2016 IEEE International Conference on Robotics and Automation (ICRA)
Stockholm, Sweden, May 16-21, 2016

Drifting Gaussian Processes with Varying Neighborhood Sizes for Online Model Learning

Franziska Meier[1,2] and Stefan Schaal[1,2]

Abstract— **Computationally efficient online learning of non-stationary models remains a difficult challenge. A robust and reliable algorithm could have great impact on problems in learning control. Recent work on combining the worlds of computationally efficient and locally adaptive learning algorithms with robust learning frameworks such as Gaussian process regression has taken a step towards both robust and real-time capable learning systems. However, online learning of model parameters on streaming data – that is strongly correlated, such as data arriving along a trajectory – can still create serious issues for many learning systems. Here we investigate the idea of drifting Gaussian processes which explicitly exploit the fact that data is generated along trajectories. A drifting Gaussian process keeps a history of a constant number of recently observed data points and updates its hyper-parameters at each time step. Instead of optimizing the neighborhood size on which the GP is trained on, we propose to use several – in parallel – drifting GPs whose predictions are combined for query points. We illustrate our approach on synthetically generated data and successfully evaluate on inverse dynamics learning tasks.**

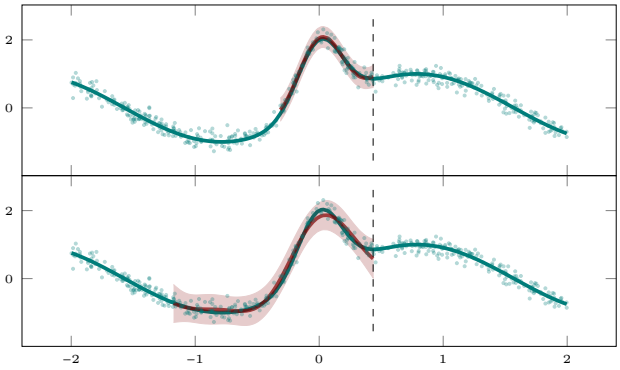

Fig. 1. Snapshot of 2 drifting Gaussian process regression models. The model shown on top is considering a shorter history of the trajectory than the model displayed on the bottom and thus is better able to adapt to the landscape.

I. INTRODUCTION

Machine learning has great potential to significantly improve model-based control. For instance, estimating an accurate dynamics model is a key component of compliant control and force control for complex robots, like humanoids. However, due to unknown and hard to model nonlinearities, analytical models of the dynamics for such systems are often only rough approximations. For that reason, this is one setting in which machine learning algorithms have been investigated to provide automatic model learning [1], [2], [3].

Because sensors of a robot can generate thousands of data points per second, a necessary focus of these approaches has been computational efficiency – in addition to producing accurate models. To this end, local learning approaches have been developed [4], [5], which are fast enough for real-time learning while being able to handle non-stationary data by learning local distance metrics. However, besides computational cost, another crucial design criterion has to be reliability and robustness of the learning and prediction process. Therefore, recent work has moved towards using Gaussian process regression methods [6], [2], [3].

[1]Computational Learning and Motor Control Lab, University of Southern California, Los Angeles, CA 90089, USA. [2]Max-Planck-Institute for Intelligent Systems, 72076 Tübingen, Germany. Email:fmeier@usc.edu

This research was supported in part by National Science Foundation grants IIS-1205249, IIS-1017134, CNS-0960061, EECS-0926052, the DARPA program on Autonomous Robotic Manipulation, the Office of Naval Research, the Okawa Foundation, and the Max-Planck-Society.

Gaussian process regression (GPR) offers a powerful and robust hyper-parameter learning framework. On top of that, GPR enables us to associate uncertainty estimates with the models prediction, which can be useful in gauging the reliability of the prediction. The main challenge is to create computationally efficient approximations of the standard GP framework, while retaining its robust learning framework. Two promising research directions addressing this challenge are sparse approximations of Gaussian processes [7], [8], [9] and localizing Gaussian process learning [6], [3]. Both of these approaches have also been applied to learning control settings, such as inverse dynamics learning [2], [10], [6], [3].

Sparse approximations aim at identifying a small set of (pseudo) data points [11], [12] to approximate the kernel matrix or at sparsifying the spectrum of the kernel matrix [8], [9]. These approaches typically require an a-priori estimation of how many resources to allocate for a sparse Gaussian process model. This becomes problematic when a model needs to be able to cover a growing workspace. Alternatively, local GP approaches [6], [3] aim at reducing computational complexity by localizing inference and learning on a small neighborhood. These methods allow the allocation of new resources, enabling the model complexity to grow when necessary. A challenge of these type of approaches is to identify the size of the neighborhood(s), which generally reduces to a difficult nonlinear optimization problem.

To make either of these approaches applicable for the setting of learning inverse dynamics, incremental learning of model parameters is key. Online sparse approaches exist [10], [2], however they typically learn one global distance

metric, making it difficult to fit the non-stationary data encountered in robotics. To address the issue of adapting to non-stationary data online, one can utilize forgetting rates [13] that allow to forget old data and focuses the parameter learning on the currently observed data. Alternatively, local online model learning approaches [1], [3] are able to deal with non-stationary data by allowing each local model to maintain its own set of parameters. Note that both, local models as well as forgetting rates, are essentially ways of determining neighborhoods in which data points contribute to parameter updates.

A concern often not addressed is the online learning of (hyper-)parameters on highly correlated data. Global models trained through online learning on correlated data would tend to unlearn previously optimized parameters to fit the currently observed data. This is true especially on tasks like learning kinematics or dynamics, where the generated data lies on a trajectory. While stochastic learning of sparse GPs [14] exists, their success is dependent on randomly drawn mini batches, as is the case for most global models trained stochasticly. Since it is often not feasible to generate a representative dataset for offline (mini-)batch training, algorithms need to be able to perform online learning from correlated streaming data.

To this end, we explicitly account for the fact that data comes in on a trajectory. Intuitively, at any given point in time one should be able to build a locally valid model from the last few observed data points. Here we explore the idea of a drifting GP. This allows us to be locally adaptive when experiencing curvature changes, but it comes with the advantages of GPs, namely robust parameter learning and uncertainty estimates. The key question is how many recently observed data points to utilize for training. Intuitively, one might think to use as many points as one can process in real time. However, Figure 1 illustrates that how many data points to use is also a question of the current function curvature. While we might be able to use a lot more of the recent observations, a model with *shorter attention span* might actually perform better because it can better adapt to the quickly changing curvature.

In the following, we review Gaussian process regression and sparse Gaussian process regression in Section II, which we will utilize to achieve low computational complexity. Next, in Section III we introduce the notion of drifting Gaussian process models with varying neighborhood sizes. Finally, in Section IV we illustrate our approach on synthetic data and evaluate it on inverse dynamics learning tasks.

II. Background: Sparse Gaussian Process Regression

Gaussian process regression (GPR) [15] offers principled inference for hyper-parameters. However, the computational cost of GPR grows cubically with the number of data points [15]. Thus, in its standard form GPR is not practical in applications like inverse dynamics learning where thousands of data points are observed in just a few seconds of moving. Specifically, let N be the number of noisy observations $\boldsymbol{y}$, at input locations $\boldsymbol{X}$ with hidden (true) function values $\boldsymbol{f}$. In standard Gaussian process regression the likelihood function $p(\boldsymbol{y} \mid \boldsymbol{f})$ and the prior over hidden function values $p(\boldsymbol{f})$ are defined as

$$p(\boldsymbol{y} \mid \boldsymbol{f}) = \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{f}, \beta^{-1}\mathbf{I}_N) \tag{1}$$

$$p(\boldsymbol{f}) = \mathcal{N}(\boldsymbol{f} \mid 0, \mathbf{K}_{\text{NN}}) \tag{2}$$

where for simplicity we have assumed a zero-mean prior, and where $\mathbf{K}_{\text{NN}}$ is the kernel matrix evaluated between all input points $\boldsymbol{X}$. Optimization of the the hyper-parameters (noise precision β and parameters of the kernel) is performed by maximizing the log marginal likelihood

$$\log p(\boldsymbol{y}) = \iint p(\boldsymbol{y} \mid \boldsymbol{f})\, p(\boldsymbol{f})\, \mathrm{d}\boldsymbol{f} = \log \mathcal{N}(\boldsymbol{y} \mid 0, \mathbf{K}_{y,\text{NN}}) \tag{3}$$

Both, evaluation and optimization of the log marginal likelihood involves the inversion of $\mathbf{K}_{y,\text{NN}} = \beta^{-1}\mathbf{I}_N + \mathbf{K}_{\text{NN}}$ - causing the high computational cost of standard GPR.

As mentioned above, recent progress in sparsifying Gaussian processes [11], [16] has resulted in scalable implementations of GPR. These sparsification methods can broadly be classified into two approaches: Identification of M support points, where $M << N$, [12], [17] or sparsification of the spectrum of the GP [8], [9], which essentially transforms GPR to the parametric Bayesian regression domain. Both of these sparsification methods have been successfully applied in a variety of domains.

A. Variational Sparse Gaussian Process Regression

Here, we are following the work of [12], which introduces a variational framework for the optimization of M pseudo input locations. To create a sparse GP approximation [12] starts by introducing M additional hidden function values $\boldsymbol{u}$ at input locations $\mathbf{Z}$, to form the following hierarchical model

$$p(\boldsymbol{y} \mid \boldsymbol{f}) = \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{f}, \beta^{-1}\mathbf{I}_N) \tag{4}$$

$$p(\boldsymbol{f} \mid \boldsymbol{u}) = \mathcal{N}(\boldsymbol{f} \mid \mathbf{K}_{\text{NM}}\mathbf{K}_{\text{MM}}^{-1}\boldsymbol{u}, \mathbf{K}_{\text{NN}} - \mathbf{K}_{\text{NM}}\mathbf{K}_{\text{MM}}^{-1}\mathbf{K}_{\text{MN}}) \tag{5}$$

$$p(\boldsymbol{u}) = \mathcal{N}(\boldsymbol{u} \mid 0, \mathbf{K}_{\text{MM}}) \tag{6}$$

where $\mathbf{K}_{\text{MM}}$ is the kernel matrix evaluated between all M input points $\mathbf{Z}$. Marginalizing out $\boldsymbol{u}$ results in the standard GP regression setup from above, and thus marginalizing out both $\boldsymbol{u}$ and $\boldsymbol{f}$ results in the same expression for the log marginal likelihood as in (3),

$$\log p(\boldsymbol{y}) = \iint p(\boldsymbol{y} \mid \boldsymbol{f}) p(\boldsymbol{f} \mid \boldsymbol{u}) p(\boldsymbol{u})\, \mathrm{d}\boldsymbol{u}\, \mathrm{d}\boldsymbol{f}. \tag{7}$$

Therefore, with exact inference this model is equivalent to standard GPR and has the same high computational complexity. As a remedy, in [18] a variational approximation is sought that reduces the computational complexity. Specifically, [18] seeks to approximate the true posterior $p(\boldsymbol{f}, \boldsymbol{u} \mid \boldsymbol{y})$ with an approximate posterior $q(\boldsymbol{f}, \boldsymbol{u})$ by

minimizing the KL-divergence $\mathrm{KL}(q(\boldsymbol{f},\boldsymbol{u})\|p(\boldsymbol{f},\boldsymbol{u} \mid \boldsymbol{y}))$. The key idea of using an approximate posterior $q(\boldsymbol{f},\boldsymbol{u})$ is to capture the idea of $\boldsymbol{u}$ being a sufficient statistic for $\boldsymbol{f}$. Thus, optimizing pseudo-input locations $\mathbf{Z}$ *shifts* around $\mathbf{Z},\boldsymbol{u}$ such that the sufficient statistics approximation is as exact as possible.

Marginalizing out $\boldsymbol{u},\boldsymbol{f}$ under the approximate posterior induces a lower bound on the exact log marginal likelihood. This bound is given through

$$\mathcal{L}_{\text{sparse}}(\beta,\mathbf{Z},\boldsymbol{\theta}) = \log\mathcal{N}(\boldsymbol{y} \mid 0,\beta^{-1}\mathbf{I}_N + \mathbf{K}_{\text{NM}}\mathbf{K}_{\text{MM}}^{-1}\mathbf{K}_{\text{MN}}) - \frac{\beta}{2}\operatorname{tr}\left[\mathbf{K}_{\text{NN}} - \mathbf{K}_{\text{NM}}\mathbf{K}_{\text{MM}}^{-1}\mathbf{K}_{\text{MN}}\right] \tag{8}$$

which depends on pseudo-input locations $\mathbf{Z}$, noise β and any kernel parameters $\boldsymbol{\theta}$. This lower bound on the log marginal likelihood becomes tight when $\mathbf{Z} = \boldsymbol{X}$. Note, how the inversion of the $N \times N$ kernel matrix $\mathbf{K}_{\text{NN}}$ has been replaced by inverting the much smaller kernel matrix $\mathbf{K}_{\text{MM}}$.

III. Drifting Gaussian Processes with Varying Neighborhood Sizes

In this work we utilize sparse GP regression in the setting of continuous online model learning. Specifically, we aim to address the scenario in which data comes in on a trajectory, as is the case for learning an inverse dynamics model $\mathcal{M}$, for instance. In this setting, at time step t, we want to use the current model $\mathcal{M}_t$ to make a prediction for a new input $\boldsymbol{x}_{t+1}$. Then, after observing the true y_{t+1} we want to include this new data point to update our model parameters.

Concretely, let $\mathcal{M}^t$ denote a Gaussian process model at time step t, with parameter set $\{\beta^t,\theta^t,\mathbf{Z}^t\}$, where β^t is the noise precision, θ^t denote the kernel parameters and $\mathbf{Z}^t$ are the current pseudo input locations. As new data points $\{\boldsymbol{x}^{t+1},y^{t+1}\}$ are being observed, they should be included in the GP optimization step. However, to avoid increasing computational complexity, we propose to drift the model along the trajectory instead of growing it. This allows us to keep the number of used data points N constant. The fact that data arrives on a trajectory can be used, to simply drop the *oldest* data point and replace it with the newest data point $\{\boldsymbol{x}_{t+1},y_{t+1}\}$. Specifically, a drifting GP model at time step $t+1$ is trained on $\mathcal{D}^{t+1}$, with $\mathcal{D}^{t+1} = \mathcal{D}^t - \{\boldsymbol{x}_{t-N},y_{t-N}\} + \{\boldsymbol{x}_{t+1},y_{t+1}\}$. Furthermore, instead of training models at each time step from scratch we can use the model from the previous time step to initialize the new one, and perform only a few optimization iterations per time step.

A. Parallel GPs with Varying Neighborhood Sizes

The key question now is how many past data points N to use to train the model at each time step with. It is likely that this number is not fixed and that it rather depends on the current curvature of the function to fit. Note, that in local methods this issue is being addressed by learning localized distance metrics, which define how large the neighborhood of active data points per local model is. However, this approach typically results in a highly non convex optimization problem.

Algorithm 1 Update Parallel Drifting GPs

Require: $\{\boldsymbol{x}_{t+1},y_{t+1}\}$, D_k^t, $\{\mathcal{M}_k^t\}_{k=1}^K$

// update data set

$\mathcal{D}_k^{t+1} \leftarrow \mathcal{D}_k^t - \{\boldsymbol{x}_{t-N_k},y_{t-N_k}\} + \{\boldsymbol{x}_{t+1},y_{t+1}\}$

// initialize new model for time step $t+1$ with parameters

$\mathcal{M}_k^{t+1} \leftarrow \mathcal{M}_k^t \forall k \in 1,\ldots,K$

$Z = 0$

for $k = 1 \rightarrow K$ **do**

  // maximize log marginal likelihood (lml) of k^{th} GP and return final lml

  $\mathcal{L}_k = \mathcal{M}_k^{t+1} \rightarrow \text{optimize}(\mathcal{D}_{t+1})$

  $Z = Z + \exp\{\mathcal{L}_k\}$

end for

// compute weight per GP, OPTION 1

$w_k^{t+1} \leftarrow 1/Z \exp\{\mathcal{L}_k\}$

// or via OPTION 2

$k_{\max} = \arg\max \mathcal{L}_k$

$w_{k_{\max}}^{t+1} = 1.0, \forall k \neq k_{\max}, w_k^{t+1} = 0.0$

return $\{\mathcal{M}_k^{t+1},w_k^{t+1}\}_{k=1}^K$

Here we investigate a different approach to determining neighborhood dimensions. Instead of formulating neighborhood estimation as a continuous learning problem, we choose to run K drifting GP models $\mathcal{M}_k, \forall k = 1,\ldots,K$ in parallel, each of which takes into account a different number N_k of data samples. As a result, the Gaussian process models cover neighborhoods of varying magnitudes. At each time step all models $\mathcal{M}_k$ are updated with the new incoming data point. An overview of this algorithm is given in Algorithm 1. What remains to be discussed is how to make predictions for new incoming data points within this setup.

B. Prediction

Given a query point $\boldsymbol{x}^*$, each of the K Gaussian process models independently makes a prediction with mean μ_k^* and variance σ_k^*. A variety of options exist to form a final prediction. In this work we form the final prediction through a weighted combination of all GPs. Here we investigate two options:

a) Option 1, denoted as GP_w: We can use the relative value of the lower bound of each GP to weigh their predictions. Specifically, this option evaluates the lower bound on the log marginal likelihood value $\mathcal{L}(\mathcal{M}_k^t,\mathcal{D}_k^t)$ of each sparse GP, and computes the weights as

$$w_k = \frac{\exp\mathcal{L}(\mathcal{M}_k^t,\mathcal{D}_k^t)}{\sum_{i=1}^K \exp\mathcal{L}(\mathcal{M}_i^t,\mathcal{D}_i^t)} \tag{9}$$

b) Option 2, denoted as GP_{max}: At each time step we can choose the Gaussian process that has the highest log marginal likelihood value – more precisely the approximate lower bound $\mathcal{L}$, Equation (8). Using this option, the weight of the

model with the maximum lower bound is set to one, and all others are set to zeros:

$$k_{\max} = \arg\max \mathcal{L}_k \tag{10}$$

$$w_{k_{\max}}^{t+1} = 1.0, \quad \forall k \neq k_{\max}, w_k^{t+1} = 0.0 \tag{11}$$

Given these weights we can form a convex combination of all K predictions to compute the final predictive mean and variance as

$$\mu^* = \sum_{k=1}^{K} w_k \mu_k^* \qquad \text{and} \qquad \sigma^* = \sum_{k=1}^{K} w_k \sigma_k^* \tag{12}$$

The computation of the lower bound $\mathcal{L}(\mathcal{M}_k^t, \mathcal{D}_k^t)$ does not incur any additional cost, as this is the objective function that is being maximized with respect to the hyper-parameters. Thus, after each optimization step we automatically receive the current value of the lower bound, as illustrated in the pseudo algorithm 1.

IV. Evaluation: Inverse Dynamics Learning

We evaluate our proposed approach on the task of online learning of inverse dynamics for robotic manipulators. Currently, the go to method – when possible – for estimating an inverse dynamics model is to perform system identification of the rigid body dynamics (RBD) model. The robot dynamics can be modeled via the general RBD equations of motions [19]

$$\boldsymbol{\tau}_{\text{rbd}} = \mathrm{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \mathrm{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \mathrm{G}(\boldsymbol{q}) \tag{13}$$

where $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}$ denote the vectors of joint positions, velocities, and accelerations, respectively, and $\boldsymbol{\tau}$ denotes the torques. The matrix $\mathrm{M}(\boldsymbol{q})$ is the RBD inertia matrix, the matrix $\mathrm{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ has terms about coriolis and centripetal forces, and the vector $G(\boldsymbol{q})$ represents torques due to gravity. The to be estimated RBD parameters appear linearly in Equation 13, such that the system identification problem for RBD can be treated as a linear regression problem [20]. However, in practice, the system identification of the RBD parameters is not always straightforward. Due to unknown and hard to model nonlinearities, the estimated RBD model for complex robotic systems is typically only a rough approximation. Furthermore, for complex systems such as humanoids it may not be possible to collect sufficiently rich data to estimate the rigid body dynamics parameters.

Thus, there has been an increased interest in learning inverse dynamics in a black-box manner [1], [2], [3], $\tau = f(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$ where we only assume that the inverse dynamics mapping is some nonlinear function f, that takes as input the joint position, velocities and accelerations. In previous work [3] we have experimentally shown that collecting data for offline learning is unlikely to work well for high-dimensional systems. Parameters learned on the offline data would create a model that works well in the covered workspace, however it is infeasible to collect data for the whole workspace, leading to suboptimal performance in parts of the input space for which no data was collected. Thus online learning - on highly correlated trajectory data - is necessary.

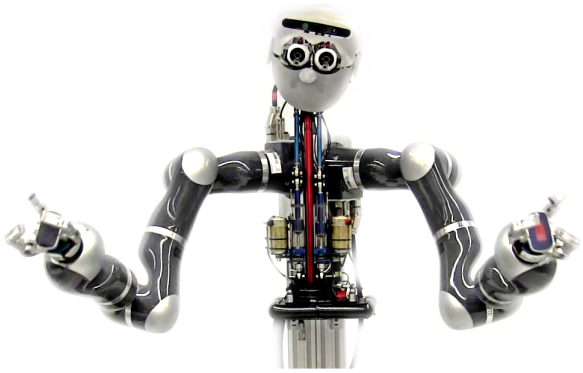

Fig. 2. One of our experimental platforms: The robot *Apollo* is a dual arm manipulation platform consisting of two 7 DoF KUKA light weight arms, two Barrett hands and a SARCOS humanoid head.

A. Experimental Setup

Before looking at inverse dynamics learning we first use a synthetic dataset with varying curvature to provide a detailed illustration of parallel drifting GPs. Then we use the publicly available SARCOS dataset, often used as a benchmark for inverse dynamics learning [15], for a quantitative analysis of the proposed method. Finally, we deploy our proposed approach on data of the KUKA lightweight arm, shown in Figure 2, which was collected in simulation.

In all experiments we use the squared exponential kernel

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_f^2 \exp\{-0.5 \sum_d \frac{(x_{i,d} - x_{j,d})^2}{l_d^2}\} \tag{14}$$

with automatic relevance determination. We perform 5 iterations of scaled conjugate gradients to update the hyper-parameters given the current data sets $\mathcal{D}_k^t$, and given these models perform a one step look ahead prediction for input $\boldsymbol{x}_{t+1}$. Errors reported are the (normalized) mean squared errors (nMSE) over these one step look ahead predictions:

$$\text{MSE} = \frac{1}{T}\sum_{t=1}^{T}(y^t - \hat{y}^t)^2 \quad \text{and} \quad \text{nMSE} = \frac{\text{MSE}}{\text{var}(\boldsymbol{y})}$$

where T is the length of the trajectory, y^t and $\hat{y}^t$ are the ground truth and prediction, respectively. Note, $\hat{y}^t$ is the combination of the predictive means μ_k^t of all K GP models $\mathcal{M}_k^{t-1}$, optimized on the last N_k data points. The normalized MSE, is the mean squared error normalized by the variance of all ground truth targets $\boldsymbol{y}$.

B. Synthetic Data

For the synthetic data set we draw 1000 points from $f(x) = \sin(2x) + 2\exp(-16x^2)$ and add Gaussian noise with variance 0.1^2. We use $M = 5$ pseudo inputs for the approximate sparse Gaussian process model, which are optimized along with the other hyper-parameters at each time step. In this experiment we use $K = 3$ Gaussian process models that keep a history of the last $N_1 = 50, N_2 = 100, N_3 = 150$ observed data points to optimize their parameters. In Figure 3 we visualize a snapshot of the 3 drifting Gaussian process models GP_1, GP_2 and GP_3

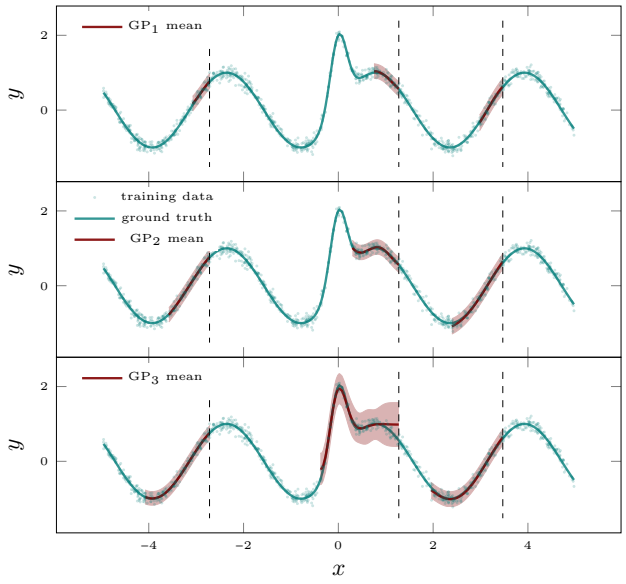

Fig. 3. Snapshots of all 3 Gaussian process regression models at 3 different time steps as they drift across the function. Black vertical dashed lines indicate the snapshot times. Noisy data points drawn from $f(x)$ shown in green. The mean and variance of each Gaussian process are shown in dark and light red, respectively.

TABLE I

PREDICTIVE PERFORMANCE ON SYNTHETIC DATA

MSE GP_1	MSE GP_2	MSE GP_3	MSE GP_w	MSE GP_{max}
0.01593	0.0154	0.0194	0.0150	0.0148

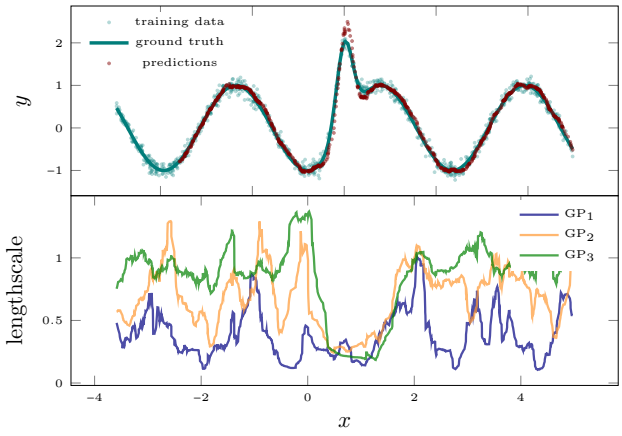

Fig. 4. Illustration of one instance of one step look ahead predictions using option 2 to combine predictions (top). Evolution of lengthscales for all 3 GP models (bottom) as they are drifting along the trajectory.

TABLE II

PREDICTIVE PERFORMANCE ON SARCOS INVERSE DYNAMICS

Joint	nMSE GP_1	nMSE GP_2	nMSE GP_w	nMSE GP_{max}
j1	0.035	0.034	0.033	0.034
j2	0.037	0.037	0.034	0.036
j3	0.027	0.028	0.025	0.026
j4	0.015	0.132	0.013	0.012
j5	0.037	0.036	0.035	0.035
j6	0.056	0.054	0.050	0.053
j7	0.019	0.018	0.017	0.018

at 3 different time steps. We can see that GP_3, which keeps the longest history (bottom), has a problem with the quickly changing curvature in the middle of $f(x)$. In Figure 4 we visualize the weighted one step look ahead predictions (top), computed using Option 2. We can see that the predictions are not affected by the suboptimal fit of GP_3 however. The bottom graph of Figure 4 displays a typical evolution of the lengthscale parameter l, showing how the lenghtscale decreases to account for the high curvature region of $f(x)$. We performed this experiment with 10 randomly seeded runs and report the average MSE per Gaussian process model, and for both options of combining the predictions. GP_w and GP_{max} stand for predictions made via option 1) and option 2), respectively. On average, the weighted predictions of all 3 drifting GP outperforms the single GPs. Note however, that more importantly, in a real setting we wouldn't be aware which GP is performing how well. Thus, automatically combining the predictions of all K models is essential, and is here shown to perform at least as well as the best single GP.

C. Sarcos Data

For our second set of experiments we use the publicly available SARCOS data [15], which consists of rhythmic motions and contains $44,484$ training data points and $4,449$ test data points. The Sarcos arm has seven degrees of freedom, thus for the inverse dynamics learning tasks we have 21 input variables $\boldsymbol{x}$, representing joint positions, velocities and accelerations for the 7 joints. The task is to predict the 7 joint torques, each of which we treat as independent regression problems. We exclusively use the training set here, and simulate the setting of data incrementally arriving on a trajectory. We use $K = 2$ models, with GP_1 keeping a history of the last $N_1 = 500$ and GP_2 of the last $N_2 = 1000$ data points. Each sparse GP uses $M = 10$ pseudo inputs that are being optimized via scaled conjugate gradients at each time step (along with the other hyper-parameters).

In Table II we present the normalized mean squared error made on the one step look ahead predictions, similar to the evaluation on the synthetic data set. The results are reported as the normalized mean squared error of the predicted torques in Newton metre (Nm). It is noteworthy that these results are on par with recently reported results for local and global learning methods in [3]. Thus, drifting GPs can keep up with the current state of the art. Also, the weighted combinations for making the one step look ahead predictions, perform better as the best single drifting GP for all joints. These results indicate, that depending which single GP performs best depends on the part of the input space. Furthermore, our proposed approach of combining the predictions of all drifting GPs is able to put more weight on the GP that is currently performing best.

D. KUKA LWR

Finally, we solidify our experimental evaluation on the KUKA data consisting of rhythmic motions. The KUKA lightweight arm also has 7 degrees of freedom, resulting in a 21-dimensional input for each of the 7 regression problems. The dataset, consisting of 100000 data points, was collected at 500Hz using [21]. The experimental setup and parameter initialization is exactly the same as for the SARCOS experiments. In Table III the normalized mean squared errors of the one step look ahead predictions is reported. The results show the same pattern as observed on the SARCOS data, only more pronounced. For almost all joints the combined predictions of both drifting GPs is significantly better than each of the single GPs. It is noteworthy, that our implementation processed the data at approximately 50Hz using our unoptimized C++ optimization (on a 2.8GHZ Intel Core i7) and performing parameter updates and predictions sequentially.

E. Parameter initialization:

A common concern in learning control settings is the robustness to initial parameter settings. Here, we have to initialize $\{\beta^t, \theta^t = l^t, \mathbf{Z}^t\}$. We have found that a simple strategy for initializing works best: We set the noise precision $\beta^0 = 10$, expecting a low signal to noise ratio on the inverse dynamics data. The initial pseudo input locations $\mathbf{Z}^0$ are linearly spaced on the first N_k data points for each GP, and the lengthscale parameters of the kernel matrix were initialized with $l = 0.3$. These settings were used for all experiments on the SARCOS and KUKA LWR data.

V. CONCLUSIONS

We have presented an online model learning approach based on drifting Gaussian process regression models and have shown that it can achieve performance on par with current state of the art on the task of learning inverse dynamics. We circumvent learning a neighborhood size - which typically involves gradient descent on non-convex objective functions - by drifting several GP models in parallel and have proposed two possible ways of combining them to form predictions. Because our proposed method forgets data by design, it can adapt to non-stationary changes of the function space. In contrast to other current approaches in this field, no additional parameters such as forgetting rates or learning rates for neighborhood estimation are required, resulting in a simple and robust, yet powerful and accurate learning method.

Future work will investigate possibilities to introduce some form of memory. Currently, the drifting GPs forget everything by design, and each time a part of the input space is revisited the learning problem is as hard as the first time. By introducing some form of a global model that serves as memory, learning could become easier and some of the computational burden of the drifting GPs could be reduced. Finally, we will investigate how to speed up our proposed approach such that it becomes a viable approach for real-time, on-the-fly, learning of inverse dynamics.

TABLE III
PREDICTIVE PERFORMANCE ON KUKA LWR

Joint	nMSE GP_1	nMSE GP_2	nMSE GP_w	nMSE GP_{max}
j1	0.113	0.034	0.013	0.014
j2	0.108	0.031	0.012	0.012
j3	0.006	0.035	0.008	0.009
j4	0.004	0.043	0.009	0.009
j5	0.489	0.011	0.007	0.007
j6	0.510	0.057	0.102	0.102
j7	0.011	0.019	0.008	0.009

REFERENCES

[1] S. Vijayakumar and S. Schaal, "Locally weighted projection regression: Incremental real time learning in high dimensional space," in *ICML*, 2000, pp. 1079–1086.

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

[3] F. Meier, P. Hennig, and S. Schaal, "Incremental Local Gaussian Regression," in *NIPS*, 2014.

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

[5] S. Schaal and C. G. Atkeson, "Constructive incremental learning from only local information," *Neural Computation*, vol. 10, no. 8, pp. 2047–2084, 1998.

[6] 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.

[7] L. Csató and M. Opper, "Sparse on-line gaussian processes." *Neural computation*, vol. 14, no. 3, pp. 641–68, Mar. 2002.

[8] A. Rahimi and B. Recht, "Random features for large-scale kernel machines." in *NIPS*, 2007.

[9] M. Lázaro-Gredilla, J. Quiñonero-Candela, C. E. Rasmussen, and A. R. Figueiras-Vidal, "Sparse spectrum Gaussian process regression," *JMLR*, vol. 11, pp. 1865–1881, 2010.

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

[11] J. Quiñonero Candela and C. E. Rasmussen, "A unifying view of sparse approximate Gaussian process regression," *JMLR*, vol. 6, pp. 1939–1959, 2005.

[12] M. K. Titsias, "Variational learning of inducing variables in sparse Gaussian processes," in *International Conference on Artificial Intelligence and Statistics*, 2009, pp. 567–574.

[13] S. Van Vaerenbergh, M. Lázaro-Gredilla, and I. Santamaria, "Kernel recursive least-squares tracker for time-varying regression." *IEEE transactions on neural networks and learning systems*, Aug. 2012.

[14] J. Hensman, N. Fusi, and N. D. Lawrence, "Gaussian Processes for Big Data," *Proceedings of UAI*, pp. 282–290, 2013.

[15] C. E. Rasmussen and C. K. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[16] K. Chalupka, C. K. Williams, and I. Murray, "A framework for evaluating approximation methods for Gaussian process regression," *JMLR*, vol. 14, no. 1, pp. 333–350, 2013.

[17] E. Snelson and Z. Ghahramani, "Sparse Gaussian processes using pseudo-inputs," *Advances in neural information processing systems*, vol. 18, p. 1257, 2006.

[18] M. Titsias, "Variational Learning of Inducing Variables in Sparse Gaussian Processes," in *AISTATS*, vol. 5, 2009, pp. 567–574. [Online]. Available: http://eprints.pascal-network.org/archive/00006353/

[19] L. Sciavicco and B. Siciliano, "Modelling and control of robot manipulators," 2000.

[20] C. H. An, C. G. Atkeson, and J. M. Hollerbach, "Estimation of Inertial Parameters of Rigid Body Links of Manipulators," in *Proceedings of the 34th Conference on Decision and Control*, no. December, 1985, pp. 990–995.

[21] S. Schaal, "The SL simulation and real-time control software package," Tech. Rep., 2009. [Online]. Available: http://www-clmc.usc.edu/publications/S/schaal-TRSL.pdf