Proceedings of the RAAD 2013
22nd International Workshop on Robotics in Alpe-Adria-Danube Region
September 11-13, 2013, Portorož, Slovenia

Learning control for positionally controlled manipulators

Domagoj Herceg[a], Dana Kulić[b] and Ivan Petrović[a]

[a]*Departament of Computer and Control Engineering*
Faculty of Electrical and Computer Engineering
University of Zagreb, Unska 3, 10000 Zagreb, Croatia
E-mail: {domagoj.herceg,ivan.petrovic}@fer.hr

[b]*Department of Electrical and Computer Engineering*
University of Waterloo, Ontario, Canda
E-mail: dana.kulic@uwaterloo.ca

Abstract. The majority of the widely available robotic arms employ the joint position control paradigm. Additionally, these kind of arms are usually closed architecture, meaning that the user has little knowledge or control over the inner workings of the controller. The user can only specify a position trajectory that the arm needs to follow. In the unstructured environment this can be a serious drawback. By exploiting the knowledge of the system, the performance of the closed architecture robotic arm system can be improved. Recently, nonparametric regression methods have been shown to improve performance of torque controlled arms. In this paper, we investigate the effectiveness of those methods in the case of closed architecture robotics arms. We apply Gaussian Process Regression (GPR) to learn the dynamic model between the input and the output signal, including the dynamics of the robot plant and controller. We also consider a sparse variant of GPR, called Sparse Spectrum Gaussian Process Regression, which enables faster training and prediction times. It is demonstrated by simulation that the proposed approach significantly enhances the trajectory following performance of closed architecture robotic arms.

Keywords. Robot Manipulators, Learning Control, Gaussian Process Regression

1. Introduction

Robots are prevalent in industrial settings, where the operating workspace of the robot is highly structured and known in advance. This implies that most of the robot's actions can be accurately planned in advance with little or no disturbance. With such reliable plans, robot tasks can be pre-programmed. To ensure positioning accuracy, rigid links and joints are utilized, using geared motors with a high gearing ratio. Simple decentralized proportional derivative (PD) control can be employed where a PD controller for every joint acts independently of the other joints (Siciliano and Khatib, 2008). This control strategy is suboptimal as the dynamics of the joints are coupled, which the PD controller treats as a disturbance. Furthermore, robots in use are subject to wear and tear of inner mechanisms which can change the properties of the robot.

A dynamics model is needed when striving for precise control during high velocity trajectories. Having a model of the robot can provide for safer trajectories, decreased power consumption, improved accuracy and compliance control. In the setting of robotic arm control, model-based controllers apply the dynamic equation of the robotic arm to cancel out nonlinearities and coupling effects.

Traditionally, controllers which utilize the dynamic model of the system have relied on either offline identification of the dynamic parameters (Armstrong et al., 1986; Radkhah et al., 2007) or adaptive control (Craig et al., 1986). These approaches require that the structure of the dynamic model is known; the model parameters are then estimated. The needed information can also be extracted from CAD models if available. However, even the detailed CAD models do not account for some nonlinearities such as backlash or elasticity, etc. As we plan to learn the dynamics of closed architecture robotic arms, such CAD models are not available, and the dynamic structure of the complete system may not be known.

Recently, enabled by research in machine learning, there have been a number of attempts to learn nonparametric dynamic models for robot control. Examples include Gaussian Process Regression (GPR) (Rasmussen and Williams, 2006), Support Vector Regression (SVR) (Smola and Schlkopf, 2003) and Locally Weighted Projection Regression (LWPR) (Vijayakumar et al., 2005).

Recent applications of the above methods include (Nguyen-Tuong et al., 2008) where the authors compare GP and LWPR methods for inverse dynamics learning. The authors show that learning methods can substantially improve over PD control with gravity compensation. In (Droniou et al., 2012), the authors compare different methods to learn the visuo-motor relationship for visual servoing. The tests are carried out on a humanoid platform and it is shown that the robot is able to achieve better performance than a model derived from the CAD description. Learning models for control have also been applied directly in operational space by (Peters and Schaal, 2008). None of the above works exploit prior knowledge of the CAD model of the robots.

Although these methods are able to learn the nonlinear dynamics of robotic arms, they suffer from some drawbacks. The first is the unfavorable cubic scaling of the GPR algorithm with the number of data samples. Sparse Spectrum GP (Lzaro-gredilla et al., 2010) is a variant which makes the scaling linear in the number of data points and square in the number of spectral frequencies that are used to approximate the full GPR. Another approach developed with the robotic application in mind is Local Gaussian Process Regression (Nguyen-Tuong and Peters, 2008). Data points are partitioned into local regions and a separate GPR model is learned for every region. Prediction is done by averaging over weighted local models. To handle the continuous data stream from the robot sensors, the authors provide a method for on-line incorporation of new data points into each local model. The second significant problem of the learned models is that performance can degrade rapidly for the predictions outside the region in which the model has been trained. To address this issue (de la Cruz et al., 2012) propose to take advantage of the prior knowledge of the rigid body equation as a globally valid model.

While many works in this area exploit torque control capabilities of high performance robotic arms, in this paper, we investigate the effectiveness of nonparametric regression methods for closed architecture robotics arms. Our work is partially inspired by the position to torque transformer of (Khatib et al., 2008), but with an important advantage. Namely, while their approach is based on detailed knowledge of the mechanics of the joints, our approach does not assume any knowledge of the structure of the robot, i.e. apply nonparametric regression methods to learn a model without a-priori knowledge of the mechanism structure or inner controller parameters.

2. Problem formulation

The equation of motion describing the robot dynamics is a function of the robot state and the unknown dynamical parameters. Let us denote with $\mathbf{q}$ a state vector of robot joint positions. This vector consists of n_J elements, where n_J is the number of joints of the robot arm. With $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ we denote joint velocities and accelerations, i.e. the first and second time derivatives of the joint positions. The vector $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ is referred to as the robot state. Assuming a rigid body robot, the equations of motion can be stated as:

$$\tau = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) + \varepsilon(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}), \quad (1)$$

where $\mathbf{M}(\mathbf{q})$ is the $n_J \times n_J$ inertia matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is the $n_J \times 1$ centripetal and Coriolis torque vector and $\mathbf{G}(\mathbf{q})$ is the gravity loading vector. τ is the n_J sized vector of joint torques. The term $\varepsilon(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ accounts for motor dynamics, backlash and other nonlinearties generally not captured by the rigid body model.

High performance robot arms usually come with torque control capabilities, meaning that it is possible to directly control the torque the motors exert at every joint. This is possible due to torque sensors in every joint which allows for a high frequency control loop based on the deviation from the desired torque. However, robotic manipulators with direct torque control capabilities are not very common.

In contrast, most of the cheaper and widely available robotic arms are position controlled. When controlling the arm in this manner, there is a PD controller for every joint that compensates for the positional error in a particular joint. The current or voltage reference for the joint motor is based on a linear combination of the positional and velocity errors, i.e. we cannot directly control motor voltage or current. Positional feedback is provided by encoders in the joints. It is worth noting that the parameters of the PD controller are usually constant.

Our goal is to investigate possibilities for enhancing performance of closed-architecture position controlled robotic manipulators. In closed architecture robotic arms the inner controller is hidden form the user. Manufacturers usually provide the user with the input interface (desired position) and the encoder output (measured position). This is depicted in Figure 1. The manner in which a trajectory is followed is completely hidden from the user. In this paper, we propose to learn the inverse function between the control input and system output, and use the model to improve trajectory tracking performance.

In the closed architecture case, the torque is a function of the positional reference $\mathbf{q_{ref}}$, the robot's current state and the unknown dynamics of the combined inner controller and robot system. The inverse model can be described as the nonlinear mapping from the full state space to the positional input.

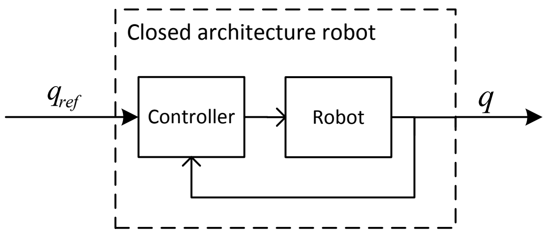

Fig. 1. Schematics of a closed architecture robotic arm

$$\mathbf{q_{ref}} = f_n(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}). \tag{2}$$

Equation (2) is a standard regression model and our goal is to infer the unknown function $f_n(.)$.

3. Gaussian Process Regression

Gaussian Process Regressions is a relatively new technique in the machine learning community. It falls into the category of nonparametric Bayesian methods. The applicability of GPR and its derivatives has been demonstrated for learning the inverse dynamics of manipulators as discussed in Section. 1. To briefly introduce GPR, let us consider a regression task of mapping an input vector $\mathbf{x}$ to an output scalar y. The model for the regression can be stated as follows:

$$y = f(\mathbf{x}) + \varepsilon, \tag{3}$$

where $f(\mathbf{x})$ is the latent function that we are trying to infer and ε is a noise therm distributed as $\varepsilon \sim \mathcal{N}(0, \sigma_n^2)$ and independent of the latent function. In GPR a Gaussian prior is assumed over the possible latent functions. The definition of a Gaussian Process (GP) states that a GP is a collection of random variables, any finite number of which have a joint Gaussian distribution. For any number N of input vectors $\mathbf{x_i}$ and a vector of evaluations $\mathbf{f} = [f(\mathbf{x_i}), \dots, f(\mathbf{x_N})]$ we have:

$$p(\mathbf{f}|\mathbf{x_i}, i \in 1, \dots, N) = \mathcal{N}(f|\mathbf{0}, \mathbf{K}). \tag{4}$$

$\mathbf{K}$ is the covariance matrix between input points where each element is calculated as $\mathbf{K}_{ij} = k(x_i, x_j)$. The choice of the covariance function $k(.,.)$ is not arbitrary, it has to be positive a definite function in order to have a valid covariance matrix. For the sake of convenience we will aggregate the input data into matrix $\mathbf{X}$ and output into vector $\mathbf{Y}$. Assuming a test point $\mathbf{x}_*$, we would like to infer the unknown output y_*, i.e. we would like to know the distribution $p(y_*|\mathbf{x}_*, \mathbf{X}, \mathbf{Y})$. Having in mind the basic definition of GP, the estimation model can be rewritten as follows:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{y}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} + \sigma_n^2 \mathbf{I} & \mathbf{K}_* \\ \mathbf{K}_*{}^T & k_{**} + \sigma_n^2 \end{bmatrix}\right),$$

where k_{**} is prior variance of $f(\mathbf{x}_*)$, $\mathbf{K}_*$ is a vector of covariance between $f(\mathbf{x}_*)$ and training values.

By conditioning on the observed values we have for the predictive mean and variance:

$$p(y_*|\mathbf{x}_*, \mathbf{X}, \mathbf{Y}) = \mathcal{N}(\mu_*, \sigma_*^2), \tag{5}$$

where

$$\mu_* = \mathbf{k}_*(\mathbf{K} + \sigma_n^2 \mathbf{I_n})^{-1}\mathbf{y}, \tag{6}$$

$$\sigma_*^2 = \sigma_n^2 + \mathbf{k}_{**} - \mathbf{k}_*(\mathbf{K} + \sigma_n^2 \mathbf{I_n})^{-1})\mathbf{k}_*{}^T. \tag{7}$$

The properties of the latent function f are governed by the covariance function, which in turn depends on a set of *hyperparameters*. These can be estimated using gradient based optimization of the log marginal likelihood:

$$\log p(\mathbf{Y}|\mathbf{X}) = \overbrace{-\frac{1}{2}\mathbf{y}^T \mathbf{K_f}^{-1}\mathbf{y}}^{\text{data fit}} - \overbrace{\frac{1}{2}\log|\mathbf{K_f}|}^{\text{complexity penalty}} - \frac{n}{2}\log 2\pi, \tag{8}$$

where $\mathbf{K_f} = \mathbf{K} + \sigma_n \mathbf{I}$. Equation (8) provides a built in regularization mechanism as the first term encourages a fit to the data, while the second term penalizes complex models.

In the rest of the paper we will use the automatic relevance determination (ARD) kernel function:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{M}(\mathbf{x}_i - \mathbf{x}_j)\right), \tag{9}$$

where diagonal matrix $\mathbf{M}$ contains $\mathbf{M}_{ii} = l_i^{-2}$. Hyperparamethers l_i are usually called lengthscales and by growing they diminish the relevance of the i^{th} input dimension. A downside of the GPR is the inversion of the covariance matrix of size $n \times n$, which is $O(n^3)$. This is unpractical for large n. However, once the inversion is known each subsequent prediction is $O(n)$ for the mean and $O(n^2)$ for the variance.

3.1. Sparse spectrum Gaussian Process Regression

To decrease the computational complexity of GPR, several simplifying approximation methods have been proposed. Some of the more popular include Snelson and Ghahramani (2006) and Csató and Opper (2002).

Rahimi and Recht (2007) show that shift invariant kernels can be approximated to an arbitrary precision using random feature mapping. A kernel is shift invariant if it only depends on the difference of its inputs, i.e. $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{x} - \mathbf{y})$. Lzaro-gredilla et al. (2010) show that this approach can be successfully used in the GPR case, yielding a new class of sparse GPR methods called Sparse spectrum Gaussian Process Regression (SSGPR). Using the Bochner theorem one can write:

$$\begin{aligned} k(\mathbf{x}_i - \mathbf{x}_j) &= \int_{\mathbb{R}^n} e^{-i\omega^T(\mathbf{x}_i - \mathbf{x}_j)}\mu(\omega)d\omega \\ &= \mathbb{E}_\omega[z_\omega(\mathbf{x}_i)^T z_\omega(\mathbf{x}_j)], \end{aligned} \tag{10}$$

where $z_\omega(\mathbf{x}) = [\cos(\omega^T\mathbf{x}), sin(\omega^T\mathbf{x})]^T$. If the spectral frequency ω is drawn according to the measure μ, the inner product gives the unbiased estimate of the shift invariant kernel. The accuracy of the approximation can be made arbitrarily good by drawing more samples and averaging over them. We then have a feature mapping:

$$\phi(\mathbf{x}) = \frac{1}{\sqrt{m}}[z_{\omega_1}(\mathbf{x})^T, \ldots, z_{0\omega_m}(\mathbf{x})^T]^T, \qquad (11)$$

where m is the number of spectral frequencies drawn form the distribution. For big enough m we have

$$k(\mathbf{x}_i, \mathbf{x}_i) \approx \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle. \qquad (12)$$

Finally, for the ASE kernel used in this paper, the feature mapping is as follows:

$$\phi(\mathbf{x}) = \frac{\sigma_f}{\sqrt{m}}[\sin(\omega_1^T\mathbf{x}), \cos(\omega_1^T\mathbf{x}), \ldots \\ , \sin(\omega_m^T\mathbf{x}), \cos(\omega_D^T\mathbf{x})]. \qquad (13)$$

Building on the previous results, Gijsberts and Metta (2013) implement an incremental version of SSGPR called Incremental SSGPR (ISSGPR). By using a rank one update to the covariance matrix they show that it can be incrementally updated practically forever. Hyperparameters tuning has to be done offline. The number of random features can be chosen to address the trade-off between accuracy and computational requirements. Another practical side of the ISSGPR is that the algorithm is very predictable. A breakdown of arithmetic operations in each step can be stated precisely as is given in Gijsberts and Metta (2013). The complexity of SSGPR is $O(nm^2)$ per conjugate gradient step for training, $O(m)$ and $O(m^2)$ for predictive mean and variance respectively. ISSGPR enables updating the covariance matrix with a new data sample in $O(m^2)$, with the predictive complexity the same as in SSGPR.

4. Proposed method

To learn the model for control of the robot we collect the data from the robot and perform regression on it. In every discrete instance two different values are recorded from the simulated robot. The input to our inverse model is the state vector $(\mathbf{q}^k, \dot{\mathbf{q}}^k, \ddot{\mathbf{q}}^k)$ at each discrete instance k. The state vector is $3 \times n_J$ with n_J being the number of joints. The output of the model is a scalar positional command given to the inner PD controller at the same instance - q_{ref}^k. In total N data pairs $\left(q_{ref}^k, (\mathbf{q}^k, \dot{\mathbf{q}}^k, \ddot{\mathbf{q}}^k)\right)$ are recorded with $k \in 1, 2, \ldots, N$. From the recorded data we identify the underlying function that describes the robot dynamics with respect to the issued positional command. We are interested in the unknown mapping $f_n(.)$ that maps input to output as given in Eq. 2

Fig. 2. Unimate's Puma 560 (picture taken form http://www.robotics.tu-berlin.de/)

The manipulator used for testing is the simulated Puma 560 model, whose dynamics parameters have been identified in (Corke and Armstrong-Helouvry, 1994). The Puma 560 is a 6 degree of freedom manipulator with revolute joints. A picture of the physical arm can be seen in Fig 2. In the simulation, the Puma arm input interface is a vector of torques acting on every joint. The output is the complete state vector. To simulate a positional controller, we introduce a feedback signal based on positional error and another one based on the velocity error. Our PD controller has the following parameters: $K_p = 2500$, $K_v = 50$ and the torque reference is calculated as follows:

$$\tau_{ref} = K_p\left(\mathbf{q_{ref}} - \mathbf{q}\right) - K_v\dot{\mathbf{q}} \qquad (14)$$

The torque reference $\tau_{\mathbf{ref}}$ is then passed on to the robot's interface. The simulation is implemented using the Robotics Toolbox software by (Corke, 2011).

To collect the data we use the motor babbling technique as described by (Peters and Schaal, 2008). A number of random joint position set points is generated for the arm to reach. Points are connected by 5^{th} order polynomials to have a smooth trajectory with a duration of 1 second for every interval between two adjacent set points. Data points are sampled every 0.01 seconds, while the inner control loop is running at 1 kHz. We collect data samples from the positional reference, which is the output for our regression task, and the joint state, which is the input vector. This can be seen in Fig. 3. A model is identified from the recorded data. Afterwards, the model is used to control the robotic arm.

Prediction from the model is used to control the robot. The desired joint state vector $(\mathbf{q_{ref}}, \dot{\mathbf{q}}_{\mathbf{ref}}, \ddot{\mathbf{q}}_{\mathbf{ref}})$ is fed into the inverse model, which outputs the pseudo reference $\mathbf{q_{ps}}$. To handle the remaining modeling error and any unmodelled disturbances, an outside loop with a small gain is added in order to compensate for small deviations from the desired trajectory. The outside loop control signals are denoted by $\mathbf{q_{fb}}$. The control

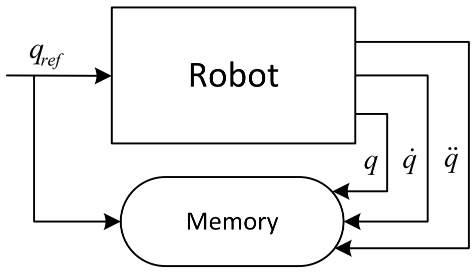

Fig. 3. Data collection procedure

Tab. 3. One-step prediction errors for different trajectories. Trajectories are ordered by how much they differ from the training state space with trajectory 3 being the most different

Method	GPR	SSGPR	ISSGPR
Trajectory 1	7.9×10^{-9}	0.0011	3.2×10^{-4}
Trajectory 2	6.5×10^{-8}	0.0016	8.7×10^{-4}
Trajectory 3	369.1862	0.1985	0.0036

signal passed on to the inner controller is $\mathbf{q_c} = \mathbf{q_{ps}} + \mathbf{q_{fb}}$. This is depicted in Figure 4. The ratio of the contribution from the outside loop and the output of the model indicates if the inverse model is providing satisfactory accuracy. A large magnitude of the control signal from the outside loop indicates that the learned model is not sufficient.

5. Simulation results

Regression results for the first joint are reported in Tab. 1. The Recorded data is partitioned into n data points used for training, and n_{test} data points used for testing. The number of spectral points used in the SSGPR is denoted by m. The mean and standard deviation are reported for the predictive absolute error of the model. Root mean square (RMS) error is also reported. GPR is used as the baseline method. We can see that the SSGPR method approaches the full GPR in modeling accuracy, while being much faster both in the optimization and the prediction stage.

To verify that the proposed solution works in closed loop in a realistic setting, simulations are carried out emulating the real operating conditions of a robotic arm. Trajectories used in testing the closed loop control are a finite sum of sine and cosine functions. This type of trajectory allows for analytic differentiation to obtain velocity and acceleration references. For the i^{th} joint we have:

$$q_i(k) = \sum_{l=1}^{N_i} \left(a_l^i \sin(\omega_f lkT_s) - b_l^i \sin(\omega_f lkT_s) \right), \quad (15)$$

where l is the number of elements of the finite Fourier series, a_l^i and b_l^i are the amplitudes of the sine and cosine functions. The behavior of the trajectory can be controlled by modifying the free parameters ω_f, N_i, a_l^i and b_l^i.

As the output of our model is a scalar, we use a separate model for every joint, i.e. 6 models for the Puma 560 arm. The input vector is the same for every model, because the joint dynamics is coupled, but a different positional command is issued to every joint controller. The learned controllers are all used in nonlinear feedforward control as depicted in Fig. 4.

The Performance of FF control with SSGPR and decentralized PD control are shown in Figs. 5 to 10. It can be seen from the figures that the learned controllers significantly enhance the performance of the trajectory tracking. We can observe that the first two joints are the most difficult to learn. A statistical evaluation is given in Tab. 2. The free parameters of Eq. 15 are randomly generated to obtain different test trajectories. In total, closed loop performance for two different control strategies was evaluated on 100 random trajectories. RMS of tracking error is used as a measure of performance. RMS is calculated for every trajectory and mean and variance are reported for this measure over all of the test trajectories. It is worth noting that the test trajectories were not used in the training procedure. Additionally, a measure $\frac{\sum_k q_{fb}(k)^2}{\sum_k q_{ff}(k)^2}$ was used to verify that feedback loop in nonlinear FF control is only marginally contributing to overall control signal. Summation over k indicates summation over all discrete instances for a single trajectory. The ratio was below 10^{-6} for all the joints on all of the test trajectories, indicating that our model is accurate enough.

Another important criterion for the closed loop usage of learned models is the ability for on-line adaptation. To show on-line adaptation performance we will use one-step ahead prediction. A PD controller is assigned to track a certain trajectory. Afterwards, prediction is made by incorporating the previous sample from the trajectory in the model, hence the name one-step ahead prediction. Three trajectories are generated which are the same except for the offset in the joint positions. We have already stated that prediction with GPR is very sensitive to deviations from the training state space and we are interested if the ISSGPR method can alleviate this problem, by incorporating new training data on-line. Trajectory 1 has all of the joint values well inside the training region of the position state space. By position training state we mean just the 6 position variables, in contrast to the full training state space of 18 variables for the Puma 560 robot. Trajectory 2 has an offset in the first joint so that the first joint position is outside the training space approximately half the time. Finally, trajectory 3 is offset in such a manner that all of the joints are outside of the training region approximately half of the time.

In total, three methods are compared. The results

Tab. 1. Inverse model identification for the simulated Puma 560 robotic manipulator. Two variations of SSGPR method are presented. SSGPR *fix* means that only the hyperparameters are optimized, while SSGP *full* means that spectral frequencies are optimized as well

Method	m	n	n_{test}	Mean	Std. dev.	RMS error	Training time
GPR	-	15000	5000	0.0003	0.0007	0.0008	12 [h]
SSGPR fix	100	15000	5000	0.0103	0.0462	0.0173	92 [s]
SSGPR fix	300	15000	5000	0.0069	0.0310	0.0129	495 [s]
SSGPR fix	500	15000	5000	0.0038	0.0250	0.0167	742 [s]
SSGPR fix	800	15000	5000	0.0016	0.0126	0.0089	1444 [s]
SSGPR fix	1000	15000	5000	0.0009	0.0181	0.0056	18029 [s]
SSGPR fix	2000	15000	5000	0.0002	0.0089	0.0014	51054
SSGPR full	100	15000	5000	0.0094	0.0125	0.0156	142 [s]
SSGPR full	500	15000	5000	0.0006	0.0020	0.0021	1624 [s]

Tab. 2. Closed loop performance of PD and Feedforward control approaches. A SSGPR learned model with 500 spectral points was used. Mean and variance of the RMS of the tracking performance over 100 runs are reported

Value	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6
FF mean	$2.599e-3$	$2.556e-3$	$645.168e-6$	$112.460e-6$	$98.753e-6$	$101.983e-6$
PD mean	$28.775e-3$	$35.688e-3$	$22.342e-3$	$20.979e-3$	$20.845e-3$	$20.430e-3$
FF std. dev	$1.674e-3$	$1.996e-3$	$409.393e-6$	$77.7326e-6$	$77.433e-6$	$80.513e-6$
PD std. dev.	$5.717e-3$	$6.381e-3$	$3.828e-3$	$2.799e-3$	$2.859e-3$	$3.125e-3$

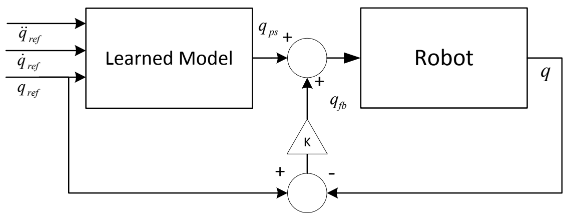

Fig. 4. Nonlinear feedforward control strategy

are reported in Tab. 3. The baseline method is the GPR learned model. The other two methods include models learned by SSGPR and ISSGPR. The hyper-parameters of the ISSGPR are the same as with the SSGPR method, but in each time step we include the measured input-output pair into the covariance matrix via rank one update. As covariance updates are made for ISSGPR, results for the ISSGPR model is bound to give better performance. Incorporating a single new sample into GPR model would only result in the increase the covariance matrix and is not practical for real time control. Results for the GPR and SSGPR are presented just to give insight into the performance gain that is achieved by ISSGPR. The measure used for comparing different methods is the normalized mean square error, i.e. squared error divided by variance of the target vector. For the first trajectory the GPR method is superior as expected. Similar results are obtained for the second trajectory. The third trajectory is the most illustrating. GPR performs poorly because it is very sensitive to training state, SSGPR preforms much better, but it can be seen that ISSGPR is able to achieve superior performance by incorporating the data on-line.

6. Conclusion

In this paper we investigate methods for learning an input-output model for control of closed architecture robotic manipulators. To achieve this goal, GPR methods were used to learn the mapping between the reference position input and the joint positions, velocities and accelerations. To avoid the exponential scaling of the computational complexity of the GPR method with the training data size, we also investigate a sparse approximation of the GPR called SSGPR. The regression is performed offline, but the learned model is used on-line in a nonlinear feedforward

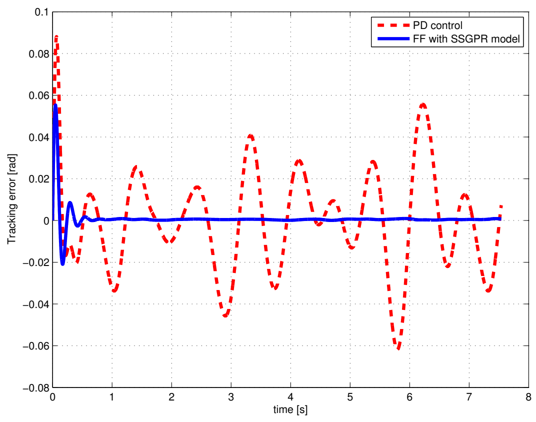

Fig. 5. Joint 1 tracking error comparison

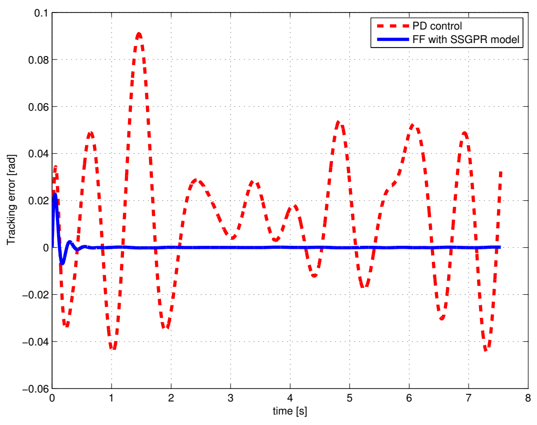

Fig. 6. Joint 2 tracking error comparison

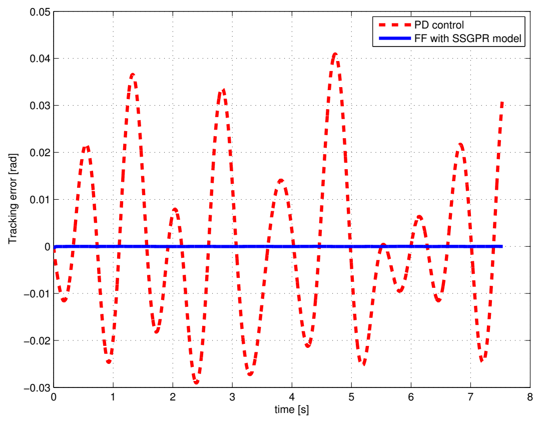

Fig. 9. Joint 5 tracking error comparison

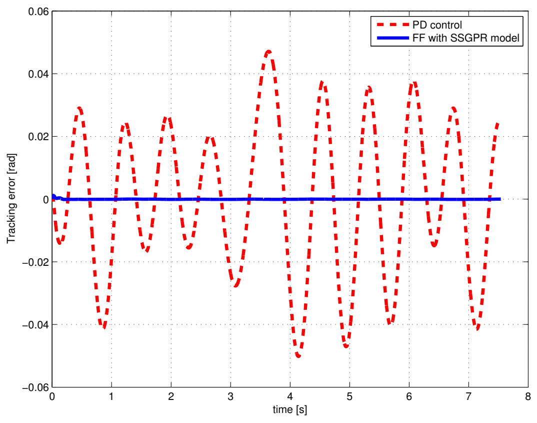

Fig. 7. Joint 3 tracking error comparison

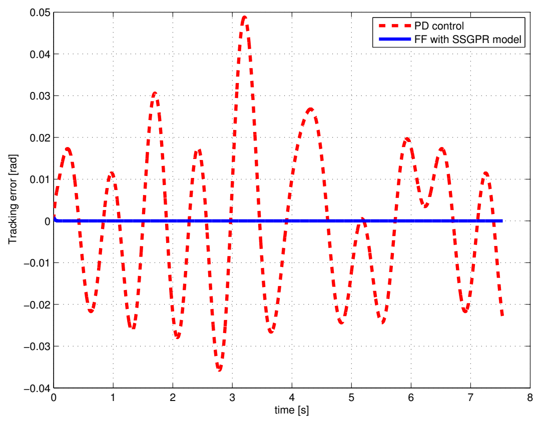

Fig. 10. Joint 6 tracking error comparison

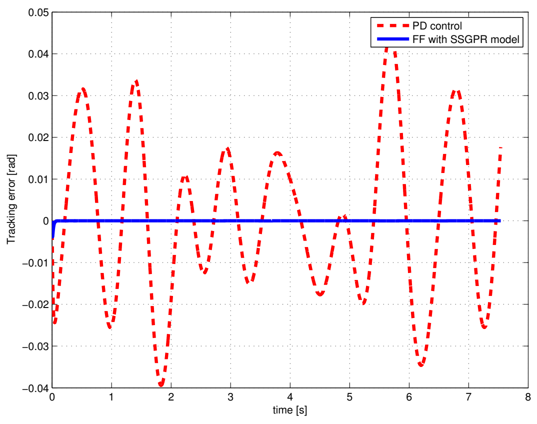

Fig. 8. Joint 4 tracking error comparison

control approach. The proposed approach is tested in simulation on the Puma 560 robot model. The results presented suggest that the proposed method can be used to control the robot arm. A significant performance boost in comparison to decentralized PD control is observed. Additionally, the SSGPR method can be extended to ISSGPR, an incremental version, so that the learned model can be adapted in an on-line manner. This is useful to decrease computation effort and improve closed loop accuracy, particularly in the face of changing plant parameters, e.g., increasing friction due to wear and tear.

7. Future work

For future work, our priority will be to perform an experimental evaluation of the results presented in this paper. State estimation is going to play a major role, because our robot (Schunk Powerball Lightweight Arm LWA 4.6) is currently only equipped with positional encoders. We also plan to investigate the state estimation problem more closely in the simulation environment to simulate real operating conditions such as sensor noise and quantization.

8. Acknowledgments

This work has been supported by the European Community's Seventh Framework Programme under grant agreement no. 285939 (ACROSS) and is also funded in part by the Natural Sciences and Engineering Council of Canada.

9. References

Armstrong, B., Khatib, O., and Burdick, J. (1986). The explicit dynamic model and inertial parameters of the puma 560 arm. In *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, volume 3, pages 510–518.

Corke, P. and Armstrong-Helouvry, B. (1994). A search for consensus among model parameters reported for the puma 560 robot. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 1608–1613 vol.2.

Corke, P. I. (2011). *Robotics, Vision & Control: Fundamental Algorithms in Matlab*. Springer.

Craig, J., Hsu, P., and Sastry, S. (1986). Adaptive control of mechanical manipulators. In *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, volume 3, pages 190–195.

Csató, L. and Opper, M. (2002). Sparse on-line gaussian processes. *Neural Comput.*, 14(3):641–668.

de la Cruz, J., Owen, W., and Kulic, D. (2012). Online learning of inverse dynamics via gaussian process regression. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 3583–3590.

Droniou, A., Ivaldi, S., Padois, V., and Sigaud, O. (2012). Autonomous online learning of velocity kinematics on the icub: a comparative study. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1–6.

Gijsberts, A. and Metta, G. (2013). 2013 special issue: Real-time model learning using incremental sparse spectrum gaussian process regression. *Neural Netw.*, 41:59–69.

Khatib, O., Thaulad, P., Yoshikawa, T., and Park, J. (2008). Torque-position transformer for task control of position controlled robots. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1729–1734.

Lzaro-gredilla, M., Quionero-candela, J., Edward, C., Engineering, R. D., Figueiras-vidal, A. R., and Jaakkola, T. (2010). Sparse spectrum gaussian process regression.

Nguyen-Tuong, D. and Peters, J. (2008). Local gaussian process regression for real-time model-based robot control. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 380–385.

Nguyen-Tuong, D., Seeger, M., and Peters, J. (2008). Computed torque control with nonparametric regression models. In *American Control Conference, 2008*, pages 212 –217.

Peters, J. and Schaal, S. (2008). Learning to control in operational space. *International Journal of Robotics Research*, pages 197–212.

Radkhah, K., Kulic, D., and Croft, E. (2007). Dynamic parameter identification for the crs a460 robot. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 3842–3847. IEEE.

Rahimi, A. and Recht, B. (2007). Random features for large-scale kernel machines. In *NIPS 2007 - Advances in Neural Information Processing Systems*.

Rasmussen, C. E. and Williams, C. (2006). *Gaussian Processes for Machine Learning*. MIT Press.

Siciliano, B. and Khatib, O., editors (2008). *Springer Handbook of Robotics*. Springer.

Smola, A. J. and Schlkopf, B. (2003). A tutorial on support vector regression. Technical report, STATISTICS AND COMPUTING.

Snelson, E. and Ghahramani, Z. (2006). Sparse gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems 18*, pages 1257–1264. MIT press.

Vijayakumar, S., D'souza, A., and Schaal, S. (2005). Incremental online learning in high dimensions. *Neural Comput.*, 17(12):2602–2634.