Model Learning with Local Gaussian Process Regression [1]

Yash Bagla[2]

I. INTRODUCTION

In technical applications precise models of technical systems can be crucial. Only a well-estimated inverse dynamics model of a robot control allow both high accuracy and compliant. For complex robots such as humanoids or lightweight arms, it is often hard to analytically model the system sufficiently well and, thus, modern regression methods can offer a viable alternative [1, 2]. However, it is difficult to get a high accuracy model with low computational cost. Models like Gaussian process regression (GPR) suffer from high computational cost, while fast real-time learning algorithms such as locally weighted projection regression (LWPR) are not straightforward to use, as they require manual adjustment of many data dependent parameters. But we can combine these two approaches to get a better model of our dynamical system.

This approach of combining these two models in a sensible way can give us better performance with lower computational cost. This work proposes a similar algorithm to improve the tracking control model of a robotic arm.

II. MODEL

For sufficiently precise robot models, the torque control enables high speed and compliant robot control while achieving accurate control with small tracking errors.The system dynamics of the moving robot can be given by the equation:

$$M(q)\ddot{q} + C(q,\dot{q}) + G(q) + \varepsilon(q,\dot{q},\ddot{q}) = u \tag{1}$$

where $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$, are joint angles, velocities and accelerations of the robot, respectively, $\mathbf{u}$ denotes the applied torques, $\mathbf{M}(\mathbf{q})$ the inertia matrix of the robot and $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ Coriolis and centripetal forces, $\mathbf{G}(\mathbf{q})$ gravity forces and $\varepsilon(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ represents nonlinearities of the robot which are not part of the rigid-body dynamics due to hydraulic tubes, friction, actuator dynamics, etc.

The model-based tracking control law determines the joint torques $\mathbf{u}$ necessary for following a desired trajectory $\mathbf{q_d}, \dot{\mathbf{q}}_\mathbf{d}, \ddot{\mathbf{q}}_\mathbf{d}$ using a dynamics model while employing feedback in order to stabilize the system. For example, the dynamics model of the robot can be used as a feed-forward model that predicts the joint torques $\mathbf{u_{FF}}$ required to track the desired trajectory. While a feedback term $\mathbf{u_{FB}}$ ensures the stability of the tracking control with a resulting control law of $\mathbf{u} = \mathbf{u_{FF}} + \mathbf{u_{FB}}$. The feedback term can be a linear control law such as $\mathbf{u} = \mathbf{K_p e} + \mathbf{K_v \dot{e}}$, where $\mathbf{e} = \mathbf{q_d} - \mathbf{q}$ denotes the tracking error and $\mathbf{K_p}, \mathbf{K_v}$ position-gain and velocity-gain, respectively. If an accurate model in the form of Equation (1) can be obtained, e.g., for negligible unknown nonlinearities, the resulting feed-forward term $\mathbf{u}$ will largely cancel the robots nonlinearities [5].

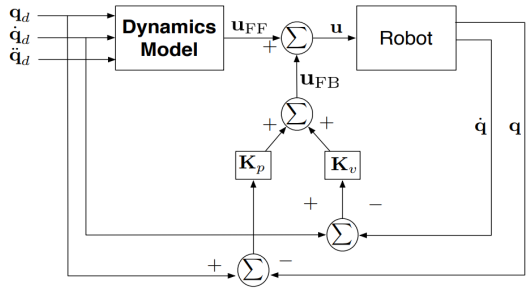

Fig. 1. Schematic showing computed torque robot control

III. REGRESSION METHODS

A. Locally Weighted Projection Regression

In this section, we briefly outline the schematic layout of the LWPR learning mechanism. Fig. 2 shows the associated local units and the inputs which feed into it. Here, a weighting kernel (determining the locality) is defined that computes a weight $\mathbf{w_{k,i}}$ for each data point $(\mathbf{x_i}, \mathbf{y_i})$ according to the distance from the center $\mathbf{c_k}$ of the kernel in each local unit. For a Gaussian kernel,$\mathbf{w_{k,i}}$ becomes

$$w_{k,i} = \exp(-\frac{1}{2}(x_i - c_k)^T D_k (x_i - c_k)) \tag{2}$$

where $\mathbf{D_k}$ corresponds to a distance metric that determines the size and shape of region of validity of the linear model. Here we assume that there are $\mathbf{K}$ local linear models combining to make the prediction.The distance matrix $\mathbf{D_k}$ determines the size and shape of each local model; it can be updated incrementally using leave-one-out cross validation [2].

Given an input vector$\mathbf{x}$, each linear model calculates a prediction $\mathbf{y_k}$. The weighted prediction $\hat{\mathbf{y}}$ is then given by

$$\hat{y} = \mathbb{E}\{\bar{y}|\mathbf{x}\} = \sum_{k=1}^{M} \bar{y_k} p(k|\mathbf{x}) \tag{3}$$

According to the Bayesian theorem, the probability of the model k given query point x can be expressed as:

$$p(k|\mathbf{x} = \frac{p(k,\mathbf{x}}{p(\mathbf{x})} = \frac{p(k,\mathbf{x}}{\sum_{k=1}^{K} p(k,\mathbf{x})} = \frac{w_k}{\sum_{k=1}^{K} w_k} \tag{4}$$

[1] I've submitted a similar version of this paper in my other Graduate project course. [2] Yash Bagla is a student in the Department of Mechanical Engineering, Michigan State University - East Lansing

The total output of the network is the weighted mean of all linear models:

$$\hat{y} = \frac{\sum_{k=1}^{K} w_k y_k}{\sum_{k=1}^{K} w_k} \tag{5}$$

B. Gaussian Process Regression

A powerful alternative for accurate function approximation in high-dimensional space is Gaussian process regression (GPR) [4]. In this section we first discuss the Bayesian treatment of the linear model. The Bayesian analysis of the standard linear regression model with Gaussian noise. Given a set of n training data points $\{\mathbf{x_i}, \mathbf{y_i}\}_{\mathbf{i=1}}^{\mathbf{n}}$, we would like to learn a function $\mathbf{f}(\mathbf{x_i})$ transforming the input vector $\mathbf{x_i}$ into the target value $\mathbf{y_i}$ given a model $\mathbf{y_i} = \mathbf{f}(\mathbf{x_i}) + \varepsilon_\mathbf{i}$, where $\varepsilon_\mathbf{i}$ is Gaussian noise with zero mean and variance $\boldsymbol{\sigma^2}$[4]. As a result, the observed targets can also be described by a Gaussian distribution $\mathbf{y} \sim \mathscr{N}(\mathbf{0}, \mathbf{K}(\mathbf{X},\mathbf{X})\boldsymbol{\sigma^2}\mathbf{I})$, where $\mathbf{X}$ denotes the set containing all input points $\mathbf{x_i}$ and $\mathbf{K}(\mathbf{X}, \mathbf{X})$ the covariance matrix computed using a given covariance function.

Gaussian kernels are probably the frequently used covariance functions [4] and are given by:

$$k(x_p, x_q) = \sigma^2 \exp(-\frac{1}{2}(x_p - x_q)^T \mathbf{W}(x_p - x_q)) \tag{6}$$

where $\boldsymbol{\sigma^2}$ denotes the signal variance and $\mathbf{W}$ represents the widths of the Gaussian kernel. The joint distribution of the observed target values and predicted value $\mathbf{f}(\mathbf{x}_*)$ for a query point $\mathbf{x}_*$ is given by

$$\begin{bmatrix} y \\ f(x_*) \end{bmatrix} \sim \mathscr{N}(0, \begin{bmatrix} \mathbf{K}(\mathbf{X},\mathbf{X}) + \sigma^2\mathbf{I} & \mathbf{k}(\mathbf{X},x_*) \\ \mathbf{k}(x_*,\mathbf{X}) & \mathbf{k}(x_*,x_*) \end{bmatrix})$$

Conditioning the joint distribution yields the predicted mean value $\mathbf{f}(\mathbf{x}_*)$ with the corresponding variance $\mathbf{V}(\mathbf{x}_*)$

$$f(x_*) = k_*^T (K + \sigma^2 I)^{-1} y = k_*^T \alpha \tag{7}$$

$$V(x_*) = k(x_*, x_*) - k_*^T (K + \sigma^2 I)^{-1} \tag{8}$$

with $\mathbf{k}_*$ = $mathbf{k}(x_*, x_*)$, $mathbf{K} = K(X,X)$ and $mathbf{\alpha}$ denotes the so-called prediction vector.[4]

IV. LOCAL GAUSSIAN PROCESS REGRESSION

GPR suffers from high computational cost as it involves inverse of the matrix $(\mathbf{K} + \boldsymbol{\sigma^2}\mathbf{I})^{-\mathbf{1}}$, which in turn increases the computational cost to $O(n^3)$. We propose a method for speed-up the training and prediction process by partitioning the training data in local regions and learning an independent Gaussian process model for each region. The local models have limited number of data points, where insertion and removal of data points can be treated in a principled manner. The prediction for a query point is performed by weighted average similar to LWPR. For partitioning and weighted prediction we use a kernel as similarity measure. Thus, our algorithm consists out of three stages: (i) clustering of data, i.e., insertion of new data points into the local models, (ii) learning of corresponding local models and (iii) prediction for a query point. [1]

The distance measure $\mathbf{w_k}$ is given by the kernel used to learn the local GP models, i.e Gaussian Kernel. The kernel width $\mathbf{W}$ is obtained by maximizing the log likelihood on a subset of the whole training data points. As the trajectories become more complex, the number of local models is allowed to increase.

A. Partitioning of Training Data

For real-time learning clustering of input data can be performed efficiently using a similarity measure between the input point $\mathbf{x}$ and the centers of the respective local models. The proximity of data points (distance measure) can be defined in terms of a kernel as the Kernel functions naturally incorporate the similarity measure between data points. We have a basic assumption on nearby inputs as they have similar target values. Thus the training points that are in periphery of same center are informative about the prediction of a new query point in this local region.

As mentioned in section III B, the distance measure for a Gaussian Kernel can be given by

$$w_{k,i} = \exp(-\frac{1}{2}(x_i - c_k)^T \mathbf{W}(x_i - c_k)) \tag{9}$$

where $\mathbf{c_i}$ denotes the mean(center) of the i^{th} local region, and $\mathbf{W}$ is the Kernel width represented by a diagonal matrix.

The incoming data point will be included to the nearest local model, i.e., the one with the maximal value of w_i. This can be easily computed by calculating the distance measure with respect to every center which in turn gives the proximity to every local model. If a particular data point is added to a local model the new center is then computed by calculating the mean of the new model with new data point included in it. Also, the number of local models is allowed to increase, new models with center c_{i+1} is created if all similarity measures w_i fall below a threshold w_{gen}.

B. Prediction using Local Models

As mentioned in the above section III A, prediction of the mean value is similar to LWPR. In other words, using weighted averaging over $\mathbf{M}$ local GP predictions $\bar{y_i}$, predictions for a mean value $\hat{y}$ is performed. This is given by equation(5), in which K corresponds to M local models and k corresponds to the i^{th} data entry.Thus, each local GP prediction $\bar{y_i} = k(X_i, x)^T \alpha_i$ is additionally weighted by the similarity $w_i(\mathbf{x}, \mathbf{c}_i)$ between the corresponding center $\mathbf{c}_i$ and the query point $\mathbf{x}$. The search for M local models can be quickly done by evaluating the proximity between the query point $\mathbf{x}$ and all model centers $\mathbf{c}_i$.

V. CONCLUSIONS

This model can be used to perform fast inverse dynamics computations which eliminates the error terms due to various non-linearities such as friction in the joints, deformities in the link, etc. In addition we also compare standard rigid-body dynamics (RBD) models with several models learned offline on training data sets. We are able to perform equivalently good with difference in control gains of up-to 3 orders. This

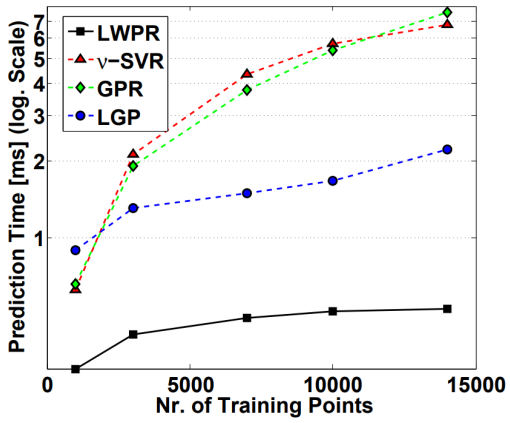

Fig. 2. The computation time is plotted logarithmically with respect to the number of training examples for a 7 DoF robot. LWRP is the fastest due to low computational cost but if we are not able to do manual tuning of control inputs LGP gives overall better results.

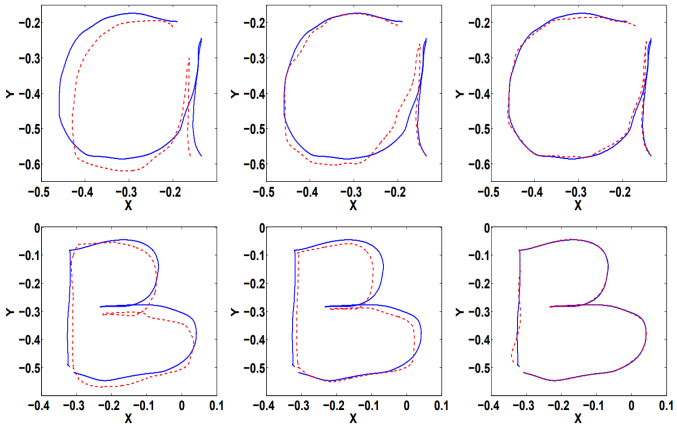

Fig. 3. **(A)** Tracking test-characters using rigid-body model **(B)** Tracking test-characters using offline-learned GP model **(C)** Tracking test-characters after online-learning with LGP

is a significant amount of change and it makes tracking of a desired trajectory more viable without using high gains.

We can deduce from Fig. 2 that the computational cost of LGP is significantly lower than that of GPR or v-SVR. This is a good enough performance to to online-learning with our model.

The trajectory tracking also gives better results with our model using LGP. We are able to predict not only the non-linear error teams but also the trajectory becomes much smoother throughout the tracking process.

Future work will be focused on partitioning in higher-dimensional space, here we only used 2-D characters for tracking. We can also use different kernel to understand the complexity and try to get a faster algorithm. Other choices for possible kernels can be found in [4, 6]. We also want to improve the algorithm by developing a new and more effective criterion for insertion and deletion of the data points as if we delete a data point randomly(which is done in our work) it can degrade/improve the performance randomly as we cannot state the importance of a randomly selected data point in our algorithm. I also want to implement this algorithm on a 5 DoF robotic chair being developed in MSU for disabled kids. This will help in performing precise actions using a robotic arm and if we can minimize the errors significantly we can do even complex tasks such as drinking water, painting, etc.

REFERENCES

[1] S. Schaal, C. G. Atkeson, and S. Vijayakumar, Scalable techniques from nonparametric statistics for real-time robot learning, Applied Intelligence, pp. 4960, 2002.
[2] S. Vijayakumar, A. DSouza, and S. Schaal, Incremental online learning in high dimensions, Neural Computation, no. 12, pp. 26022634, 2005.
[3] Duy Nguyen-Tuong, Jan Peters, "Local Gaussian process regression for real-time model-based robot control," Intelligent Robots and Systems, 2008.
[4] C. E. Rasmussen and C. K. Williams, Gaussian Processes for Machine Learning. Massachusetts Institute of Technology: MIT-Press, 2006.
[5] M. W. Spong, S. Hutchinson, and M. Vidyasagar, Robot Dynamics and Control. New York: John Wiley and Sons, 2006.
[6] B. Scholkopf and A. Smola, Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond. Cambridge, MA: MIT-Press, 2002.