On-line regression algorithms for learning mechanical models of robots: a survey

Olivier Sigaud*, Camille Salaün, Vincent Padois

Université Pierre et Marie Curie,
Institut des Systèmes Intelligents et de Robotique - CNRS UMR 7222,
Pyramide Tour 55 - Boîte Courrier 173,
4 Place Jussieu, 75252 Paris CEDEX 05, France
firstname.name@isir.upmc.fr

Please cite as: Sigaud, O. and Salaün, C. and Padois, V. (2011),
On-line regression algorithms for learning mechanical models of robots: a survey,
Robotics and Autonomous Systems, 59:1115-1129

Abstract

With the emergence of more challenging contexts for robotics, the mechanical design of robots is becoming more and more complex. Moreover, their missions often involve unforeseen physical interactions with the environment. To deal with these difficulties, endowing the controllers of the robots with the capability to learn a model of their kinematics and dynamics under changing circumstances is becoming mandatory. This emergent necessity has given rise to a significant amount of research in the Machine Learning community, generating algorithms that address more and more sophisticated on-line modeling questions. In this paper, we provide a survey of the corresponding literature with a focus on the methods rather than on the results. In particular, we provide a unified view of all recent algorithms that outlines their distinctive features and provides a framework for their combination. Finally, we give a prospective account of the evolution of the domain towards more challenging questions.

Keywords: Adaptive and learning systems, Adaptive control, Mechanical models

1. Introduction

A major trend in robotics research is a shift from industrial applications, where everything can be planned in advance, to service, intervention, and exploration applications where robots must be able to address a large diversity of tasks while reacting to unforeseen situations [95] – in particular when they have to interact with humans. In front of these versatility requirements, the robots themselves get more and more complex. Due to the presence of humans, they must also be less dangerous, thus lighter, which results in stiffness limitations [121]. Taking all these constraints together, the accurate low-level control of these systems for achieving elementary tasks is an increasingly difficult matter, not to mention higher level decision making problems.

In this context, efficient control methods are generally *model-based* and call upon mechanical models of the robot [25]. For instance, when the robot is redundant with respect to a task, a forward model of its velocity kinematics is mandatory to make profit of the redundancy. Besides, a model of the dynamics is required when the dynamics effects can generate disturbances that are larger than what local PID controllers with a limited stiffness can reject instantaneously. However, the accuracy of model-based control directly depends on the accuracy of these models and of the sensors used to close the control loop.

Up to a certain extent, mechanical engineering knowledge can help to provide such models. Indeed, the kinematic equation of a rigid poly-articulated plant is easy to obtain given a precise knowledge of its geometry. Similarly, its dynamic equation can be written as the combination of well-modeled factors such as the inertia of the plant, Coriolis and centrifugal effects as well as external forces (including gravity). But, for instance, the geometry of the mechanical chain changes when the robot is using a tool or when its structure is altered accidentally. And, at the dynamics level, there are additional unmodeled effects such as joint and cable frictions or backlashes [35] that are often difficult to model accurately. Furthermore, some of these effects are inherently non-stationary.

In that context, on-line adaptation of models is a valuable strategy because it can capture non-stationarities in the mechanical properties of the plant.

Two approaches in that respect are the on-line parameter identification step of *adaptive control* [94], which tunes parameters of a given model given its mechanical structure [59] and *model learning*, which identifies mechanical models using supervised learning methods without any knowledge of the structure. Whereas classical identification methods generally assume a rigid body model and then incorporate some enhance-

*corresponding author, phone: (+33).1.4427.8853, fax: (+33).1.4427.5145

ments to take non-linear phenomena such as friction or flexibilities into account, the model learning approach makes no assumption on the structure and includes all phenomena in a general function built out of experimental data.

This paper provides a survey of state-of-the-art methods in the supervised learning approach. A related survey paper has been published recently in Nguyen-Tuong & Peters [71]. That previous publication covers many topics in that domain, from the different control methods that make use of the learned models to the presentation of specific robotics applications that demonstrate the feasibility of the methods. Another related and recent survey paper is Hoffmann et al. [44]. It investigates the notion of *body schema* and helps discriminating between different kinds of representation that the controller of a robot can have of its body. It covers the vision processing questions related to the acquisition of such representations (e.g. Jagersand [47]) and more general work on the acquisition of a body schema or a body image that directly relate visual features to mechanical properties of the plant [43, 62, 64, 99]. It also investigates somewhat superficially the relationships to computational neurosciences issues related to model learning in animals and humans.

Here, in the terminology of Hoffmann et al. [44], we focus on mechanical models that are either forward or inverse models, without addressing all the other points listed above. More precisely, we present the local and incremental regression algorithms used for learning mechanical models so as to provide a unifying view through a study of their similarities and differences. In particular, we highlight the features that make these algorithms unique, and investigate whether they may be combined into a new generation of methods.

The paper is organized as follows. In Section 2, we give some background knowledge about the mechanical models of robots. In Section 3, we present machine learning methods that are used to learn these models. Each time we present a method, we give an overview of the applications to robot model learning it has given rise to. All these methods are discussed in Section 4. Section 5 is devoted to a prospective presentation of the evolution of the field. Finally, we summarize our main messages in Section 6.

2. Mechanical Models of Robots

Before presenting the methods used to learn mechanical models in Section 3, in this section, we explain what these models are. The control methods based on such models are not covered in the paper, we refer the reader to Nguyen-Tuong & Peters [71], Salaün et al. [86] or to other papers cited in Section 3 to learn more about these control methods.

2.1. Joint space and task space

Rigid poly-articulated systems are defined by a finite number of rigid bodies interconnected by ideal joints. An ideal joint is a kinematic relationship between two rigid bodies.

The n parameters chosen to describe the configuration of a robot along with the joint equations are sufficient to describe the pose of each body composing it. These parameters can be gathered to form the vector of configuration parameters $\boldsymbol{q}$ also named generalized coordinates. The configuration of a robot is thus defined on an n-dimensional space called *joint space* or *configuration space*.

Though configuration parameters are often chosen so that they can be intuitively related to the motion of the actuators of the plant, it is very difficult to use them to describe most of the tasks of a robot. In order to facilitate the description of these tasks, an alternative space can be used, which is often called *task space* or *operational space*.

A unitary task consists in positioning the coordinate frame of a body, for instance an end-effector, relatively to a reference coordinate frame. This task can be described with a vector of task space parameters $\boldsymbol{\xi}$ of size $m \leq 6$, since six parameters are sufficient to locally span the space of positions and orientations.

2.2. Joint space to task space mappings

The joint space to task space mapping can be described at three different levels. At the geometric level, the forward kinematics model is a non-linear function $\boldsymbol{f}$ such that $\boldsymbol{\xi} = \boldsymbol{f}(\boldsymbol{q})$.

If the robot is redundant with respect to the task, there is an infinite number of possible inverses for $\boldsymbol{f}$. In that case, there is no simple method to span the set of possible solutions at the geometric level and the mapping is often described at the velocity level by the Jacobian matrix $J(\boldsymbol{q}) = \partial \boldsymbol{f}(\boldsymbol{q}) / \partial \boldsymbol{q}$ such that

$$\dot{\boldsymbol{\xi}} = J(\boldsymbol{q})\, \dot{\boldsymbol{q}}. \tag{1}$$

$J(\boldsymbol{q})$ is an $m \times n$ matrix and thus can be inverted using linear algebra techniques. In the redundant and non-singular cases, i.e. $rank(J(\boldsymbol{q})) = m$ and $m < n$, there exists an infinity of possible generalized inverses of $J(\boldsymbol{q})$ (see Ben Israel & Greville [9]). Given the redundancy, a system can make profit of internal motions that do not induce any perturbation on the task to achieve other tasks. The resulting trajectory of the plant depends on the specific inverse that has been chosen. Different possible redundancy resolution schemes can be applied, depending on the compatibility of the tasks or constraints, which have to be solved. We refer the reader to Salaün et al. [86] for a formal treatment of this topic. Conversely, in the case of parallel plants, the inverse model is unique whereas there can be an infinity of forward models.

2.3. Forward and Inverse Dynamics

The forward dynamics mapping relates forces applied on the plant, among which the applied m-dimensional control input vector $\boldsymbol{\Gamma}$, to the acceleration $\ddot{\boldsymbol{q}}$. The Dynamics ($\mathcal{D}$) of a plant is described as

$$\ddot{\boldsymbol{q}} = A^{-1}(\boldsymbol{q})\,(\boldsymbol{\Gamma} - \boldsymbol{\Gamma}^{ext} - \boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{g}(\boldsymbol{q}) - \boldsymbol{\epsilon}(\boldsymbol{q}, \dot{\boldsymbol{q}})), \tag{2}$$

where $A(\boldsymbol{q})$, $\boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, $\boldsymbol{g}(\boldsymbol{q})$, $\boldsymbol{\epsilon}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ and $\boldsymbol{\Gamma}^{ext}$ are respectively the $n \times n$ inertia matrix of the plant, the vector of Coriolis and centrifugal effects, the vector of gravity effects, the vector of unmodeled effects and the torques resulting from external forces applied to the plant.

At the dynamics level, the state of the plant is $\boldsymbol{s} = \left[\boldsymbol{q}^T, \dot{\boldsymbol{q}}^T\right]^T$ thus the induced dimensionality is larger than at the kinematics level where it is $\boldsymbol{q}$.

The Rigid Body Dynamics (RBD) of the plant corresponds to the dynamics of the system with rigid parts and ideal joints. It is given by (2) where the term corresponding to unmodeled effects is removed. More generally, (2) can be written

$$\ddot{\boldsymbol{q}} = \mathcal{D}\left(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{\Gamma}, \boldsymbol{\Gamma}^{ext}\right). \tag{3}$$

Similarly, the inverse dynamics ($\mathcal{ID}$) is described as

$$\boldsymbol{\Gamma} = A(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{\epsilon}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{\Gamma}^{ext}, \tag{4}$$

or, more generally,

$$\boldsymbol{\Gamma} = \mathcal{ID}\left(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, \boldsymbol{\Gamma}^{ext}\right). \tag{5}$$

In any given state $\boldsymbol{s}$, the joint space to joint space mapping represented by (5) gives the torques to be applied by the actuators of the robot to obtain the desired acceleration $\ddot{\boldsymbol{q}}$ at the next step.

2.4. Inverse Dynamics in the task space

In Khatib [51], Khatib formulated an alternative expression of the dynamics of the plant in the task space

$$\boldsymbol{F} = \Lambda(\boldsymbol{q})\,\ddot{\boldsymbol{\xi}} + \boldsymbol{\mu}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{p}(\boldsymbol{q}) + \bar{\boldsymbol{\epsilon}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{F}^{ext} \tag{6}$$

$$\boldsymbol{\Gamma} = J(\boldsymbol{q})^T\,\boldsymbol{F} \tag{7}$$

where $\Lambda(\boldsymbol{q})$, $\boldsymbol{\mu}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, $\boldsymbol{p}(\boldsymbol{q})$ and $\bar{\boldsymbol{\epsilon}}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ are the projection in the task space of the inertia matrix, the Coriolis and centrifugal forces, the gravity forces and the unmodeled effects forces. $\boldsymbol{F}^{ext}$ is the vector of external forces applied to the plant at the end-effector level. These equations relate the task space acceleration vector $\ddot{\boldsymbol{\xi}}$, with a dimension less than or equal to 6, to the joint torques vector, with a dimension equal to n.

In general, inverse dynamics in the task space is a mapping from $\mathbb{R}^{2n} \times \mathbb{R}^m$ to $\mathbb{R}^n$ whereas inverse dynamics in joint space is a mapping from $\mathbb{R}^{3n}$ to $\mathbb{R}^n$. Poly-articulated systems have a task space dimension which is usually smaller or equal to the dimension of the joint space. Thus, the model of the inverse dynamics in the task space is generally smaller than the combination of the kinematics model with the standard inverse dynamics model.

3. Regression of Mechanical Models

In supervised learning, a class of machine learning techniques, a supervisor provides input/output pairs that can be used to learn a relationship between the corresponding training data. When the output is discrete, the *classification* problem consists in associating the presented input to an output value, its *class*. In contrast, when the output is continuous, the *regression* problem consists in finding a function that approximates as accurately as possible a *latent function* that describes the input/output relationship.

Formally, given a set of k training samples $\mathcal{S} = \{\boldsymbol{x_i}, y_i\}_{i=1}^k$, where $\boldsymbol{x_i} \in \mathcal{X} \subseteq \mathbb{R}^l$ is a vector of dimension l and $y_i \in \mathcal{Y} \subseteq \mathbb{R}$ the regression problem consists in approximating a latent function $f : \mathcal{X} \to \mathcal{Y}$ with a function $\hat{f}$ taken in some predetermined family so as to minimize some measurement of the approximation error.

For instance, the Least Square (LS) regression method considers the family of linear functions $f(\boldsymbol{x}) = \boldsymbol{\beta}^T\boldsymbol{x}$ where $\boldsymbol{\beta}$ is a weight vector minimizing the quadratic error

$$\min_{\boldsymbol{\beta}} \|\boldsymbol{y} - X\boldsymbol{\beta}\|^2, \tag{8}$$

where $\boldsymbol{y} = [y_1, \ldots, y_k]^T$ and $X = [\boldsymbol{x_1}, \ldots, \boldsymbol{x_k}]^T$ is a $k \times l$ matrix.

Learning a mechanical model of a plant is a regression problem where training samples are obtained from the state and controls of the plant along time.

A global view of the regression methods presented in this paper is given in Fig. 1. The next section describes early regression methods used to learn such models, from Artificial Neural Networks (ANNS) to Locally Weighted Regression (LWR) methods. Then we compare several state-of-the-art methods that all incrementally split the input space into regions and perform regression on local models associated to these regions.

In order to insist on what distinguishes these methods from their competitors and before discussing the impact of these differences, we focus on the following questions for all algorithms. How are the regions defined? How does the algorithm perform local regression? How do the region boundaries evolve? How does the algorithm add (or delete) a region? How does it compute the global output?

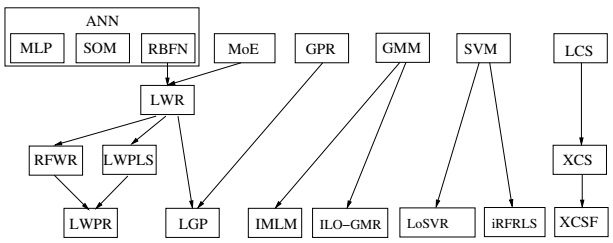

Figure 1: Historical view of several lines of research on supervised learning algorithms giving rise to incremental and local regression methods to infer mechanical models

3.1. Artificial Neural Networks

This section is dedicated to ANNS and their evolution towards LWR methods. ANNS are a wide class of non-linear function approximation tools. They are declined under different forms such as Multi-Layer Perceptrons (MLPS) [83], Self-Organizing Maps (SOMS) [53] or Radial Basis Function networks (RBFNS) [29]. Other forms of neural networks such as *population coding* are not covered in this paper because they are less relevant to the introduction of the subsequent sections, although they have been applied to robot model learning (see e.g. Butz et al. [12]). For

a classical survey about learning mechanical models with ANNs, we refer the reader to Jordan & Rumelhart [48].

In all the sections below, α is the learning rate, n_n is the number of neurons of a network and n_r is the number of regions.

3.1.1. Multi-Layer Perceptrons

An MLP approximates the latent function f through a directed graph of artificial neurons whose nodes convey a non-linear function and edges convey a weight that is tuned by the algorithm. The excitation level $v_i(\boldsymbol{x})$ of each neuron i is calculated as

$$v_i(\boldsymbol{x}) = \sum_{i=1}^{n_n} \beta_i x_i$$

where x_i is the input of neuron i and β_i is the associated weight. The activation function $y_i = f(v_i)$ uses the excitation level to determine if neuron i fires or not and is its output. Classical activation functions are $y_i = \tanh(v_i)$ or the sigmoid function $y_i = (1 + e^{-v_i})^{-1}$.

The learning process consists in computing the error of the global network output to update all weighted connections. The weights can be updated by back-propagating through the network an error δy_i between the desired value and the real output of each neuron i

$$\delta\beta_{ij} = -\alpha \frac{\partial \delta y_i}{\partial \beta_{ij}}. \tag{9}$$

where β_{ij} is the weight between input x_i and neuron i.

MLPs are used to learn the inverse kinematics (e.g. Ahmad & Guez [1], Barhen et al. [7], Pourboghrat [78]), the Jacobian matrix [56] or the forward kinematics model of a robot [11, 67]. More recently, Sang & Han [87] and Sadjadian & Taghirad [84] use ANN methods to learn the forward models of parallel robots which involves highly coupled non-linear equations that are difficult to model analytically.

3.1.2. Self-Organizing Maps

A SOM is another type of ANN. It is trained using unsupervised learning to produce a low-dimensional discrete representation of the input space. An important feature of SOMs is that they can preserve the topological properties of the input space.

Barretto et al. [8] provide a good survey of the application of SOMs to robot identification problems. Most authors use these methods in the context of visuo-motor control, apart from Walter & Schulten [109] and Wang & Zilouchian [110] who respectively learn the inverse kinematics of a six dofs industrial robot Puma 562 and the forward kinematics model of a simple plant. In this context, the evaluation is based on the accuracy of the model itself rather than on its control capabilities.

In the visuo-motor case, Ritter et al. [81] and Martinetz et al. [63] use SOMs to learn a mapping between the three-dimensional end-effector position measured by two cameras and joint positions on a three dofs robot. More recent work in the domain can be found in Kumar et al. [55].

Furthermore, specific SOMs called *Growing Neural Gas* are capable of expanding in the relevant dimensions of a domain. They have been combined with locally weighted learning methods presented hereafter [36]. The resulting algorithm is endowed with interesting properties for incremental function approximation in a large domain, but to our knowledge it has not been applied to the identification of mechanical models. Thus, we do not investigate the topic in more detail.

3.1.3. Radial Basis Function Networks

Radial Basis Function Networks (RBFNS), illustrated in Fig. 2, can be seen as a simplification over MLPs, retaining only one layer and using a Radial Basis Function (RBF) as activation function.

An RBF ϕ is a function whose output value $\hat{y}$ depends only on the distance r_i from its center c_i to the input value x. A standard RBF is a Gaussian function defined as

$$\phi(r_i) = \frac{1}{\sqrt{2\pi {\sigma_i}^2}}\, e^{-\frac{r_i^2}{2{\sigma_i}^2}} \tag{10}$$

where ${\sigma_i}^2$ is the variance of the Gaussian function i which determines its size and shape. The values of all σ_i are often set to a unique value σ. To perform regression with RBFNS, the output of all functions are weighted and summed together to get the global network output

$$\hat{y} = \sum_{i=1}^{n_n} \beta_i \phi(r_i) \tag{11}$$

where the β_i are weights.

In RBFNS, the weights are updated incrementally with different methods such as gradient descent based on the error between the output of the network and the desired value. As it can be considered as a one layer ANN, the back-propagation can be written as

$$\delta\beta_i = -\alpha(y_i - \hat{y}_i)\phi(r_i). \tag{12}$$

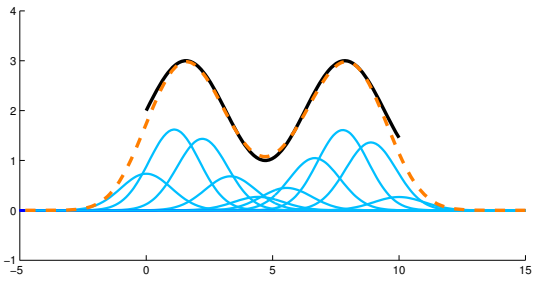

Figure 2: Approximation (doted) of a sinus function (top in black) with an RBFN (bottom in cyan).

In Sun & Scassellati [100, 101], an RBFN is used to learn the forward kinematics of a robot. Each basis function is derived to obtain a Jacobian matrix, using the Resolved Motion Rate Control (RMRC) [111] framework to generate trajectories. In Lin & Goldenberg [58], an RBFN is used to learn an inverse dynamics model of a 4 dofs manipulator in three dimensions.

3.1.4. Locally Weighted Regression

Locally Weighted Regression (LWR) methods [4] use Gaussian features like RBFNS, but the weights are approximated with the LS method (see Fig. 3).

The main difference between RBFNS and LWR methods lies in the introduction of a linear model associated to each RBF and used to perform LS regression. Each RBF defines a region of validity for the corresponding linear model.

More precisely, around each point $\boldsymbol{x}$, the contribution of each receptive field to the global output is computed with a Gaussian function (10). Thus LWR globally approximates non-linear functions by combining local linear models.

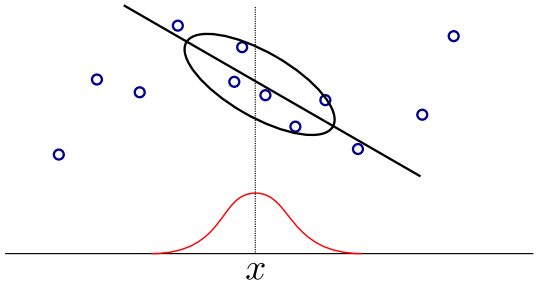

Figure 3: In LWR, $\boldsymbol{x}$ is the input of a latent function. A Gaussian, centered on $\boldsymbol{x}$, weights stored point with respect to their proximity to $\boldsymbol{x}$. A regression is then computed using the weighted points [26].

Compared to ANNS or RBFNS, LWR benefits from the power of linear estimation methods. But LWR needs to retain all training data to perform the LS approximation, which may be infeasible in large spaces given memory limitations. However, the linear models β can also be computed incrementally, using the Recursive Least Square (RLS) algorithm. This extension to LWR gave rise to RFWR [89], that is an incremental local regression method avoiding storage of data. In parallel, LWR was improved in another direction by LWPLS [91], that uses the Partial Least Square (PLS) [33, 117] projection mechanisms to neglect irrelevant dimensions in the input space.

3.2. Locally Weighted Projection Regression

Locally Weighted Projection Regression (LWPR) [108] is probably the most popular learning method for learning mechanical models of robots. As described in Schaal et al. [90], it can be seen as combining the properties of LWPLS and RFWR. Indeed, it performs simultaneously incremental regression of local linear models like RFWR and input projection like LWPLS. Furthermore, the shape of the region corresponding to each linear model is also tuned incrementally. LWPR provides a fast, incremental and reasonably accurate approximation, thus it is a good reference for comparisons.

Definition of regions. In LWPLS, RFWR and LWPR, a region is called a *receptive field*. Like its ancestors, LWPR uses Gaussian functions $\phi_i(\boldsymbol{x})$

$$\phi_i(\boldsymbol{x}) = e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{c}_i)^T W_i (\boldsymbol{x}-\boldsymbol{c}_i)} \tag{13}$$

where the positive distance metric W_i of the Gaussian delimits a region which is updated during learning to match the training data.

Local regression mechanism. As in LWR, each region has a corresponding local linear model β_i which is used to predict a local scalar output y_i relative to an input vector $\boldsymbol{x}$

$$\hat{y}_i(\boldsymbol{x}) = \beta_i \boldsymbol{x}. \tag{14}$$

LWPR inherits from the capacity of LWPLS to reduce the input dimensionality using PLS. Some projection is made on the input vector to model high dimensional functions only on their most relevant directions. Moreover, like RFWR, the linear model approximation is performed efficiently using an incremental version of PLS called NIPALS [117]. Several other ways to perform dimensionality reduction in the context of local linear regression are discussed in Vijayakumar et al. [106].

Evolution of regions. The distance metric W is adapted when new data are received to improve the coverage of the input space. It is computed as $W = M^T M$ where M is an upper-triangular matrix to ensure W is symmetric and positive definite. M is modified recursively and individually to regulate the size of the regions with the update law

$$M = M - \alpha \frac{\partial C}{\partial M} \tag{15}$$

where C is a minimized cost function. This cost function is more complex than the weighted mean square error criterion in LWR. It is computed by a leave one out cross-validation method and includes a penalty term that prevents the population of models from collapsing. The reader can refer to Schaal et al. [90] or Vijayakumar & Schaal [108] for a detailed presentation of the incremental version of the algorithm.

Note that, in contrast with the distance metrics, the centers $\boldsymbol{c}_i$ of the regions are not adapted. Their position depends on the mechanism used to add a new region.

Adding new regions. The rule for adding a new region is the following. For a given input $\boldsymbol{x}$, the Gaussian function $\phi_i(\boldsymbol{x})$ defining each region can be seen as a level of activation of the local model corresponding to this region. So, if, for the current $\boldsymbol{x}$, the level of activation of no region is above a threshold w_{gen}, a new region is created with $\boldsymbol{x}$ as center and an initial D_{init} as distance metrics. The value of w_{gen} plays a major role in the number of regions created for a given approximation problem.

Global model approximation. The approximation performed by LWPR is the sum of local linear models weighted by their respective Gaussian weights

$$\hat{y}(\boldsymbol{x}) = \frac{1}{\Phi(\boldsymbol{x})} \sum_{i=1}^{n_r} \phi_i(\boldsymbol{x})\, \hat{y}_i(\boldsymbol{x}) \tag{16}$$

where $\Phi(\boldsymbol{x}) = \sum_{i=1}^{n_r} \phi_i(\boldsymbol{x})$.

Furthermore, the gradient of the latent function, differentiated with respect to the input (and not with respect to time) and the confidence bounds on the predicted output are also provided through an incremental algorithm [52].

Applications to robot model learning. LWPR is certainly the most used algorithm in the context of learning mechanical models of a plant. As a result, it is used as a basis for comparisons in most if not all the papers presented in the next sections. Hereafter, we list a few of the most impressive applications of LWPR to learning mechanical models of robots.

In D'Souza et al. [32], an inverse kinematics model is learned with LWPR. The plant is an ATR humanoid robot [50] with 26 actuated joints among which only 4 are relevant for the learning algorithm. An end-effector is controlled in Cartesian space. Even though 22 of these dimensions are irrelevant, the dimension of the input (ξ, q) is 29 and the dimension of the output $\dot{q}$ is 26. The model is learned along a task space trajectory, which is much easier than learning in the whole space. The method solves redundancy by minimizing a cost function that depends on the configuration. Doing so, no inversion is involved and singularity problems are avoided.

In Schaal et al. [90], Vijayakumar & Schaal [107, 108], the authors use LWPR to learn the inverse dynamics model of a seven dofs anthropomorphic SARCOS robotic arm, again along a task space trajectory. The trajectory is followed with a task space $\mathcal{PD}$ controller. Some noise is added, making the plant explore and learn an envelope of possible configurations around the trajectory. The input of the learning algorithm is $(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})$ of dimension 21 and the output is the torque $\boldsymbol{\Gamma}$ of dimension 7. The number of training data points is 5 10^4 and the model contains 260 regions.

In Vijayakumar et al. [106], the same approach is used to learn the inverse dynamics model of the ATR humanoid robot. The input dimension is 90 and the output dimension is 30. Here, 7.5 10^6 samples and 2200 regions are required to obtain an accurate model. To our knowledge, this is the largest space in which a function has been learned with LWPR, but once again the function is learned along a trajectory in a small sub-part of the input space. Moreover, the number of joints used to control the arm is much smaller than the number of joints taken as input of the learning algorithm. Besides, since learning 30 parallel LWPR models would be computationally too expensive, the authors choose to learn one single model with 30 dimensional outputs projected, thanks to the PLS algorithm, onto the input data. Indeed, as a consequence of using the PLS regression, it is possible to learn one single model with a vector output instead of a scalar one.

More recently, LWPR was used in Peters & Schaal [75] to learn the inverse dynamics in the task space of an anthropomorphic SARCOS arm and a Mitsubishi PA-10 robot. For both plants, the input dimension is 17 and the output dimension is 7. As underlined in Section 2.4, those dimensions have to be compared to the size of two separate models which can be learned independently: an inverse velocity kinematics model which links a 3 dimensional space to a 7 dimensional space and an inverse dynamic model which links a 21 dimensional space to a 7 dimensional space. The inverse dynamics in the task space is also learned with LWPR in Sun de la Cruz et al. [27], where the system deals with redundancy by choosing a solution that minimizes the torque output used to control the plant.

Finally, in Mitrovic et al. [66], the inverse dynamics model of a two dofs planar arm actuated with three artificial agonist-antagonist pairs of muscles is learned with LWPR in the whole planar space. The input is $(\boldsymbol{q},\dot{\boldsymbol{q}},\boldsymbol{u})$ where $\boldsymbol{u}$ is the activation of muscles and the output is $(\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})$. The input dimension is $dim(\boldsymbol{q},\dot{\boldsymbol{q}},\boldsymbol{u}) = 10$ and the output dimension is $dim(\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}) = 4$. The authors use an optimal feedback control loop based on the ILQG algorithm [57] to find the muscular activations that minimize the muscular input while performing some reaching movements. After a learning period generating 1.2 10^6 training points, the model is composed of 852 regions and reaches three different targets.

3.3. Extended classifier system for function approximation

The XCSF algorithm is a regression method that shares many similarities with LWPR but comes from Learning Classifier Systems (LCSS), a family of adaptive rule-based systems first proposed by Holland [45] that combines reinforcement learning mechanisms [102] and genetic algorithms [40].

The immediate ancestor of XCSF is XCS [114], an efficient accuracy-based LCS designed to solve discrete classification problems and sequential decision problems. XCS finds accurate solutions to a wide range of problems with high probability in polynomial time [15]. XCSF [115, 116] is an evolution of XCS towards function approximation.

As any LCS, XCSF manages a population P of rules, called *classifiers*. These classifiers contain a condition part and a prediction part. The condition part defines the region of validity of a local model whereas the prediction part contains the local model itself. XCSF is a generic framework that can use different kinds of prediction models (linear, quadratic, etc.) and can pave the input space with different families of regions (Gaussian, hyper-rectangular, etc.). In this paper, we only consider the case of linear prediction models and Gaussian regions. In this case, the structure of the function approximation built by XCSF is similar to the one built by LWPR.

Definition of regions. An important specificity of XCSF is that it distinguishes a *prediction input space* and a *condition space*. The *prediction input space*, noted $\mathcal{X}$, corresponds to the input used to compute the linear models with regression. The *condition space*, noted $\mathcal{Z}$, is paved with Gaussian classifiers, corresponding to the regions of LWPR. A classifier defines a domain $\phi_i(z)$ in the following way

$$\phi_i(z) = e^{-\frac{1}{2}(z-\boldsymbol{c}_i)^T W_i (z-\boldsymbol{c}_i)} \tag{17}$$

where c_i is the center of the Gaussian i and W_i is a positive distance metric, inverse of the squared variance of Gaussian i. The input of the condition space, $z \in \mathcal{Z}$, defines the space where the Gaussian influences the global prediction.

The match set M is the subset of all reliable classifiers in the population P for which $\phi_i(z)$ is above a threshold ϕ_0[1].

Local regression mechanism. Each classifier has a corresponding linear model β_i which is used to predict a local output vector $\boldsymbol{y}_i$ relative to an input vector $\boldsymbol{x}$, provided an augmented state $\boldsymbol{x}_{aug} = \begin{bmatrix}\boldsymbol{x}^T & 1\end{bmatrix}^T$ to deal with the presence of an offset β_0.

The linear model is updated using the RLS algorithm, the incremental version of the LS method. If a classifier i makes an error $\delta\boldsymbol{y}_i = \boldsymbol{y} - \hat{\boldsymbol{y}}_i$, it is updated with $\delta\beta_i = \delta\boldsymbol{y}_i\boldsymbol{x}_i^T\alpha$.

Evolution of regions. A specificity of XCSF is that it uses a genetic algorithm (GA) instead of a gradient-based method to update its population P. As a result, there is no evolution of a region *per se*, but the GA rather uses a fitness function to add new regions similar to the old ones after applying mutation operators and deleting some regions. Here we explain how the fitness is obtained.

The prediction error of a classifier i approximates the mean absolute deviation of its prediction using the Widrow-Hoff delta rule [112]

$$\delta\boldsymbol{\epsilon}_i = \alpha\left(|\delta\boldsymbol{y}_i| - \boldsymbol{\epsilon}_i\right)$$

where $\boldsymbol{\epsilon}_i$ is the mean absolute error of the classifier i at the previous time step. Then the classifier accuracy is determined by the scaled inverse of $\boldsymbol{\epsilon}_i$. Classifiers with an $\boldsymbol{\epsilon}_i$ below a threshold $\boldsymbol{\epsilon}_0$ are considered trustworthy. Finally, the fitness value is derived from the classifier accuracy relative to all others in the match set M (see Butz & Herbort [17], Butz et al. [18] for algorithmic details).

Adding and deleting regions. In XCSF, a classifier is created when there is no classifier in the match set corresponding to an input. This mechanism is called *covering*, it creates a matching condition which corresponds to that input with a randomly generated condition space.

But classifiers can also be created by an evolutionary component based on a niched steady-state GA [116] improved with a set-size-relative tournament selection [16]. This favors the recombination of efficient classifiers by crossover and mutations. A GA is run if the time since the last GA invocation exceeds a threshold θ_{GA} to balance non-uniform problem-space sampling.

The reproduction of classifiers is performed within the match set M, whereas the deletion of inaccurate classifiers is performed over the whole population P. This combination of mechanisms ensures a good generalization capability [18, 114]. In particular, general classifiers are reproduced more often as they match the input data more often than the others.

In more detail, the GA selects in the current match set M two classifiers based on their relative predictive accuracy with respect to the other classifiers. The centers c_i^{new} of their offspring are moved within an interval $\|c_i - c_i^{new}\| \leq \sqrt{-2W_i \ln\phi_0}$ where ϕ_0 is the threshold used to generate the match set M. The shape of the ellipsoids, defined by σ_i which is the square-root of W_i, is also randomly increased or decreased within a limited interval.

[1]This threshold is named θ_m in Butz & Herbort [17]

Deletion takes place in the population when the number of classifiers in P reaches the maximum n_P^{max}. Supernumerary classifiers are deleted from P with a probability proportional to an estimate of the size of the match sets that they occur in. If a classifier is sufficiently experienced and its fitness F is significantly lower than the average fitness of classifiers in P, its deletion probability is further increased [54].

Finally, two additional processes that further optimize the population, namely subsumption and compaction, are described in Butz & Herbort [17].

Global model approximation. The global population P partitions the input space into a set of overlapping prediction models. XCSF generates the global approximation based on the current match set M. The learning output $\hat{\boldsymbol{y}}$ is given for a $(\boldsymbol{x}, z)$ pair as the weighted sum of the linear models of each classifier i in M

$$\hat{\boldsymbol{y}}(\boldsymbol{x}, z) = \frac{1}{\Phi(z)}\sum_{i=1}^{n_M}\phi_i(z)\,\hat{\boldsymbol{y}}_i(\boldsymbol{x}) \tag{18}$$

where $\Phi(z) = \sum_{i=1}^{n_M}\phi_i(z)$ and n_M is the number of classifiers in the match set M.

In sum, the evolutionary mechanism is designed to evolve partitions in which linear approximations are maximally accurate. Indeed, XCSF strives to evolve complete, maximally accurate, and maximally general population of local approximations [14]. A more complete description of XCSF can be found in Butz et al. [18] or in Butz & Herbort [17].

Applications to robot model learning. In Butz et al. [13], Butz & Herbort [17], Stalph et al. [97], the authors use XCSF to learn the forward kinematics model of a four dofs simulated arm in order to model motor adaptation in arm reaching experiments. They invert the model with classical analytical methods in order to control an end-effector in three dimensions while controlling the redundancy. They choose three different secondary tasks in the joint space which consist in minimizing angular velocities, avoiding joint limits or specifying a global posture.

3.4. Support Vector Regression

In Support Vector Regression (SVR), the regression extension of Support Vector Machines (SVMS), it is assumed that the latent function $f(\boldsymbol{x})$ can be parametrized as $f(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{w}$, where $\boldsymbol{\phi}$ is a feature vector mapping the input $\boldsymbol{x}$ into some high dimensional space and $\boldsymbol{w}$ is the corresponding weight vector [31, 69, 96]. The weight vector $\boldsymbol{w}$ can be represented as a linear combination of the input vectors in the feature space with coefficients β_i, i.e. $\boldsymbol{w} = \sum_{i=1}^{n_f}\beta_i\boldsymbol{\phi}(x_i)$, where n_f is the number of features.

Given this representation, the prediction $\hat{y}$ of an input $\boldsymbol{x}$ is

$$\begin{aligned}\hat{y} = \hat{f}(\boldsymbol{x}) &= \sum_{i=1}^{n_f}\beta_i < \boldsymbol{\phi}(x_i), \boldsymbol{\phi}(x) >, \\ &= \sum_{i=1}^{n_f}\beta_i k(\boldsymbol{x_i}, \boldsymbol{x})\end{aligned} \tag{19}$$

where $<,>$ denotes the dot product between two vectors and $k(\boldsymbol{x}, \boldsymbol{x'})$ is a kernel function. A standard kernel is the Gaussian function

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \sigma^2 e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{x'})^T W(\boldsymbol{x}-\boldsymbol{x'})}, \tag{20}$$

where W denotes the kernel width.

By comparing (19) with (16) and (18), one can see that the global structure of the approximation in SVMs is different from the one used in LWPR and XCSF. Whereas the former is a linear combination of kernel functions, the latter is a kernel-weighted combination of linear models.

The standard way to obtain the parameters β_i of the models in SVMs if through a quadratic programming optimization process based on all training samples [96]. There also exists a LS formulation called LS-SVM [103, 104]. In Fumagalli et al. [37], the authors are interested in obtaining an accurate estimation of internal forces in the arm of the humanoid robot JAMES. The arm of the robot is equipped with a force/torque sensor that estimates a total force from which the internal and external forces have to be determined. This in turn requires the estimation of a dynamics model, for which the authors compare LS-SVM and a batch neural network approach to the analytical derivation of the model from CAD parameters. The results show that LS-SVM converges faster than the neural network approach, even though the final performance is similar.

An important parameter for computational complexity is the number of support vectors, hence algorithms like ν-SVR use a specific meta-parameter to control this number [92]. Given its availability in the LIBSVM library [23], this method has been used as a baseline for performance comparisons in the context of comparisons for model learning methods [21, 72].

3.4.1. LoSVR

The focus of this survey being incremental and local approaches, we must mention that an SVM approach endowed with these properties and called LOSVR is presented in Choi et al. [24]. LOSVR is built on a previous incremental SVM algorithm presented in Ma et al. [61]. It is used to learn the inverse dynamics model of a simulated 2 dofs arm and then a real robot. However, the addressed problem is rather small and the authors are expecting poor generalization capabilities from their system, so we do not discuss it further in this paper.

3.4.2. Random Features Regularized Least Squares

An important concern in locally weighted regression methods is the model complexity, expressed in terms of the number of regions. A standard way to deal with this concern is called *regularization*. It consists in adding a term related to model complexity in the cost function that is optimized by the approximation method.

For instance, in *Regularized Least Squares*[2] (RGLS), the LS cost function (8) becomes

$$\min_{\boldsymbol{\beta}} \frac{\lambda}{2}||\boldsymbol{\beta}||^2 + ||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}||^2, \tag{21}$$

where $\lambda > 0$ is a regularization parameter that balances the trade-off between complexity and accuracy.

The optimal $\boldsymbol{\beta}$ can be obtained by setting the gradient of (21) to 0, which gives

$$\boldsymbol{\beta} = (\lambda \boldsymbol{I} + \boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{22}$$

Kernel Regularized Least-Squares (KRGLS)[3] is the extension of RGLS to non-linear functions using the same kernel trick as in SVMs [80]. Like GPR, the problem of KRGLS is its cubic cost in the number of samples.

In Rahimi & Recht [79], the authors propose to approximate the kernel matrix with a set of random features, giving rise to *Random Features Regularized Least Squares* (RFRLS), an algorithm that converges to KRGLS in $O\left(\frac{1}{\sqrt{D}}\right)$, where D is the number of features on which the kernel is projected.

Finally, in Gijsberts & Metta [39], the authors present an incremental version of RFRLS (named IRFRLS hereafter) that introduces the incremental estimation method of RLS in RFRLS. The complexity of this method is in $O\left(D^2\right)$, thus it is independent of the number of samples. Furthermore, the accuracy versus time complexity trade-off can be easily tuned by changing the number of features D. Experiments on several databases confirm that KRGLS is competitive with respect to GPR and that RFRLS and IRFRLS can be set close to KRGLS with a small enough processing time.

3.5. Gaussian Processes Regression

In SVR (Section 3.4), one is looking for a unique vector of weights $\boldsymbol{\beta}$ such that the approximation

$$\hat{f}(x) = \sum_{i=1}^{n_f} \beta_i \phi_i(x)$$

is as close as possible to the latent function f. Bayesian linear regression methods provide an alternative basis to this problem. Instead of just looking for a specific vector $\boldsymbol{\beta}$, they look for the posterior distribution over $\boldsymbol{\beta}$ given all the approximation samples and some prior. Given the resulting distribution, the best model can be determined either by averaging over the distributions of models or by selecting the Maximum A Posteriori (MAP) element of the distribution. As a result, such methods provide a more accurate but also more expensive way of finding an appropriate model.

As clearly explained in Williams [113], Gaussian Processes Regression (GPR) methods are mathematically equivalent to Bayesian linear regression. The main assumption in GPR is that the data presented to the learning process is corrupted by Gaussian noise with zero mean and variance σ_n^2.

An important property of Gaussian vectors is the following. If $n \geq 1$ and $\boldsymbol{X} = [X_1, \ldots, X_n, X_{n+1}]$ is a Gaussian vector of

[2] Not to be confused with Recursive Least Squares

[3] Again, not to be confused with KRLS

average $\mu = 0$ and whose variance $\sum$ can be decomposed in the following way:

$$\sum = \begin{bmatrix} K\boldsymbol{k} \\ \boldsymbol{k}^T k^* \end{bmatrix},$$

where K is a covariance matrix, $\boldsymbol{k}$ is a vector and k^* is a scalar, then

$E(\hat{f}(x_{n+1})|Y_1 = y_1, \ldots, Y_n = y_n) = \boldsymbol{k}^T K^{-1} \boldsymbol{y}$ and $Var(\hat{f}(x_{n+1})|Y_1 = y_1, \ldots, Y_n = y_n) = k^* - \boldsymbol{k}^T K^{-1} \boldsymbol{k}$ where $\boldsymbol{y}^T = (y_1, \ldots, y_n)$, $E(.)$ denotes the expectancy and $Var(.)$ denotes the variance.

If the latent function f is assumed to be a Gaussian process, a function $C : (x, x') \rightarrow E(\hat{f}(x), \hat{f}(x'))$ must be chosen to estimate the covariance matrix K out of data. Once this function is chosen and K is estimated, the approximation is obtained by just generating a Gaussian random variable of average $\boldsymbol{k}^T K^{-1} \boldsymbol{y}$, and of variance $k^* - \boldsymbol{k}^T K^{-1} k$ (see Williams [113] for details).

The advantage of GPR methods over Bayesian linear regression is that one does not need to define a set of feature functions over the input space. But, given the necessity to compute K^{-1}, standard GPR has $O(n^3)$ computation and $O(n^2)$ storage cost, where n is the number of samples used for approximation. Thus, as Williams [113] points out, GPR methods can be preferred only if n is not too big with respect to the number of feature functions that would be required to perform Bayesian linear regression.

3.5.1. Practical approaches to GPR

Given the cubic cost in the number of samples, there are two practical approaches to GPR. The *sparse* approach consists in limiting n by choosing adequately the points to remember and forgetting the rest. A good overview of this approach can be found in Quiñonero Candela & Rasmussen [20]. For instance, it has been used recently in *Sparse On-line Gaussian Processes* (SOGP), in the context of learning control policies from demonstrations [41]. The *mixture of experts* (MOE) approach rather consists in splitting the global domain of the latent function into smaller regions. In MOE methods, the regions are called *experts*. Indeed, if one splits the function domain into k regions, the standard GPR complexities are $k.O(n_i^3)$ and $k.O(n_i^2)$ respectively, where n_i is the number of samples that fall in region $i \in \{1, ..., k\}$.

This approach is also used to learn a policy from demonstration in Woods et al. [118], that describes an incremental version of *Infinite Mixture of Gaussian Process Experts* (IMGPE) [65] and compares it to LWPR. From the analysis of the authors themselves, even if their algorithm benefits from the Bayesian linear regression properties in contrast with LWPR and provides more accurate results is some contexts, it is still too slow to be used in an on-line robotics set-up. To our knowledge, this system has not been applied to learning mechanical models of a robot. The system presented below can be seen as a better compromise following this line of research.

3.5.2. LGP

Local Gaussian Processes (LGP) is a system that combines the accuracy of GPR methods on one region with the good capability of LWPR to split the latent function domain into regions [68, 72–74]. Many mechanisms being identical to those of LWPR, we only give a brief presentation, insisting on the points where both systems differ.

In LGP, regions are defined as in LWPR. Furthermore, the mechanism for adding a new region in LGP is identical to the one used in LWPR: a region is added if the distance of the current point to all activated region is more than a threshold w_{gen}.

By contrast, the mechanisms for the evolution of regions differ. In LWPR, the parameters W determining the shape of the regions evolve whereas the centers of all regions are fixed. In LGP, the parameters W are fixed whereas the centers are moving each time a region receives a new point, the center being the barycenter of all the points associated to the region.

The linear approximation provided in each region is based on a local GPR method. In that context, the covariance function used is the standard Gaussian kernel function (20).

In order to compute the global model output, instead of using the models of all regions as done in LWPR, LGP uses only the M regions whose center is closest to the current point.

In order to prevent the $k.O(n_i^3)$ computation cost to get too expensive, once a threshold is reached, the algorithm removes the points that bring the least information. In [69], LGP is augmented with an on-line sparsification algorithm inspired from what is used in *Kernel Recursive Least-Squares* (KRLS) [34]. This additional process addresses the same issue as SOGP, i.e. the incremental selection of samples that are used to train the local Gaussian processes.

In terms of application to robot model learning, in Nguyen-Tuong et al. [72], the authors compare the performance of LGP with LWPR, standard GPR and ν-SVR (see Section 3.4). The comparison is performed on learning the inverse dynamics in the task space of a seven dofs SARCOS arm. It shows that LGP is competitive and provides a good trade-off between LWPR and GPR.

Finally, in Nguyen-Tuong et al. [74], the authors also show that using LGP could give rise to better results than just using the parametric model provided by the robot manufacturer of a torque controlled manipulator.

3.6. Gaussian Mixture Models

A *mixture model* is a probabilistic model for representing the presence of sub-populations within an overall population, without requiring that an observed data-set should identify the sub-population to which an individual observation belongs. In that context, the Expectation-Maximization (EM) algorithm [30] can be used to make statistical inferences about the properties of the sub-populations given only observations on the global population.

The EM algorithm is an iterative method for finding maximum likelihood or MAP estimates of parameters in statistical models, where the model depends on unobserved latent variables. EM alternates between performing an *expectation* (E) step and a *maximization* (M) step. The E step computes the expectation of the log-likelihood evaluated using the current estimate for the latent variables. The M step computes parameters

maximizing the expected log-likelihood found on the E step. These estimates are then used to determine the distribution of the latent variables in the next E step, up to convergence. In the mixture model context, the latent variable in the EM algorithm is the identity of the sub-population to which a specific data point should be attributed.

The specific computation of the E step and the M step depends on the underlying model. In *Gaussian Mixture Models* (GMMS), the population is a predetermined collection of Gaussian regions, thus the function approximation is built as in RBFNS [38].

3.6.1. ILO-GMR

In the context of machine learning methods for robotics, GMMS are used mostly as an alternative to GPR methods for learning policies by demonstration (see Calinon [19] for a review). The corresponding methods are called *Gaussian Mixture Regression* (GMR), they are cubic in the number of samples, like GPR.

ILO-GMR, presented in Cederborg et al. [21], is one such method. However, one section of the paper is devoted to the comparison of ILO-GMR with LWPR, GPR, ν-SVR, GMR, and LGP, to the problem of approximating the inverse velocity kinematics model of a SARCOS robot arm using the database provided by the authors of Nguyen-Tuong et al. [72]. The results show that ILO-GMR performs close to GPR and slightly better than LGP and standard GMR.

With respect to standard GMR, the main advantage of ILO-GMR lies in an efficient partitioning of the set of samples into regions based on an algorithm similar to *kd-trees* described in Angeli et al. [2]. This allows one to break the cubic complexity of GMR and to quickly access the output for a given input.

In all other respects, ILO-GMR is identical to standard GMR, thus the model is defined as a fixed collection of Gaussian regions whose weights are trained with an EM algorithm.

3.6.2. Incremental Mixture of Linear Models

All the methods presented so far are approximating mechanical models as a mapping between an input space and an output space. This mapping is a function that returns a unique output for a given input. It can represent a forward model or an inverse model depending on which variables are used as input and which are used as output. Moreover, two functions may represent the inverse of each other if part of the input and the output are exchanged in the learning process.

In contrast with this standard approach, the method presented in Lopes & Damas [60] approximates a *manifold* in the space spanned by all input and output variables. This manifold can represent both a forward and an inverse model at the same time. It corresponds to a subspace that is compatible with the constraints generated by the considered model. Formally, if we call $\boldsymbol{\nu}$ a $[\boldsymbol{x}^T\ \boldsymbol{y}^T]^T$ point in the global space, the manifold can be represented by a constraint $H(\boldsymbol{\nu}) = 0$ that imposes the restriction that the point $\boldsymbol{\nu}$ must comply with the model.

For instance, let us consider the kinematics model. The concatenation of any $\boldsymbol{q}$ vector and the corresponding $\boldsymbol{\xi}$ vector of task space position defines a point in a $\boldsymbol{q} \times \boldsymbol{\xi}$ space. All such points are lying on the manifold that characterizes the kinematics of the corresponding plant.

In practice, the authors are trying to approximate this manifold using a GMM approach. They try to estimate the probability of a point $\boldsymbol{\nu}$ belonging to the manifold, *i.e.*,

$$p(H(\boldsymbol{\nu}) = 0|\boldsymbol{\nu}_1, \boldsymbol{\nu}_2, \cdots, \boldsymbol{\nu}_N).$$

Definition of regions. More precisely, the manifold verifying $H(\boldsymbol{\nu}) = 0$ is approximated with a collection of regions that are defined as Gaussian functions with a center $\boldsymbol{c}_m$ and a local covariance matrix $\boldsymbol{C}_m$. The probability of a data point $\boldsymbol{\nu}_i$ belonging to a region m is noted as $p(\boldsymbol{\nu}_i|m)$.

Evolution of regions. The evolution of the parameters $\boldsymbol{c}_m$ and $\boldsymbol{C}_m$ of these regions is driven by an incremental implementation of the EM algorithm (see Lopes & Damas [60] for the equations realizing this incremental implementation).

Adding and deleting regions. The system starts with only one region and adds new regions when necessary. More precisely if, for some point $\boldsymbol{\nu}$, $p(\boldsymbol{\nu}|m)$ is below a threshold p_{gen} for all regions, then a new region n is added centered on this point $\boldsymbol{\nu}$ and with a covariance initialized to maximize $p(\boldsymbol{\nu}|n)$.

The system also computes an adjacency matrix between all pairs of regions based on the idea that if, for a point $\boldsymbol{\nu}$, $p(\boldsymbol{\nu}|m_1)$ and $p(\boldsymbol{\nu}|m_2)$ are both above a threshold, then m_1 and m_2 are very likely to be adjacent.

Global model approximation. The global probability of a data point $\boldsymbol{\nu}_i$ belonging to the manifold verifying $H(\boldsymbol{\nu}) = 0$ results from a combination of the probabilities of all local regions. The difficulty here is that the system must provide either a forward or an inverse model depending on the need of the controller. Thus it is designed to answer to a *query*: given a specification of some coordinates of a point, what are (if any) the points lying on the manifold that match this specification?

For a given query $\boldsymbol{\nu}_q$, after simple mathematical calculations, the system can provide for each region m the corresponding answer $\hat{\boldsymbol{\nu}}_{am}$ with a degree of confidence given by $p(\hat{\boldsymbol{\nu}}|m)$, where $\hat{\boldsymbol{\nu}} = [\boldsymbol{\nu}_q^T\ \hat{\boldsymbol{\nu}}_{am}^T]^T$.

But, in many cases, there will be either several discrete possible answer values or a continuous region corresponding to an infinity of such potential answers. Thus, a simple weighted average over the answers of all reliable regions does not provide a satisfactory solution.

Instead, the system uses the adjacency relationship described above to group together the answers of connected component and it provides one averaged answer and the corresponding degree of confidence for each group. By doing so, a set of answers may sample a continuous space of solutions.

Applications to robot model learning. To our knowledge, this system has not been applied so far to a robot. It was just compared with LWPR in simulated experiments. For the sake of easy reference, we call this system IMLM (for *Incremental Mixture of Linear Models*) below.

System	LWPR	XCSF	iRFRLS	LGP	IMLM	ILO-GMR
Based on	LWR	LCSs	SVR	GPR + LWR	GMMs	GMR
Main feature	Dimensionality reduction	Prediction vs. condition space distinction	Constant computational complexity	Sparsification	Bidirectional mapping	Use of *kd-trees* for fast access
Other assets	Available in C++ and matlab with cookbook. Fast. Widely used	Available in Java. Flexible	Simple. Easy tuning	Less meta-parameters than LWPR	No matrix inversion required	Simple
Drawbacks	Hard to tune. Initialization stage required	Complex. Sensitive to tuning	No improvement of features	Slower than LWPR	Less mature, not used on real robots	Fixed number of features
Favorite context of use	Large systems, along a trajectory, with dynamics	Learning on whole space	Learning on whole space	Sparse data, requiring accuracy	Redundant and parallel systems	Learning on whole space

Table 1: Summary of the discussion of the systems

4. Discussion

All the systems studied in Section 3 have been applied to learning the mechanical model of some plant. However, many other regression algorithms exist that could be applied to this problem. In particular, the regression methods used to learn a control policy from demonstration can generally be used as such to model identification problems. In that category, we already mentioned SOGP and IMGPE, for instance. We did not present this broader class of systems because it is well covered in Argall et al. [3], Calinon [19].

In this section, we focus on the incremental and local regression algorithms used for learning a mechanical model of robots presented in more details above. We do not discuss anymore the ANNS approaches that are not local because they do not fall into the unified view that is presented below. In practice, we focus on LWPR, XCSF, iRFRLS, LGP, IMLM and ILO-GMR. The purpose of this section is to compare the systems by first providing a unifying view before discussing their strengths and weaknesses from the presentation of their properties. This part of the discussion is summarized in Table 1. Finally, we investigate from the unifying view whether their specific features could be combined into a new generation of systems.

4.1. Performance comparisons

Several papers present a performance comparison between some of the systems reviewed here [21, 39, 72, 74]. These comparisons are made possible thanks to the availability of a SARCOS and a BARRETT arm database first published in Nguyen-Tuong et al. [72]. The papers generally compare the accuracy of the models, measured with a standard NMSE calculation and sometimes the prediction time as a function of the number of samples.

However, the corresponding results are to be taken with care. Indeed, most of the algorithms are sensitive to meta-parameter tuning and there is no standard way to accurately perform this tuning so that all algorithms are given a fair chance. As a consequence, the same algorithm is attributed different performances in different papers.

A striking outcome of a survey of these results is that, despite its popularity, LWPR is generally less accurate than most of its competitors. This paradox can be explained by the fact that LWPR obtained impressive results when learning along a trajectory for very large plants whereas in the comparisons listed above it is used for learning over the whole space for smaller plants. We refer the reader to the list of papers above for other performance comparisons.

4.2. A unifying view

From the survey of the systems presented in Section 3, a unifying view emerges. We consider four aspects of these systems: the structure of the model, the way this structure evolves or not, the local regression method used and finally the meta-parameter tuning issue.

4.2.1. Structure of the models

If we consider Gaussian regions and linear models for XCSF, and Gaussian kernels for iRFRLS, then all the systems listed above include a Gaussian component and a linear component in their model. However, two broad classes of different structures must be distinguished. In LWPR, LGP and XCSF, the model is made of a Gaussian weighted combination of linear models, whereas in iRFRLS, IMLM and ILO-GMR, it is made of a linear combination of Gaussian kernels. In that respect, the first class of systems is endowed with a more flexible structure, as already pointed out about LWR in Section 3.1.4.

4.2.2. Evolution of the structure

Table 2 summarizes in what respect the structure of the model can evolve in different systems. The systems are organized from the less flexible one, iRFRLS, where regions are generated randomly once and for all to the most flexible one, XCSF, where all elements of the structure can evolve.

4.2.3. Regression method

All the algorithms listed above perform local regression based on the minimization of an LS criterion, but the way to

System	iRFRLS	ILO-GMR	LGP	LWPR	IMLM	XCSF
Addition of regions			•	•	•	•
Deletion of regions						•
Moving centers		•	•		•	•
Moving shapes		•		•	•	•

Table 2: Capabilities of the systems to evolve the structure of the model. A • indicates that the system is endowed with the corresponding capability

perform this minimization differs. As clearly stated in Nguyen-Tuong & Peters [69], standard SVR approaches rely on a batch quadratic programming optimization process to get the linear parameters whereas GPR gets them through a matrix inversion (see Section 3.5). iRFRLS is an SVR algorithm, but the batch optimization is replaced by an incremental version known as the QR algorithm in the field of adaptive filtering [88]. ILO-GMR relies on a standard EM algorithm. Finally, XCSF uses standard RLS whereas LWPR relies on the more sophisticated NIPALS algorithm.

4.2.4. Tuning meta-parameters

In order to approximate an arbitrarily complex function in a large space, one must deal with a trade-off between the accuracy of the approximate model and the number of regions used to perform this approximation. Dealing with this trade-off is a matter of meta-parameter tuning. Different algorithms use a different strategy in that respect.

- In iRFRLS, the only meta-parameter is the number of Gaussian kernels. This makes it very easy to tune and very fast, but this approach should meet its limitations when many kernels are required
- In LWPR and LGP, the indirect parametrization is based on setting in advance the size of the regions through the w_{gen} parameter. This is a poor compromise since one must try a w_{gen} value, monitor the number of regions and the accuracy, and try again as long as the number of regions is too large or the accuracy too low.
- IMLM sets a threshold p_{gen} on the accuracy of the local models. Using this threshold provides guarantees in terms of accuracy, at the expense of computational complexity if a segmentation into many regions is necessary to reach the threshold.
- XCSF rather sets a maximum number of regions to perform the approximation. Such parametrization provides a good grip on the computational complexity of the process, at the expense of accuracy. But XCSF also uses a threshold on the global accuracy of prediction. Indeed, if the maximum population size is reached, the algorithm tries to improve individual classifiers in the population until the accuracy threshold is reached. However, if both constraints are not set with care, the process may diverge due to incompatible evolutionary pressures.

4.3. Key features of systems

The unifying view presented above reveals some of the features on which the systems can readily be compared. We can now highlight the strengths and weaknesses of these systems and exhibit the features that provide them with a unique advantage over their competitors.

4.3.1. LWPR

There are several reasons for the popularity of LWPR. First, a good documentation and an efficient and versatile code are available [46]. Second, LWPR has been convincingly applied to the identification of diverse models (kinematics, velocity kinematics and dynamics, forward and inverse) of large mechanical systems such as humanoid robots. This success was possible because LWPR is fast, reasonably accurate and because it performs dimensionality reduction in the prediction space of each local model. This dimensionality reduction feature is unique to LWPR among the systems we presented. Apart from its very appealing features, LWPR also suffers from several important limitations.

First, it comes with many meta-parameters and its performance is sensitive to their tuning. To temper this criticism, Klanke et al. [52] provides a useful "cookbook" to help determine how the meta-parameters should be tuned.

A second problem with LWPR is initialization. Because the centers of the region do not move during the learning process, LWPR is sensitive to the samples received in the first iterations. Thus, in order to get a good prediction accuracy with LWPR, it is necessary to start with an initialization stage that must ensure a good coverage of the latent function domain. However, this initialization stage can be difficult to realize in a concrete robotics set-up if the system is not endowed with the model it is supposed to learn.

Third, all the impressive results published on plants with many dofs have been obtained along some specific trajectory in the state space of the plant which makes the problem easier but also less relevant. It is easier because learning along a trajectory dramatically reduces the domain that has to be split into regions. With LWPR, covering a domain requires a number of regions that is roughly exponential in the number of dimensions of the domain: the dimensionality reduction property provided by NIPALS is applied to the local model in each region but does not result in a significantly lower number of regions. Furthermore, experiments show that the performance of LWPR collapses as soon as the number of regions gets above a few tens of thousands. The problem is also less relevant because controlling a plant along an articular trajectory prevents making profit of redundant dofs. To face this difficulty, the learning mechanism has to be endowed with a good extrapolation capability, which is generally not the case with the local methods studied in this paper in general and with LWPR in particular.

4.3.2. XCSF

XCSF shares a lot of similarities with LWPR in the structure of the learned models. However, it differs in the algorithms used to update this structure and in the meta-parameters that constrain

the algorithms. A specific comparison between both systems has shown that XCSF generally outperforms LWPR in standard function approximation problems, even if LWPR usually converges faster [98]. Among other things, XCSF can be tuned more intuitively and its evolutionary component can modify the centers of the approximation regions. Given that XCSF is available in Java [119], the main reason for its lesser use in the model learning community is probably its lesser maturity in the field.

But the unique feature of XCSF is the condition space versus prediction space distinction. This feature can result in important savings in the context of learning mechanical models of a plant. We illustrate these savings in the context of learning the forward velocity kinematics model and the inverse dynamics model.

According to (1), learning a forward velocity kinematics model with LWPR requires the joint positions and velocities $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ as input and the task velocity $\dot{\boldsymbol{\xi}}$ as output. But, taking a closer look at the equation $\dot{\boldsymbol{\xi}} = J(\boldsymbol{q})\,\dot{\boldsymbol{q}}$, one can see that it relates linearly $\dot{\boldsymbol{\xi}}$ to $\dot{\boldsymbol{q}}$ for different contexts based on $\boldsymbol{q}$. Thus one can use the values of $\boldsymbol{q}$ to define the condition space and the corresponding relation between $\dot{\boldsymbol{\xi}}$ and $\dot{\boldsymbol{q}}$ to learn the local models in the prediction space.

Instead of segmenting a $(\boldsymbol{q}, \dot{\boldsymbol{q}})$ space into regions to approximate the forward velocity kinematics function, one just needs to segment a $(\boldsymbol{q})$ space. As a result, the condition space is twice smaller with XCSF than with any other local regression method.

The same is true when one considers learning an inverse dynamics model. Equation (5) describing the dynamics of a poly-articulated system can be reformulated

$$\boldsymbol{\Gamma} = A(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \boldsymbol{B}(\boldsymbol{q}, \dot{\boldsymbol{q}})\,. \tag{23}$$

This equation can be viewed as a linear relation between $\ddot{\boldsymbol{q}}$ and $\boldsymbol{\Gamma}$ with dependencies on $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$

$$\boldsymbol{\Gamma} = \begin{bmatrix} A(\boldsymbol{q}) & \boldsymbol{B}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ 1 \end{bmatrix}. \tag{24}$$

In this context, $\boldsymbol{B}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ can be seen as an offset. Thus, similar to the velocity kinematic case, one can learn an inverse dynamics model by providing at each time step a state $(\boldsymbol{q}, \dot{\boldsymbol{q}})$ as condition parameters and a $(\ddot{\boldsymbol{q}}, \boldsymbol{\Gamma})$ pair as input to the prediction space.

This approach is less expensive in terms of dimensionality than the standard one. Indeed, one needs to span $(\boldsymbol{q}, \dot{\boldsymbol{q}})$ instead of $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$ to learn the model. As a result, the space is $1/3$ smaller.

In Section 3.1.4, we noted that the major improvement in function approximation techniques brought by LWR has consisted in separating a linear model from the radial basis function that delimits the corresponding region of influence. In XCSF, the idea is put one step further: since those entities are separated, they do not need to be based on the same dimensions, thus the global input vector can be split into a condition part and a prediction part. The autonomous determination of which input should be used in the condition part, in the prediction part or just ignored is a matter for future research.

4.3.3. IRFRLS

The main feature of IRFRLS is its constant computational complexity. This property results from the segmentation of the latent function domain into a set of random kernels, which is the simplest possible mechanism. The number of kernels is the only meta-parameter that needs to be tuned. On the data sets presented in Nguyen-Tuong et al. [72] and other data sets extracted from the JAMES and ICUB humanoid robots, IRFRLS obtains a surprisingly good performance compared to its competitors. However, the random kernel approach obviously results in a suboptimal structure of the model, which may become critical when the model has to be very large. This system being very recent, more work is required to better determine its empirical properties, but given its simplicity, it is a good starting point for a new family of systems.

4.3.4. LGP

LGP was designed after LWPR and has been compared favorably to its ancestor in several contexts [72, 74]. One difficulty with LWPR being meta-parameter tuning, a strong incentive for calling upon GPR methods is that they generally come with much less meta-parameters. Moreover, by splitting the domain of approximation into regions, LGP breaks down the computational complexity of standard GPR methods. It is still slower than LWPR, but it also inherits the greater accuracy of GPR. However, some meta-parameters inherited from LWPR for the segmentation into regions such as w_{gen} are still difficult to tune.

Because it performs local regression based on a GPR process, LGP is strongly driven towards the use of sparse data for each region. Sparsification is the main feature brought into this survey by LGP.

Along this line of research, one may wonder if using standard Bayesian linear regression instead of GPR in a mixture of experts approach would not result in a more efficient approach, the features necessary for Bayesian linear regression being provided by the segmentation into regions.

4.3.5. IMLM

As explained in Section 2.2, the forward velocity kinematics model of a redundant plant is unique whereas there exists an infinity of corresponding inverse models. Conversely, for a parallel plant, the inverse velocity kinematics model is unique whereas there is an infinity of forward models. A first appealing property of the approach of Lopes & Damas [60] is that it can store all such models from data without knowing in advance if the model is unique nor losing the information corresponding to the potential infinity of solutions.

A second appealing property is that no matrix inversion is necessary to retrieve either a forward or an inverse information from the stored data. Intuitively, in the case of kinematics, one may provide a vector $\boldsymbol{q}$ or a vector $\boldsymbol{\xi}$ and retrieve the complementary part by looking for all points in the manifold that share the specified coordinates.

Nevertheless, retrieving the required information from such a manifold is not straightforward. The manifold being approximated by a set of Gaussian regions, the system returns one value

for each region whose probability of belonging to the manifold is above a threshold. As a consequence, when there is an infinity of correct answers to a query, the system only returns a set of discrete answers sampling the domain of correct answers. Furthermore, the accuracy of the sampling is sensitive to the size of the regions.

Finally, when there are several correct answers, one must take additional constraints into consideration to choose a specific answer among the possible ones. Despite a first solution proposed in Lopes & Damas [60], this point is still an open problem.

4.3.6. ILO-GMR

ILO-GMR is a straightforward extension of GMR systems designed to perform learning from demonstration. It is a simple approach, whose main drawback is that the number of kernels have to be determined in advance. However, an interesting feature of ILO-GMR is its use of *kd-trees* to organize the regions and the corresponding samples so as to quickly access them. An extension of GPR with *kd-trees* has also been proposed in Shen et al. [93].

4.4. Combining Features from Several Systems

In the previous section, the key features of the studied systems have been highlighted. Now, we discuss whether these features could be combined in order to build a new generation of systems.

4.4.1. Dimensionality Reduction

The incremental dimensionality reduction property provided to LWPR by the NIPALS algorithm is a key feature of LWPR. It is particularly adequate for learning models along trajectories in spaces with many dimensions, some of which being irrelevant along the given trajectory. XCSF learns the same kind of linear models as LWPR, using RLS instead of NIPALS. It seems that NIPALS could replace RLS in XCSF without major modifications of the rest of the algorithm. Whether a similar dimensionality reduction property could be introduced into GPR or GMM based methods such as LGP and IMLM is a more difficult open issue.

4.4.2. Condition Space versus Prediction Space Distinction

As outlined above, LWPR, LGP and XCSF both build their model as a Gaussian weighted combination of linear models. As a consequence, the distinction between prediction space and condition space used in XCSF could certainly be transferred to LWPR and LGP. In contrast, such a transfer would be much less immediate for all the other systems that use a different structure for the approximation of the latent function.

4.4.3. Learning the Input-Output Manifold

Learning the $input \times output$ manifold is unique to IMLM. All the other methods are designed to approximate a function rather than a manifold. Anyway, even if replacing the GMM foundation of that system by other regression methods such as the one used in XCSF or LGP is probably an option, more work is required on top of IMLM in order to efficiently use its unique feature.

4.4.4. Meta-parameter optimization

From Section 4.2.4, the meta-parameters used in IMLM and XCSF look easier to tune than the one used in LWPR and LGP, with a slight advantage to XCSF. Whether the approach used in XCSF can be transposed to the other systems is an open question.

Alternatively, one may think of dealing with the accuracy versus number of regions trade-off with a multi-objective function optimization approach. Such an approach would generate a population of non-dominated solutions among which a very accurate solution, a very sparse one or any compromise between both extrema could be chosen by the user.

4.4.5. Sparsification and fast access to samples

It seems that the sparsification algorithm proposed in LGP is not specific of this algorithm in any way. Thus, it could probably be included in most of the other systems, giving rise to immediate computational savings.

Similarly, the fast access to samples brought by the use of *kd-trees* in ILO-GMR could certainly be imported in most if not all of the systems listed in this survey.

A new system that incorporates these properties as well as some of those that we listed in the previous sections remain to be built, giving rise to a new generation of more efficient systems.

5. Towards more advanced systems

Most of the experimental work presented so far consists in learning mechanical models of a robot without any interactions with its environment. In this context, the resulting benefit is limited. Indeed, the mechanical structure of a robot is generally almost perfectly known through the CAD model, thus a good engineering work and an eventual additional parametric identification process can suffice to get an accurate model.

Obviously, the methods presented in this paper are becoming more advantageous when they address the robotics contexts outlined in the introduction. Future robots will interact physically with unknown objects and will even do so with unknown tools given the fast development of dexterous manipulation capabilities [10, 42, 49]. Even more convincingly, they will interact with human users that are highly unpredictable. In all these cases, a model provided in advance cannot generally account for unspecified interactions. In this section, we first stress the challenges raised by these more difficult contexts of use, then we survey some new lines of research that start addressing them.

5.1. New Challenges

To face the challenges of interactions with the environment, the next generation of systems will have to address problems with many more dimensions, they will have to adapt faster, and they will have to memorize several models corresponding to different contexts of use, together with the capability to switch between these models appropriately.

5.1.1. Challenge 1: Plants with more dimensions

Among the algorithms surveyed here, LWPR is the only one that has been used on a plant as complex as a humanoid. But only one arm of the robot was moving and, above all, learning was performed along a specific trajectory. Learning kinematics and dynamics model of a more than 10 dofs robot over the whole space is still beyond the capabilities of all state-of-the-art methods. However, for a humanoid engaged in a complex everyday-life mission, having such a complete model is necessary. A key property for addressing these larger problems without facing hard computational complexity challenges is generalization. Future systems should definitely be able to extrapolate the models they are learning beyond the domain where they were trained, which is generally not the case of the local learning methods surveyed in this paper.

5.1.2. Challenge 2: Faster adaptation

A barely addressed issue in all the papers surveyed here is speed of adaptation. Indeed, when a robot interacts with different objects or users, one would like the model to adapt to a new context of interaction fast enough to deal immediately with this new context. For instance, if a robot is pouring water from a container, the weight is changing quickly. Or, if a robot is in physical interaction with an user, the force exerted by the user may vary at any moment.

It is hard to accurately present the state-of-the-art on that topic in the absence of any systematic study, but the methods presented in this survey are not equivalent in that respect and are probably slower than what is required for the kind of non-stationary interactions just described above.

5.1.3. Challenge 3: Switching between contexts

If a robot has to physically interact with several users or objects, eventually with some tools, the systems described in this paper must be enriched with two capabilities: the capability to learn different models corresponding to the different contexts of interaction and the capability to switch between these models appropriately. Otherwise, the system would spend a lot of time re-learning the same models back and forth. An additional requirement is that the system is able to recognize which model it should be using in a given context or to determine that a new model is necessary.

5.2. New Lines of Research

Corresponding to the above challenges, two of the lines of research below consist in incremental improvements of the existing systems whereas the third line consists in an extension of the addressed domain. However, each of the two first lines contributes to all challenges simultaneously.

5.2.1. Line 1: Combining knowledge and regression

A way to increase the speed of adaptation and to deal with larger plants is to rely more on expert knowledge about the plant. Indeed, the model learning approach does not benefit from prior knowledge of the RBD model of the plant, though this knowledge is generally available. Recently, the authors of Nguyen-Tuong & Peters [70] combined the strengths of RBD identification and machine learning methods, using GPR for learning the difference between the parametric mapping and the observed behavior.

Their method provides a higher model accuracy and better generalization for unknown trajectories compared to RBD identification and GPR-based methods. Furthermore, it shows a good learning performance even on sparse and noisy data.

Though their results are promising, it is still unclear how prior knowledge helps learning a robot dynamics, notably depending on the control law used to actuate the robot. Furthermore, their semi-parametric regression approach does not alleviate the methodological difficulty of performing identification, which, in most cases, is not straightforward.

A different approach to the same problem is presented in Sun de la Cruz et al. [28], where the authors incorporate prior knowledge of the RBD model of the plant in LWPR by initializing all linear models with a first order approximation of the known RBD. The application to a simple simulated system convincingly improves the learning and generalization performance.

Other methods based on visual processing such as Hersch et al. [43], Mansard et al. [62], Martinez-Cantin et al. [64], Sturm et al. [99] incorporate prior knowledge of the structure of the body of the robot. These methods are surveyed in Hoffmann et al. [44]. The promising line of research corresponding to all these works should be pursued whatever the model learning method used.

With a very different perspective, the system presented in Ulbrich et al. [105] learns the forward kinematics model of mechanical systems using Bezier curves. Their method is based on the insight that the forward kinematics model of a mechanical system with rotational joints mainly calls upon trigonometric functions. It spans efficiently the parametric space of such functions in order to get an accurate model from sparse data.

Though this method does not rely on a parametric model of the robot in the classical sense, it is based on a strong assumption about the structure of the model. On the one hand, this assumption makes the parameter tuning more efficient. On the other hand, it makes the method less general. In particular, it can only be efficient when learning the forward kinematics model of a plant whose dofs all result from rotational joints. The extension to prismatic joints and to dynamics must call upon additional assumptions.

5.2.2. Line 2: Active learning

Different ways to span the input/output space can result in a large difference in the time necessary to learn an accurate model over the whole space. Thus it can increase both the speed of adaptation and the size of the problems that can be addressed. In Machine Learning, choosing appropriately the set of learning examples so as to increase the learning speed is called *active learning*. Active learning of a mechanical model is a difficult topic because it generates a "chicken and egg" problem. Indeed, in order to choose the samples that feed the learning process, it is necessary to know how to drive the plant towards the corresponding states. But to do so, having an accurate model is generally required. A way to escape this difficulty consists in

calling upon model-free control methods such as articular PID controllers to drive the system towards areas of interest. So far, there are few attempts to design specific active learning methods for learning mechanical models of robots, exceptions being Baranes & Oudeyer [5, 6], Martinez-Cantin et al. [64], Robbel [82], Saegusa et al. [85]. Given the growing interest in active learning in general, important progress can be expected in this domain in the near future.

5.2.3. Line 3: Switching models and sharing between models

In Petkos et al. [76], Petkos & Vijayakumar [77], the authors address the problem of switching between models raised in Section 5.1.3. They rely on LWPR combined with an EM algorithm to learn elementary models and use a latent variable to account for the different unknown contexts of interaction. So far, their work is just a proof of concept. It should be extended with perception processes to help the system recognizing the different contexts based on richer information than just the articular state of the robot. Along a different line of research, given the infinity of inverse velocity kinematics models of a redundant plant, a different model can be used to achieve each different task in a different way. Thus, addressing a complex mission composed of several tasks can also be seen as a problem of learning several models that must be memorized in parallel. Some authors are using this approach and learn mechanical models in the context of multiple tasks based on GPR. More interestingly, since the inverse dynamics model is unique but may be involved in the realization of diverse tasks at the kinematic level, one can see the problem of learning this inverse dynamics model through a first task and using it for other tasks as a *transfer learning* problem [22, 120]. On top of those ideas, a next challenge will consist in generalizing what is learned in interaction with one object to interaction with a class of similar objects.

6. Conclusion

The on-line modeling of robotics system with regression methods is a very active field. In this paper, we surveyed in particular local and incremental algorithms currently used to solve this problem, providing a unifying view and comparing their features. Among other things, we have shown that, despite its popularity and efficiency for some problems such as learning along a trajectory in a large domain, LWPR suffers from several important drawbacks that justify the important effort dedicated to looking for a better algorithm.

Furthermore, we have shown that several systems come with interesting additional features, such as the condition space/prediction space distinction in XCSF or the capability to learn both the forward and inverse model into a unique manifold in IMLM. We highlighted that there is still room for large performance improvements in this domain just by combining some the features of these algorithms. Further ideas such as the multi-objective optimization of the trade-off between the accuracy and the number of regions might also give rise to new lines of research. Finally, we are convinced that the application of such combinations of algorithms should help addressing in the near future the "new frontier" of model learning, such as the physical interaction with unknown objects, tools and users.

Acknowledgments

This work is supported by the French ANR program (ANR 2010 BLAN 0216 01), more at http://macsi.isir.upmc.fr

References

[1] Ahmad, Z., & Guez, A. (1990). On the solution to the inverse kinematic problem. In *IEEE International Conference on Robotics and Automation* (pp. 1692–1697). volume 3.

[2] Angeli, A., Filliat, D., Doncieux, S., & Meyer, J. (2008). Fast and incremental method for loop-closure detection using bags of visual words. *IEEE Transactions on Robotics*, *24*, 1027–1037.

[3] Argall, B. D., Chernova, S., Veloso, M., & Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, *57*, 469–483.

[4] Atkeson, C. G. (1991). Using locally weighted regression for robot learning. In *IEEE International Conference on Robotics and Automation* (pp. 958–963). volume 2.

[5] Baranes, A., & Oudeyer, P.-Y. (2009). R-IAC: Robust intrinsically motivated exploration and active learning. *IEEE Transactions on Autonomous Mental Development*, *1*, 155–169.

[6] Baranes, A., & Oudeyer, P.-Y. (2010). Maturationally-constrained competence-based intrinsically motivated learning. In *IEEE International Conference on Development and Learning*.

[7] Barhen, J., Gulati, S., & Zak, M. (1989). Neutral learning of constrained nonlinear transformations. *Computer*, *22*, 67–76.

[8] Barretto, V. R., Araujo, A. F. R., & Ritter, H. J. (2003). Self-organizing feature maps for modeling and control of robotic manipulators. *Journal of Intelligent Robotics Systems*, *36*, 407–450.

[9] Ben Israel, A., & Greville, T. N. E. (2003). *Generalized Inverses: Theory and Applications*. (2nd ed.). Springer.

[10] Bicchi, A. (2000). Hands for dexterous manipulation and robust grasping: A difficult road toward simplicity. *IEEE Transactions on Robotics and Automation*, *16*, 652–662.

[11] Brüwer, M., & Cruse, H. (1990). A network model for the control of the movement of a redundant manipulator. *Biological Cybernetics*, *62*, 549–555.

[12] Butz, M., Herbort, O., & Hoffmann, J. (2007). Exploiting redundancy for flexible behavior: Unsupervised learning in a modular sensorimotor control architecture. *Psychological Review*, *114*, 1015–1046.

[13] Butz, M., Pedersen, G., & Stalph, P. (2009). Learning sensorimotor control structures with XCSF: redundancy exploitation and dynamic control. In *Conference on Genetic and Evolutionary Computation* (pp. 1171–1178). ACM.

[14] Butz, M. V., & Goldberg, D. (2003). Bounding the population size in XCS to ensure reproductive opportunities. In *Genetic and Evolutionary Computation* (pp. 1844–1856). Springer-Verlag volume 2724.

[15] Butz, M. V., Goldberg, D., & Lanzi, P. (2005). Computational complexity of the XCS classifier system. *Foundations of Learning Classifier Systems*, *51*, 91–125.

[16] Butz, M. V., Goldberg, D. E., & Tharakunnel, K. (2003). Analysis and improvement of fitness exploitation in XCS: bounding models, tournament selection, and bilateral accuracy. *Evolutionary Computation*, *11*, 239–277.

[17] Butz, M. V., & Herbort, O. (2008). Context-dependent predictions and cognitive arm control with XCSF. In *Conference on Genetic and Evolutionary Computation* (pp. 1357–1364). ACM.

[18] Butz, M. V., Kovacs, T., Lanzi, P. L., & Wilson, S. W. (2004). Toward a theory of generalization and learning in XCS. *IEEE Transactions on Evolutionary Computation*, *8*, 28–46.

[19] Calinon, S. (2009). *Robot programming by demonstration*. EPFL/CRC Press.

[20] Quiñonero Candela, J., & Rasmussen, C. E. (2005). A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, *6*, 1939–1959.

[21] Cederborg, T., Li, M., Baranes, A., & Oudeyer, P.-Y. (2010). Incremental local online Gaussian mixture regression for imitation learning of multiple tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 267–274).

[22] Chai, K. M. A., Williams, C. K. I., Klanke, S., & Vijayakumar, S. (2009). Multi-task Gaussian process learning of robot inverse dynamics. In *Advances in Neural Information Processing Systems*.

[23] Chang, C., & Lin, C. (2001). LIBSVM: a library for support vector machines, 2001. *Software available at http://www. csie. ntu. edu. tw/cjlin/libsvm*, .

[24] Choi, Y., Cheong, S. Y., & Schweighofer, N. (2007). Local online support vector regression for learning control. In *International Symposium on Computational Intelligence in Robotics and Automation* (pp. 13–18).

[25] Chung, W., Fu, L.-C., & Hsu, S.-H. (2008). Motion control. In B. Siciliano, & O. Khatib (Eds.), *Handbook of Robotics* chapter 6. (pp. 133–159). Springer.

[26] Cohn, D., Ghahramani, Z., & Jordan, M. (1996). Active learning with statistical models. *Journal of Artificial Intelligence Research*, *4*, 129–145.

[27] Sun de la Cruz, J., Kulic, D., & Owen, W. (2010). Learning inverse dynamics for redundant manipulator control. In *International Conference on Autonomous and Intelligent Systems* (pp. 1–6). Springer.

[28] Sun de la Cruz, J., Kulić, D., & Owen, W. (2011). Online incremental learning of inverse dynamics incorporating prior knowledge. In *International Conference on Autonomous and Intelligent Systems* (pp. 167–176). Springer.

[29] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, *2*, 303–314.

[30] Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B*, *39*, 1–38.

[31] Drucker, H., Burges, C., Kaufman, L., Smola, A., & Vapnik, V. (1997). Support vector regression machines. *Advances in neural information processing systems*, (pp. 155–161).

[32] D'Souza, A., Vijayakumar, S., & Schaal, S. (2001). Learning inverse kinematics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 298–303). volume 1.

[33] Elden, L. (2004). Partial least-squares vs. Lanczos bidiagonalization-i: analysis of a projection method for multiple regression. *Computational Statistics and Data Analysis*, *46*, 11–31.

[34] Engel, Y., Mannor, S., & Meir, R. (2004). The kernel recursive least-squares algorithm. *IEEE Transactions on Signal Processing*, *52*, 2275–2285.

[35] Featherstone, R., & Orin, D. E. (2008). Dynamics. In B. Siciliano, & O. Khatib (Eds.), *Handbook of Robotics* chapter 2. (pp. 35–65). Springer.

[36] Flentge, F. (2006). Locally weighted interpolating growing neural gas. *IEEE Transactions on Neural Networks*, *17*, 1382–1392.

[37] Fumagalli, M., Gijsberts, A., Ivaldi, S., Jamone, L., Metta, G., Natale, L., Nori, F., & Sandini, G. (2010). Learning to exploit proximal force sensing: A comparison approach. In O. Sigaud, & J. Peters (Eds.), *From Motor to Interaction Learning in Robots* (pp. 149–167). Springer.

[38] Ghahramani, Z., & Roweis, S. T. (1999). Learning nonlinear dynamical systems using an EM algorithm. In *Advances in Neural Information Processing Systems* (pp. 599–605). MIT Press.

[39] Gijsberts, A., & Metta, G. (2010). Incremental learning of robot dynamics using random features. In *IEEE International Conference on Robotics and Automation* (pp. 951–956).

[40] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley.

[41] Grollman, D., & Jenkins, O. C. (2008). Sparse incremental learning for interactive robot control policy estimation. In *IEEE International Conference on Robotics and Automation* (pp. 3315–3320).

[42] Grupen, R., Henderson, T., & McCammon, I. (1989). A survey of general-purpose manipulation. *The International journal of robotics research*, *8*, 38.

[43] Hersch, M., Sauser, E., & Billard, A. (2008). Online learning of the body schema. *International Journal of Humanoid Robotics*, *5*, 161–181.

[44] Hoffmann, M., Marques, H., Arieta, A., Sumioka, H., Lungarella, M., & Pfeifer, R. (2010). Body schema in robotics: A review. *IEEE Transactions on Autonomous Mental Development*, *2*, 304–324.

[45] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Ann Arbor, MI: University of Michigan Press.

[46] http://homepages.inf.ed.ac.uk/svijayak/software/LWPR/ (2008).

[47] Jagersand, M. (2001). Image-based predictive display for high dof uncalibrated tele-manipulation using affine and intensity subspace models. *Advanced Robotics*, *14*, 683–701.

[48] Jordan, M., & Rumelhart, D. (1992). Forward models: Supervised learning with a distal teacher. *Cognitive Science: A Multidisciplinary Journal*, *16*, 307–354.

[49] Kaneko, K., Harada, K., & Kanehiro, F. (2008). Development of multifingered hand for life-size humanoid robots. In *IEEE International Conference on Robotics and Automation* (pp. 913–920).

[50] Kawato, M. (1999). Internal models for motor control and trajectory planning. *Current Opinion in Neurobiology*, *9*, 718–727.

[51] Khatib, O. (1983). Dynamic control of manipulators in operational space. In *Sixth CISM-IFToMM Congress on Theory of Machines and Mechanisms* (pp. 1128–1131).

[52] Klanke, S., Vijayakumar, S., & Schaal, S. (2008). A library for locally weighted projection regression. *Journal of Machine Learning Research*, *9*, 623–626.

[53] Kohonen, T. (2001). *Self-Organizing Maps*. Berlin: Springer.

[54] Kovacs, T. (1999). Deletion schemes for classifier systems. In *Genetic and Evolutionary Computation Conference* (pp. 329–336). Orlando, Florida.

[55] Kumar, S., Premkumar, P., Dutta, A., & Behera, L. (2010). Visual motor control of a 7dof redundant manipulator using redundancy preserving learning network. *Robotica*, *28*, 795–810.

[56] Lee, S., & Kil, R. (1990). Robot kinematic control based on bidirectional mapping neural network. In *International Joint Conference on Neural Networks* (pp. 327–335). volume 3.

[57] Li, W. (2006). *Optimal Control for Biological Movement*. Ph.D. thesis University of California, San Diego.

[58] Lin, S., & Goldenberg, A. A. (2001). Neural-network control of mobile manipulators. *IEEE Transactions on Neural Networks*, *12*, 1121–1133.

[59] Ljung, L. (1986). *System identification: theory for the user*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA.

[60] Lopes, M., & Damas, B. (2007). A learning framework for generic sensory-motor maps. In *IEEE/RSJ International Conference on Intelligent Robot Systems* (pp. 1533–1540).

[61] Ma, J., Theiler, J., & Perkins, S. (2003). Accurate on-line support vector regression. *Neural Computation*, *15*, 2683–2703.

[62] Mansard, N., Lopes, M., Santos-Victor, J., & Chaumette, F. (2006). Jacobian learning methods for tasks sequencing in visual servoing. In *IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 4284–4290). Beijing, China.

[63] Martinetz, T., Ritter, H., & Schulten, K. (1990). Three-dimensional neural net for learning visuomotor coordination of a robot arm. *IEEE Transactions on Neural Networks*, *1*, 131–136.

[64] Martinez-Cantin, R., Lopes, M., & Montesano, L. (2010). Body schema acquisition through active learning. In *IEEE International Conference on Robotics and Automation* (pp. 1860–1866).

[65] Meeds, E., & Osindero, S. (2006). An alternative infinite mixture of Gaussian process experts. In *Advances in Neural Information Processing Systems* (pp. 883–890). MIT Press.

[66] Mitrovic, D., Klanke, S., & Vijayakumar, S. (2008). Adaptive optimal control for redundantly actuated arms. In *From Animals to Animats 10: 10th International Conference on Simulation of Adaptive Behavior, Sab 2008, Osaka, Japan, July 7-12, 2008, Proceedings* (p. 93). Springer.

[67] Nguyen, L., Patel, R., & Khorasani, K. (1990). Neural network architectures for the forward kinematics problem in robotics. In *International Joint Conference on Neural Networks* (pp. 393–399). volume 3.

[68] Nguyen-Tuong, D., & Peters, J. (2008). Local Gaussian processes regression for real-time model-based robot control. In *IEEE/RSJ International Conference on Intelligent Robot Systems* (pp. 380–385).

[69] Nguyen-Tuong, D., & Peters, J. (2010). Incremental sparsification for real-time online model learning. In *International Conference on Artifical Intelligence and Statistics* (pp. 557–564). volume 9.

[70] Nguyen-Tuong, D., & Peters, J. (2010). Using model knowledge for learning inverse dynamics. In *IEEE International Conference on*

Robotics and Automation.

[71] Nguyen-Tuong, D., & Peters, J. (2011). Model learning for robot control: a survey. *Cognitive Processing*, (pp. 1–22).

[72] Nguyen-Tuong, D., Peters, J., Seeger, M., & Schölkopf, B. (2008). Learning inverse dynamics: a comparison. In *European Symposium on Artificial Neural Networks*.

[73] Nguyen-Tuong, D., Seeger, M., & Peters, J. (2009). Model learning with local Gaussian process regression. *Advanced Robotics*, *23*, 2015–2034.

[74] Nguyen-Tuong, D., Seeger, M., & Peters, J. (2010). Real-time local Gaussian processes model learning. In O. Sigaud, & J. Peters (Eds.), *From Motor to Interaction Learning in Robots* (pp. 193–207). Springer.

[75] Peters, J., & Schaal, S. (2008). Learning to control in operational space. *International Journal of Robotics Research*, *27*, 197–212.

[76] Petkos, G., Toussaint, M., & Vijayakumar, S. (2006). Learning multiple models of non-linear dynamics for control under varying contexts. In *International Conference on Artificial Neural Networks (ICANN)* (pp. 898–907). Springer.

[77] Petkos, G., & Vijayakumar, S. (2007). Load estimation and control using learned dynamics models. In *IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 1527–1532).

[78] Pourboghrat, F. (1989). Neural networks for learning inverse-kinematics of redundant manipulators. In *32nd Midwest Symposium on Circuits and Systems* (pp. 760–762). volume 2.

[79] Rahimi, A., & Recht, B. (2008). Random features for large-scale kernel machines. In *Advances in neural information processing systems* (pp. 1177–1184).

[80] Rifkin, R., Yeo, G., & Poggio, T. (2003). Regularized least-squares classification. *Nato Science Series Sub Series III Computer and Systems Sciences*, *190*, 131–154.

[81] Ritter, H. J., Martinetz, T. M., & Schulten, K. J. (1989). Topology-conserving maps for learning visuo-motor-coordination. *Neural networks*, *2*, 159–168.

[82] Robbel, P. (2005). *Active Learning in Motor Control*. Master's Thesis, University of Edinburgh, UK.

[83] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, *65*, 386–408.

[84] Sadjadian, H., & Taghirad, H. D. (2004). Numerical methods for computing the forward kinematics of a redundant parallel manipulator. In *IEEE Conference on Mechatronics and Robotics* (pp. 557–562). Aachen, Germany.

[85] Saegusa, R., Metta, G., & Sandini, G. (2009). Active learning for sensorimotor coordinations of autonomous robots. In *Human System Interaction* (pp. 701–706).

[86] Salaün, C., Padois, V., & Sigaud, O. (2009). Control of redundant robots using learned models: an operational space control approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 878–885).

[87] Sang, L. H., & Han, M.-C. (1999). The estimation for forward kinematic solution of stewart platform using the neural network. In *IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 501–506). volume 1.

[88] Sayed, A. H. (2008). *Adaptive Filters*. Wiler-IEEE Press.

[89] Schaal, S., & Atkeson, C. G. (1997). Receptive field weighted regression. *ART Human Inf. Process. Lab., Kyoto, Japan, Tech. Rep. TR-H-209*, .

[90] Schaal, S., Atkeson, C. G., & Vijayakumar, S. (2002). Scalable techniques from nonparametric statistics for real time robot learning. *Applied Intelligence*, *17*, 49–60.

[91] Schaal, S., Vijayakumar, S., & Atkeson, C. G. (1998). Local dimensionality reduction. In *Advances in neural information processing systems* (pp. 633–639).

[92] Schölkopf, B., Smola, A., Williamson, R., & Bartlett, P. (2000). New support vector algorithms. *Neural computation*, *12*, 1207–1245.

[93] Shen, Y., Ng, A., & Seeger, M. (2006). Fast Gaussian process regression using kd-trees. In *Advances in neural information processing systems* (p. 1225).

[94] Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2008). *Robotics: modelling, planning and control*. Springer Verlag.

[95] Sigaud, O., & Peters, J. (2010). Introduction. In O. Sigaud, & J. Peters (Eds.), *From Motor Learning to Interaction Learning in Robots* chapter 1. (pp. 1–12). Springer.

[96] Smola, A. J., & Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and Computing*, *14*, 199–222.

[97] Stalph, P., Butz, M., & Pedersen, G. (2009). Controlling a four degree of freedom arm in 3d using the XCSF learning classifier system. *KI 2009: Advances in Artificial Intelligence*, (pp. 193–200).

[98] Stalph, P., Rubinsztajn, J., Sigaud, O., & Butz, M. (2010). A comparative study: Function approximation with LWPR and XCSF. In *13th International Workshop on Advances in Learning Classifier Systems*.

[99] Sturm, J., Plagemann, C., & Burgard, W. (2008). Unsupervised body scheme learning through self-perception. In *IEEE International Conference on Robotics and Automation* (pp. 3328–3333). IEEE.

[100] Sun, G., & Scassellati, B. (2004). Reaching through learned forward model. In *4th IEEE/RAS International Conference on Humanoid Robots* (pp. 93–112). Los Angeles, USA volume 1.

[101] Sun, G., & Scassellati, B. (2005). A fast and efficient model for learning to reach. *International Journal of Humanoid Robotics*, *2*, 391–414.

[102] Sutton, R., & Barto, A. (1998). *Reinforcement learning*. MIT Press.

[103] Suykens, J., Van Gestel, T., De Brabanter, J., De Moor, B., & Vandewalle, J. (2002). Least squares support vector machines.

[104] Suykens, J., & Vandewalle, J. (1999). Least squares support vector machine classifiers. *Neural processing letters*, *9*, 293–300.

[105] Ulbrich, S., Ruiz de Angulo, V., Asfour, T., Torras, C., & Dillmann, R. (2009). Rapid learning of humanoid body schemas with kinematic bezier maps. In *IEEE-RAS International Conference on Humanoid Robots* (pp. 431–438).

[106] Vijayakumar, S., D'Souza, A., & Schaal, S. (2005). *LWPR: A Scalable Method for Incremental Online Learning in High Dimensions*. Technical Report Edinburgh University Press.

[107] Vijayakumar, S., & Schaal, S. (1997). Local dimensionality reduction for locally weighted learning. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation* (pp. 220–225).

[108] Vijayakumar, S., & Schaal, S. (2000). Locally weighted projection regression: An o(n) algorithm for incremental real time learning in high dimensional space. In *International Conference on Machine Learning* (pp. 1079–1086).

[109] Walter, J., & Schulten, K. (1993). Implementation of self-organizing neural networks for visuo-motor control of an industrial robot. *IEEE Transactions on Neural Networks*, *4*, 86–96.

[110] Wang, D., & Zilouchian, A. (1997). Solutions of kinematics of robot manipulators using a kohonen self-organizing neural network. In *IEEE International Symposium on Intelligent Control* (pp. 251–255).

[111] Whitney, D. E. (1969). Resolved motion rate control of manipulators and human prostheses. *IEEE Transactions on Man-Machine Systems*, *10*, 47–53.

[112] Widrow, B., & Hoff, M. (1960). Adaptive switching circuits. *Western Electronic Show and Convention*, (pp. 96–104).

[113] Williams, C. K. I. (1997). *Prediction with Gaussian Processes: From Linear Regression to Linear Prediction and Beyond*. Technical Report NCRG/97/012 Neural Computing Research Group.

[114] Wilson, S. W. (1995). Classifier Fitness Based on Accuracy. *Evolutionary Computation*, *3*, 149–175.

[115] Wilson, S. W. (2001). Function approximation with a classifier system. In *Genetic and Evolutionary Computation Conference* (pp. 974–981). San Francisco, California, USA: Morgan Kaufmann.

[116] Wilson, S. W. (2002). Classifiers that Approximate Functions. *Natural Computing*, *1*, 211–234.

[117] Wold, H. (1975). Soft modelling by latent variables: the non-linear iterative partial least squares (NIPALS) approach. *Perspectives in Probability and Statistics*, (pp. 117–142).

[118] Woods, F., Grollman, D., Heller, K., Jenkins, O. C., & Black, M. (2008). *Incremental Nonparametric Bayesian Regression*. Technical Report Brown University Department of Computer Science.

[119] http://www.coboslab.psychologie.uni wuerzburg.de/code/ (2009).

[120] Yeung, D.-Y., & Zhang, Y. (2009). Learning inverse dynamics by Gaussian process regression under the multi-task learning framework. In *The Path to Autonomous Robots* (pp. 131–142).

[121] Zinn, M., Khatib, O., Roth, B., & Salisbury, J. K. (2004). Playing it safe. *IEEE Robotics and Automation Magazine*, *june*, 12–21.