# Indirect Robot Model Learning for Tracking Control

Botond Bócsi[1] Lehel Csató[1] Jan Peters[2,3]

[1] *Faculty of Mathematics and Informatics, Babeş-Bolyai University, Kogalniceanu 1, 400084 Cluj-Napoca, Romania,* BBOTI@CS.UBBCLUJ.RO *and* LEHEL.CSATO@CS.UBBCLUJ.RO

[2] *Technische Universitaet Darmstadt, Intelligent Autonomous Systems Group, Hochschulstr. 10, 64289 Darmstadt, Germany,* PETERS@IAS.TU-DARMSTADT.DE

[3] *Max Planck Institute for Intelligent Systems, Spemannstr. 38, 7206 Tuebingen, Germany,* JAN.PETERS@TUEBINGEN.MPG.DE

**Abstract**

Learning task-space tracking control on redundant robot manipulators is an important but difficult problem. A main difficulty is the non-uniqueness of the solution: a *task-space* trajectory has multiple joint-space trajectories associated, therefore averaging over non-convex solution space needs to be done if treated as a regression problem. A second class of difficulties arise for those robots when the physical model is either too complex or even not available. In this situation machine learning methods may be a suitable alternative to classical approaches. We propose a learning framework for tracking control that is applicable for underactuated or non-rigid robots where an analytical physical model of the robot is unavailable. The proposed framework builds on the insight that tracking problems are well defined in the *joint* task- and joint-space coordinates and consequently predictions can be obtained via local optimization. Physical experiments show that state-of-the art accuracy can be achieved in both online and offline tracking control learning. Furthermore, we show that the presented method is capable of controlling underactuated robot architectures as well. [1]

## 1 Introduction

Efficient tracking control algorithms are essential in robot control [4, 5], and most algorithms are based on an accurate inverse models of the robot. However, such a mapping from task-space to joint-space is non-unique: given a task-space position, there will be several possible joint-space configurations forming a non-convex solution space [6] and one has to choose from these solutions [4].

Classical control methods are based on physical models of the robot [7], but it has several drawbacks [see 6, 2]: in modern applications the exact robot models might not be available, e.g., for complex robots, non-rigid manipulators, flexible joint robots, or when uncalibrated cameras provide noisy Cartesian coordinates. A second drawback appears when a system can change over time, therefore we need to adapt

[1] This article is an extended work of conference papers [1], [2], and [3].

the tracking control model as well, a time-consuming procedure. Hence, instead of traditional methods, *learning tracking control* using sampled data may be a suitable alternative. Traditional methods also break down for non-rigid robot systems with insufficient sensing, as well as for systems with an unknown nonlinear perceptual transformation. In our experiments, a ball was attached to the end-effector of a Barrett Whole-Arm-Manipulator (WAM) robot arm [3], and the ball at the end of a string had to follow a desired trajectory instead of the end-effector of the robot (see Figure 5(c) for illustration). For this system even the forward kinematics is undefined since the same joint configuration can result in different ball positions due to the swinging of the ball. Such an underactuated and insufficiently sensed target is difficult to track since swinging produces non-linear effects that lead to undefinable forward or inverse models. Nevertheless we were able to track the system and show that our method is capable of capturing non-linear effects, like the centrifugal force, into the control algorithm, despite incomplete sensing.

We present tracking control framework: (1) approximation of the joint input-output model of task- and joint-spaces with machine learning techniques, (2) obtaining the inverse kinematics mapping with local optimization, and (3) based on the mapping, using the joint-space controller of the robot to obtain the desired torques. We assume that the inverse dynamics controller of the robot is available.

The second step of the previous algorithm calculates the inverse kinematics function. In this article we focus on learning a *direct mapping* from the task-space position to the joint-space configuration using structured output prediction [8, 9], exploiting the idea that a model of the *joint input-output space* is well-defined and can be inferred from samples. This joint model can be viewed as an energy function with low energies at pairs which fit together well. Predictions for target outputs, i.e., joint-space configurations, can be obtained by minimizing this model for given inputs, i.e., task-space positions.

The potential ambiguity in the output space, i.e., different joint configurations resulting in the same end-effector position, is resolved by finding the most probable output solutions, which explain the current input trajectory. We propose three different models of the joint energy function:[2] joint kernel support estimation (JKSE), structured output Gaussian processes (SOGP) and an indirect method based on a learned forward kinematics model. We show the strengths and weaknesses of the proposed approximations both from a theoretical point of view and based on results of real robot experiments.

The paper is organized as follows: Section 1.1 gives an overview of existing inverse kinematics solutions which are based on the physical parameters of the robot, while Section 1.2 details the approaches based on machine learning. Section 2.1 presents the basic idea behind the proposed approach. Next, we detail the three proposed methods in Section 2.2, Section 2.3, and Section 2.4 respectively. Results of real world experiments are presented in Section 3 with conclusions drawn in Section 4.

## 1.1 Analytical and Numerical Solutions for Inverse Kinematics

The forward kinematics are given by the correspondence $\mathbf{x} = \mathbf{f}(\boldsymbol{\theta})$, mapping the $\mathfrak{n}$-dimensional joint angles $\boldsymbol{\theta} \in \mathbb{R}^{\mathfrak{n}}$ into $\mathfrak{p}$-dimensional task-space positions $\mathbf{x} \in \mathbb{R}^{\mathfrak{p}}$. The inverse kinematics – being the

[2]The term "joint" refers to the joint distribution over inputs and outputs and not to the joints of the robot. We will use the "joint" with both meanings, actual meaning being clear from the context.

inverse of the forward correspondence – transforms the end-effector coordinates into joint angles

$$\theta = f^{-1}(x). \tag{1}$$

Finding $f^{-1}(\cdot)$ is a significantly harder problem than modeling $f(\cdot)$. For redundant systems when the dimension of the task-space is smaller than the dimension of the joint-space ($p < n$), the inverse $f^{-1}(\cdot)$ is not a function since there are values for $x$ with different joint angles $\theta$.

Classical approaches solve inverse kinematics on the velocity level, i.e., differential inverse kinematics [7], for example, the Jacobian transpose method or by the resolved velocity method $\dot{\theta} = J(\theta)^{\dagger}\dot{x}$, where $J(\theta)^{\dagger}$ is the pseudo-inverse [7]. It is important that resolved velocity methods are numerically unstable if $J(\theta)$ is singular, where the robot looses a degree of freedom. Numerical solutions to $\theta = f^{-1}(x)$ can be found by iterating the above inverse kinematics methods until convergence.

Instead of using the physical architecture of the robot to find the inverse kinematics mapping, learning this model became attractive. The first attempts to approximate the inverse kinematics mapping, modeled the physical architecture of the robot based on sampled data. Iterative and adaptive solutions can be found in Xu et al. [10]. Next, we present the prevailing approaches where we not assume any knowledge of the robot model.

## 1.2 Learning Inverse Kinematics

Among the most well-known online learning approaches for inverse kinematics is D'Souza et al. [6], based on "locally weighted projection regression" (LWPR) [11]. LWPR learns the inverse kinematics mapping on the velocity level. Its central idea is to partition the input space into regions and learn local piecewise linear models for each region. The input space, i.e., desired end-effector velocity, is augmented with the current joint configuration of the robot and, subsequently, the mapping $(\dot{x}, \theta) \rightarrow \dot{\theta}$ is learned. By adding the current joint configuration to the input space, the localization of the linear models becomes valid. This augmentation leads to a locally consistent algorithm, however, global consistency is not guaranteed. D'Souza et al. [6] argue that the main disadvantage of LWPR is that the partitioning of the augmented input space becomes difficult for robots with many degrees of freedom (DoFs).

A similar modular construction of $f^{-1}(\cdot)$ is employed by Oyama et al. [12] using neural networks as local models and a gating neural network that chooses among them. The construction of the gating network that determines model responsibilities becomes difficult in high dimensions.

The multivalued nature of inverse kinematics is addressed by Jordan and Rumelhart [13], who introduced an algorithm for learning multivalued functions and applied it for inverse kinematics. A neural network was used for forward kinematics and a second neural network for the inverse kinematics, and set up so that their composition to yield the identity function. They found that training the inverse model was difficult due to local minima, instability, and problems in selecting the network structure.

One of the approaches discussed later uses the fact that modeling forward kinematics is significantly easier than modeling its inverse. This setup was investigated in the literature using different approaches [5]: radial basis functions were used for forward modeling by Sun and Scassellati [14] on a 4-DoF robot.

The system was trained on data collected offline and was only tested on reaching tasks. The inverse kinematics mapping has been achieved on velocity level by differentiating the neural network. The authors claimed that the structure of the network and the parameter settings have a major effect on the generalization capacity, therefore a tedious task dependent tuning is required. The same idea of using the forward model has been used by Ulbrich et al. [15] who used Bèzier curves to model the forward kinematics function.

Task space tracking has been investigated in a more general framework, known as operational space control, when the direct mapping between task-space and torque-space is modeled. Peters and Schaal [16] applied local (reward-weighted) linear regression to obtain this model and reformulated the problem in a reinforcement learning framework. Nguyen-Tuong and Peters [17] proposed a kernel based approach of operational space control that was also applicable for online usage.

# 2 Indirect Robot Model Learning

We now focus on learning the direct inverse kinematics function on the position level. We use structured output learning to model the multivalued inverse kinematics relationship by learning the model in the *joint input-output space* and obtaining prediction for target outputs by local search by fixing the input. The local optimization routine addresses the non-uniqueness of $\mathsf{f}^{-1}(\cdot)$. We process with a short introduction to structured output learning, and show its use for inverse kinematics modeling.

## 2.1 Structured Output Learning

In structured output learning, we aim at finding a function $\mathsf{g} : \mathcal{X} \rightarrow \Theta$ but unlike in the usual setup, here $\Theta$ has a *structure*. We consider that the input space $\mathcal{X}$ is domain dependent, and usually is a high dimensional Euclidean real space. If we consider a robot model, $\Theta$ is the space of joint angles and usually we cannot simply consider $\Theta = \mathbb{R}^n$ since there are joint configurations that are not allowed due to the physical constrains of the robot. Therefore, in this case, the structure aims to capture the physical constrains of the robot.

In structured learning in what follows we assume that the input-output mapping is modeled via a function $\mathsf{E}(\mathbf{x}, \boldsymbol{\theta})$ that measures the quality of a given $(\mathbf{x}, \boldsymbol{\theta})$ *pair* with $\mathbf{x} \in \mathcal{X}$ and $\boldsymbol{\theta} \in \Theta$, and finding the most fit $\boldsymbol{\theta}$ for a given $\mathbf{x}$ as prediction. Given $\mathsf{E}(\mathbf{x}, \boldsymbol{\theta})$, the prediction $\mathsf{f}^{\text{pred}}(\mathbf{x})$ is defined as

$$\mathsf{f}^{\text{pred}}(\mathbf{x}) \quad \stackrel{\circ}{=} \quad \underset{\boldsymbol{\theta} \in \boldsymbol{\Theta}}{\operatorname{argmin}} \, \mathsf{E}(\mathbf{x}, \boldsymbol{\theta}). \tag{2}$$

In this paper, we model inverse kinematics using structured output learning, i.e., $\mathsf{f}^{\text{pred}} = \mathsf{f}^{-1}$. An important issue is raised by the non-uniqueness of the inverse kinematics function: how minimization ensures that a "good" solution is chosen when the end-effector position $\mathbf{x}^{\text{desired}}$ has at least two different joint configurations $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$, as shown in Figure 1. We want our algorithm to direct to $\boldsymbol{\theta}_2$ to avoid large movements and to achieve smooth trajectories in both the output and the joint-spaces, therefore we start a gradient search from the current joint position vector $\boldsymbol{\theta}^{\text{current}}$. Our algorithm of computing the

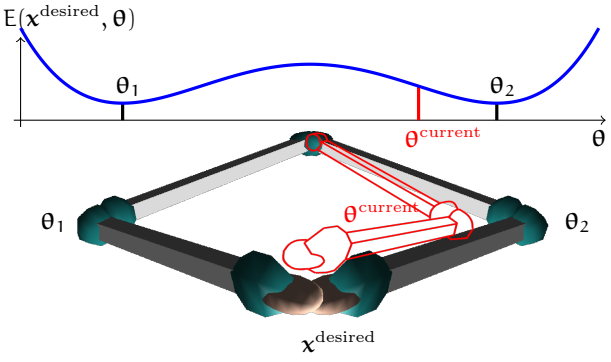



Figure 1: Inverse kinematics prediction. During training $x^{desired}$ has been reached by two different joint configurations $\theta_1$ and $\theta_2$, therefore, $E(x^{desired}, \theta_1) = E(x^{desired}, \theta_2)$. As prediction our algorithm will chose $\theta_2$ since the current joint configuration $\theta^{current}$ is in the attraction range of $\theta_2$.

inverse kinematics mapping benefited from boosting the iterative gradient method by using improved second order gradient search methods like conjugate gradient. The usage of the gradient search puts constrains on the energy function $E(\cdot, \cdot)$ we can use: it has to be smooth and we must be able to calculate its gradient with respect to $\theta$ analytically. Since our model of $E(\cdot, \cdot)$ is based on the sample set which is always finite, there must be finite number of local minima and the search method will always converge to one of the local minima. By carefully selecting the training points, we can ensure that $E(\cdot, \cdot)$ contains no physically inconvenient solutions, remaining within the safe operating stage.

There are similarities between the proposed gradient based search and iterative learning control [10]. The most prominent similarity is that the solution is obtained via several steps of calculations. While iterative learning control methods make first order steps toward the solution, gradient based search can use second order methods, like conjugate gradient. Although, second order search converges faster, the evaluation of the second derivative of the model is required that may be computationally expensive – whether it is beneficial depends on the complexity of the model. Another benefit of the proposed method over iterative learning control is that the later assumes linear models, while we do not limit ourselves to such models.

To model the energy function $E(\cdot, \cdot)$, we propose three different approaches: (1) *joint kernel support estimation* that is based on support vector machines [18], (2) *structured output Gaussian processes* that uses GPs on the joint input-output space, and (3) an approximation with a GP of the forward kinematics model. The methods are presented in the next three sections.

## 2.2 Joint Kernel Support Estimation

Joint kernel support estimation (JKSE) models the energy function as the negative joint probability of $x$ and $\theta$, i.e., $E(x, \theta) = -p(x, \theta)$. JKSE models the joint probability distribution of inputs and outputs as a log-linear model of a joint feature function [9] by

$$p(x, \theta) = \frac{1}{Z} \exp\left(w^\top \phi(x, \theta)\right), \tag{3}$$

where $\boldsymbol{w}$ is a vector of weights, $\phi(\cdot,\cdot)$ is a joint feature function, generally used to express knowledge related to the specific task (an implicit definition of $\phi(\cdot,\cdot)$ is also available using kernels), and $Z$ is the normalizing constant to the density function – $Z = \int_{\mathcal{X}} \int_{\Theta} \exp\left(\boldsymbol{w}^\top \phi(\boldsymbol{x},\theta)\right) d\theta\, d\boldsymbol{x}$ –, that is not needed since it will not affect the location of the optimum.

We are looking for $\boldsymbol{w}$ which generates a $p(\boldsymbol{x},\theta)$ that explains the training dataset $\mathbf{D} = \{(\boldsymbol{x}_i,\theta_i)\}_{i=1}^{m}$ best, where $m$ is the number of training points. The normally high dimension of the joint space prevents us from computing the entire distribution, we relax the problem to determine the *support* of the distribution $p(\boldsymbol{x},\theta)$ instead, done with the one-class support vector machines (OC-SVM) [18]. Using OC-SVMs leads to the energy function $E(\boldsymbol{x},\theta) = -\mathbf{k}_{(\boldsymbol{x},\theta)}^\top \boldsymbol{\alpha}$ with the following prediction:

$$f^{-1}(\boldsymbol{x}) \quad = \quad \underset{\theta\in\boldsymbol{\Theta}}{\operatorname{argmax}}\, \mathbf{k}_{(\boldsymbol{x},\theta)}^\top \boldsymbol{\alpha}, \tag{4}$$

where $k : (\mathcal{X}\times\Theta)\times(\mathcal{X}\times\Theta) \to \mathbb{R}$ is a kernel function [19, 20] defined on joint input-output space. The weights $\alpha_i$ are determined by the OC-SVM learning procedure. We use the shorter notation $\mathbf{k}_{(\boldsymbol{x},\theta)} \in \mathbb{R}^{m\times 1}$, with $\mathbf{k}^i_{(\boldsymbol{x},\theta)} = k((\boldsymbol{x},\theta),(\boldsymbol{x}_i,\theta_i))$ where $i$ runs over the training examples, and $\boldsymbol{\alpha} = [\alpha_i]$.

### 2.2.1 Joint Data Representation

In the previous section, we did not assume how the joint data are defined and more precisely what is stored in the state- and joint spaces. A natural choice is the simple concatenation of the vectors $\boldsymbol{x}$ and $\theta$ but we found that adding the sines and cosines of the joint angles, i.e.

$$\psi(\boldsymbol{x},\theta) = [\boldsymbol{x} \;\; \theta \;\; \sin(\theta) \;\; \cos(\theta)]^\top,$$

improves the prediction, since forward kinematics highly depends on these values (see also Sciavicco and Siciliano [7]). Here, $\psi(\boldsymbol{x},\theta)$ is a vector of length $p + 3n$. The notations $\sin(\theta)$ and $\cos(\theta)$ define the element-wise sines and cosines of the vector $\theta$. This data representation will be used for other methods as well. When calculating the gradient of the energy function (see Section 2.2.2), we will need the Jacobian of the function $\psi(\cdot,\cdot)$ that look as follows

$$\nabla_\theta \psi(\boldsymbol{x},\theta) = [0_3 \;\; \mathbf{I}_n \;\; \operatorname{diag}(\cos(\theta)) \;\; -\operatorname{diag}(\sin(\theta))], \tag{5}$$

where $0_3$ is a $n\times 3$ matrix with zero elements, $\mathbf{I}_n$ is the identity matrix of size $n$, and $\operatorname{diag}(\boldsymbol{x})$ defines a diagonal matrix with the elements of $\boldsymbol{x}$ on the diagonal.

### 2.2.2 Gradient of the Energy Function

To calculate the gradient of the energy function, we use the form of the prediction from Equation (4). Using the chain rule, the gradient of $E(\cdot,\cdot)$ with respect to $\theta$ looks as follows:

$$\nabla_\theta E(\boldsymbol{x},\theta) = -\frac{\partial \mathbf{k}^\top_{\psi(\boldsymbol{x},\theta)}}{\partial\theta} \nabla_\theta \psi(\boldsymbol{x},\theta)\boldsymbol{\alpha}, \tag{6}$$

with $\nabla_\theta\psi(\boldsymbol{x},\theta)$ defined in Equation (5) and the partial derivatives of the kernel $\partial \mathbf{k}^\top_{\psi(\boldsymbol{x},\theta)}/\partial\theta$ depend on the choice of the kernel function. In our experiments, we obtained good performance for

the popular squared exponential kernel $k(\mathbf{x}_1, \mathbf{x}_2) = C \exp(-\|\mathbf{x}_1 - \mathbf{x}_2\|^2/2\omega)$, with gradient as follows: $\partial k(\mathbf{x}_1, \mathbf{x}_2)/\partial \mathbf{x}_1 = -k(\mathbf{x}_1, \mathbf{x}_2)(\mathbf{x}_1 - \mathbf{x}_2)^\top/\omega$, where $C$ and $\omega$ are hyper-parameters of the kernel function.

### 2.3 Structured Output Gaussian Processes

In this section, we propose to model the energy function using Gaussian processes (GPs). Gaussian processes are non-parametric Bayesian models which define a distribution over functions; with the *prior* characterized fully by the mean – $\mu(\cdot)$ – and covariance (or kernel) functions – $k(\cdot,\cdot)$ –, see [19]. Given a training set $\mathbf{D}$, GP inference aims to find the mapping $\mathbf{x} \to \theta$ that explains the data-set best. The posterior distribution of a test point $\mathbf{x}_*$ is Gaussian-distributed with mean $\mu_*$ and variance $\sigma_*^2$:

$$\begin{aligned} \mu_* &= \mathbf{k}_*^\top (\mathbf{K} + \sigma_0^2 \mathbf{I}_m)^{-1} \theta \\ \sigma_*^2 &= k_{**} - \mathbf{k}_*^\top (\mathbf{K} + \sigma_0^2 \mathbf{I}_m)^{-1} \mathbf{k}_*, \end{aligned} \tag{7}$$

where $\mathbf{K} \in \mathbb{R}^{m \times m}$ with $\mathbf{K}^{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, $\mathbf{k}_* \in \mathbb{R}^{m \times 1}$ with $\mathbf{k}_*^i = k(\mathbf{x}_i, \mathbf{x}_*)$, $k_{**} = k(\mathbf{x}_*, \mathbf{x}_*)$, $k(\cdot,\cdot)$ is a kernel function [20], $\mathbf{I_m}$ is the identity matrix, and $\sigma_0^2$ is the variance of the measurement noise.

Let us now consider the structured output learning framework presented in Section 2.1 and let us model $E(\cdot,\cdot)$ using GPs. A key insight is that the training data provides only *positive* examples of $(\mathbf{x}, \theta)$, i.e., we know that $E(\mathbf{x}_i, \theta_i)$ has a high value for all $(\mathbf{x}_i, \theta_i) \in \mathbf{D}$, assume it is 1. Such an unbalanced training set can easily lead to over-fitting – e.g. the constant 1 function would give an "exact" solution. To avoid over-fitting, the definition of a strong prior is essential. In the rest of this section, we assume a zero mean prior and to model $E(\cdot,\cdot)$, we define a GP on the joint data $(\mathbf{x}_i, \theta_i)$ as input, and 1 as output. We also have to define a joint kernel function $k(\cdot,\cdot)$ on the space $\mathcal{X} \times \Theta$, the same kernel function that was presented in Section 2.2.2.

Using the predictive distribution of a GP form Equation (7), the posterior distribution of $E(\cdot,\cdot)$ at point $(\mathbf{x}, \theta)$ is $p(E|\mathcal{D})(\mathbf{x}, \theta) = \mathcal{N}(\mu_{(\mathbf{x},\theta)}, \sigma^2_{(\mathbf{x},\theta)})$ where

$$\begin{aligned} \mu_{(\mathbf{x},\theta)} &= \mathbf{k}_{(\mathbf{x},\theta)}^\top (\mathbf{K} + \sigma_0^2 \mathbf{I_m})^{-1} \mathbf{1} = \mathbf{k}_{(\mathbf{x},\theta)}^\top \boldsymbol{\alpha} \\ \sigma^2_{(\mathbf{x},\theta)} &= k_{(\mathbf{x},\theta)(\mathbf{x},\theta)} - \mathbf{k}_{(\mathbf{x},\theta)}^\top (\mathbf{K} + \sigma_0^2 \mathbf{I_m})^{-1} \mathbf{k}_{(\mathbf{x},\theta)}, \end{aligned}$$

where $\mathbf{K} \in \mathbb{R}^{m \times m}$ with $\mathbf{K}^{ij} = k((\mathbf{x}_i, \theta_i), (\mathbf{x}_j, \theta_j))$, $\mathbf{k}_{(\mathbf{x},\theta)} \in \mathbb{R}^{m \times 1}$ with $\mathbf{k}^i_{(\mathbf{x},\theta)} = k((\mathbf{x}_i, \theta_i), (\mathbf{x}, \theta))$, $k_{(\mathbf{x},\theta)(\mathbf{x},\theta)} = k((\mathbf{x}, \theta), (\mathbf{x}, \theta))$, $\mathbf{I_m}$ is the identity matrix of size $m$, $\sigma_0^2$ is the variance of the measurement noise, and $\mathbf{1}$ is the unit vector of length $m$. We use the shorter notation $\boldsymbol{\alpha} = (\mathbf{K} + \sigma_0^2 \mathbf{I_m})^{-1} \mathbf{1}$.

We define the energy function as the *posterior mean* of the GP, i.e.,

$$E(\mathbf{x}, \theta) = -\mu_{(\mathbf{x},\theta)} = -\mathbf{k}_{(\mathbf{x},\theta)}^\top \boldsymbol{\alpha}. \tag{8}$$

Note that the expression of the prediction of JKSE from Equation (4) and SOGP from Equation (8) have similar forms. Thus, the gradient has the same form as Equation (6) as well.

## 2.4 Indirect Learning with Forward Models

Here, the key observation is that forward kinematics models are significantly simpler functions than the inverse mappings. Once the forward kinematics model is known, the energy function is defined as the Euclidean distance between the desired task-space position and the one predicted by the forward model:

$$E(\mathbf{x}, \boldsymbol{\theta}) \quad = \quad \|\mathbf{x} - f(\boldsymbol{\theta})\|^2. \tag{9}$$

A similar construction has been used by Salaün et al. [5] and Ulbrich et al. [15] modeling $f(\theta)$ with LWPR and Bèzier curves respectively. Here, we use a GP to do the respective approximation and re-interpret in the structured prediction framework.

A general description of GPs has been given in Section 2.3, here, we highlight how they are used in this slightly different context. As opposed to the inverse kinematics case in Section 2.3, here, the inputs are the joint configurations and the outputs are the task-space positions as in a forward kinematics setting. Given a training set $\mathbf{D} = \{(\mathbf{x}_i, \boldsymbol{\theta}_i)\}_{i=1}^m$ with inputs $\boldsymbol{\theta}_i$ and labels $\mathbf{x}_i$, the prediction for a new $\boldsymbol{\theta}$ is a Gaussian–distributed random variable with mean $\mu_{\boldsymbol{\theta}}$ and variance $\sigma^2_{\boldsymbol{\theta}}$ where

$$\begin{aligned} \mu_{\boldsymbol{\theta}} &= \sum_{i=1}^{m} \alpha^i k(\boldsymbol{\theta}, \boldsymbol{\theta}_i) = \mathbf{k}_{\boldsymbol{\theta}}^\top \boldsymbol{\alpha}, \\ \sigma^2_{\boldsymbol{\theta}} &= k_{\boldsymbol{\theta}\boldsymbol{\theta}} - \mathbf{k}_{\boldsymbol{\theta}}^\top \left(\mathbf{K} + \mathbf{I}_m \sigma_0^2\right)^{-1} \mathbf{k}_{\boldsymbol{\theta}}. \end{aligned} \tag{10}$$

Here, $k_{\boldsymbol{\theta}\boldsymbol{\theta}} = k(\boldsymbol{\theta}, \boldsymbol{\theta})$, $\mathbf{k}_{\boldsymbol{\theta}} \in \mathbb{R}^{m \times 1}$ is a vector with elements $\mathbf{k}^i_{\boldsymbol{\theta}} = k(\boldsymbol{\theta}, \boldsymbol{\theta}_i)$ and $\mathbf{K} \in \mathbb{R}^{m \times m}$ is a matrix with elements $\mathbf{K}^{ij} = k(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j)$. The function $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is a positive definite kernel and $\boldsymbol{\alpha} \in \mathbb{R}^{m \times 1}$, where $\boldsymbol{\alpha} = \left(\mathbf{K} + \mathbf{I}_m \sigma_0^2\right)^{-1} \mathbf{x}$ are the parameters of the GP. The predictive mean of the GPs is used as the prediction of the forward model[3] from Equation (10) given by $f(\boldsymbol{\theta}) = \mu_{\boldsymbol{\theta}}$. We adopt *"forward Gaussian process modeling"* (FWGP) for the approach presented in this section. The energy function from Equation (9) is different from the JKSE and SOGP approaches, with its gradient derived from Equation (9) as

$$\nabla_{\boldsymbol{\theta}} E(\mathbf{x}, \boldsymbol{\theta}) = 2(\mathbf{x} - f(\boldsymbol{\theta}))^\top \nabla f(\boldsymbol{\theta}).$$

Observe that this approach seems to be equivalent to the iterative pseudo inverse method, therefore it is sufficiently fast, disregarding the size of the training data, speed improvement is presented next.

## 2.5 Dealing with Large Amounts of Data

In robot control applications it is essential to hard limit the prediction time since robot architectures require inputs at regular intervals. The main problem in keeping this limit is the large amount of data available, e.g., by sampling at 500Hz we collect 30000 training points in every minute. JKSE– see Section 2.2 – solves the problem of the large amount of data by using OC-SVM as its base method. By definition, OC-SVMs provide a sparse usage of the training data, i.e., only the elements of $\boldsymbol{\alpha}$ from

[3] We used separate GP's for each output, i.e., $f_j(\boldsymbol{\theta}) = \mu^j_{\boldsymbol{\theta}}$, $j = \overline{1,3}$ where $f_j(\boldsymbol{\theta})$ is the j-th component of $f(\boldsymbol{\theta})$.

**Algorithm 1** General Algorithm for Task-Space Tracking Control

INPUT: $\mathbf{D} = \{(\mathbf{x}_i, \boldsymbol{\theta}_i)\}_{i=1}^{m}$, $\gamma$ {parameters}

$\mathcal{M} \leftarrow$ train-model ($\mathbf{D}$, $\gamma$) {e.g., OC-SVM or GP}
**while** task is not over **do**
  $\mathcal{M} \leftarrow$ update-model($\mathbf{x}^{\text{current}}, \boldsymbol{\theta}^{\text{current}}$) {optional}
  $\mathbf{x}^{\text{desired}} \leftarrow$ next-position ()
  $\boldsymbol{\theta}^{\text{desired}} \leftarrow$ gradient-minimization ($\boldsymbol{\theta}^{\text{current}}$, $\mathsf{E}_{\mathcal{M}}(\mathbf{x}^{\text{desired}}, \cdot)$)
  inverse-dynamics ($\boldsymbol{\theta}^{\text{current}}$, $\boldsymbol{\theta}^{\text{desired}}$)
**end while**

Equation (4) corresponding to support vectors will be non-zero. We can control the desired complexity explicitly by applying a different parametrization of OC-SVM.

GPs have the same complexity as support vector machines, with time and space complexities growing cubically with the number of the *data points*. To overcome this problem, several sparsification methods have been proposed [21, 22, 23, 24]. We adopt a method proposed by Csató and Opper [21] that can be applied online. The method is the online approximation to the posterior distribution using a sequential algorithm [25] where we combine the likelihood of a single data point with the GP prior from the result of the previous approximation step.

The sparsification algorithm defines an approximation to the exact Gaussian process posterior, chosen such that the discrepancy between the exact and the approximate posterior is minimized. An important feature of the method is that the parameters of the GP are updated even when the new data point is not included into the base set. Thus, the accuracy of the approximation improves during the learning process while the base points might not change.

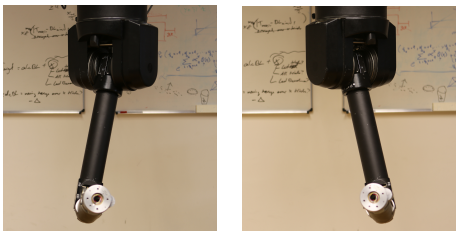
By setting an upper limit to the size of the base set, one can set an upper limit to the algorithm, therefore fully controlling the computation time.

## 2.6 Practical Considerations and Implementation

All three approaches can be phrased and applied within the same general framework. Algorithm 1 contains an overview of the task-space tracking control method in pseudo-code. The training phase is straightforward: given a data-set $\mathbf{D} = \{(\mathbf{x}_i, \boldsymbol{\theta}_i)\}_{i=1}^{m}$, we train a model $\mathcal{M}$ on the joint data. In the presented paper $\mathcal{M}$ refers to OC-SVM (Section 2.2), online GP on the joint input-output space (Section2.3), or online GP of the forward model (Section 2.4).

The trajectory tracking looks as follows. We generate the desired end-effector position $\mathbf{x}^{\text{desired}}$, and predict the corresponding joint configuration $\boldsymbol{\theta}^{\text{desired}}$. This joint configuration is the one that minimizes the energy function defined by the model $\mathcal{M}$. To perform the minimization, we use gradient search starting from the current joint position $\boldsymbol{\theta}^{\text{current}}$. The algorithm ends when the tracking task is finished.

Note that the previous framework can be used for online learning if the used approximation for the energy function can be updated online. In this case, the input of the algorithm may contain no data but at the beginning of each prediction we update the model $\mathcal{M}$ with the current task-space and joint-space




(a) Rest posture 1. (b) Rest posture 2. (c) Support points of FWGP.

Figure 2: (a) (b) Illustration of different rest postures resulting in an ambiguous data-set. (c) Illustration of the support points in the case of FWGP (no points under the caption).

coordinates $(\mathbf{x}^{\text{current}}, \boldsymbol{\theta}^{\text{current}})$.

An advantage of online adaptive sparsification is its automatic adaptation to the complexity of the task: simple trajectories on low-dimensional manifolds can be represented with fewer data points than complex trajectories in high-dimensional spaces.

A major concern in task-space control is keeping joint-space stability [7, 6]. This requirement can be fulfilled by augmenting the energy function from Equation (2) with a regularization term $\lambda h(\boldsymbol{\theta})$ that helps keeping the joint-space stability. Here, $\lambda$ denotes a regularization parameter. We propose a simple regularization term, i.e., $h(\boldsymbol{\theta}) = 0.5(\boldsymbol{\theta} - \boldsymbol{\theta}_{\text{rest}})^\top(\boldsymbol{\theta} - \boldsymbol{\theta}_{\text{rest}})$, where $\boldsymbol{\theta}_{\text{rest}}$ is a joint configuration far from the joint limits of the robot.

# 3 Evaluations

In this section, we present the evaluation of the proposed methods for task-space tracking. The algorithm is applied to learn the tracking control of a Barrett WAM robot arm [2] (see Figure 5(c) for illustration), and track a figure eight with different settings[4]. If not mentioned otherwise (in Section 3.4), we refer to real-world experiments on the physical robot architecture.

We showed (1) that the proposed framework can be used for ambiguous data-sets, (2) good accuracy can be achieved in real-time setting, (3) it can be used online (learning the robot model while performing the task), and (4) it can be used in non-rigid robot architecture. We refer to the method presented in Section 2.2 as JKSE, in Section 2.3 as SOGP, and in Section 2.4 as FWGP.

## 3.1 Learning from ambiguous data

In this experiment, the task was to track a figure eight defined in task-space with two different postures. The goal of this experiment was to show that the presented framework was capable of learning from ambiguous data-sets. Such data-sets may contain elements where a given end-effector position has been

[4] A figure eight is described by the following equations: $x = \sin(t), y = \text{const}, z = \sin(2t)$, where $t = (0, 2\pi]$. Note that we omitted some scaling parameters for clarity.

reached from different joint configurations – see Figure 4(b) for illustration. We constructed such a data-set by using the analytical controller of the Barrett arm with the Jacobian transpose method [7]. The desired figure eight has been tracked (with the analytical controller) with two different postures (see Figure 2 for an illustration of the two rest postures). Then, the collected training sets have been merged. Data have been sampled at 500Hz. To obtain task-space coordinates, the nominal forward kinematics model of the Barrett WAM has been used.

To apply JKSE, we had to train a OC-SVM model. For this purpose, we used a modified version of libSVM [26] with parameters $\nu = 0.1$ and $g = 1000$. Both for SOGP and FWGP, a modified version of sparse online Gaussian process [27] has been used. For SOGP, we used the parameter settings $C = 1$ and $\omega = 0.007$, while for FWGP $C = 1$ and $\omega = 0.7$. For SOGP the maximum number of base points has been set to 800, for FWGP 100. Both softwares were implemented in C++. These parameter values were obtained by collecting a training set offline and running standard hyper-parameter optimization algorithms, e.g., evidence maximization for GPs. Experiments show that the obtained values can be shared among tasks with the same robot arm. Running hyper-parameter optimization for each learning/tracking task is slow and does not provide significant improvement.

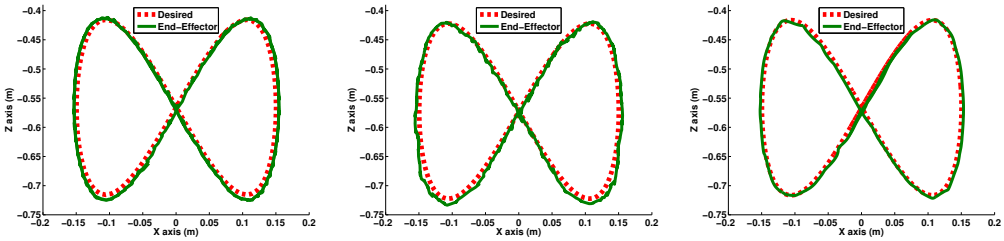
The proposed methods (JKSE, SOGP, and FWGP) were able learn the proper inverse kinematics function on the merged dataset. During the evaluation, they were able to choose the right predictions depending on the initial joint configuration of the robot arm. We also trained a simple GP regression model on the merged data, but it completely failed to learn the inverse kinematics function. This result is not surprising. The previous data-set contained pairs of data $(\mathbf{x}, \boldsymbol{\theta}_1)$ and $(\mathbf{x}, \boldsymbol{\theta}_2)$ where $\boldsymbol{\theta}_1 \neq \boldsymbol{\theta}_2$.

Merging data-set obtained from more than two rest postures does not affect the accuracy of the prediction as long as the rest postures are not *too close* to each other. In this case the inverse kinematics predictions will be averaged. This phenomena is expected since inverse kinematics solutions form a locally convex solution set [6]. How the *closeness* of the rest postures is measured depends on the parametrization of the energy function, e.g., noise level or length scale of a GP.

### 3.2 Offline Task-Space Tracking Control

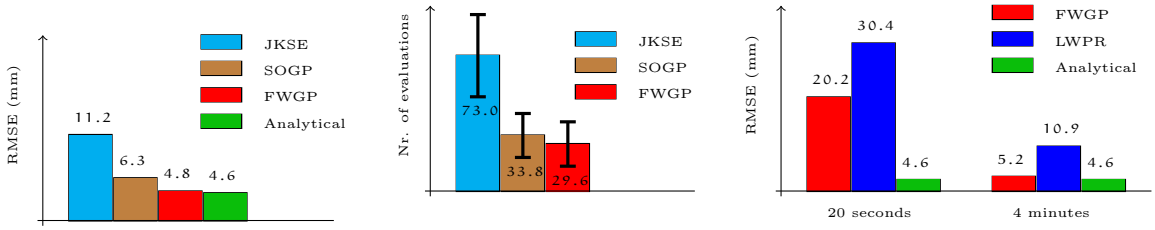
In this experiment, we were interested in the real-time applicability of the proposed method along with the achieved accuracy. The task was to follow a figure eight as a standard benchmark problem for tracking control [4]. We learned the parameters of the models offline, while predictions were calculated in real-time. In order to perform the parameter learning of the model, we required a set of training points. A training set must contain pairs of end-effector positions and the corresponding joint configurations. To avoid oversampling, we only sampled trajectories with end-effector positions in the area where the actual task would take place. For example, when considering the task of drawing a figure eight as shown in Figure 3, the desired end-effector trajectory lies in a plane. Hence, sampling in a volume around this plane fully suffices[5]. Here, we again used the analytical controller (with the Jacobian transpose method)

[5] Note that by this type of training data generation process we evaluated only the interpolation capabilities of the methods (no extrapolation). The extrapolational behaviour is presented in the next Section.




(a) JKSE. (b) SOGP. (c) FWGP.

Figure 3: Results of tracking control for figure eight tracking, when the models have been trained offline. Good accuracy have been achieved with all three approaches.




(a) Tracking accuracy. (b) Number of function (or gradient) evaluations. (c) Tracking accuracy of online learning.

Figure 4: (a) Tracking accuracy and (b) the number of function (or gradient) evaluations for one inverse kinematics predictions. FWGP produced the best results (close to the analytical solution) and at the same time it needed the fewest function evaluations. (c) FWGP outperforms LWPR both after 20 seconds and four minutes of learning and its accuracy converges to the analytical solutions.

with a rectangle representing the plane of the figure eight as the reference trajectory. Note that instead of using the analytical controller, it is also possible to move the arm manually to different positions and record the joint end-effector pairs. Since the automatic movement is faster, we preferred this solution.

Beside JKSE, SOGP, and FWGP, we compared our algorithms with the analytical model that uses the pseudo-inverse method based on the known physical parameters of the robot. We used the same parameters as in the previous experiment, except for SOGP, where we have changed the maximum number of base points to 200.

The results of figure eight tracking are shown on Figure 3, where it can be seen that good accuracy was achieved (the drawing of the figure took 30 seconds). We measured the accuracy of tracking control and showed the results on Figure 4(a). Experiments show that FWGP outperformed JKSE and SOGP. Furthermore, FWGP almost achieved the accuracy of the analytical model. This result is slightly surprising in the light of the number of the points the models were based on. The JKSE model was based on 8456 support vectors, the SOGP model on 200 base points, and the FWGP model on 31 base points, shown on Figure 2(c).

JKSE, SOGP, and FWGP are indirect methods, meaning that the prediction is a result of a min-

imization process. To perform the minimization[6], the function (or its gradient) has to be evaluated several times. We measured the average number of evaluations needed for each method to give one inverse kinematics prediction. Results are shown in Figure 4(b). One can see that SOGP and FWGP needed almost half of evaluations as JKSE, thus, we expect them to be significantly faster.

### 3.3 Online Task-Space Tracking Control

An important feature of tracking control algorithms is their capability if they can be used online, i.e., learning the parameters of the model while performing the task. We argue that JKSE and SOGP are not feasible in real-time online learning. Although, the inclusion of a new data point is theoretically possible for both algorithms – see Laskov et al. [30] and Section 2.5 –, it is too slow when the model contains too many points – 8456 and 200 respectively. Experiments showed that the base points in the FWGP model have never exceeded 40, thus the inclusion of a new points was relatively fast. We used the parameters as in the previous experiments, and for stability, we used $\theta_{\text{rest}} = [0\ 0.5\ 0\ 1.9\ 0\ 0\ 0]^\top$ as rest posture from Section (2.6). This term ensured that we never add physically inconvenient solutions to the base set. FWGP has been compared with LWPR as one of the best online inverse kinematics learning methods. Here, LWPR has been used for the direct kinematics model as presented by Vijayakumar et al. [11]. For LWPR the initial learning rate was set to $\alpha = 250$ and the initial forgetting rate to $\lambda = 0.99$ [11].

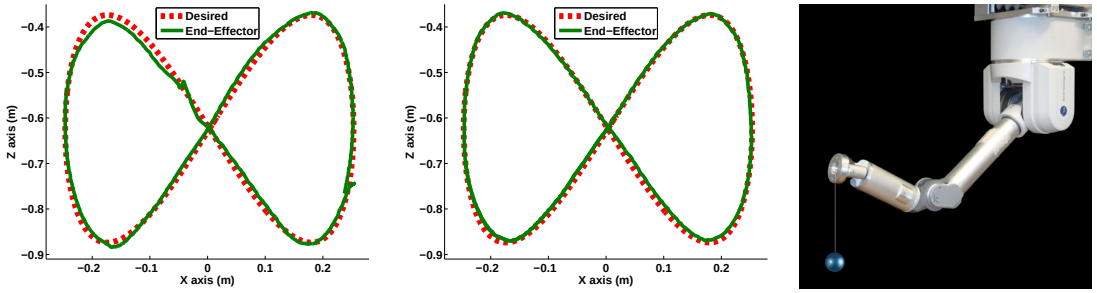
Figure 5(a) shows that after 20 seconds of learning an acceptable tracking accuracy was achieved by FWGP. After four more minutes of learning and tracking, high accuracy was achieved, see Figure 5(b). We present a comparison of tracking accuracy on this task in Figure 4(c). FWGP outperforms LWPR after 20 seconds of interaction time, and approaches the performance of the analytical model.

No comparison with the kinematic Bèzier maps [15] has been made, although it is considered to be a similar method. The reason is that when experimenting with a real robot, authors used some model of the robot (e.g., CAD model), whilst we assumed such a model is not available. Note that the authors claim that kinematic Bèzier maps require $3^7 = 2187$ support points for a robot with seven DoF assuming no noise. FWGP required only 31 points for a 7-DoF robot with noisy measurements (see Figure 2(c)). Figure 2(c) also reveals the extrapolation capabilities of FWGP. Good accuracy is achieved outside the volume defined by the support points.

### 3.4 Task-Space Tracking Control for Non-rigid Robots

In this experiment, we made the *simulated* model more complex by attaching a ball to the end-effector of the arm with a 20 cm string (see Figure 5(c) for illustration). The mass of the ball was negligible compared to the mass of the robot arm and air friction was neglected. The swinging motion of the ball produced a non-linear system. In this new setting, we performed task-space tracking where the position of the ball was considered as the desired position. The task was to track a circle figure with 20 cm radius, see Figure 6(a), on the horizontal plane with the ball. The task was performed in two different

[6] We used the Fletcher-Reeves conjugate gradient algorithm [28] from the GSL library [29].




(a) Tasks-space tracking of a figure eight after 20 seconds of online learning.

(b) Tasks-space tracking of a figure eight after 4 minutes of online learning.

(c) Barrett WAM.

Figure 5: Online task-space tracking learning of the figure eight task by a 7-DoF Barrett WAM robot arm. (a) The tracking performance of FWGP after 20 seconds of online learning is acceptable. (b) After four minutes of learning, very good tracking accuracy is obtained. (c) Illustration of the non-rigid robot when a ball has been attached to the end of the Barrett WAM.

settings: (1) when the desired point moved slowly along the circle (one turn took 24 seconds) and (2) when the desired point moved fast along the circle (one turn took 0.62 seconds). One may think that tracking a circle is an easy task. However, it is hard to perform even for humans who cannot achieve such a good accuracy as our method did.

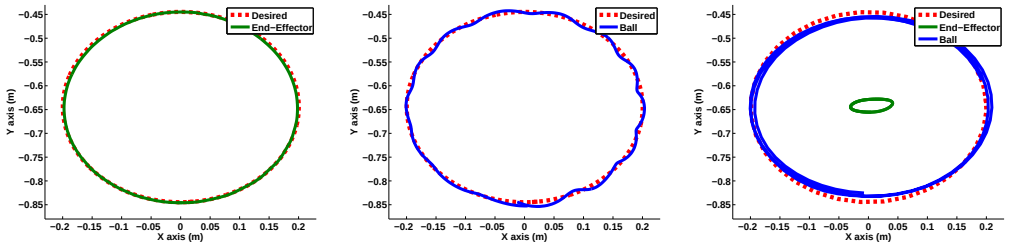
Similar to the previous experiment, only FWGP was able to learn the task. JKSE and SOGP failed due to the high computational requirements. When the ball was moving fast, it was essential to make fast prediction. No comparison with LWPR has been made either as it could not learn the previous tasks sufficiently well.

In the first case, FWGP learned to move the end-effector of the arm right *above* the desired circle while the ball was moving along the desired trajectory since the swinging of the ball was damped. It took four minutes of learning to achieve the learning accuracy presented on Figure 6(a) and Figure 6(b). The GP model was built from 20-25 data points.

In the second case, the trajectory of the end-effector of the arm was moving fast *inside* the desired circle and used the centrifugal force to swing the ball around, see Figure 6(c). After 20 minutes of learning, the GP model was built from 13-15 data points. Following the fast circle movement with the arm itself would be impossible, since it would reach the physical limits of the robot.

We emphasize that the same parameter settings were used for both experiments (with GP hyper-parameters $C = 1$ and $w = 0.7$) Thus, the adaptive behavior depends weakly on the hyper-parameters of the GP. In the first case, FWGP considered the swinging motion of the ball as noise and the trajectory of the ball followed a similar trajectory as the end-effector. On the other hand, in the second experiment, the GP model incorporated into the control model the centrifugal force as well.

We highlight that controlling such an underactuated robot would require dynamic information about the movement of the ball. However, the samples of joint–ball position pairs contain dynamic information as well, since they are a result of an experiment where the dynamic effects were *applied*. How the dynamic




(a) End-effector trajectory of the arm with slow circle learning.

(b) Ball trajectory with slow circle learning.

(c) End-effector and ball trajectory with fast circle learning.

Figure 6: Online task-space tracking learning of a circle following task by a simulated 7-DoF Barrett WAM robot arm with a ball attached on it. (a) When the movement of the desired point on the circle is slow the end-effector of the arm is placed above the desired trajectory while (b) the ball is hanging down and its swinging is damped – the swinging is considered noise by FWGP. (c) When the desired point moves fast on the circle the end-effector of the arm moves inside the desired circle while the ball swings around along the desired trajectory – the centrifugal force is incorporated into the control model.

information is extracted from the samples is implicitly done by the learning algorithm.

We found that the applicability of FWGP, in the case of underactuated robots, is limited to relatively simple trajectories (although, as we have mentioned above, circle tracking is not an easy task, despite a circle being a simple trajectory). For example, the learning process of a figure eight resulted in a controller which accuracy was barely beyond random movements. We believe that accurate tracking exceeds the physical limitations of the robot, specially the wrist joints. Note that our method does not provide any guarantees about the solvability of an underactuated control problem.

# 4 Discussion

In this paper, we proposed a framework to learn the inverse kinematics function of robots with redundant manipulators using structured output machine learning methods. The method learns the *direct* inverse kinematics function on position level. By modeling the function in an indirect way, we also addressed the problem of non-uniqueness of inverse kinematics, highlighting that regular regression algorithms are not capable of modeling it. For the joint energy function, we proposed three different approximations based on support vector machines and Gaussian processes. Real world experiments show that all three methods could achieve good performance and FWGP produced the best results. The presented results show that the accuracy of the state-of-the-art method LWPR and the analytical model is achieved.

We presented one experiment where FWGP was capable of learning the inverse kinematics model of a non-rigid robot, where the centrifugal force had to be incorporated into the kinematics model. This experiment suggest that learning such a control is possible, although, it is not *always* possible.

We proposed a general framework, where the energy function of the joint input-output data can be modeled arbitrarily. Other approximations may lead to faster or more accurate inverse kinematics

prediction. Another possible future direction may be the application of the same idea of joint data representation for other robot control tasks, such as inverse dynamics learning [31].

## Acknowledgments

We acknowledge the support of the Romanian Ministry of Education [grant PN-II-RU-TE-2011-3-0278], and the support of the European Community's F.P.7 programme [grant ICT-270327 CompLACS].

## REFERENCES

[1] Botond Bócsi, Lehel Csató, and Jan Peters. Structured output Gaussian processes. Technical report, Babes-Bolyai University, 2011. URL http://www.cs.ubbcluj.ro/~bboti/pubs/sogp_2011.pdf.

[2] Botond Bócsi, Duy Nguyen-Tuong, Lehel Csató, Bernhard Schoelkopf, and Jan Peters. Learning inverse kinematics with structured prediction. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, pages 698–703, San Francisco, USA, 2011.

[3] Botond Bócsi, Philipp Hennig, Lehel Csató, and Jan Peters. Learning tracking control with forward models. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 259–264, St. Paul, MN, USA, 2012.

[4] J. Nakanishi, R. Cory, M. Mistry, J. Peters, and S. Schaal. Operational space control: A theoretical and emprical comparison. *International Journal of Robotics Research*, (6):737–757, 2008.

[5] Camille Salaün, Vincent Padois, and Olivier Sigaud. Learning forward models for the operational space control of redundant robots. In *From Motor Learning to Interaction Learning in Robots*, volume 264 of *Studies in Computational Intelligence*, pages 169–192. Springer, 2010.

[6] Aaron D'Souza, Sethu Vijayakumar, and Stefan Schaal. Learning inverse kinematics. In *Proceedings of the IEEE Inter. Conf. on Intelligent Robots and Systems*. Piscataway, NJ: IEEE, 2001.

[7] Lorenzo Sciavicco and Bruno Siciliano. *Modelling and Control of Robot Manipulators (Advanced Textbooks in Control and Signal Processing)*. Advanced textbooks in control and signal processing. Springer, 2nd edition, January 2005.

[8] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

[9] Christoph H. Lampert and Matthew B. Blaschko. Structured prediction by joint kernel support estimation. *Machine Learning*, 77:249–269, December 2009.

[10] J.X. Xu, S.K. Panda, and T.H. Lee. *Real-time Iterative Learning Control: Design and Applications*. Advances in Industrial Control. Springer, 2008.

[11] Sethu Vijayakumar, Aaron D'Souza, and Stefan Schaal. Incremental online learning in high dimensions. *Neural Computation*, 17:2602–2634, 2005.

[12] Eimei Oyama, Nak Young, and Chong Arvin Agah. Inverse kinematics learning by modular architecture neural networks with performance prediction networks. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1006–1012, 2001.

[13] Michael I. Jordan and David E. Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16:307–354, 1992.

[14] Ganghua Sun and Brian Scassellati. Reaching through learned forward model. In *Proceedings of the IEEE-RAS/RSJ International Conference on Humanoid Robots*, Santa Monica, 2004.

[15] Stefan Ulbrich, Vicente Ruiz de Angulo, Tamim Asfour, Carme Torras, and Rüdiger Dillmann. Kinematic bézier maps. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 42(4): 1215–1230, 2012.

[16] J. Peters and S. Schaal. Learning to control in operational space. *International Journal of Robotics Research*, 27(2):197–212, 2008.

[17] Duy Nguyen-Tuong and Jan Peters. Online kernel-based learning for task-space tracking robot control. *IEEE Transactions on Neural Networks and Learning Systems*, 23(9):1417–1425, 5 2012.

[18] Bernhard Schölkopf, John C. Platt, John C. Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13:1443–1471, July 2001.

[19] Carl E. Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2005.

[20] Bernhard Schölkopf and Alex Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. MIT-Press, Cambridge, MA, 2002.

[21] Lehel Csató and Manfred Opper. Sparse on-line Gaussian processes. *Neural Computation*, 14(3): 641–668, 2002.

[22] Joaquin Quiñonero Candela and Carl E. Rasmussen. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959, 2005.

[23] Neil D. Lawrence, Matthias Seeger, and Ralf Herbrich. Fast sparse Gaussian process methods: The informative vector machine. In *Neural Information Proces. Sys.*, pages 609–616. MIT Press, 2002.

[24] Edward Snelson. Local and global sparse Gaussian process approximations. *Artificial Intelligence and Statistics*, 11, 2006.

[25] Manfred Opper. *A Bayesian approach to on-line learning*, pages 363–378. Cambridge University Press, 1998.

[26] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[27] Dan Grollman. *Sparse Online Gaussian Process C++ Library*, 2012. Software `http://cs.brown.edu/people/dang/code.shtml`.

[28] Jan A. Snyman. *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms*. Applied Optimization. Springer-Verlag New York, 2005.

[29] Mark Galassi, Jim Davies, James Theiler, Brian Gough, Gerard Jungman, Michael Booth, and Fabrice Rossi. *Gnu Scientific Library: Reference Manual*, 2003. Software `http://www.gnu.org/software/gsl/manual/html_node/index.html`.

[30] Pavel Laskov, Christian Gehl, Stefan Krüger, and Klaus-Robert Müller. Incremental support vector learning: Analysis, implementation and applications. *J. Mach. Learn. Res.*, 7:1909–1936, 2006.

[31] Duy Nguyen-Tuong and Jan Peters. Using model knowledge for learning inverse dynamics. In *IEEE International Conference on Robotics and Automation*, pages 2677–2682, 2010.