussian process.

While the Gaussian process framework comes with a robust learning framework, we can receive quite noisy acceleration measurements, which influence both the inputs and targets of the GPR process. Further, we have to assume that, depending on the state space we are moving in, the squared exponential kernel may be too smooth and thus not always well suited to fit the inverse dynamics. Both of these issues may lead to extreme hyper parameter updates that could lead to instabilities. This is a research topic outside of the scope of this work, however, we can try to protect against this unwanted behavior of the parameters by exponentially smoothing them over time. Thus, when performing the gradient descent on θ, resulting in new parameters θ^t, we compute smoothed parameters as

$$\theta^t_{\text{smoothed}} = (1-\epsilon)\theta^{t-1}_{\text{smoothed}} + \epsilon\theta^t$$

where ϵ determines the degree of smoothing. The smoothed parameters become the new hyper parameter set if the new log likelihood value of the GPR model increased with respect to the previous parameter set. This exponential smoothing, prevents large jumps from time step to time step. This effectively means that we are slowing down the reactiveness of the GP, and so there is an inherent trade-off between the reactiveness and robustness of the hyper parameter optimization of the GP.

Note that we are using a localizing kernel, so we are protected against extrapolating into parts of the state space that we have no information about (assuming the length scale is in a reasonable range). However, to protect against erratic predictions—that may be caused for the same reasons as stated above—we also perform exponential smoothing of the predictions over time.

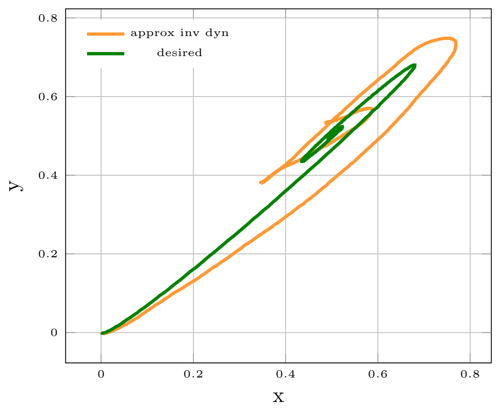


Fig. 1. Simulated system: In green the desired trajectory that is supposed to be tracked. In orange the trajectory resulting from applying inverse dynamics control with the wrong inverse dynamics model and no feedback terms.

$$\Psi^t_{\text{smoothed}} = (1-\kappa)\Psi^{t-1}_{\text{smoothed}} + \kappa f^t_{\text{rbdErr}}(q^t, \dot{q}^t, \ddot{q}^t_d, \theta^t_{\text{smoothed}})$$

Smoothing at the prediction level further changes the timescale in which the GPR model operates. A positive effect of this formulation is that the online offsets can correct fast time scale perturbations while the GPR model corrects for longer term effects such as friction. Furthermore, the risk of oscillations between the two learning systems is reduced is greatly reduced.

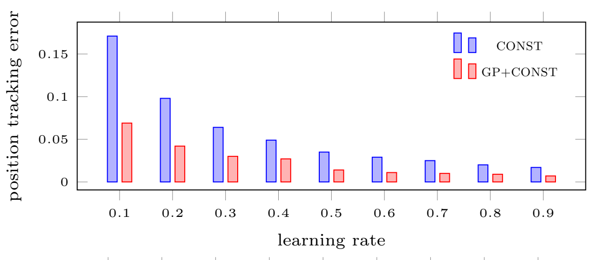
IV. Experiments

We evaluate our approach in three different settings. First we create a simulated 2D inverse dynamics problem that allows use to explore noise levels and parameter settings extensively. Second, we deploy our method in simulation for our KUKA lightweight arm. In this setting, we are also able to add noise and torque perturbations while being able to test real time capabilities of our approach. Finally, we evaluate our approach on the KUKA lightweight arm. In all these experiments we evaluate and compare the tracking error and the magnitudes of $\boldsymbol{w}$ – the adaptive feedback terms – of 2 different versions:

1) Const: Inverse dynamics control with adaptive constant offsets only, meaning only feedback terms are learned online
2) GP+Const: CONST plus a drifting Gaussian process to capture the inverse dynamics model errors

We compute the tracking error as the mean absolut difference between desired positions q_d and actual positions q_a

$$\frac{1}{T}\sum_{t=1}^{T}|q^t_d - q^t_a|$$



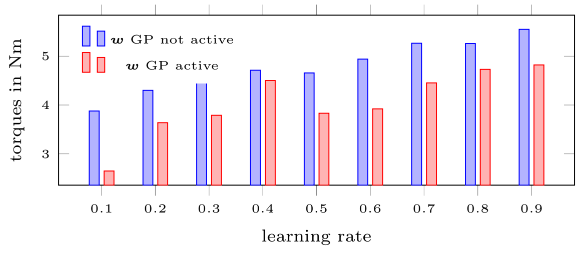


Fig. 2. (top) Position tracking error per learning rate of the stochastic gradient descent on the constant offsets. Results are averaged across noise with $1e-3, 1e-4, 1e-5$ standard deviation. (bottom) Average magnitude of $\boldsymbol{w}$, averaged across the same noise levels.

TABLE I
POSITION TRACKING ERROR

noise level	no GP	GP
0.01	0.058	0.058
0.001	0.056	0.037
0.0001	0.057	0.018
0.00001	0.057	0.018

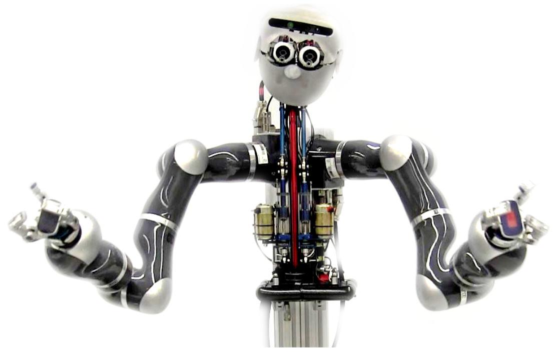


Fig. 3. Our robotic system with 2 KUKA lightweight arms, used to perform experiments

A. Simulated 2D System

We generate a system with simplistic inverse dynamics, where we model the true system to have forward dynamics

$$\tau_{\text{rbd}} = \mathbf{M}\,\ddot{q}_d = 5.0\mathbf{I}\,\ddot{q}_d \tag{12}$$

and the simulated robot uses an approximate model

$$\hat{\tau}_{\text{rbd}} = \hat{\mathbf{M}}\,\ddot{q}_d = 2.5\mathbf{I}\,\ddot{q}_d \tag{13}$$

Additionally, we model the true system such that it experiences some friction and stiction effects, that the simulated robot has no knowledge of. Finally, to simulate noisy sensor readings, we add zero mean Gaussian noise with standard deviations between $1e-5$ and $1e-2$ on the positions.
To evaluate our approach under these system settings we generate a trajectory, which takes 5 seconds assuming a control loop of 100Hz. Figure 1 shows the desired position trajectory in green. In orange, we see the trajectory that is being generated trying to track desired accelerations based on the approximate inverse dynamics model without a feedback term. Clearly, some feedback is required to improve tracking. We evaluate the two versions each with different learning rates for the online gradient descent on the constant offsets, from 0.1 to 0.9 in 0.1 increments. For the versions using GPR to estimate the modeling errors, we use a window size of 50, and perform online parameter learning. Further, we run both frameworks using each of the learning rates with 4 different sensing noise levels. Table I shows the tracking errors for each noise level, averaged across the different learning rates. As we can see, using the GP improves tracking in all cases but the highest noise level. Note that in our simulation we add noise on positions and finite difference twice to simulate actual accelerations. Thus, the resulting accelerations used as data points for the GP are exteremely noisy, and it is likely that the actual signal that we are trying to fit is lost.
To analyze the effect of the learning rate of the online gradient descent on the feedback terms, we visualize the same results in Figure 2(top) as function of the learning rate, this time we average across the 3 noise-levels (we exclude the results with the highest noise level for the reason stated above). We can clearly see, that for each learning rate, using the GP to learn the modeling errors helps improve tracking performance. The most gain is achieved for the lowest learning rate. Note, this is exactly the scenario that we want to deploy our framework in, a low learning rate on the constant offsets (the feedback term). The bottom plot, shows the resulting magnitudes of the learned offsets $\boldsymbol{w}$ (the feedback), both when using no additional feedforward term (blue), and when deploying the GP (red). As expected, the average magnitude of $\boldsymbol{w}$ drops when deploying the Gaussian process model. The lowest learning rate, creates the largest drop off. Note, that our goal is to use our framework with a small learning rate, while retaining or improving tracking accuracy through the online estimation of larger modeling errors. While we can see smaller tracking errors for higher learning rates, they come at the price of larger offset torques that are send to the system, resulting in reduced compliancy. We have observed oscillations, between the two learning processes when using a high learning rate on the feedback gains, a phenomenon that can also be observed for tradtional PID control frameworks with high gains. Thus our simulation results confirm that using a nonlinear function approximator to estimate inverse dynamics modeling errors leads to smaller feedback terms.

B. Robot Experiments

For both robot experiments, simulation and real, we were using the platform shown in Fig. 3. The platform has two KUKA lighweight arms, each of which has 7 degrees of freedom. All experiments were performed on one arm resulting in a 21-dimensional input for the online learning problem. Without loss of generality we only investigated the prediction

TABLE II

DESIRED ACCELERATION TRACKING ERROR.

	Simulation				Real			
	I	II	III	IV	I	II	III	IV
0.20: CONST	1.76	1.77	2.58	3.66	1.20	0.95	1.38	0.80
0.20: +GP	1.85	2.09	2.60	3.49	1.50	0.84	1.52	0.99
0.05: CONST	1.93	1.36	2.48	3.29	1.28	0.91	1.28	0.68
0.05: +GP	1.77	1.93	2.41	3.29	1.28	0.74	1.46	0.91
0.01: CONST	2.87	1.78	2.82	3.29	1.42	0.98	1.37	0.56
0.01: +GP	1.76	1.96	2.33	3.18	1.16	0.81	1.37	0.80

TABLE III

ONLINE OFFSETS FOR DESIRED ACCELERATION TRACKING.

	Simulation				Real			
	I	II	III	IV	I	II	III	IV
0.20: CONST	1.96	1.57	1.53	1.10	0.70	0.79	0.32	1.05
0.20: +GP	0.13	0.13	0.14	0.08	0.47	0.09	0.19	0.13
0.05: CONST	1.61	1.51	1.06	0.64	0.53	0.60	0.21	0.31
0.05: +GP	0.07	0.04	0.05	0.03	0.06	0.03	0.08	0.07
0.01: CONST	0.65	0.53	0.44	0.22	0.20	0.25	0.08	0.13
0.01: +GP	0.04	0.02	0.02	0.01	0.03	0.01	0.02	0.04

for a single joint, using the inputs of all joints. Notice, prediction for all joints is a simple extension, since each joint can be treated as an independent learning problem. Since our predictions and optimizations are performed in a hard real-time loop of 1 kHz, the current naive implementation requires 0.8 ms for optimization and prediction of the Gaussian process model with a window size of 50 plus online adaption of the feedback terms. By exploiting the intrinsic parallelism of the problem, this approach can be scaled to any number of joints.

The experiment consists of a pre-planned sequence of LQR [12] policies. These policies generate the desired accelerations online and if the end configuration of a policy is reached the next policy in the sequence is performed. Both the simulation and real robot experiments use the same policies and robot model. In the simulation case we add artificial noise to the accelerations obtained from the system which is dependent on the magnitude of the acceleration. Additionally, in simulation, we add both a torque bias and noise, to simulate model misspecifications. Thus, even without noise on the accelerations the system starts drifting since the feedforward torque computed with the inverse dynamics model would not compensate for our artificially introduced perturbations and noise.

In Table II we present the average acceleration tracking error for the 4 different policies, both for our very noisy simulation and the real system. Each experiment is performed with and without the online GPR model for three different learning rates of the online learning adaptive control. Notice that the highest learning rate results in good tracking, yet, it also applies higher torques. Furthermore, higher learning rates result in slightly worse results when combining online learning adaptive control with our proposed GPR model. For lower learning rates these two models perform very well together and for the lowest learning rate, the combination of both learning processes outperforms the high learning rate online adaptive control approach. Hence, our approach allows to significantly reduce the learning rate, thus achieving more compliant behavior, while increasing tracking performance.

In Table III the average offsets predicted by the online learning adaptive control approach are presented for the same experiment run. This illustrates that the proposed GPR model can patch the inverse dynamics model almost perfect, with the help of the online learning adaptive control which will compensate for minor errors and thus, automatically provide more meaningful data for the online GPR training.

V. DISCUSSION AND FUTURE WORK

We have presented an online learning control framework, that combines recent advances on online learning of feedback terms with an online Gaussian process regression to learn a locally valid inverse dynamics error model. We have shown that the Gaussian process model can estimate the larger inverse dynamics model, which has the effect of smaller feedback terms. This already works effectively, for low learning, however future work could explore reducing the learning rate further, once the nonlinear function approximator has reached a certain level of accuracy.

The presented framework uses a fixed size Gaussian process model, and due to hard real time constraints cannot use a large window size. In the future we plan to explore sparse approximations of Gaussian processes to be able to explore the effect of the window size. Furthermore, the approach presented here opens up the possibility of exploiting the uncertainty estimates of the Gaussian process model. We believe one interesting avenue to pursue is to use the uncertainty estimate to determine how much influence the online updated GP model should have on the total torque command.

REFERENCES

[1] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*, 2nd ed. Springer, 2010.

[2] D. Nguyen-Tuong, J. R. Peters, and M. Seeger, “Local Gaussian process regression for real time online model learning,” in *Advances in Neural Information Processing Systems*, 2008, pp. 1193–1200.

[3] M. F. Huber, “Recursive Gaussian process: On-line regression and learning,” *Pattern Recognition Letters*, vol. 45, pp. 85–91, 2014.

[4] A. Gijsberts and G. Metta, “Real-time model learning using incremental sparse spectrum Gaussian process regression,” *Neural Networks*, vol. 41, pp. 59–69, 2013.

[5] F. Meier, P. Hennig, and S. Schaal, “Incremental Local Gaussian Regression,” in *NIPS*, 2014.

[6] N. Ratliff, F. Meier, D. Kappler, and S. Schaal, “DOOMED: Direct Online Optimization of Modeling Errors in Dynamics,” arXiv:1608.00309 [cs.RO], Cornell University Library, Tech. Rep., 2016.

[7] F. Meier and S. Schaal, “Drifting Gaussian Processes with Varying Neighborhood Sizes for Online Model Learning,” in *ICRA*, 2016.

[8] C. H. An, C. G. Atkeson, and J. M. Hollerbach, “Estimation of Inertial Parameters of Rigid Body Links of Manipulators,” in *24th IEEE Conference on Decision and Control*, 1985.

[9] C. G. Atkeson, A. W. Moore, and S. Schaal, “Locally weighted learning for control,” *Artificial Intelligence Review*, 1997.

[10] S. Schaal and C. G. Atkeson, “Constructive incremental learning from only local information,” *Neural Computation*, 1998.

[11] M. Zinkevich, “Online Convex Programming and Generalized Infinitesimal Gradient Ascent,” in *ICML*, 2003.

[12] R. Stengel, *Optimal Control and Estimation*. Dover, New York, 1994.