**2014 14th IEEE-RAS International Conference on
Humanoid Robots (Humanoids)
November 18-20, 2014. Madrid, Spain**

Computed Torque Control with Variable Gains through Gaussian Process Regression

Nicolas Torres Alberto[1], Michael Mistry[2], Freek Stulp[1,3]

Abstract— **In computed torque control, robot dynamics are predicted by dynamic models. This enables more compliant control, as the gains of the feedback term can be lowered, because the task of compensating for robot dynamics is delegated from the feedback to the feedforward term. Previous work has shown that Gaussian process regression is an effective method for learning computed torque control, by setting the feedforward torques to the mean of the Gaussian process. We extend this work by also exploiting the variance predicted by the Gaussian process, by lowering the gains if the variance is low. This enables an automatic adaptation of the gains to the uncertainty in the computed torque model, and leads to more compliant low-gain control as the robot learns more accurate models over time. On a simulated 7-DOF robot manipulator, we demonstrate how accurate tracking is achieved, despite the gains being lowered over time.**

I. Introduction

A critical safety aspect of physical human-robot interaction is that robots should avoid high impact forces with humans, and should rather be compliant. One approach to achieving this is active compliance. Here, the torque u of each actuator is determined by a feedback term, e.g. a PD controller with gains K_P and K_D, and a feedforward torque u_{ff}. The feedforward term compensates for (predictable) robot dynamics, and the feedback term achieves robustness towards (unpredictable) perturbations.

$$u = \underbrace{K_P(q_{\text{ref}} - q) + K_D(\dot{q}_{\text{ref}} - \dot{q})}_{\text{feedback}} + \underbrace{u_{\text{ff}}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})}_{\text{feedforward}} \tag{1}$$

If the feedforward term u_{ff} would not compensate for the robot's dynamics, we would have to dramatically increase the feedback gains, as the feedback term would now be responsible also for compensating these dynamics. Therefore, knowing u_{ff} is essential to achieving low-gain compliant control, and thus safe physical human-robot interaction.

But how do we acquire an accurate computed torque function $u_{\text{ff}}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ for each joint? As discussed in more detail in Section II, there are three basic approaches: • determine the rigid body dynamics model of a robot and compute its open kinematic and dynamic parameters from its design, i.e. a CAD model of the robot [1]. • as above, but using parameter identification to determine the dynamic parameters from observations [1], [2]. • *learn* the computed torque, for instance through iterative learning control [3], or with Gaussian process regression (GPR) [4]–[7]. In this latter supervised learning approach, a Gaussian process (GP) is trained with example data with known feedforward torques. During control, the mean of the Gaussian process, the most likely estimate for an input given the training data, is then used as a feedforward torque.

Our main innovation in this paper to also exploit the *variance* of the Gaussian process, with the purpose of lowering feedback gains. High variance indicates that the computed torque is not yet known accurately. This implies that we should use higher gains to compensate for modelling inacurracies. By using variable gains that are adapted to the uncertainty in the model, we are able to achieve more accurate tracking, whilst being compliant whenever the accuracy of the model allows it, as illustrated in Fig. 1.

Furthermore, because more training data leads to more accurate models, the gains are automatically reduced overall as learning progresses. This is also visible in Fig. 1, where the overall variance reduces as the number of training trajectories increases.

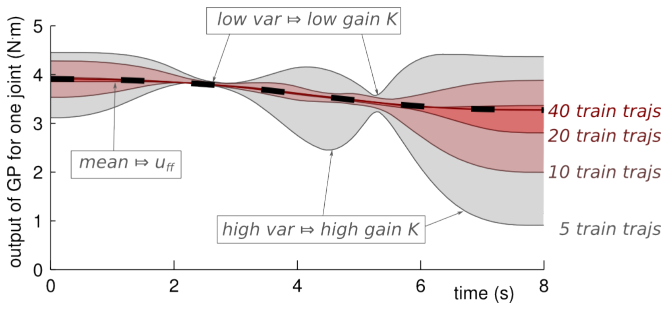

Fig. 1. Mean and variance of the Gaussian process ($\mu \pm 2\sigma$) on the same test trajectory of 8 seconds, after having been trained with 5, 10, 20 and 40 training trajectories. With an increasing number of training data, the mean of the GP approaches the true function (black dashed line). The known values for u_{ff} are plotted as a black dotted line. Previous work [4]–[7] has used the mean of the GP to provide computed torques $\mathbf{u}_{\text{ff}}$ in the controller in (1). In this paper, we also use the variance, in order to reduce the feedback gains, as indicated in the graph.

The rest of this paper is structured as follows. In the next section, we provide an overview of related work. Section III then describes in more detail the approach of applying GPR to computed torque control. In Section IV we present our contribution: lowering feedback gains by considering the variance of the GP. The method is empirically evaluated in simulation in Section V, after which we conclude with Section VI.

[1]Robotics and Computer Vision, ENSTA-ParisTech, Paris, France
[2]School of Computer Science, University of Birmingham, UK
[3]FLOWERS Team, INRIA Bordeaux Sud-Ouest, Talence, France

This paper was partly supported by the European Commission, within the CoDyCo project (FP7-ICT-2011-9, No. 600716).

II. Related Work

Traditional algorithms for learning computed torque exploit the linearity of inertial parameters with respect to the inverse dynamics of articulated rigid bodies [2]. In this case, least squares regression can be used to fit a set of parameters from observed kinematic and force measurements.

The methods of [8] use an adaptation law to update the inertial parameters online, while providing guarantees for tracking error convergence. However, these techniques must assume an underlying model structure, and thus fail to estimate or learn unmodeled dynamics such as flexible elements or nonlinear friction. Alternatively, the mapping $\mathbf{u}_{\text{ff}} = f(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ may be learned directly from the data, as in for instance [4]–[7], [9], [10]. This is quite a challenging problem, because the input space is $3\dim(\mathbf{q})$.

Gaussian process regression (GPR) [7] has been shown to be particularly effective in learning computed torques, but is not readily applicable to real-time control for computational reasons. These may be overcome by using local [4] or sparse [5] GPR. Multi-task GPR is able to infer inverse dynamics for multiple task contexts [6].

Iterative learning control schemes do not require any assumptions on model structure, but rather use a simple adaptation scheme to learn a time dependant feed-forward input, to reduce tracking error over several repetitions of a task [3]. Unmodeled dynamics, or even repeating disturbances, can be compensated for in an anticipatory fashion [11].

The above methods, however, work by adapting the feed-forward input, keeping feedback gains, and thus the impedance of the robot, constant. The work of [12] developed a biologically inspired algorithm to simultaneously adapt feed-forward input, desired trajectory, and impedance to reduce tracking error and effort over repeated trials. Impedance is adjusted based on the tracking error from the previous trial. In contrast to that work, we learn a state dependant feed-forward model (i.e. computed torque) and adapt impedance based on the confidence of the learned model. In this manner, a robot may anticipate high errors before entering a new workspace or attempting a new task, and subsequently increase impedance to prevent errors from occurring.

III. Gaussian Processes for Computed Torque

In this section, we present the theory behind Gaussian process regression (GPR) [7], [13], and its application to computing feedforward torques as in [4]–[7], [9], [10].

Gaussian processes (GP) assume that (any subset of) observed data is generated from (zero mean) multi-variate Gaussian distribution [7], [13], that is

$$y \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}). \tag{2}$$

The covariance matrix $\mathbf{\Sigma}$ is determined by the covariance function g. It is often chosen to be a radial kernel, e.g. a Gaussian function

$$g(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\tfrac{1}{2}(\mathbf{x}-\mathbf{x}')^\intercal \mathbf{W}^{-1}(\mathbf{x}-\mathbf{x}')\right), \tag{3}$$

as for smooth functions, we expect in general higher covariance between points that are closer. Here, σ_f represents signal variance, and $\mathbf{W}$ the covariance matrix of the Gaussian kernel[1]. Note that the maximum value of the Gaussian function is σ_f^2, which occurs when $\mathbf{x} = \mathbf{x}'$.

For a given covariance function g and N training points $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ the corresponding Gaussian Process is:

$$y \sim \mathcal{N}(\mathbf{0}, \mathbf{G}(\mathbf{X}, \mathbf{X})), \tag{4}$$

where

$$\mathbf{G}(\mathbf{X}, \mathbf{X}) = \begin{bmatrix} g(\mathbf{x}_1,\mathbf{x}_1) & \dots & g(\mathbf{x}_1,\mathbf{x}_N) \\ g(\mathbf{x}_2,\mathbf{x}_1) & \dots & g(\mathbf{x}_2,\mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ g(\mathbf{x}_N,\mathbf{x}_1) & \dots & g(\mathbf{x}_N,\mathbf{x}_N) \end{bmatrix} \tag{5}$$

is the Gram matrix, which is a covariance matrix containing the covariance function values for all pairs of input examples. In the above, the input examples $\{\mathbf{x}_n\}_{n=1}^N$ have been combined in one design matrix $\mathbf{X}$, where each row contains one example.

Predicting y^* for a novel input $\mathbf{x}^*$ – Gaussian process regression (GPR) – is done by assuming the that the novel output y^* will also be sampled from a multivariate Gaussian with

$$\begin{bmatrix} \mathbf{y} \\ y^* \end{bmatrix} \sim \mathcal{N}(\mathbf{0}, \begin{bmatrix} \mathbf{G}(\mathbf{X},\mathbf{X}) & \mathbf{g}(\mathbf{x}_*,\mathbf{X})^\intercal \\ \mathbf{g}(\mathbf{x}_*,\mathbf{X}) & g(\mathbf{x}_*,\mathbf{x}_*) \end{bmatrix}), \text{ and} \tag{6}$$

$$\mathbf{g}(\mathbf{x}_*, \mathbf{X}) = [g(\mathbf{x}_*, \mathbf{x}_1), \dots, g(\mathbf{x}_*, \mathbf{x}_n)]. \tag{7}$$

Conditioning this joint distribution to predict y_* given $\mathbf{X}$, $\mathbf{y}$, and $\mathbf{x}_*$ yields another multivariate Gaussian:

$$\begin{aligned} y_*|\mathbf{X}, \mathbf{y}, \mathbf{x}_* \sim \mathcal{N}(&\mathbf{g}(\mathbf{x}_*, \mathbf{X})\mathbf{G}(\mathbf{X}, \mathbf{X})^{-1}\mathbf{y}, \\ &g(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{g}(\mathbf{x}_*, \mathbf{X})\mathbf{G}(\mathbf{X}, \mathbf{X})^{-1}\mathbf{g}(\mathbf{x}_*, \mathbf{X})^\intercal) \end{aligned} \tag{8}$$

Thus, the best estimate for y_* is the mean, and the uncertainty in y_* is captured by the variance:

$$\overline{y}_* = \mathbf{g}(\mathbf{x}_*, \mathbf{X})\mathbf{G}(\mathbf{X}, \mathbf{X})^{-1}\mathbf{y} \tag{9}$$

$$\text{var}(y_*) = g(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{g}(\mathbf{x}_*, \mathbf{X})\mathbf{G}(\mathbf{X}, \mathbf{X})^{-1}\mathbf{g}(\mathbf{x}_*, \mathbf{X})^\intercal \tag{10}$$

If each measurement itself assumed to be a noisy observation, with additive independent identically distributed Gaussian noise with variance σ_n^2, the Gaussian process becomes

$$y \sim \mathcal{N}(\mathbf{0}, \mathbf{G}(\mathbf{X}, \mathbf{X}) + \sigma_n^2\mathbf{I}). \tag{11}$$

With this model, all occurences of the Gram matrix $\mathbf{G}(\mathbf{X}, \mathbf{X})$ in (4)-(10) must be replaced with $\mathbf{G}(\mathbf{X}, \mathbf{X}) + \sigma_n^2\mathbf{I}$ when performing GPR.

The open parameters of the Gaussian process, for instance the parameters σ_f and $\mathbf{W}$ of the covariance function, are known as the *hyperparameters*. It is customary to tune the hyperparameters automatically by minimizing the log

[1]We follow the notation in [4], except for using g for the kernel, rather k; the latter may cause confusion with the gains K.

likelihood of the meta-parameters on a training dataset, i.e. $\arg\min_\theta \log p(\mathbf{y}|\mathbf{X},\theta)$, where θ contains all the hyperparameters, e.g. $\mathbf{W}$ and σ_f.

GPR is applicable to computed torque control if we define the input space to be the angles/angular velocities/accelerations of the joints, and the output space to be the computed torque. Thus, a set of N training examples $\{\{([\mathbf{q}_n\ \dot{\mathbf{q}}_n\ \ddot{\mathbf{q}}_n], u^d_{\text{ff,n}})\}_{n=1}^N\}_{d=1}^D$ for each of the D joints must be available. With this training data, $\mathbf{X} = \{[\mathbf{q}_n\ \dot{\mathbf{q}}_n\ \ddot{\mathbf{q}}_n]\}_{n=1}^N$ and $\mathbf{y}^d = \{u^d_{\text{ff,n}}\}_{n=1}^N$, using the symbols $\mathbf{X}$ and $\mathbf{y}$ from the previous section. In our experimental evaluation in Section V, we describe two methods to acquire this training data, i.e. from model-based predictions, or through iterative learning control.

Given the training data, we define the corresponding GP with (4). The covariance function typically used in learning computed torque is the Gaussian kernel [4]. Each joint is modeled with a separate GP, i.e. for a 7-DOF arm there will be 7 GPs predicting the u_{ff} for that joint, with each GP having a 21-D input space (angles, velocities, accelerations for all joints).

On-line during execution, the feed-forward torques $u_{\text{ff,t}}$ for the current joint states $\mathbf{x}_t = [\mathbf{q}_t\ \dot{\mathbf{q}}_t\ \ddot{\mathbf{q}}_t]$ is computed as the mean of the GP, i.e.

$$\overline{u}^d_{\text{ff,t}} = \mathbf{g}(\mathbf{x}_t,\mathbf{X})\mathbf{G}(\mathbf{X},\mathbf{X})^{-1}\mathbf{y}^d \tag{12}$$

For control on physical robotics systems, local [4] or sparse [5] GPR approaches are necessary to overcome real-time issues in computing the above equation.

IV. Variance-dependent Gains

The key idea behind using GPR for variable gain control is that, if there is a high uncertainty in our feed-forward torque u_{ff}, we need higher gains to compensate for the inaccurately modelled robot dynamics to ensure accurate tracking. But, if u_{ff} is known with high accuracy, we can lower the gains. This idea has been illustrated in Fig. 1, and is implemented as follows.

Given a GP modelling of the computed torque training data, the variance in u_{ff} is computed with (10) as

$$\text{var}(u_{\text{ff,t}}) = g(\mathbf{x}_t,\mathbf{x}_t) - \mathbf{g}(\mathbf{x}_t,\mathbf{X})\mathbf{G}(\mathbf{X},\mathbf{X})^{-1}\mathbf{g}(\mathbf{x}_t,\mathbf{X})^\intercal. \tag{13}$$

The standard deviation is thus $\text{std}(u_{\text{ff,t}}) = \sqrt{\text{var}(u_{\text{ff,t}})}$, which has the same unit as $\overline{u}_{\text{ff}}$, i.e. $N \cdot m$. We know that the maximum value of $\text{std}(u_{\text{ff,t}})$ is σ_f (signal variance) and its minimum is σ_n (observation noise). Thus, we should choose high gains when $\text{std}(u_{\text{ff,t}}) = \sigma_f$ and low gains $\text{std}(u_{\text{ff,t}}) = \sigma_n$. As we expect an exponentially decaying learning curve for $\text{std}(u_{\text{ff,t}})$, we have chosen an exponential function to map $\text{std}(u_{\text{ff,t}})$ to K_P:

$$K_{P,t} = K^{\text{min}} + (1-z)(K^{\text{max}} - K^{\text{min}}) \tag{14}$$

$$z = \exp\left(-C\frac{\text{std}(u_{\text{ff,t}}) - \sigma_n}{\sigma_f - \sigma_n}\right) \tag{15}$$

The open parameters K^{min} and K^{max} are chosen by hand; but they must also be if our method is not applied. We determine K_D from K_P using the method described in [14]. Finally, the parameter C determines how quickly gains decrease as learning progresses. Throughout the paper, we use $C = 10$. The effect of this parameter on the mapping is shown in Fig. 2.

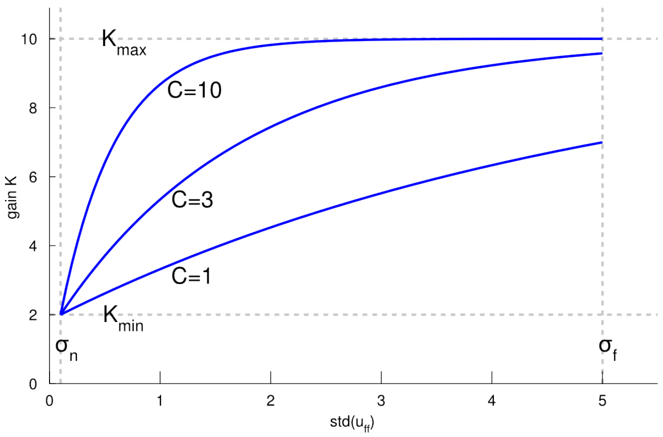

Fig. 2. Mapping uncertainty in $u_{\text{ff,t}}$ to gains K_P.

Thus, in summary, we have one GPR for each degree of freedom for which we compute a feedforward torque. The feedforward torque is the mean $\overline{u}_{\text{ff}}$ of the GPR for the input $\mathbf{x}_t = [\mathbf{q}_t\ \dot{\mathbf{q}}_t\ \ddot{\mathbf{q}}_t]$. The feedback gain is determined by computing the standard deviation of the GPR $\text{std}(u_{\text{ff,t}})$ for $\mathbf{x}_t$, and using the exponential function in (14) to determine K from $\text{std}(u_{\text{ff,t}})$.

V. Experimental Evaluation

For our experimental evaluation, we use a simulation of the SARCOS master arm, depicted in Fig. 3. The benchmark dataset used in for instance [4], [7], [15] was generated with this robot. We use a simulation of this robot in Simulation Lab (SL) [16], as in [4]. SL uses the same dynamic model for both forward dynamics, as needed for the simulation, and inverse dynamics (using recursive Newton-Euler), as needed for control. As the same deterministic and ideal model is used for both simulation and control, we thus know the ground-truth feedforward torque required to compensate for the robot dynamics.

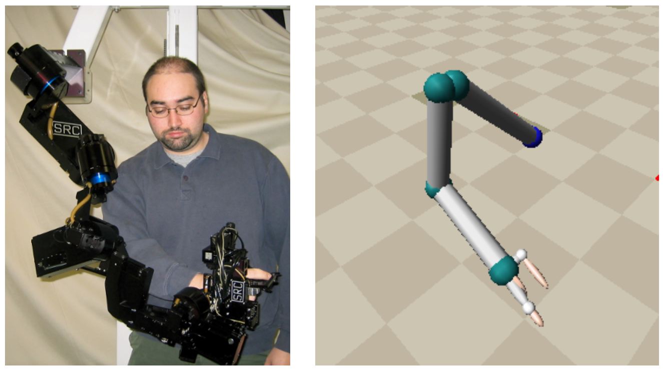

Fig. 3. The SARCOS master arm (left), and its simulation in SL (right)

The Gaussian process regression implementation we use is the GPML [7] Matlab Code version 3.4 [2]

The parameters of the Gaussian process – the diagonal of $\mathbf{W}$, and the scalars σ_f, σ_n in (3) and (11) – are determined with hyperparameter optimization. In the Appendix A, we provide an alternative analytical method for determining σ_f. We use the low default gains, which have been tuned previously for the physical robot, as K_{min}. Furthermore, $K_{\text{max}} = 6K_{\text{min}}$. The specific values for each joint can be read from Fig. 6. The decay rate C in (15) from is set to 10.

To compare our method to state-of-the-art in model-based and learned computed torque, we run four different scenarios. • GPR: In this scenario, we learn the feed-forward torque with GPR, but not the gains. This thus corresponds to current state-of-the-art [4]–[7]. • GPR-VG: This is the method proposed in this paper, i.e. we learn the feed-forward torque with GPR ($\overline{u}_{\text{ff}}$) *and* decrease the feedback gains as the variance of the GPR $\text{var}(u_{\text{ff}})$ decreases. • MB-VG: Here, the feedforward torques are computed based on the analytical model. We also decrease the feedback gains for this method, so as to enable a fair comparison between the model-based approach and our approach (GPR-VG), for the same gains. • 0-VG: Pure PD control without feedforward torques, again with decreasing gains over time. This is a baseline, rather than state-of-the-art.

A. Experiment 1: Learning from ground-truth data

In this experiment, we generate a training database of 100 trajectories with known feedforward torques $\mathbf{u}_{\text{ff}}$, as well as 10 test trajectories. The start/end positions of the trajectories are uniformly sampled in a range 45° around the joint angles of the robot when each joint is centered between its joint limits. Minimum-jerk trajectories in joint space are used to go from start to end position. The duration of the trajectories varies between 5 and 10s, and training data is sampled at 10Hz. In simulation, model-based computed torques $\mathbf{u}_{\text{ff}}$ are computed with a recursive Newton-Euler method; these torques are used as target variables in the training trajectories.

a) Results: The top graph in Fig. 4 shows how the error between the predicted and ground truth u_{ff} on the test trajectories decreases with an increasing number of training trajectories. This is thus a standard learning curve. The top plot in Fig. 6 show the same results for all 7 joints.

The bottom graph in Fig. 4 shows how the uncertainty in the model, i.e. the average $\text{std}(u_{\text{ff}})$ over the test trajectories, decreases over time. For GPR-VG (solid line), the average gains decrease along with $\text{std}(u_{\text{ff}})$ (according to the function depicted in Fig. 2), whereas for GPR (dashed line), the gains are always high. Please note that $\text{std}(u_{\text{ff}})$ is on the left y-axis, and the gains on the right y-axis (green).

The graphs in Fig. 5 show the effect of learning computed torque on the tracking error. Three variables are visualized: the average gains (x-axis), the tracking error (y-axis) and the number of training trajectories (numbers alongside the lines). Thus, we see how the four methods traverse the gain/tracking

[2] http://www.gaussianprocess.org/gpml/code/matlab/

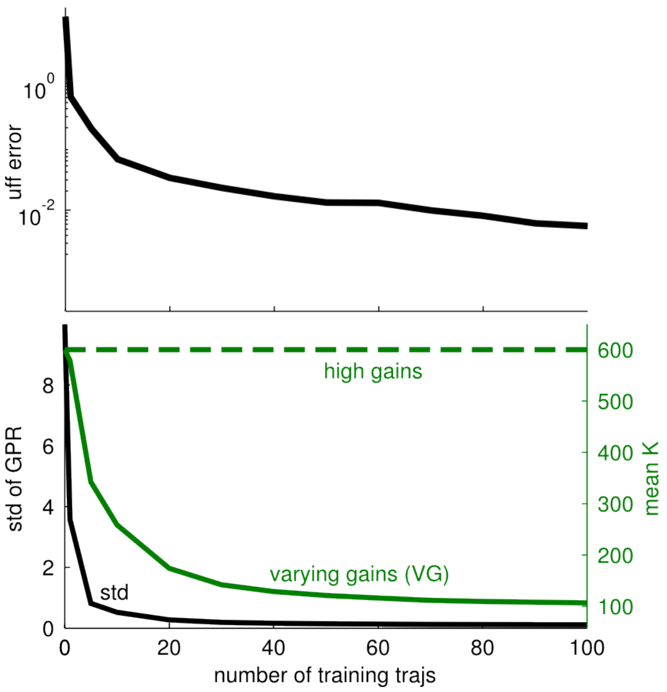

Fig. 4. Top: decreasing error in u_{ff} as the number of training trajectories increases. Bottom: decreasing model uncertainty, represented by $\text{std}(u_{\text{ff}})$. In high gain mode (GPR), the gains are independent of $\text{std}(u_{\text{ff}})$ (dashed line). With varying gains (GPR-VG), K decreases with decreasing $\text{std}(u_{\text{ff}})$. All values represent the average over all 10 test trajectories, and all time steps in a trajectory. Results are for the first joint (the shoulder extensor/flexor) only.

error space as more training data becomes available. The top graph depicts the joint tracking error (in radians) for the first joint, whereas bottom graph depicts the end-effector tracking error (in cm). Fig. 6 repeats these two graphs, but includes the results for for all 7 joints.

b) Discussion: From these results, we derive the following conclusions:

- Not using feedforward torques leads to bad tracking, which gets worse as we lower the gains (see 0-VG).
- We verify the feasability of learning computed torque with GPR (see GPR). At point Ⓐ in Fig. 5, the tracking performance of GPR is the same as that of the analytical model for the same gains.
- With a near-perfect analyical model, feasible in simulation, lowering the gains hardly affects the tracking error, as the dotted line labeled MB-VG is almost horizontal.
- With our approach (GPR-VG), we see that the joint tracking error decreases, even though we are lowering the gains, which is the main aim of our work. After 100 training trajectories, our approach is able to achieve almost the same tracking error as the analytical model, for the same low gains, see Ⓑ.

B. Experiment 2: Autonomous generation of training data

In Experiment 1, the training signal consists of torques computed with a model-based method. For real robots however, an accurate model of the robot dynamics is not always

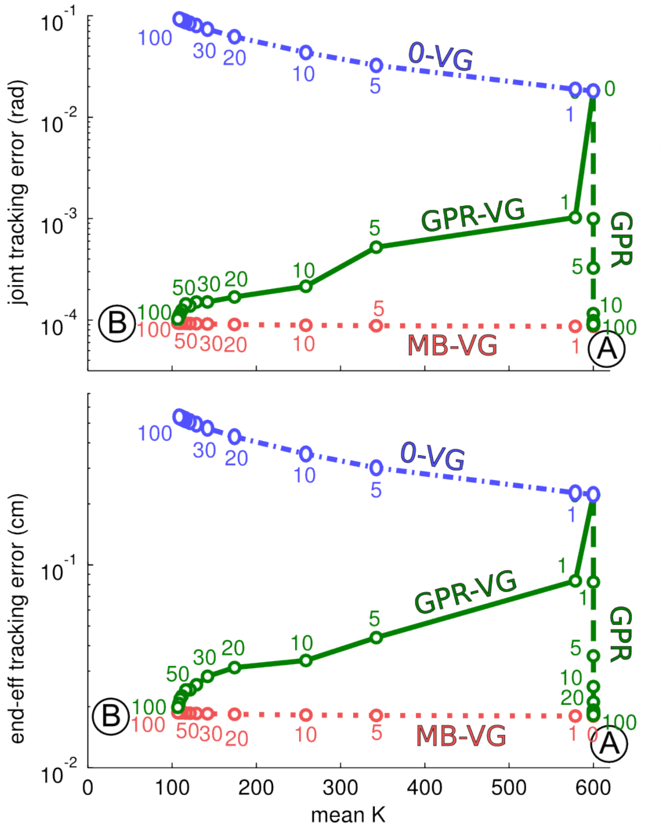

Fig. 5. Top: Joint tracking error (in radians) against gains. The numbers next to the lines represent the number of training trajectories used. Bottom: as above, but for the end-effector tracking error (in cm).

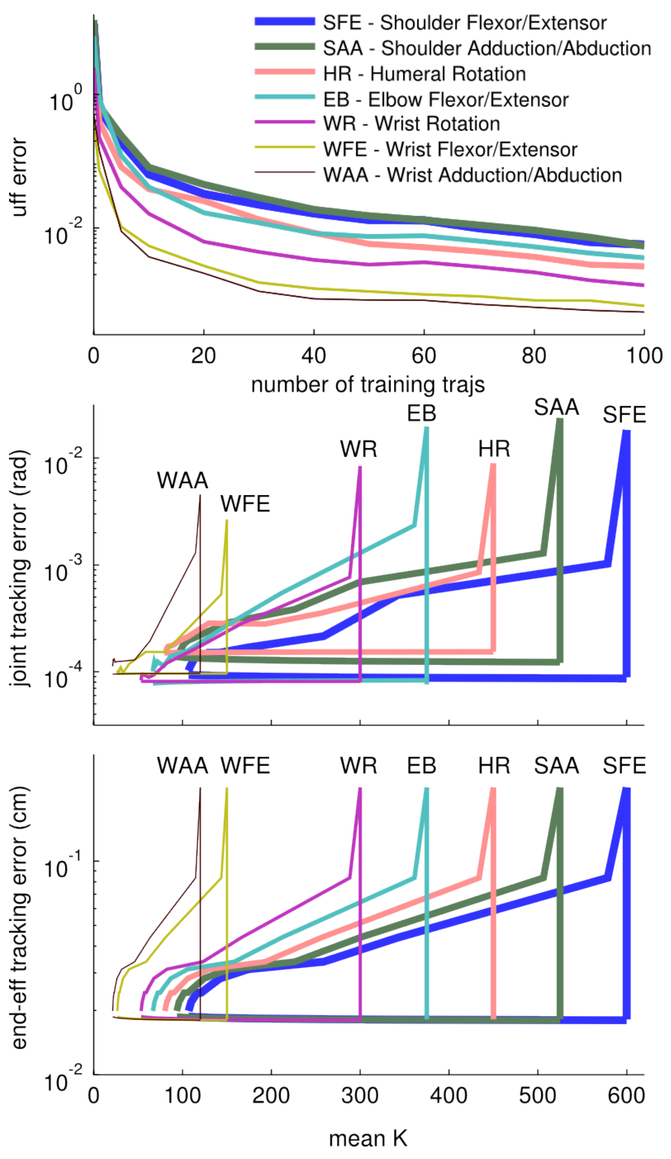

Fig. 6. Analogous to Fig. 4 and Fig. 5, but with results for all 7 joints. In the bottom two plots, annotation has been supressed to avoid clutter. The approximately triangular shape that can be observed for each joint corresponds to the GPR →MB-VG →GPR-VG triangle in Fig. 5. The 0-VG method is excluded, also to avoid clutter. Note that more proximal joints require larger torques, and thus higher gains.

available [4]. Thus, the robot should preferably generate its own training data.

We enable this by 1) not using the model-based feedforward torques 2) using the total torque $\mathbf{u}$ in (1) as a target, rather than only $\mathbf{u}_{\text{ff}}$. Initially, there are therefore no feedforward torques (the untrained zero-mean GPR returns 0), and $\mathbf{u}$ is determined entirely by the high-gain feedback term. As the GPR model is trained with $\mathbf{u}$, it will start generating increasingly accurate $\mathbf{u}_{\text{ff}}$, and the gains are lowered as in the previous experiment. This is a variation of iterative learning control (ILC) [3] that has also been used in [17].

The results of using $\mathbf{u}$ instead of $\mathbf{u}_{\text{ff}}$ are plotted in Fig. 7. Tracking errors are slightly higher than when using $\mathbf{u}_{\text{ff}}$. However, the advantage of this approach is that we do not need to know the ground truth torques, but can generate the training data autonomously from scratch.

VI. Conclusion

We have shown that in Gaussian process approaches to learning computed torque control, the variance in the estimate of the predicted torques may be used to automatically lower the gains. When the variance in $\mathbf{u}_{\text{ff}}$ is low, we know that we have a good estimate of $\mathbf{u}_{\text{ff}}$, and we can lower our feedback gains, because the feedforward term will do the “heavy lifting”. However, if the variance in $\mathbf{u}_{\text{ff}}$ is still high, it is the feedback term that will need to the work, and the gains are kept high.

This approach has the advantage that the gains can be lowered over time, whilst simultaneously decreasing tracking errors. This is true on both the trajectory level – gains may vary within a trajectory, depending on whether data has already been observed along parts of the trajectory (see Fig. 1) – and also on a developmental scale – overall, the gains decrease as more training data becomes available (see Fig. 6). We see this work as a first step towards life-long learning robots, that continuously and autonomously adapt their control parameters (feedforward *and* feedback) over extended periods of time.

Implementing our approach on a real robot will require two improvements. First of all, since we are using a non-optimized version of GPR, predictions are slow; too slow for

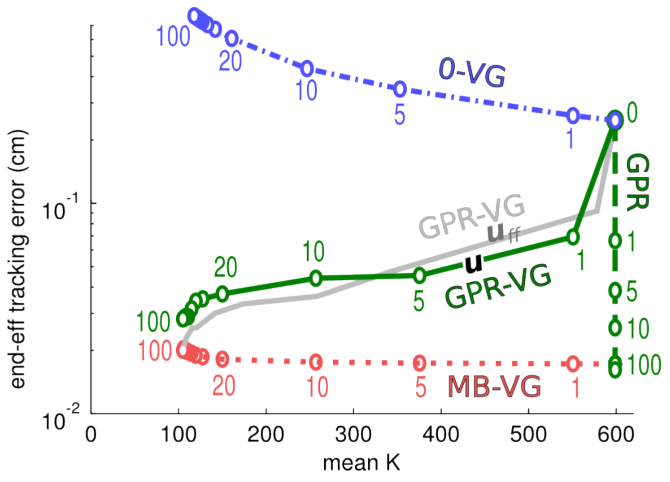

Fig. 7. Results of training the Gaussian process with $\mathbf{u}$ instead of $\mathbf{u}_{\text{ff}}$. End-effector tracking error (in cm) plotted against the gains. Numbers indicate number of training trajectories. For comparison, the result of training with $\mathbf{u}_{\text{ff}}$ – as in Experiment 1 – has been included (light gray) for our GPR-VG approach.

use on in a real-time system. However, this is merely a question of implementation, as our GPR implementation could be readily replaced with Local GPR [4] or Sparse GPR [5], whose query times have been shown to be fast enough for real-time requirements on robotic systems. Secondly, varying the gains may lead to unexpected instabilities. Therefore, the stability of the system must be further analyzed, especially in relation to the parameters K^{max}, K^{min} and C in (14). These two improvements are on our immediate research agenda.

References

[1] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics – Modelling, Planning and Control*. Springer, London, 2009.

[2] C. G. Atkeson, C. H. An, and J. M. Hollerbach, “Estimation of inertial parameters of manipulator loads and links,” *Int. J. Rob. Res.*, vol. 5, no. 3, pp. 101–119, Sept. 1986.

[3] S. Arimoto, S. Kawamura, and F. Miyazaki, “Bettering operation of Robots by learning,” *Journal of Robotic Systems*, vol. 1, no. 2, pp. 123–140, 1984.

[4] D. Nguyen-Tuong, M. Seeger, and J. Peters, “Model learning with local gaussian process regression,” *Advanced Robotics*, no. 15, pp. 2015–2034, 2009.

[5] J. S. de la Cruz, W. Owen, and D. Kulic, “Online learning of inverse dynamics via gaussian process regression,” in *IROS*, 2012.

[6] K. M. A. Chai, C. K. I. Williams, S. Klanke, and S. Vijayakumar, “Multi-task gaussian process learning of robot inverse dynamics,” in *NIPS*, 2008, pp. 265–272.

[7] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[8] J.-J. E. Slotine and W. Li, “On the adaptive control of robot manipulators,” *The International Journal of Robotics Research*, vol. 6, no. 3, pp. 49–59, 1987.

[9] S. Vijayakumar, A. D’Souza, and S. Schaal, “Incremental online learning in high dimensions,” *Neural Computation*, vol. 17, pp. 2602–2634, 2005.

[10] D.-Y. Yeung and Y. Zhang, *In The Path to Autonomous Robots*. Springer, 2009, ch. Learning Inverse Dynamics by Gaussian Process Regression under the Multi-Task Learning Framework, pp. 131–142.

[11] D. A. Bristow, M. Tharayil, and A. G. Alleyne, “A survey of iterative learning control,” *Control Systems, IEEE*, vol. 26, no. 3, pp. 96–114, 2006.

[12] G. Ganesh, A. Albu-Schaffer, M. Haruno, M. Kawato, and E. Burdet, “Biomimetic motor behavior for simultaneous adaptation of force, impedance and trajectory in interaction tasks,” in *ICRA, 2010*. IEEE, 2010, pp. 2705–2711.

[13] M. Ebden, “Gaussian processes for regression: A quick introduction,” 2008, `http://www.robots.ox.ac.uk/mebden/reports/GPtutorial.pdf`.

[14] J. Buchli, F. Stulp, E. Theodorou, and S. Schaal, “Learning variable impedance control,” *International Journal of Robotics Research*, vol. 30, no. 7, pp. 820–833, 2011.

[15] S. Vijayakumar and S. Schaal, “Locally weighted projection regression: An o(n) algorithm for incremental real time learning in high dimensional spaces,” in *Proceedings of the 17th International Conference on Machine Learning (ICML)*, 2000, pp. 288–293.

[16] S. Schaal, “The SL simulation and real-time control software package,” University of Southern California, Tech. Rep., 2007.

[17] T. Petric, A. Gams, L. Zlajpah and A. Ude, “Online learning of task-specific dynamics for periodic tasks”, *International Conference on Intelligent Robots and Systems (IROS)*, 2014

Appendix

A. Determining signal variance

Because σ^f is a hyperparameter of the covariance function, it can be automatically tuned through hyperparameter optimization on a training set [4], [5], [7]. However, since we know that σ^f represents the standard deviation of our data, we can directly compute σ^f from a training set with N examples with the formula for the standard deviation

$$\sigma^f = \sqrt{\frac{1}{N}\sum_{t=1}^{N}(u_{\text{ff,t}} - E[\mathbf{u}_{\text{ff}}])^2}. \tag{16}$$

However, knowing that we will use a zero mean GPR for improved stability, σ^f should rather be computed *as if* the data has zero mean, i.e.

$$\sigma^f = \sqrt{\frac{1}{N}\sum_{t=1}^{N}u_{\text{ff,t}}^2}. \tag{17}$$

If no data is available, one may also use the bounds on the robot’s torques $u_{\text{ff}}^{\text{max}}$ (which we hope your robot has) as a heuristic. As u_{ff} will not exceed $u_{\text{ff}}^{\text{max}}$, we may choose $\sigma^f = u_{\text{ff}}^{\text{max}}/3$ expressing that we expect only 0.1% of observed u_{ff} it to be larger than $u_{\text{ff}}^{\text{max}}$ (whereas it will in fact be 0%, due to the bounds).

1) Experiment 3: We use the benchmark dataset from the real SARCOS master arm robot – used in [4], [7], [15] – to verify that Equation (17) corresponds to the value acquired through hyperparameter tuning. The results for each of the seven joints of an 7DOF articulated robot on the benchmark database are listed in Table I. This table confirms that (17) yields approximately the same value.

Joint #	1	2	3	4	5	6	7
optimization	26.0	23.4	19.3	29.1	1.7	2.6	5.7
equation (17)	23.5	27.7	13.8	30.3	1.3	1.8	4.8

TABLE I
Comparison of signal variance (σ_f) estimates.