Real-Time Inverse Dynamics Learning for Musculoskeletal Robots based on Echo State Gaussian Process Regression

Christoph Hartmann*†, Joschka Boedecker†, Oliver Obst‡§, Shuhei Ikemoto¶ and Minoru Asada†
*Institute of Cognitive Science, University of Osnabrueck, 49069 Osnabrueck, Germany
Email: chrhartm@uos.de
†Graduate School of Engineering, Osaka University, Suita 565-0871, Osaka, Japan
‡ CSIRO ICT Centre, PO Box 76, Epping NSW 1710, Australia
§ School of Information Technologies, The University of Sydney, NSW 2006, Australia
¶ Graduate School of Information Science and Technology, Osaka University, Suita 565-0871, Osaka, Japan

***Abstract*—A challenging topic in articulated robots is the control of redundantly many degrees of freedom with artificial muscles. Actuation with these devices is difficult to solve because of nonlinearities, delays and unknown parameters such as friction. Machine learning methods can be used to learn control of these systems, but are faced with the additional problem that the size of the search space prohibits full exploration in reasonable time. We propose a novel method that is able to learn control of redundant robot arms with artificial muscles online from scratch using only the position of the end effector, without using any joint positions, accelerations or an analytical model of the system or the environment. To learn in real time, we use the so called online "goal babbling" method to effectively reduce the search space, a recurrent neural network to represent the state of the robot arm, and novel online Gaussian processes for regression. With our approach, we achieve good performance on trajectory tracking tasks for the end effector of two very challenging systems: a simulated 6 DOF redundant arm with artificial muscles, and a 7 DOF robot arm with McKibben pneumatic artificial muscles. We also show that the combination of techniques we propose results in significantly improved performance over using the individual techniques alone.**

I. Introduction

For articulated robots, pneumatic actuators have grown in popularity and found increasing use in biorobotics, medical, industrial and aerospace applications [1]. Pneumatic artificial muscles (PAMs), such as the McKibben artificial muscle [16], are favourable for developing biologically inspired robots since they are backdriveable, have high flexibility, and a high power-to-weight ratio, similar to biological muscles. As a result, PAMs have often been used to drive the structural parts of complex anthropomorphic musculoskeletal robots. On the other hand, PAMs generally exhibit very strong nonlinearity, and their response is often delayed. For individual PAMs and single-DOF systems, there have been several proposals for analytical models [6, 12, 7]. When these are combined for use in a multi-DOF robot, it is, however, extremely hard to model the dynamics analytically for accurate motion control. In fact, related studies, using well-established methods from control theory, had to resort to limiting the number of DOF of the systems being modeled, and/or address the the problem only in the context of static tasks like position control [3, 21, 2].

Machine learning techniques have played an important role in modeling the complex dynamics of a manipulator driven by PAMs in [10, 8]. In these studies, feedforward neural networks were used to model the complex inverse dynamics of the manipulator, and feedback error learning [11, 13] was employed to train the networks. However, as the structure of PAM-driven robots gets more complex, expressing its dynamics in the form of a static mapping is becoming difficult. Additionally, in order to employ feedback error learning to realize trajectory tracking of an end effector, an initial feedback controller has to be provided, requiring an inverse kinematics model in advance. Here, we aim for a model-free approach to the control problem. Other relevant methods for learning dynamics of high-DOF systems in real time include for example "locally weighted learning" [20].

In order to deal with the increasing difficulty of learning the inverse dynamics of musculoskeletal robots driven by PAMs, in this work, we combine a variant of goal babbling [19] with an echo state network (ESN) [9], a particular kind of recurrent neural network, to represent the state of the system. Gaussian process regression (GPR) is used to train the network in real-time (see Sects. II-A, II-B, and II-C for more details on goal babbling, ESNs, and GPR, respectively).

ESNs [9] have recently become more prominent in the field of robotics, e.g., for learning the inverse kinematics of an industrial robot [18]. However, to the best of our knowledge, ESNs have not been successfully applied to musculoskeletal robots and inverse dynamics yet, despite their known ability to provide highly nonlinear mappings, and a fading memory of past inputs. We employ these features of ESNs to represent the state of a redundant robot arm with artificial muscles, and thereby implicitly compute information such as velocity or acceleration relevant for inverse dynamics tasks.

Gaussian process regression has been applied in previous work in [4] as well as [14, 15] to learn robust inverse dynamics on industrial robot arms. The latter authors succeeded to

use GPR in a real-time setting, and demonstrated that this method outperforms other state of the art learning methods for inverse dynamics. A notable difference of these approaches to our work is that GPR was only used on industrial robot arms and did not have to face the challenges associated with musculoskeletal robots like elastic deformation of the actuator and long dead-times in the control. Additionally, features such as joint angles, velocities, and accelerations were directly available for the learning algorithm, which is not the case for our system.

We briefly introduce the individual methods that our approach builds on, as well as the learning task in Sect. II. After reviewing basic GPR, we will present a modification for fast online GPR in Sect. III. In Sect. IV, we explain how the individual methods are combined in our approach, and how they can be applied to both a simulated 6 DOF redundant arm with 12 artificial muscles, and to a 7 DOF robot arm driven by 17 McKibben pneumatic artificial muscles. Results are presented in Sect. V. Finally, in Sect. VI, we briefly summarize and discuss our results, and give an outlook over possible future directions of our work.

II. Basic Methods

A. Goal Babbling

Goal babbling is based on work by Rolf et al. [19], where the comparison is made to infants who tend to make goal-directed movements from at a very early age even if they do not succeed. Accordingly only a smaller relevant subspace of possible motor commands is explored, as opposed to random motor babbling. By directly learning from the perturbed goal directed movements, this bootstraps and increases the speed of learning.

In [19], the inverse kinematics is learned in 2D space by online construction of local linear regressions over positions that are weighted by prototype vectors. A similar local weighting is discussed in more detail in Sect. IV. Goal babbling is responsible for generating the training samples: At every step, the inverse kinematics for a point between the last goal and the next goal is computed according to the current weight matrix and then perturbed to facilitate exploration. Stability in learning is ensured by always returning to a "home posture" for which the inverse kinematics are known. Continuing the comparison to infants, this corresponds to the child relaxing its muscles and resting.

In order to transfer these ideas to inverse dynamics learning, a new state vector has to be generated that captures the dynamics of the robot arm. This is done using an echo state network as described below.

B. Echo State Networks

ESNs [9] have been introduced as an alternative to more traditional recurrent neural network (RNN) approaches. Two major differences that enable ESNs to overcome problems of some earlier algorithms for RNN training, such as slow convergence and instabilities, are:

- to initialize the network weights with random values and only adapt the weights from the hidden layer to the output units (output or *readout* weights);
- to scale hidden layer weights so that the network has the *echo state property*, i.e., it implements a fading memory of past inputs (see [9] for more details).

A random input-matrix $\mathbf{W}_{in}$ combines input values $\mathbf{u}$ linearly and sends them to the units in the high-dimensional hidden layer, also referred to as the *reservoir*. The units in the reservoir also have recurrent connections amongst each other, collected in the matrix $\mathbf{W}_{res}$. Through these loops, information can remain in the system for some time. In this context, the metaphor of a reservoir is often used since the hidden layer can be seen as a water reservoir that gets disturbed by a drop, but slowly returns to its initial state after the ripples from the input have decayed. This reservoir state $\mathbf{r}$ is mapped at timestep $t+1$ by an activation function $f()$ such as a hyperbolic tangent or a Fermi function, in the following way:

$$\mathbf{r}_{t+1} = f(\mathbf{W}_{res} * \mathbf{r}_t + \mathbf{W}_{in} * \mathbf{u}_{t+1}) \tag{1}$$

This mapping has several consequences. First, it projects the input in a high-dimensional space so that regression gets easier as it is the case for several kernel methods. Second, the repeated mapping by the activation function of neurons in the reservoir leads to many nonlinearities that, together with the fading memory, provide an implicit representation of nonlinear input properties such as the velocity or acceleration of a robot arm. Third, the input decays slowly. This way, ESNs are a natural choice to represent delayed and nonlinear signals. Usually, the output weights are trained with some form of linear regression over the reservoir states. In our case, GPR is used to train the readout instead.

C. Gaussian Process Regression

As Chatzis and Demiris [5] demonstrated, GPR is a more powerful and robust tool for regression on ESNs than the linear regression or ridge regression that has been employed so far. Moreover, it seems to be a good choice for learning control with echo state networks, because Nguyen-Tuong and Peters [14] showed that this works very well with regular inverse dynamics learning and that it can be adjusted to learn in real-time.

The idea behind GPR is to perform a parameter-free regression over the the space of possible functions that gave rise to the training data. In order to do so, a kernel function defines the correlations or – more intuitively – distances between training points. When a test point is then evaluated, its function value should be close to the function values of the training points that covary most with the test point. The hyperparameters of this kernel function represent a *prior* distribution over possible functions. Conceptually, these hyperparameters correspond to different activation functions and connections of neural networks while parameters would be the actual connection weights. When data points are added and correlations are evaluated, this *prior* is transformed to a *posterior* distribution which can be used to predict new targets.

A Gaussian process is defined as a distribution over functions $f(\mathbf{x})$ with the constraint that any subset of evaluated data points is always jointly Gaussian distributed. With a given kernel function $k(\mathbf{x}, \mathbf{x}')$ and the unproblematic assumption of a zero mean, the distribution over functions can therefore be written as

$$p(\mathbf{f}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{C}) \tag{2}$$

with $C_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j) + \delta_{i,j}\sigma$ where σ stands for the noise in the observed target values, and δ is the Kronecker delta. Given this distribution, a prediction for a new test point $\mathbf{x}_{N+1}$, that is the mean function value $m(\mathbf{x}_{N+1})$ given all previous N targets $\mathbf{t}$ and the correlations $\mathbf{k}$ between the previous N training points and the new one, can be calculated as follows:

$$m(\mathbf{x}_{N+1}) = \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{t} = \mathbf{k}^T \alpha \tag{3}$$

with $\alpha = \mathbf{C}_N^{-1}\mathbf{t}$ as the later precomputed prediction vector.

We have to address two issues to learn with GPRs: A kernel has to be selected and associated hyperparameters have to be learned. We will use both the exponential kernel

$$k(\mathbf{x}, \mathbf{x}') = \theta \exp\left(-0.5 \sum_{i=1}^{D} \eta_i (x_i - x_i')^2\right) \tag{4}$$

because it is known for its good predictions and robustness, as well as the linear kernel

$$k(\mathbf{x}, \mathbf{x}') = \theta \mathbf{x}^T \mathbf{x}. \tag{5}$$

It was shown in [5] that this kernel resembles ridge regression. Using that, we can easily compare the performance difference between GPR and ridge regression. The hyperparameters σ, θ and η_i are optimized by applying conjugate gradient descent to the log-likelihood of the data fit given the hyperparameters. The best way to think of η_i is to see it as a relevance measure for the i^{th} dimension in the input data since if it is high, the corresponding data x_i has a high influence on the covariance, while if it is low the covariance is not affected. While η_i scales the resulting functions horizontally, θ can be seen as a vertical scale representing the standard deviation of the GP.

The runtime cost for making a prediction is dominated by the inversion of the covariance matrix $\mathbf{C}$ which is $O(N^3)$. Obviously, this is not practical for online learning if a reasonable amount of data points is assumed. We propose a faster method to approach this problem.

III. Online Gaussian Processes

To deal with the expensive matrix inversion at every step, Nguyen-Tuong and Peters [14] suggested to use many small local GPRs to keep N reasonably small and to linearly combine the predictions of the closest GPs. This approach reduces the runtime significantly, but it still grows cubic. In [15], the same authors further reduce the runtime to $O(N^2)$ using a Cholesky factorization. We achieve a similar speedup with our GPR modifications, presented below, using block-inversions.

We take advantage of the incremental manner of online learning by exploiting the fact that a single data point is added to one covariance matrix at every step. For a matrix with sub-blocks $\mathbf{A}, \mathbf{B}, \mathbf{D}$, and $\mathbf{E}$, the block inversion formula states:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{D} & \mathbf{E} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}\mathbf{S}^{-1}\mathbf{D}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}\mathbf{S}^{-1} \\ -\mathbf{S}^{-1}\mathbf{D}\mathbf{A}^{-1} & \mathbf{S}^{-1} \end{bmatrix}, \tag{6}$$

where $\mathbf{S}$ is the Schur complement of $\mathbf{A}$, i.e., $\mathbf{S} = (\mathbf{E} - \mathbf{D}\mathbf{A}^{-1}\mathbf{B})$. In the case of an incrementally growing covariance matrix $\mathbf{C}$, it holds that $\mathbf{A} = \mathbf{C}$, $\mathbf{B} = \mathbf{k}(\mathbf{x}_n, \mathbf{x}_{N+1})$, $\mathbf{D} = \mathbf{k}(\mathbf{x}_{N+1}, \mathbf{x}_n)^T \ \forall n \leq N$ and $\mathbf{E} = k(\mathbf{x}_{N+1}, \mathbf{x}_{N+1}) + \sigma$. Because $\mathbf{A}^{-1} = \mathbf{C}^{-1}$ was already computed in the previous step and the Schur complement $\mathbf{S}$ is a single number, no matrix inversion is left and the runtime reduces to $O(N^2)$ for the matrix multiplications in every step. In order to make use of this incremental inversion, the hyperparameters have to be optimized beforehand and therefore represent a true *prior*.

IV. The Combined Model

The combination of the methods described above is explained in the following paragraphs. Actual parameter settings can be found in Table I.

In order to formally describe the algorithm implementing the combined model, we define useful terms for online Gaussian processes with a fixed delay of *delay* timesteps:

- G_t is the set of all local GPs at time point t.
- $c_t^g = \langle D_t^g \rangle$ is defined as the center of $g_t \in G_t$.
- closeness$(\mathbf{x}, g_t)$ is adapted from [14] as $k(\mathbf{x}, c_t^g)$ which takes a high value for small distances.
- $D_t^g = \{\mathbf{x}_i | (\text{delay} < i < t) \wedge$ $(\forall h \in G_i : \text{closeness}(\mathbf{x}_i, g_i) \geq \text{closeness}(\mathbf{x}_i, h_i))\}$ is the set of all data points belonging to a local GP.
- $T_t^g = \{y_i | (0 < i < t - \text{delay}) \wedge (\mathbf{x}_{i+\text{delay}} \in g_{i+\text{delay}})\}$ is the set of all target points belonging to a local GP. This is used as $\mathbf{t}$ in formula 3.

The general flow of information goes as follows: The sensor information (length or pressure) from a set of muscles is used to update an ESN every $esntime$ milliseconds. Parallel to this, a growing group of local GPs, G, is used to map the reservoir state $\mathbf{r}$ of the ESN and information about the current and desired position of the arm to commands (Hill-parameters or pressures) for muscle activation.

A. Predicting and Learning

The following paragraphs are a detailed description of the algorithms for predicting and learning for one muscle (Algorithms 1 and 2, respectively). Information flow in these is also visualized in Fig. 1.

The input to the system is provided by the current reservoir state $\mathbf{r_t}$ and by the position $\mathbf{pos}_t$ of the robot arm end-effector. Depending on this position a random new goal point $\mathbf{goal}_t$ is selected from a larger version of the later trajectory if either the distance between the goal and the current position is smaller than a threshold (the reaching movement was successful), or if $\mathbf{goal}_{t-1}$ is older than 20 steps.

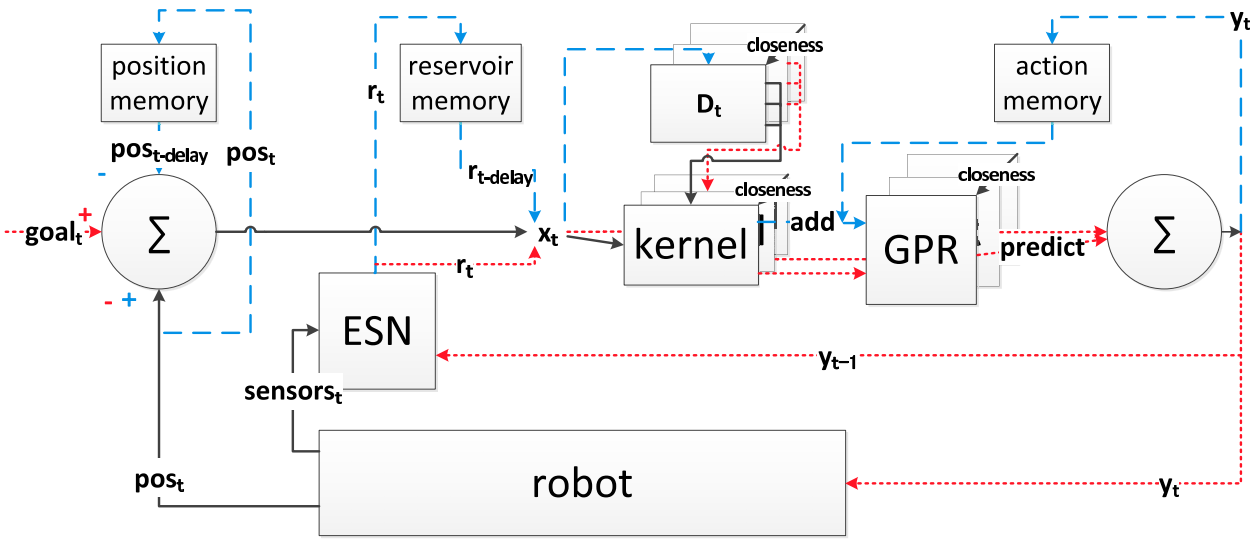

Fig. 1. The flow of information in the combined model. Dotted red arrows correspond to information flow during action selection whereas dashed blue arrows are only relevant for training the GPs. The subdevided rectangles present in the upper half of the figure are memory-queues that save informations between time t and $t-$ delay. The GPR-box subsumes the set of targets, the matrix inversion and the prediction of the mean.

Next, a state vector is generated by concatenating the reservoir state $\mathbf{r}_t$ with the difference $\mathbf{d}_t^{\text{pred}} = \mathbf{goal}_t - \mathbf{pos}_t$ to the state $\mathbf{x}_t^{\text{pred}}$. To predict the next action y_t, the covariance between $\mathbf{x}_t^{\text{pred}}$ and the observed states D_t of the closest num_{pred} GP is then computed and used in formula 3 to determine the prediction of each local GPR for the muscle. Our closeness measure is defined at the beginning of this section. These local approximations are weighted by the closeness of each GP to $\mathbf{x}_t^{\text{pred}}$. The final prediction y_t is then perturbed by uniform noise to resemble the exploration in goal babbling. In our experiments we found that the properties of noise highly determine the final outcome. Smoother trajectories were obtained by applying the same noise for several steps and to only a subset of the muscles at the same time.

After sending the action, the actual learning (Alg. 2) takes place. As in the prediction phase, $\mathbf{x}_t^{\text{learn}}$ is gained by combining a reservoir state with a position difference. However, for learning $\mathbf{r}_{t-\text{delay}}$ and $\mathbf{d}_t^{\text{learn}} = \mathbf{pos}_t - \mathbf{pos}_{t-\text{delay}}$ are concatenated. The pair $(x_t^{\text{learn}}, y_{t-\text{delay}})$ is then used as an observation-target pair for training the closest GPR if the closeness is larger than *mincloseness*. By doing so, the position difference after *delay* steps is associated to the action and reservoir state at that time. When a new action is predicted, the desired effects should therefore be visible after *delay* steps. We are using the position difference instead of direct position information since this makes the algorithm a little more independent from the arm it is applied to. If the closest GPR is further away than *mincloseness*, a new local GPR is added and used for learning this state-action pair.

After training, the algorithm waits until the rest of *steptime* milliseconds has passed and starts again with new information.

B. Learning Stages

The actual learning is subdivided into three stages:

Algorithm 1 Predicting in the combined model for a single muscle at step t. Symbols explained at the end of Sect. III. In case of trajectory tracking, a new goal is selected each step.

Input : $\mathbf{r}_t, \mathbf{pos}_t$
if $(\mathbf{goal}_{t-1} == \mathbf{goal}_{t-20}) || (\|\mathbf{goal}_{t-1} - \mathbf{pos}_t\| \leq const)$ **then**
  $\mathbf{goal}_t \leftarrow \mathbf{goal\_set}[\text{rand}()]$
else
  $\mathbf{goal}_t \leftarrow \mathbf{goal}_{t-1}$
$\mathbf{x}_t^{\text{pred}} \leftarrow [\mathbf{r}_t, \mathbf{goal}_t - \mathbf{pos}_t]$
$\mathbf{cls} \leftarrow \text{Array}(|G_t|), \mathbf{cls\_index} \leftarrow \text{Array}(|G_t|)$
for $i = 1 \rightarrow |G_t|$ **do**
  $\mathbf{cls}[i] \leftarrow \text{closeness}(\mathbf{x}_i^{\text{pred}}, G_t[i]), \mathbf{cls\_index}[i] = i$
sort $\mathbf{cls\_index}$ according to $\mathbf{cls}$ in descending order
$\mathbf{pred} \leftarrow \text{Array}(\text{num}_{\text{pred}}), \text{denom} = 0$
for $i = 1 \rightarrow \text{num}_{\text{pred}}$ **do**
  $j \leftarrow \mathbf{cls\_index}(i)$
  $g \leftarrow G_t[j]$
  $\mathbf{covar} \leftarrow$ covariances between D_t^g and $\mathbf{x}_t^{\text{pred}}$
  $\mathbf{pred}[i] \leftarrow \mathbf{covar}^T * \alpha$ (formula 3)
  $\mathbf{pred}[i] \leftarrow \mathbf{pred}[i] * \mathbf{cls}[j], \text{denom} \leftarrow \text{denom} + \mathbf{cls}[j]$
$y_t = \text{sum}(\mathbf{pred})/\text{denom} + \text{noise}$
Send : y_t

1) Hyperparameter optimization
2) Goal-directed learning
3) Trajectory tracking

Before both the hyperparameter optimization and the learning, a short phase of random activation is performed to "warm up" the echo state network.

The first stage makes use of random muscle activation to mimic the later activations and thereby explore sensorimo-

Algorithm 2 Learning in the combined model for a single muscle at step t. Symbols explained at the end of Sect. III.

$\mathbf{Input}: \mathbf{r}_t, \mathbf{pos}_t$
$\mathbf{x}_t^{\text{learn}} \leftarrow [\mathbf{r}_{t-\text{delay}}, \mathbf{pos}_t - \mathbf{pos}_{t-\text{delay}}]$
for $i = 1 \rightarrow |G_t|$ **do**
  $\mathbf{cls}[i] \leftarrow \text{closeness}(\mathbf{x}_i^{\text{learn}}, G_t[i])$
$(\text{cls\_max}, \text{max\_index}) \leftarrow \max(\mathbf{cls})$
if cls_max $>$ max_closeness **then**
  $g \leftarrow G_t[\text{max\_index}]$
  $\mathbf{covar} \leftarrow$ covariances between D_t^g and $\mathbf{x}_t^{\text{learn}}$
  covar_self $\leftarrow k(\mathbf{x}_t^{\text{learn}}, \mathbf{x}_t^{\text{learn}}) + \sigma$
  resize covariance matrix $\mathbf{C}_t^g$ to $|\mathbf{C}_t^g| + 1$
  add covar_self and $\mathbf{covar}$ to $\mathbf{C}_t^g$
  update $(\mathbf{C}_t^g)^{-1}$ with formula 6 using $\mathbf{B}, \mathbf{D} = \mathbf{covar}$ and $\mathbf{E} =$ covar_self
  $\alpha \leftarrow (\mathbf{C}_t^g)^{-1} * T_t^g$ (3)
else
  $g \leftarrow \mathbf{new}\ GP$ with $\mathbf{C}_t^g = k(\mathbf{x}_t^{\text{learn}}, \mathbf{x}_t^{\text{learn}}) + \delta_{i,j}\sigma$
Add $y_{t-\text{delay}}$ to T_{t+1}^g and $\mathbf{x}_t^{\text{learn}}$ to D_{t+1}^g
$\mathbf{Return}: g$

tor contingencies. A number of $optobs$ training samples are recorded and then used to optimize the hyperparameters for the kernel of the GPR. This process is also called automatic relevance determination (ARD) [17].

In the learning phase, the algorithm explained above is used to learn actions in the relevant subspace for the later tracking. Also, every 500 steps the muscle activations are set to 0 for several steps to reach a "home posture" and stabilize the learning process as described in Sect. II-A.

For this architecture, the tracking of a trajectory reduces to changing the goal to reach at every step to the next goal on the trajectory. The speed can be adjusted by increasing or decreasing the distance between two succeeding goals. Because of the fixed and identical *steptime* in learning and tracking, the muscle activation will then be adjusted to reach the required goal faster or slower.

C. Notes

To limit complexity and thereby decrease the runtime, we set the maximal size of one GPR to *maxgprobs* observations. In the case that the closest GPR has reached *maxgprobs*, the information of a previous observation in the inverse covariance matrix $\mathbf{C}^{-1}$ can be substituted with the new one according to the Sherman-Morrison formula in $O(N^2)$. In our trials with *steptime* = 60 ms, this substitution is feasible up to a matrix size of 1100 rows while the blockwise matrix inversion needs the same time for 1500 rows. Therefore this method is only used if *maxgprobs* is small enough.

Because the standard deviation between the optimized hyperparameters for each muscle in the simulation and on the real arm is about ten times smaller than the standard deviation in the hyperparameters itself, the same hyperparameters are used for all muscles. This leads to identical covariance matrices $\mathbf{C}$ and cuts the runtime by the number of muscles, because at every step only one covariance matrix has to be inverted.

Parameter	Sim1	Sim2	Real	Comments
learn steps	16000	30000	20000	$\approx$ max gprs $*$ gpr obs[a]
step time	60	60	100	60 ms $\rightarrow$ 1500 $\frac{\text{obs}}{\text{GPR}}$ [b]
esn time	30	30	50	should be step time /2
esn units	200	180	200	improvement stops at 200
gpr obs	800	1500	1100	improves fit
opt obs	800	600	400	many $\rightarrow$ var, few $\rightarrow$ bias
max gprs	5	25	10	maximal number of GPs
num_{pred}	3	3	3	many $\rightarrow$ bias, few $\rightarrow$ var
delay	1	2	6	multiply with step time
min close.	0.7	0.3	0.35	inserts new GP
spect. rad.	0.7	0.6	0.7	ESN memory measure

[a] Sim1, Real: substituting observations possible in second half
[b] on a Thinkpad x220t (2.7Ghz, 4GB memroy)

TABLE I
PARAMETER VALUES USED FOR THE SHOWN TRAJECTORIES.

Also, please note that the position vectors of the simulated arms only consist of the 3D-position of a marker on the "hand". No metainformation such as positions, velocities or accelerations of joints are given, since information about these are encoded in the memory of the ESN reservoir. In fact, correlation analysis shows that although no joint information is given to the ESN, in every simulation there are always reservoir units that are correlated with the joint angles with $\rho \approx 0.9$, with the joint velocities with $\rho \approx 0.3$ and with the joint accelarations with $\rho \approx 0.2$. This enables us to use these features for learning on the real arm, even though it does not have any joint sensors.

V. RESULTS

The described model has been successfully applied to endpoint effector tracking on simulated 2 and 6 DOF robot arms with artificial muscles as well as a real musculoskeletal arm with 7 degrees of freedom. The parameters used for the trajectories (shown in Fig. 3) can be found in Table I. The source code for the developed model and for the simulation will be made available on: www.christophartmann.de

A. Simulated Arm

The simulation has two main purposes: First, it should give a proof of concept that the developed model is providing reasonable results. Second, it can be used to investigate the influence of the different methods employed by running many different trials without wearing off the real robot arm.

The arm used for the proof of concept is quite simple. It is made of two solid capped cylinders connected by a universal 2 DOF joint. These segments are connected by four muscles spaced $90°$ apart around the cylinders. They are individually controlled by setting a parameter x, which is then mapped to a force by the following nonlinear modified Hill-equation.

$$\text{force}(x) = (\frac{750}{x+1} - 375) * \text{muscle\_length} \qquad (7)$$

The force is applied at both fixpoints of a muscle in the direction towards the opposing point. The position information

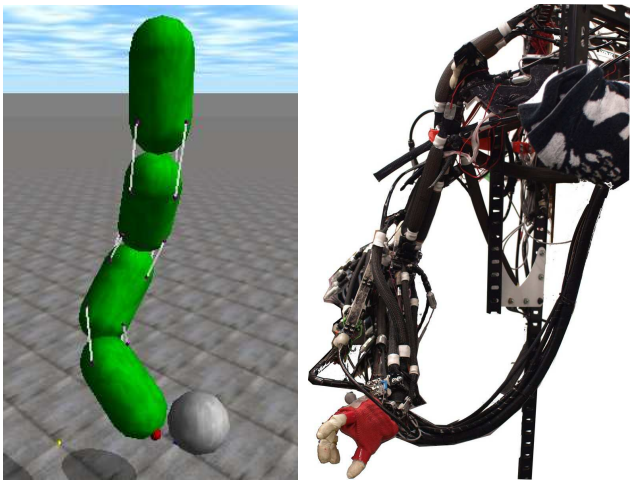

Fig. 2. Left: The simulated robot reaching for the goal visualized as a grey ball. Muscles are drawn in white and the center of the red point provides position information. Right: The musculoskeletal arm with a red glove for position tracking.

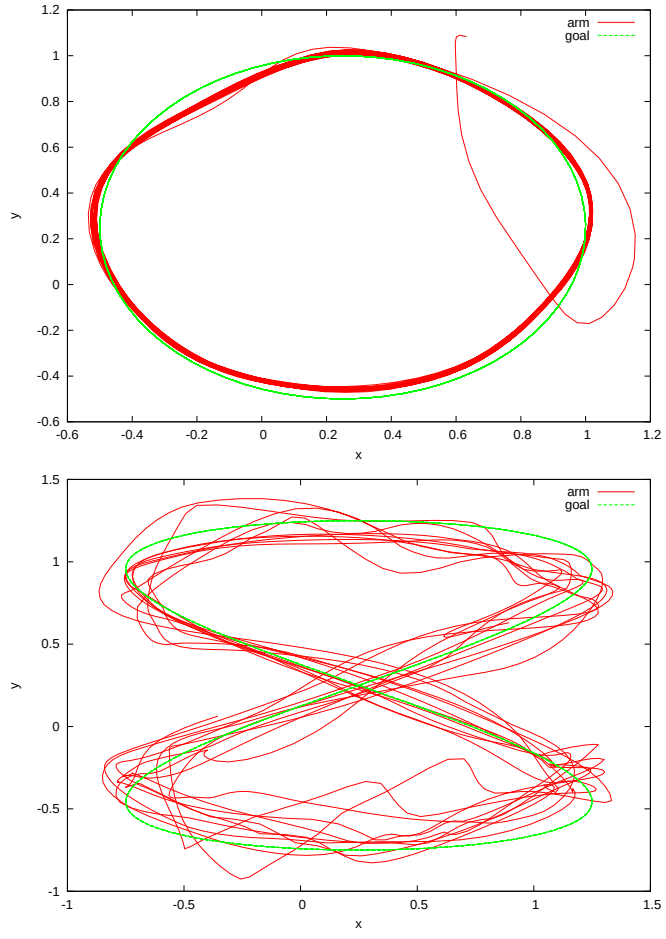

Fig. 3. 10 successive sample trajectories for the simple (top) and complex (bottom) simulated arm. In each case, a completed trajectory takes 9 s. Parameters are shown in Table I for "Sim1" and "Sim2". The respective RMSEs are 0.15 and 0.3.

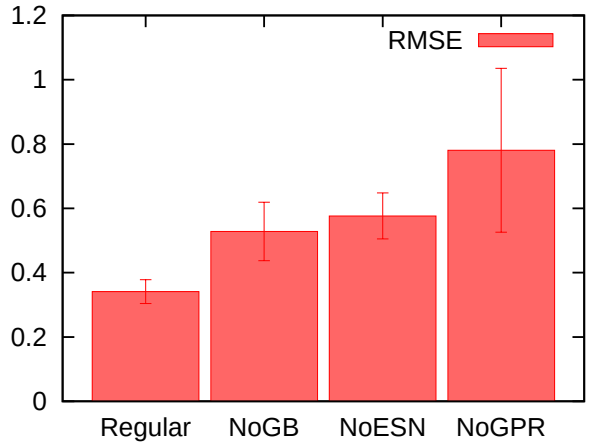

Fig. 4. The effect of not using one part of the combined model on the root mean squared error. The error bars represent the standard deviance across 10 independently trained models. All differences to the regular conditions have a significance > 0.99. The significance was computed with the two-sided Welch's t-test because the variance of the different samples cannot be assumed to be identical.

needed for the model comes from the position of the red marker at the end of the arm. The length of each muscle serves as sensory information for the model. Results for tracking an ellipsoid are shown in Fig. 3. Since this shows that the tracking error is only small and that the model works, a next step is to apply the model to more difficult and redundant problems.

The arm used to explore the relative influence of the different methods is based, as depicted in Fig. 2, on the first arm but has two more universal joints resulting in a highly redundant arm with 12 muscles. The influence of each method as well as remaining parameters such as the hyperparameters for the ESN, the kind of initial random exploration or the maximal size of individual GPRs were explored by grid-search. This led to the results in Fig. 4 for trajectories of 10 independently trained models for each condition.

These observations demonstrate that the novel combination of methods used in this project is indeed better than the previous approaches that made use of only one of the methods.

B. Real Arm

The musculoskeletal robot arm used for our experiments has a 7 DOF skeletal structure, comparable to a human arm in both the number of DOF, as well as the configuration of bones and each joint's DOF: shoulder, elbow, forearm, and wrist are realized by using a 3 DOF, 1 DOF, 1 DOF and 2 DOF joint, respectively. In order to move this structure, 17 PAMs are used and configured schematically similar to a human's arm. As a consequence, there are also PAMs which drive multiple joints at the same time. Each PAM has a pressure sensor and a tension sensor and is controlled by pressure feedback control. Two cameras are mounted on the head of the robot arm to measure the position of the end effector.

The kinematics of this arm (and similar ones) is difficult

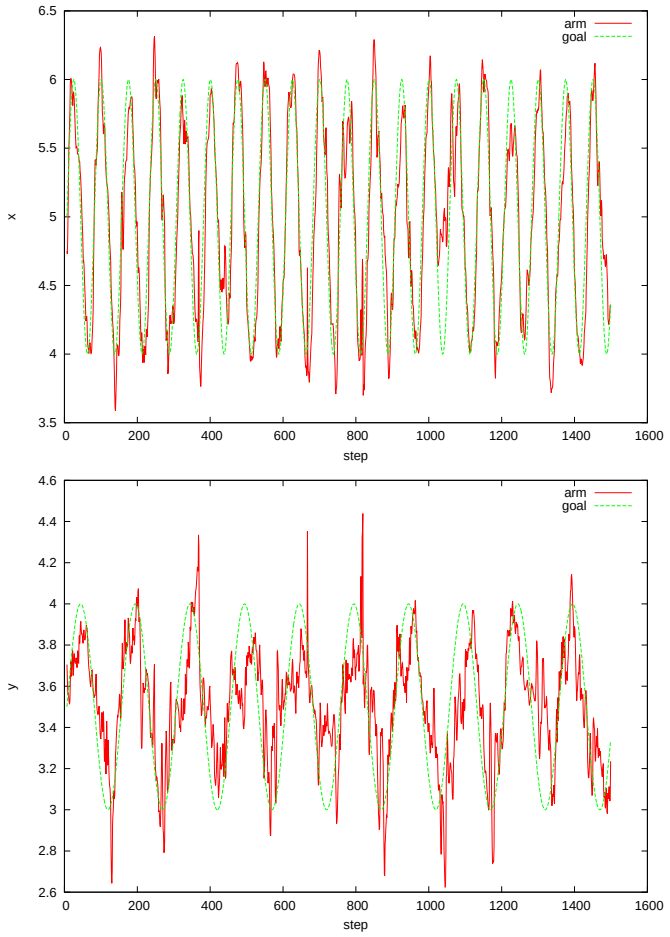

Fig. 5. The x and y coordinates of a sample trajectory tracking on the real arm when tracking 10 repetitions of a figure 8 using the parameters given in Table I. One figure consists of 150 goal points and therefore takes 15 s to complete.

for several reasons: When using PAMs, multiple actuators are needed for one joint because PAMs are only capable of pulling. Therefore, the motor command space, such as the pressures of the PAMs, has to be redundant with respect to the joint angular space. In addition to this redundancy, there is also the well-known redundancy between the position space of the end effector and the joint angular space.

One of the properties of PAMs is their high level of backdriveability – in contrast to many traditional driving systems using electrical motors and decelerators (as found, e.g., in ASIMO and other robots). This also means, however, that pressure control of individual PAMs influence each other through the dynamics of the body structure. Expressing a continuous motion by snapshots of postures has to take into account the dynamics of these systems. In other words, in order to realize trajectory tracking in a PAM-driven arm, inverse dynamics learning is necessary but has to face several challenges.

Due to the fact that the vision system of the arm can currently only detect the 2D position of the red glove of the robot (see also Fig. 2), the tracking is performed in only two dimensions. Figure 5 shows the tracking performance of the algorithm on the musculoskeletal arm using a real-time Linux system to ensure proper synchronization. As can be seen, the tracking performance for the first coordinate matches well while the performance on the second dimension is more noisy. Given the fact that the algorithm worked on the simulated arms, we believe that the main reason for this noise lies in the structure of the arm in combination with the random exploration: The freedom of the arm in the y-direction is very limited and it requires quite some effort to maintain a particular pose. Additionally, because of the random exploration in the algorithm, we were only able to use moderate pressures in the muscles. Therefore, this dimension was rarely explored. This might have led to the noisy performance. Another point to make is that the delay of the muscles is quite variable since the pressure in the muscles builds up over time. For our algorithm, a fixed delay time of 6 steps ($\widehat{=}$ 600ms) worked well. However, this long and variable reaction time is surely one reason for the oscillating noise in the second dimension. Unfortunately, the action feedback to the ESN reservoir was not able to compensate this issue. Since the tracking of the first dimension barely shows oscillations, this effect might not be as large as one would initially assume. Finally, the optimal parameters for the real arm might differ from the parameters optimized by grid-search for the simulated one. Exhaustively grid-searching all parameters of the real arm is impossible. Results of the trials we made can be found in Table I.

VI. Conclusion

We present a combination of methods that is capable of learning inverse dynamics for end-effector trajectory tracking of redundant robot arms with artificial muscles in real-time. We show that, using our approach, this can be achieved without any previous knowledge about the sensorimotor contingencies and the structure of the musculoskeletal arm. We also demonstrate that the incremental growth present in online learning tasks can be taken advantage of to reduce the runtime for GPR learning to $O(N^2)$ in a way that – to the best of our knowledge – has not been suggested in previous work yet.

While the proposed model leads to good tracking results on the simple 2 DOF arm with four muscles, the results gain more variance as the number of joints and muscles increases. This was to be expected for the challenging task of end-effector trajectory tracking without any previous knowledge on a highly redundant and nonlinear system. One of the contributions of the paper is to show that the combination of techniques we use is an improvement over using any of them individually. In the previous section, the performance substantially decreased when no ESN, no GPR, or only random exploration instead of goal babbling was used. Without the ESN the system is lacking memory and a nonlinearly expanded feature space; without the goal babbling the search space would increase dramatically since learning is not focused on the subspace relevant for the trajectory tracking; and finally, the GPR facilitates the readout

to a level necessary to deal with the complex inverse dynamics present in the robot models.

We plan to investigate causes of the tracking errors in more detail in future studies. Our experiments point in the direction that we are dealing with several problems. Regarding the real arm, the activation delay is not the same for every muscle and every state of the arm which conflicts with the fixed delay assumed in our model. This could be approached by learning state-specific delays, for example using a GP-based technique. A possible reason for the error in the complex simulation, which does not suffer from delays, may be found in the dynamics that emerge from the interactions between the different joints. These might require a more complex, maybe hierarchical, ESN structure to represent the system state. This will also be a point for future research. Additionally, the uncertainty present in the predictions of a GP is not yet used although they can be obtained fairly cheaply. This information could be used to select which state-action pairs to learn and thereby reduce the computational load on the GPRs.

Acknowledgments

The authors are grateful for machine time and help with the robotic arm provided by the Hosoda lab (http://www-hi.ise.eng.osaka-u.ac.jp). The authors also would like to thank the High Performance Computing and Communications Centre (http://www.hpccc.gov.au/) for the use of their super-computer cluster in performing some of the experiments for this paper.

References

[1] G. Andrikopoulos, G. Nikolakopoulos, and S. Manesis. A Survey on applications of Pneumatic Artificial Muscles. In *Proceeding of the 10th Mediterranean Conference on Control Automation (MED), 2011*, pages 1439–1446, june 2011. doi: 10.1109/MED.2011.5982983.

[2] J.E. Bobrow and B.W. McDonell. Modeling, identification, and control of a pneumatically actuated, force controllable robot. *Robotics and Automation, IEEE Transactions on*, 14(5):732–742, 1998.

[3] B.Tondu, S.Ippolito, J.Guiochet, and A.Daidie. A Seven-degrees-of-freedom Robot-arm Driven by Pneumatic Artificial Muscle for Humanoid Robots. *The International Journal of Robotics Research*, 24(4):257–274, 2005.

[4] K.M.A. Chai, C.K.I. Williams, S. Klanke, and S. Vijayakumar. Multi-task gaussian process learning of robot inverse dynamics. *Advances in Neural Information Processing Systems*, 21, 2009.

[5] S.P. Chatzis and Y. Demiris. Echo State Gaussian Process. *Neural Networks, IEEE Transactions on*, 22 (9):1435–1445, 2011.

[6] C.P. Chou and B. Hannaford. Measurement and modeling of McKibben pneumatic artificial muscles. *IEEE Transactions on Robotics and Automation*, 12(1):90–102, 1996. ISSN 1042-296X.

[7] M. Doumit, A. Fahim, and M. Munro. Analytical Modeling and Experimental Validation of the Braided Pneumatic Muscle. *IEEE Transactions on Robotics*, 25 (6):1282–1291, 2009. ISSN 1552-3098.

[8] H. Gomi and M. Kawato. Neural network control for a closed-loop System using Feedback-error-learning++. *Neural Networks*, 6(7):933–946, 1993.

[9] H. Jaeger and H. Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78, 2004.

[10] M. Katayama and M. Kawato. Learning trajectory and force control of an artificial muscle arm by parallel-hierarchical neural network model. In *Proceedings of the 1990 conference on Advances in neural information processing systems 3*, pages 436–442. Morgan Kaufmann Publishers Inc., 1990.

[11] M. Kawato, K. Furukawa, and R. Suzuki. A hierarchical neural-network model for control and learning of voluntary movement. *Biological cybernetics*, 57(3):169–185, 1987.

[12] G.K. Klute and B. Hannaford. Accounting for elastic energy storage in McKibben artificial muscle actuators. *Journal of dynamic systems, measurement, and control*, 122:386, 2000.

[13] J. Nakanishi and S. Schaal. Feedback error learning and nonlinear adaptive control. *Neural Networks*, 17(10): 1453–1465, 2004.

[14] D. Nguyen-Tuong and J. Peters. Local gaussian process regression for real time online model learning and control. In *Advances in Neural Information Processing Systems 22*, 2008.

[15] D. Nguyen-Tuong, M. Seeger, and J. Peters. Real-Time Local GP Model Learning. In *From Motor Learning to Interaction Learning in Robots*, volume 264. Springer Verlag, 2010.

[16] V. L. Nickel, J. Perry, and A. L. Garrett. Development of useful function in the severely paralyzed hand. *Journal of Bone and Joint Surgery*, 45A(5):933–952, 1963.

[17] C.E. Rasmussen. Gaussian processes in machine learning. *Advanced Lectures on Machine Learning*, pages 63–71, 2004.

[18] R.F. Reinhart and J.J. Steil. Attractor-based computation with reservoirs for online learning of inverse kinematics. In *Proc. ESANN*, pages 257–262. Citeseer, 2009.

[19] M. Rolf, J.J. Steil, and M. Gienger. Online Goal Babbling for rapid bootstrapping of inverse models in high dimensions. In *Development and Learning (ICDL), 2011 IEEE International Conference on*, volume 2, pages 1–8. IEEE, 2011.

[20] S. Schaal, C.G. Atkeson, and S. Vijayakumar. Real-time robot learning with locally weighted statistical learning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 288–293, 2000.

[21] X. Zhu, G. Tao, B. Yao, and J. Cao. Adaptive robust posture control of a parallel manipulator driven by pneumatic muscles. *Automatica*, 44(9):2248–2257, 2008.