2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

# Learning-Based Robot Control with Localized Sparse Online Gaussian Process

Sooho Park
Mechanical Engineering
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, Pennsylvania 15213, USA
Email: soohop@andrew.cmu.edu

Shabbir Kurbanhusen Mustafa
Singapore Institute
of Manufacturing Technology
71 Nanyang Drive
638075, Singapore
Email: mustafa@simtech.a-star.edu.sg

Kenji Shimada
Mechanical Engineering
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, Pennsylvania 15213, USA
Email: shimada@cmu.edu

***Abstract*—In recent years, robots have been increasingly utilized in applications with complex unknown environments, which makes system modeling challenging. In order to meet the demand from such applications, an experience-based learning approach can be used. In this paper, a novel learning algorithm is proposed, which can learn an unknown system model from given data iteratively using a localization approach to manage the computational costs for real time applications. The algorithm segments the data domain by measuring significance of data. As case studies, the proposed algorithm is tested on the control of the mecanum-wheeled robot and in learning the inverse kinematics of a kinematically-redundant manipulator. As the result, the algorithm achieves the on-line system model learning for real time robotics applications.**

## I. Introduction

In recent years, robots have been increasingly utilized in applications with complex and unknown environments. In such applications, there are two demanding issues that are difficult to address by conventional robot control approaches. The first issue is that it is difficult to obtain an accurate model of a highly nonlinear system. The second issue is that modeling every situation a robot encounters is often impossible. In contrast, experience-based learning approaches [1]–[4] are able to resolve these issues. Since an experience-based approach learns the system model directly from data, it can adapt to complex system behaviors that are difficult to be addressed in an idealized analytical model or to be modeled in advance. In addition, this learning approach can be applied across a wide range of robotics problems because it does not need analytical models that depend on the unique configurations of the problems. As such, this general approach can be applied .

The general framework of learning-based robot control algorithms consists of three steps. In Step 1, the algorithm applies a user-supplied initial controller on the system and observes the behavior of the system. It then learns the system model from observations (data or sensory information) in Step 2. In Step 3, it improves the controller to achieve a given task based on the learnt system model. Repeating these steps, the algorithm gradually builds the system model and updates the controller.

In this paper, we focus on an algorithm to learn the system model in Step 2. It is important to note that there are four requirements for the learning algorithms. First, the algorithm should be able to handle arbitrary nonlinear systems. For this requirement, an approach known as non-parametric regression should be used. Second, as the sensory information are usually given as a data stream, the algorithm should be able to update the existing knowledge of the system iteratively with the arrival of new data. Third, the algorithm should be computationally fast. Since the robot interact with environment in real time, the learning algorithm should perform as fast as possible. The final requirement is that the algorithm should be able to handle noisy data in a robust fashion. As far as we know, none of the previous methods used in experience-based approaches satisfy all these requirements, limiting their applications.

This paper proposes a novel algorithm - *Localized Sparse Online Gaussian Process* (LSOGP) that is based on Online Gaussian Process (OGP) [5]. Even though OGP is a non-parametric, iterative, and stochastic regression method that satisfies most of the requirements mentioned above, It still cannot achieve the real time performance speed and accuracy at the same time since the computational cost is increasing along with the learning procedure. LSOGP resolve this problem. On top of the advantages of OGP, LSOGP effectively controls the computational cost to realize real time responsiveness by using a localization approach detailed in Section III. As the result, LSOGP can learn an unknown complex system model from given data iteratively using a localization approach to manage the computational costs for real time applications.

The rest of the paper is organized as follows: Section II briefly discusses the mathematical background of OGP and related work. Section III proposes LSOGP based on the formulation from Section II. Two examples are presented to evaluate this algorithm in Section IV, while Section V concludes with some future directions for this work.

## II. Online Gaussian Process

Given a data set $\mathcal{D} = \{(\mathbf{x}_t, y_t), t = 1, \cdots, T\}$ of vector inputs $\mathbf{x}_t \in \mathcal{R}^{d_\mathbf{x}}$ and scalar outputs $y_t = f(\mathbf{x}_t) + \epsilon$,

$y_t \in \mathcal{R}$, $\epsilon \sim \mathcal{N}\left(0, \sigma_\epsilon^2\right)$, Online Gaussian Process (OGP) [5] provides a tool to represent the functional Gaussian probability distribution of underlying function $f$ based on $\mathcal{D}$ in iterative update formula as follows:

$$\Pr\left(f\left(\mathbf{x}\right) | Y_T\right) \approx \mathcal{N}\left(\boldsymbol{\mu}_T\left(\mathbf{x}\right), \Sigma_T\left(\mathbf{x}, \mathbf{x}\right)\right), \tag{1}$$

where $\boldsymbol{\mu}_T$ and $\Sigma_T$ are

$$\boldsymbol{\mu}_T\left(\mathbf{x}\right) = K\left(\mathbf{x}, X_T\right)\boldsymbol{\alpha}_T, \tag{2}$$

$$\boldsymbol{\alpha}_T = \left[\begin{array}{c} \boldsymbol{\alpha}_{T-1} \\ 0 \end{array}\right] + \left[\begin{array}{c} \Omega_{T-1}K\left(X_{T-1}, \mathbf{x}_T\right) \\ 1 \end{array}\right] \boldsymbol{\alpha}_T|_{\mathbf{x}_T}, \tag{3}$$

$$\boldsymbol{\alpha}_T|_{\mathbf{x}_T} = \left[\Sigma_{T-1}\left(\mathbf{x}_T, \mathbf{x}_T\right) + \sigma_\epsilon^2\right]^{-1}\left(y_T - \boldsymbol{\mu}_{T-1}\left(\mathbf{x}_T\right)\right), \tag{4}$$

$$\Sigma_T\left(\mathbf{x}, \mathbf{x}'\right) = K\left(\mathbf{x}, \mathbf{x}'\right) + K\left(\mathbf{x}, X_T\right)\Omega_T K\left(X_T, \mathbf{x}'\right), \tag{5}$$

$$\Omega_T = \left[\begin{array}{cc} \Omega_{T-1} & \mathbf{0} \\ \mathbf{0}^T & 0 \end{array}\right] + \left[\begin{array}{c} \Omega_{T-1}K(X_{T-1}, \mathbf{x}_T) \\ 1 \end{array}\right] \Omega_T|_{(\mathbf{x}_T, \mathbf{x}_T)} \left[\begin{array}{c} \Omega_{T-1}K(X_{T-1}, \mathbf{x}_T) \\ 1 \end{array}\right]^T, \tag{6}$$

$$\Omega_T|_{(\mathbf{x}_T, \mathbf{x}_T)} = -\left[\Sigma_{T-1}\left(\mathbf{x}_T, \mathbf{x}_T\right) + \sigma_\epsilon^2\right]^{-1}, \tag{7}$$

and $X_T = \{\mathbf{x}_1, \cdots, \mathbf{x}_T\}$, $Y_T = \{y_1, \cdots, y_T\}$.
A kernel used in this paper is the *Radial Basis Function* (RBF) kernel,

$$K\left(\mathbf{x}_i, \mathbf{x}_j\right) = \exp\left(-\frac{1}{2}\left(\mathbf{x}_i - \mathbf{x}_j\right)^T W_\mathbf{x}^T W_\mathbf{x}\left(\mathbf{x}_i - \mathbf{x}_j\right)\right), \tag{8}$$

where $W_\mathbf{x}^T W_\mathbf{x} (:= P_\mathbf{x}) \in R^{d_\mathbf{x} \times d_\mathbf{x}}$ is the weight matrix, which is symmetric positive-semidefinite, and can be adjusted to compensating for scale differences among the dimensions of $\mathbf{x}$. $P_\mathbf{x}$ and the system noise $\sigma_\epsilon$ are more commonly referred to *hyper-parameters*, $\theta$.
Note that the computational cost of this is reduced to $\mathcal{O}\left(T^2\right)$ by actively exploiting the previous regression results $\boldsymbol{\alpha}_{T-1}$ and $\Omega_{T-1}$.

## III. Localized Sparse Online Gaussian Process

Although OGP is attractive due to its simple implementation, capability to handling a highly nonlinear system, fully probabilistic expression, and iterative update property, applying OGP to robot dynamics learning is still not feasible because the computational cost increases as the number of observed data points increases. This is due to the dependency on the data set[1], whose size increases along with the training. Therefore, the computation of OGP becomes infeasible as the size of the training set increases.

There are two solutions for this problem. First, the total computational cost can be controlled by limiting the size of the data to keep. Only the most significant, finite number of data points are kept and reused. Leave One Out Cross Validation (LOOCV) can be applied to implement this. If the regression result changes appreciably after adding the new data point, the data point is considered significant and kept in a data set called the *Representative data set,* $X_R = \{\mathbf{x}_{i_1}, \cdots, \mathbf{x}_{i_R}\}$ and reused for the next iteration. If not, it can be discarded. This approach is termed *Sparse approximation* [6]. In the sparse approximation approach, the evaluation of the significance of each datum (significance measurement) and the method to exclude a certain datum from the previous learning result $\boldsymbol{\alpha}_T$ and $\Omega_T$ (pruning data) are important issues.

The second approach is a divide-and-conquer approach. It is termed *Localization* [7], and its data domain is learned by not one OGP, but by many OGPs that segment the whole domain into smaller pieces so that the size of data to be learned is confined for each local region. In this approach, the proper partitioning of data space becomes a challenge.

In this section, both of these approaches are derived to make OGP fast enough to be applied to a robotic system.

### A. Sparse Approximation

With respect to basis function representation, Eqs. (2) and (5) add a new basis function $K\left(\mathbf{x}_T, \mathbf{x}\right)$ to the OGP model at every iteration. This is prohibitive for on-line learning because the memory usage and computational cost increase with training data size. In this section, Sparse Online Gaussian Process (SOGP) [5] is shortly introduced.

- *Significance Measurement:*

As mentioned above, Eqs. (2) and (5) can be represented by only the important basis functions $K\left(X_R, \mathbf{x}\right)$ and their coefficients $\boldsymbol{\alpha}_T$ and $\Omega_T$. In order to choose $X_R$, it is necessary to measure the significance of each datum. In every iteration of OGP, the coefficients $\boldsymbol{\alpha}_{T-1}$ and $\Omega_{T-1}$ are updated, and a new kernel function $K\left(\mathbf{x}_T, \mathbf{x}\right)$ is added to the previous set of basis functions $K\left(X_{R-1}, \mathbf{x}\right)$, i.e., adding one more dimension to the functional subspace. However, in the case where $K\left(\mathbf{x}_T, \mathbf{x}\right)$ is close enough to the subspace of $K\left(X_{R-1}, \mathbf{x}\right)$, the projection of $K\left(\mathbf{x}_T, \mathbf{x}\right)$ onto the functional subspace $K\left(X_{R-1}, \mathbf{x}\right)$ can be used instead of adding one more functional dimension. Furthermore, the projection error can be used as a significance measurement. Since we are handling a functional space, Reproducing Kernel Hilbert Space (RKHS) norm [5], [8] is used for computing the functional projection error instead of the projection error in the Euclidean space. This projection finds the optimal linear combination of the bases of subspace $K\left(X_{R-1}, \mathbf{x}\right)$ to make the shortest distance to $K\left(\mathbf{x}_T, \mathbf{x}\right)$, and is represented as $\boldsymbol{\beta}^T K\left(X_{R-1}, \mathbf{x}\right)$. The coefficient $\boldsymbol{\beta}$ can be obtained analytically as

$$\boldsymbol{\beta} = K\left(X_{R-1}, X_{R-1}\right)^{-1} K\left(X_{R-1}, \mathbf{x}_T\right). \tag{9}$$

Hence, the projection error of the kernel function can be obtained as

$$\varepsilon^2\left(\mathbf{x}\right) = K\left(\mathbf{x}, \mathbf{x}_T\right) - K\left(\mathbf{x}, X_{R-1}\right)^T K\left(X_{R-1}, X_{R-1}\right)^{-1} K\left(X_{R-1}, \mathbf{x}_T\right).$$

Note that this is always positive as long as the Gram matrix is positive semi-definite. If $\varepsilon^2\left(\mathbf{x}\right)$ is small enough to be ignored, $K\left(\mathbf{x}_T, \mathbf{x}\right)$ can be approximated as $K\left(\mathbf{x}_T, \mathbf{x}\right) \approx \boldsymbol{\beta}^T K\left(X_{R-1}, \mathbf{x}\right)$. In this case, Eqs. (3) and (6) can be represented as

$$\boldsymbol{\alpha}_T = \boldsymbol{\alpha}_{T-1} + \left(\Omega_{T-1}K\left(X_{R-1}, \mathbf{x}_T\right) + \boldsymbol{\beta}\right)\boldsymbol{\alpha}_T|_{\mathbf{x}_T} \tag{10}$$

[1]This is a model parameter of OGP.

$$\begin{aligned}\Omega_T &= \Omega_{T-1}\\ &+ \left(\Omega_{T-1}K\left(X_{R-1},\mathbf{x}_T\right)+\boldsymbol{\beta}\right)\Omega_T|_{(\mathbf{x}_T,\mathbf{x}_T)}\\ &\times \left(\Omega_{T-1}K\left(X_{R-1},\mathbf{x}_T\right)+\boldsymbol{\beta}\right)^T, \qquad (11)\end{aligned}$$

From the difference between Eq. (2) with Eq. (3) and Eq. (5) with Eq. (10), the projection error in the output domain becomes

$$e\left(\mathbf{x}_T\right) = \left|\boldsymbol{\alpha}_T|_{\mathbf{x}_T}\right|\varepsilon^2\left(\mathbf{x}_T\right). \qquad (12)$$

Based on $e(\mathbf{x}_T)$, we can decide whether Eqs. (3) and (6), or Eqs. (10) and (11) should be used to update coefficients. If $e(\mathbf{x}_T)$ is bigger than a certain value $\nu$, $(\mathbf{x}_T, y_T)$ should be added as a new basis function (a new dimension of the functional space) by using Eqs. (3) and (6), and if $e(\mathbf{x}_T)$ is smaller than $\nu$, it should be projected by using Eqs. (10) and (11). Using this approach, the size of $X_R$ to be stored can be kept significantly smaller than the size of the given data.

Eq. (12) can be represented in a simpler form as follows: by using the Woodbury matrix identity,

$$K\left(X_R, X_R\right)^{-1} := \begin{bmatrix} A & \mathbf{b} \\ \mathbf{b}^T & c \end{bmatrix},$$

where

$$c = \left[K\left(\mathbf{x}_T,\mathbf{x}_T\right) - \boldsymbol{\beta}^T K\left(X_{R-1},\mathbf{x}_T\right)\right]^{-1} = \frac{1}{\varepsilon^2\left(\mathbf{x}_T\right)}.$$

Also, from Eqs. (3), (12) and $c$, the projection error is expressed as

$$e\left(\mathbf{x}_T\right) = \frac{\left|\boldsymbol{\alpha}_T|_{\mathbf{x}_T}\right|}{c} = \frac{\left|\boldsymbol{\alpha}_T|_{\mathbf{x}_T}\right|}{K\left(X_R,X_R\right)^{-1}\Big|_{(\mathbf{x}_T,\mathbf{x}_T)}}.$$

Furthermore, assuming that this error formula is consistent for all other data in $X_R$ and $Y_R$, the projection error for an arbitrary $r^{th}$ representative datum can be defined as

$$e\left(\mathbf{x}_{i_r}\right) = \frac{\left|\boldsymbol{\alpha}_T|_{\mathbf{x}_{i_r}}\right|}{K\left(X_R,X_R\right)^{-1}\Big|_{(\mathbf{x}_{i_r},\mathbf{x}_{i_r})}}. \qquad (13)$$

In summary, if $K(X_R, X_R)^{-1}$ is given, we can obtain $e(\mathbf{x}_{i_r})$ of all the representative data by using Eq. (13). Based on $e(\mathbf{x}_{i_r})$, we can measure the significant of each representative data.

In practice, calculating $\boldsymbol{\beta}$ or $e(\mathbf{x}_{i_r})$ can be expensive because the computational cost of Gram matrix inversion is $\mathcal{O}(R^3)$. We can overcome this obstacle by using the 'Woodbury matrix identity' or 'Cholesky decomposition rank one update' [9]. In both methods, calculating $K(X_R, X_R)^{-1}$ from $K(X_{R-1}, X_{R-1})^{-1}$ can be done iteratively in $\mathcal{O}(R^2)$.

- *Pruning of Representative Data:*

When $e(\mathbf{x}_{i_r}) < \nu$, we can remove the corresponding datum $\mathbf{x}_{i_r}$ from the representative data set and project it. Since the only coefficients we have at the $T^{th}$ iteration are $\boldsymbol{\alpha}_T$ and $\Omega_T$, the goal is to restore $\boldsymbol{\alpha}_{T\setminus r}$ and $\Omega_{T\setminus r}$ from $\boldsymbol{\alpha}_T$ and $\Omega_T$, and then projecting the removed datum on $\boldsymbol{\alpha}_{T\setminus r}$ and $\Omega_{T\setminus r}$ ($\cdot_{\setminus r}$ means excluding the $r^{th}$ element of $\cdot$). Although we assume only the situation in which the lastly added datum $\mathbf{x}_{i_R}$ is removed, the same strategy can be applied to all $r$. From Eqs. (3) and (6), we can obtain following equalities:

$$\boldsymbol{\alpha}_T|_{X_R} = \boldsymbol{\alpha}_{T-1} + \Omega_{T-1}K\left(X_{R-1},\mathbf{x}_{i_R}\right)\boldsymbol{\alpha}_T|_{\mathbf{x}_{i_R}}, \qquad (14)$$

$$\begin{aligned}\Omega_T|_{(X_R,X_R)} &= \Omega_{T-1} \qquad (15)\\ &+ \Omega_{T-1}K\left(X_{R-1},\mathbf{x}_{i_R}\right)\Omega_T|_{(\mathbf{x}_{i_R},\mathbf{x}_{i_R})}K\left(\mathbf{x}_{i_R},X_{R-1}\right)\Omega_{T-1},\end{aligned}$$

$$\Omega_T|_{(X_{R-1},\mathbf{x}_{i_R})} = \Omega_{T-1}K\left(X_{R-1},\mathbf{x}_{i_R}\right)\Omega_T|_{(\mathbf{x}_{i_R},\mathbf{x}_{i_R})}. \qquad (16)$$

And from Eq. (16),

$$\Omega_{T-1}K\left(X_{R-1},\mathbf{x}_{i_R}\right) = \Omega_T|_{(X_{R-1},\mathbf{x}_{i_R})}\,\Omega_T|^{-1}_{(\mathbf{x}_{i_R},\mathbf{x}_{i_R})}. \qquad (17)$$

By substituting Eq. (14) into Eq. (10) and Eq. (15) into Eq. (11) and then using Eq. (17), we can restore the coefficients by projecting $\mathbf{x}_R$ as follows:

$$\boldsymbol{\alpha}_T = \boldsymbol{\alpha}_{T-1}|_{X_{R-1}} + \boldsymbol{\beta}\,\boldsymbol{\alpha}_T|_{\mathbf{x}_{i_R}}, \qquad (18)$$

$$\begin{aligned}\Omega_T &= \Omega_T|_{(X_{R-1},X_{R-1})} + \boldsymbol{\beta}\,\Omega_T|_{(\mathbf{x}_{i_R},X_{R-1})}\\ &+ \Omega_T|_{(X_{R-1},\mathbf{x}_{i_R})}\boldsymbol{\beta}^T + \boldsymbol{\beta}\,\Omega_T|_{(\mathbf{x}_{i_R},\mathbf{x}_{i_R})}\boldsymbol{\beta}^T. \qquad (19)\end{aligned}$$

Based on this pruning strategy, we can bound $R$ by a certain value $R_{max}$ and achieve a computational efficiency of $\mathcal{O}(R^2_{max})$. The detail procedure is presented in Algorithm 1.

**Algorithm 1** Sparse Online Gaussian Process Algorithm

```
1) Given $\mathbf{x}_1$, $y_1$, $\nu$, $R_{max}$
2) **Initialization**
   $\boldsymbol{\alpha}_1 = \left[K(\mathbf{x}_1,\mathbf{x}_1) + \sigma_\epsilon^2\right]^{-1} y_1$
   $\Omega_1 = -\left[K(\mathbf{x}_1,\mathbf{x}_1) + \sigma_\epsilon^2\right]^{-1}$
   $R = 1$
3) **Learning**:
     ***Datum addition***:
       For a given new datum $(\mathbf{x}_T, y_T)$, $T > 2$,
       Compute $\boldsymbol{\mu}_{T-1}(\mathbf{x}_T)$ and $\Sigma_{T-1}(\mathbf{x}_T,\mathbf{x}_T)$ using
       Eqs. (2) and (5).
       Compute $\boldsymbol{\alpha}_T|_{\mathbf{x}_T}$ and $\Omega_T|_{(\mathbf{x}_T,\mathbf{x}_T)}$
       using Eqs. (4) and (7).
       Update coefficient $\boldsymbol{\alpha}_T$ and $\Omega_T$ using
       Eqs. (3) and (6).
       Let $R = R + 1$.
     ***Data pruning***:
       **If** $R > R_{max}$
         **For twice**
           Calculate the projection error $e(\mathbf{x}_{i_r})$, $\forall r$.
           For the $r^{th}$ datum whose $e(\mathbf{x}_{i_r})$ is smallest,
           **If** $e(\mathbf{x}_{i_r}) < \nu$,
             Remove and project the $r^{th}$ datum using
             Eqs. (18) and (19).
           **end if**
         **end for**
       **end if**
4) **Prediction**:
   For a query input $\mathbf{x}$,
   compute $\boldsymbol{\mu}_T(\mathbf{x})$ and $\Sigma_T(\mathbf{x},\mathbf{x})$ using Eqs. (2) and (5).
```

### B. Localization : Multi-Layer Clustering

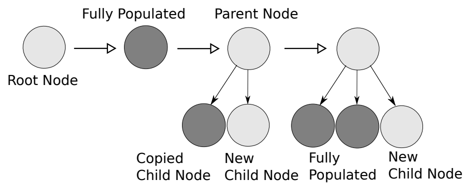
Even though we can manage the computational cost as described in Section III-A, such a limited number of representative data points may not be enough to cover a wide range of domain, and prediction results may not be accurate enough in this case. The data domain can be divided into small subregions [7] and SOGP can be performed for each subregion separately to ensure both accuracy and speed. Thus, the training data form a set of local data clusters, and the prediction for a query is then implemented by using only the nearest local SOGP from the query. In this way, the learning can be done in real time without sacrificing accuracy. This section proposes a novel localization method based on the significance measurement in Section III-A.

In order to localize the input domain, each $c^{th}$ local SOGP model has a center $\mathbf{c}_c$ which is the mean of its $X_R$. When a new datum $(\mathbf{x}_T, y_T)$ is given, the nearest local model is found based on the distance between each $\mathbf{c}_c$ and $\mathbf{x}_T$. Then $(\mathbf{x}_T, y_T)$ is learned by the nearest local SOGP model. During the learning process of the local SOGP model, if $e(\mathbf{x}_T) > \nu$ (the new datum is significant) but the local model is already populated with the maximum representative data $R_{max}$, $(\mathbf{x}_T, y_T)$ cannot be added to the local model. Instead, it spawns a new local model. Thus, the number of local model is increased by one.

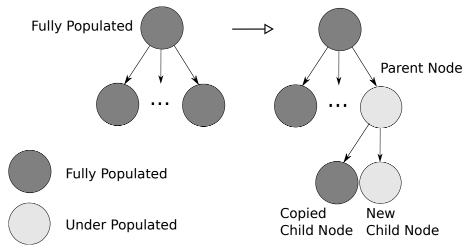
This concept can be extended to multi-layer localization, meaning that there is a local model for each local model. This naturally makes the hierarchical structure shown in Fig. 1. In this localization, two kinds of local model additions are possible: 1) A new node is added as a sibling node, which means the added node has the same parent node as the current node, or 2) The current node is considered a parent node, and the new one is added as a child node of it. When the first datum is given, the root SOGP model is initialized and learns the datum. Once the node is fully populated ($R > R_{max}$), it spawns two child nodes as illustrated in Fig. 1 (a). One child node is an exact copy of the parent node, while the other is a new node that is initialized with the new datum. From this point, a new datum is learned by these child nodes. Every time a new datum is given, the nearest child node is found, and the datum is learned by it. Note that the parent node is now populated not with data but with child nodes. So, the node becomes underpopulated when it becomes a parent node as shown in Fig. 1. Once the parent node is fully populated with $R_{max}$ child nodes, the parent node cannot have more child nodes. Then, the nearest child node from the new datum becomes a parent node, and spawns two child nodes just as the root node does (Fig. 1 (b)). With this procedure, the LSOGP has no limitation in terms of number of local models. The entire algorithm for multi-layer LSOGP is presented in Algorithm 2.

## IV. Experiments

This section demonstrates two simulation examples carried out using MATLAB® on a workstation running Ubuntu 12.04 with an Intel i7 2.8GHz quad-core processor to show the capability of LSOGP. In all these examples, system models are assumed to be unavailable and learned by LSOGP



(a) Spawning from Root Node



(b) Spawning from Parent Node

Figure 1. Multi-layer Clustering

on the fly from the new data at every iteration. By applying the same approach to solve two different example problems, we show that this learning-based robot control algorithm is generic and potentially can be applied to various robotic platforms.

### A. Mecanum-Wheeled Robot

In this example, we control a simulation of a mecanum-wheeled robot as shown in Fig. 2 (a) without any analytical dynamics model. The robot has four separate actuators to drive each wheel, and the velocity of the robot is a result of these four wheel velocities. The task of the robot is to start from $(0, 0, 0)$ to reach a given goal position $\left(0, 10, \frac{\pi}{4}\right)$, i.e., $(x, y, \theta)$ on a 2D plane as shown in Fig. 2 (b). The ideal relationship between the velocity of robot and the velocities of wheels used in the simulation is given as

$$\begin{bmatrix} v_{FL} \\ v_{FR} \\ v_{RL} \\ v_{RR} \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 \\ 1 & 1 & 1 \\ -1 & -1 & 1 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_\theta \end{bmatrix}, \tag{20}$$

where $v_{FL}$, $v_{FR}$, $v_{RL}$ and $v_{RR}$ are the velocities of Front Left (FL), Front Right (FR), Rear Left (RL), and Rear Right (RR) wheel respectively. $v_x$, $v_y$ and $v_\theta$ are the robot's linear velocities in the $x$ and $y$ direction and its angular velocity, respectively. In every pre-defined time interval, the robot applies control variables, i.e., $\mathbf{x}_T = [v_{FL}, v_{FR}, v_{RL}, v_{RR}]^T$ to the system and observes the noisy system state, i.e., $\mathbf{y}_T = [v_x, v_y, v_\theta]^T$. As the system model in Eq. (20) is assumed to be unavailable, the robot learns it from the given datum $(\mathbf{x}_T, \mathbf{y}_T)$ using LSOGP. Since the robot can control only $\mathbf{x}_T$, this is the input, and $\mathbf{y}_T$ is the output. In this test, the output is three-dimensional, and one-dimensional LSOGP in Algorithm 2 is implemented for each dimension independently.

The target function $f$ of LSOGP is assumed to be

$$\mathbf{y}_T = f(\mathbf{x}_T) + \boldsymbol{\epsilon},$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}\left(\mathbf{0}, \sigma_\epsilon^2 \otimes I\right)$ is the system noise, and $\otimes$ is element-wise multiplication.

The desired robot velocity change $\Delta\mathbf{y}$ needed to reach the goal is obtained based on the difference between the current

**Algorithm 2** Multi-layer LSOGP Algorithm

```
1) Initialization:
   Initialize a single SOGP model $Model_{LSOGP}$ as a root
   node (it is a child node at the same time).
   Let $C = 1$.
2) Learning: For a given new datum $(\mathbf{x}, y)$
   Find the nearest LSOGP model.
   Call function $LSOGP(\mathbf{x}, y, Model_{LSOGP})$
3) Prediction:
   For a query input $\mathbf{x}$, find the nearest LSOGP model.
   Compute $\boldsymbol{\mu}_T(\mathbf{x})$ and $\Sigma_T(\mathbf{x}, \mathbf{x})$ by using Eqs. (2) and
   (5).
```

```
Function $LSOGP(\mathbf{x}, y, Model_{LSOGP})$
  If $Model_{LSOGP}$ is a child node,
    If $R = R_{max}$ and projection error $e(\mathbf{x}) > \nu$,
      If parent node is $R < R_{max}$ ( for the root node,
      this is always false),
        Add a new local SOGP model at $\mathbf{x}$ as a child
        node of the parent node.
        Let $C = C + 1$, $\mathbf{c}_{C+1} = \mathbf{x}$.
      else parent node has room for a new child node.
        Make current child node a parent node.
        Copy current node to child node of the parent
        node.
        Reinitialize the parent with the two child nodes.
        Add a new local SOGP model at the removed $\mathbf{x}$,
        $C = C + 1$, $\mathbf{c}_{C+1} = \mathbf{x}$ as a child node of the
        new parent node.
      end if
    else
      Learn $(\mathbf{x}, y)$ with local SOGP model $Model_{LSOGP}$
      by using Algorithm 1.
    end if
  else $Model_{LSOGP}$ is parent node,
    Find the nearest local cluster model $Model_{LSOGP}$
    (child node of the current node) based on the
    distance between $\mathbf{x}$ and $\mathbf{c}_c$, $\forall c$.
    Call function $LSOGP(\mathbf{x}, y, Model_{LSOGP})$.
  end if
end of function
```

location $\mathbf{y}_T$ and the given goal to determine the next control variable $\mathbf{x}_{T+1}$. Then, the following optimization problem is solved.

$$\underset{\Delta\mathbf{x}}{argmin}\left\{\frac{1}{2}\left\|\Delta\mathbf{y} - \nabla_{\mathbf{x}} f(\mathbf{x})\Delta\mathbf{x}\right\|^2 + \frac{1}{2}w^2\left\|\Delta\mathbf{x}\right\|^2\right\}, \quad (21)$$

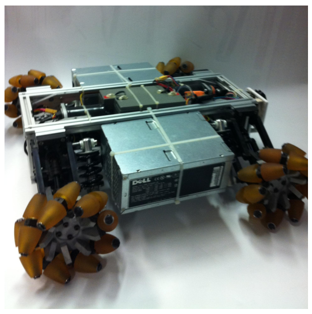
where the first term is the dynamics constraint of the system, and the second term is the penalty of the magnitude of the input change. $w^2$ is the weight coefficient. Note that the gradient of the LSOGP model, $\nabla_{\mathbf{x}} f(\mathbf{x})$, can be obtained analytically using Eqs. (23) and (24) in the Appendix. The analytical solution of this optimization problem is

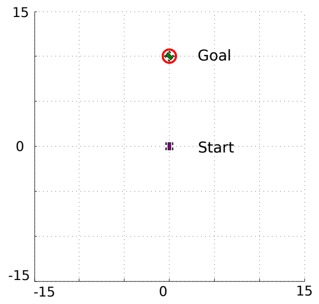
$$\Delta\mathbf{x}^* = [\nabla_{\mathbf{x}} f(\mathbf{x})]^T\left[\nabla_{\mathbf{x}} f(\mathbf{x})[\nabla_{\mathbf{x}} f(\mathbf{x})]^T + w^2 \otimes I\right]^{-1}\Delta\mathbf{y}. \quad (22)$$

Hence, $\mathbf{x}_{T+1} = \mathbf{x}_T + \rho\Delta\mathbf{x}^*$ will be the next test input to the robot system $f(\mathbf{x})$, where $\rho$ is the learning rate or the update step size. By applying $\mathbf{x}_{T+1}$, the robot reduces the error between the current position and the goal.

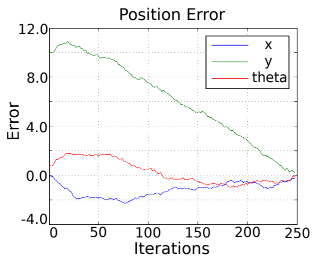
It is worth mentioning that, in the early stage of the learning, $f(\mathbf{x})$ is inaccurate and $\nabla_{\mathbf{x}} f(\mathbf{x})$ is not good enough to find a correct $\Delta\mathbf{x}^*$. In order to overcome this problem, more training data are necessary, and they can be generated by adding random variance to $\Delta\mathbf{x}^*$. By doing this, the robot explores around $\Delta\mathbf{x}^*$ and collects data to get a more accurate $f(\mathbf{x})$. However, when the robot approaches the goal, this randomness should be decreased. Thus, we use a random variable added to $\Delta\mathbf{x}^*$ whose variance is proportional to the magnitude of error between the current position and the goal. Also, to be robust in the early stages of learning, the data pruning happens only when the data size of the local model is bigger than 100.

The test results show that LSOGP successfully learns the system model on the fly, and the control algorithm can achieve the task in 250 iterations (in Fig. 2 (c)). At the same time, time consumption is limited to approximately 20 msec or less (in Fig. 2 (d)). Contrary to the original SOGP in which the computational cost increases along with the iterations, we can confirm that LSOGP can successfully limit the maximum computational cost. Note that the jump at the $100^{th}$ iteration in Fig. 2 (d) is due to the data pruning procedure. The parameters used in this test are listed in Table I. Parameters and entire algorithm used for this test are listed in Table I.

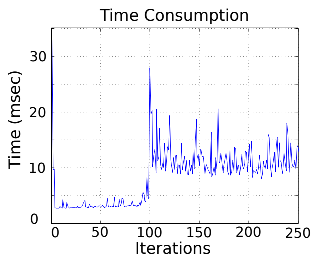

(a) Mecanum-Wheeled Robot



(b) Configuration of Simulation



(c) Position Error Graph

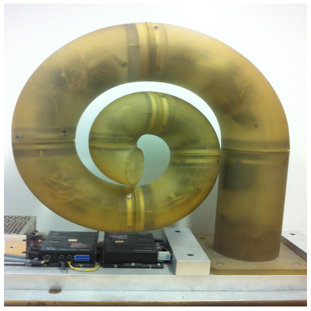


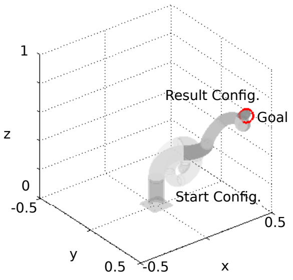
(d) Time Consumption Graph

Figure 2. Mecanum-Wheeled Robot

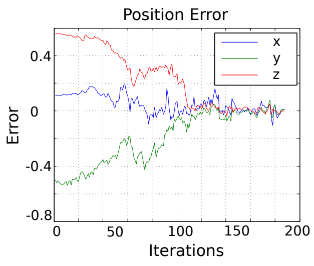
## *B. Seven Degrees of Freedom Spiral Manipulator Inverse Kinematics*

In this example, we control a redundant seven Degrees Of Freedom (DOF) spiral manipulator shown in Fig. 3 (a) without analytical inverse kinematics model. The manipulator has eight links and seven revolute joints without any mechanical joint angle limitations. The position of the end effector $\{x_{ee}, y_{ee}, z_{ee}\}$ is a result of all these seven joint angles $\{q_1, \cdots, q_7\}$. The task of the robot is to touch a sphere of radius 0.02m centered at given goal position in 3D space with its end effector. At every pre-defined time interval, the manipulator applies control variables $\mathbf{x}_T = [q_1, \cdots, q_7]^T$ to the system and observes the noisy end effector position $\mathbf{y}_T = [x_{ee}, y_{ee}, z_{ee}]^T$. The analytical kinematics model of this robot is assumed to be unavailable. Hence, the robot should learn it from the given data by using LSOGP. The desired robot end effector position change $\Delta\mathbf{y}$ to reach the goal is obtained from the difference between the current $\mathbf{y}_T$ and the goal to decide the next control variable $\mathbf{x}_{T+1}$. Then, Eq. (21) is solved using Eq. (22) in the same way as in mecanum-wheeled robot control example. Parameters and entire algorithm used for this test are listed in Table I.

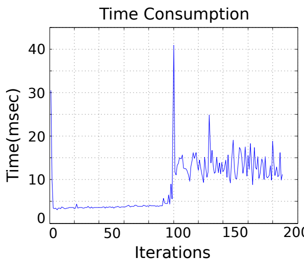

(a) 7DOF Spiral Manipulator



(b) Simulation Configuration



(c) Position Error Graph



(d) Time Consumption Graph

Figure 3. 7DOF Spiral Manipulator

Table I
PARAMETERS FOR EXPERIMENTS

| | Mecanum-Wheeled Robot | 7DOF Spiral Manipulator |
|---|---|---|
| $P$ | $I_{4\times4}$ | $I_{7\times7}$ |
| $\sigma_\epsilon$ | 0.05 | 0.01 |
| $\nu$ | 0.01 | 0.01 |
| $w$ | 0.01 | 0.1 |
| $\rho$ | 0.1 | 0.1 |
| $R_{max}$ | 200 | 200 |

## V. CONCLUSIONS

This paper proposes the novel learning algorithm - Localized Sparse Online Gaussian Process (LSOGP) for learning-based robot control. in contrast to the previous algorithms, LSOGP can learn an unknown complex system model from given data iteratively using a localization approach to manage the computational costs for real time applications.

The proposed algorithm is tested on the control of the mecanum-wheeled robot and in learning the the inverse kinematics of a kinematically redundant manipulator to demonstrate its capability. With LSOGP, the same algorithm in both tests successfully learns the system model in real time and completes the tasks without any a priori system model. This also verifies that the proposed learning-based approach is a general method which can be applied to various robot platforms in the same way.

Our future work will consider the following enhancements. First, hyper-parameters of each local OGP model can be optimized as mentioned in [10] by using the maximum likelihood approach. Since SOGP is a iterative method, the iterative numerical optimization may be implemented concurrently to save the time consumption at one iteration of LSOGP. Second, if a rough system model is available, it may be used as a prior for the LSOGP algorithm. Third, old data should be forgotten to remove the effects of early transient stage of the learning procedure in which data may be inaccurate, or to adapt to a non-stationary system. Fourth, dimensional reduction techniques such as PCA may be used to be more efficient for high dimensional $Y_T$. Fifth, the prediction can be made not only by using the nearest local model, but by mixing a few nearest ones as in [7]. This approach is helpful to guarantee the global smoothness.

## APPENDIX

In SOGP, derivatives of mean and covariance with respect to inputs are given as follows:

$$\frac{\partial \boldsymbol{\mu}(\mathbf{x})}{\partial X_R} = \left[\begin{bmatrix} K(\mathbf{x}, X_R) \\ \vdots \\ K(\mathbf{x}, X_R) \end{bmatrix} \otimes P_{\mathbf{x}} \left[(\mathbf{x}_1 - \mathbf{x}), \cdots, (\mathbf{x}_R - \mathbf{x})\right]\right]^T \otimes [\boldsymbol{\alpha}, \cdots, \boldsymbol{\alpha}], \tag{23}$$

$$\frac{\partial \Sigma(\mathbf{x}, \mathbf{x})}{\partial X_R} = [2 \otimes \Omega K(X_R, \mathbf{x}), \cdots, 2 \otimes \Omega K(X_R, \mathbf{x})] \otimes \left[\frac{\partial K(\mathbf{x}, \mathbf{x}_1)}{\partial \mathbf{x}_1}, \cdots, \frac{\partial K(\mathbf{x}, \mathbf{x}_R)}{\partial \mathbf{x}_R}\right]^T, \tag{24}$$

$$\frac{\partial K(\mathbf{x}, \mathbf{x}')}{\partial \mathbf{x}} = -P_{\mathbf{x}}(\mathbf{x} - \mathbf{x}') K(\mathbf{x}, \mathbf{x}'). \tag{25}$$

## REFERENCES

[1] M.P. Deisenroth and C.E. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. 2011.
[2] S. Schaal, P. Mohajerian, and A. Ijspeert. Dynamics systems vs. optimal control a unifying view. *Progress in brain research*, 165:425–445, 2007.
[3] J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, 2008.
[4] J. Kober and J. Peters. Policy search for motor primitives in robotics. *Machine Learning*, 84(1):171–203, 2011.
[5] L. Csató and M. Opper. Sparse online Gaussian processes. *Neural Computation*, 14(3):641–668, 2002.
[6] J.Q. Candela and C.E. Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959, 2005.
[7] D. N. Tuong and J. Peters. Local gaussian process regression for real-time model-based robot control. In *Proc. Int. Conf. Intell. Robots Sys.*, pages 380–385, 2008.
[8] S. Vijayakumar and H. Ogawa. Rkhs-based functional analysis for exact incremental learning. *Neurocomputing*, 29(1-3):85–113, 1999.
[9] Matthias Seeger. Low rank updates for the cholesky decomposition. *University of California at Berkeley, Tech. Rep*, 2007.
[10] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, Massachusetts, 2006.