Robot Learning by Demonstration with Local Gaussian Process Regression

Markus Schneider, Wolfgang Ertel
University of Applied Sciences Ravensburg-Weingarten
Email: {markus.schneider, ertel}@hs-weingarten.de

***Abstract*— In recent years there was a tremendous progress in robotic systems, and however also increased expectations: A robot should be easy to program and reliable in task execution. *Learning from Demonstration* (LfD) offers a very promising alternative to classical engineering approaches. LfD is a very natural way for humans to interact with robots and will be an essential part of future service robots. In this work we first review heteroscedastic Gaussian processes and show how these can be used to encode a task. We then introduce a new Gaussian process regression model that clusters the input space into smaller subsets similar to the work in [11]. In the next step we show how these approaches fit into the Learning by Demonstration framework of [2], [3]. At the end we present an experiment on a real robot arm that shows how all these approaches interact.**

I. INTRODUCTION

In the last years robots have turned from simple pre-programmed machines into highly flexible and complex systems. Therefore, the programming of such a robot is very difficult and other human-robot interfaces are needed. This is one reason why *Learning from Demonstration* (LfD), also referred to as *Programming by Demonstration* (PbD), has become a major topic in the context of robotics [1], [18].

Current approaches for robot skill representations can be broadly divided in two layers called *symbolic encoding* and *trajectory encoding*. An encoding at a symbolic level describes a sequence of primitives that are already known and given in advance. In most cases a graph-like representation of the task and the environment is created where each state is a node of the graph and actions are directed links between these states. In contrast, *trajectory encoding* tries to approximate the underlying teacher policy (the state-to-action mapping) directly and is less goal oriented than the symbolic approach. This is also very often referred to as *action primitive*. Typically, regression techniques are used to retrieve a generalized version of these low-level motions (joint positions, dynamics, torques). For example, *Locally Weighted Regression* (LWR) in combination with differential equations was used in [9] to form so called *motor primitives* and [8] using a condence-based learning approach for teaching low-level robotic skills.

The recent work of [2] and [3] showed, that *Gaussian Mixture Models* (GMM) and *Gaussian Mixture Regression* (GMR) can be used to encode demonstrated robot trajectories. They showed successfully that the important features of a task can be extracted using the change in variability between demonstrations over time. In this work, we show how *Gaussian processes* (GP) can be used for the same purpose and how to deal with the extremely high computational costs. Gaussian processes had already been proposed in the context of robot Learning from Demonstration [7], [17], but have never been used to extract task constraints as in this work. After a short review we extend the standard Gaussian process model using *heteroscedasticity* [10], [13]. We then show how this can be used to extract properties of a task called *constraints*. Further, we show that this encoding is able to generalize over multiple demonstrations in order to create smooth and continuous trajectories. In a second step we introduce a new *local* Gaussian process algorithm which extremely reduces the computational costs compared to the standard Gaussian process model. In contrast to the work of [11] it can also handle the GP prediction variances.

This work was supported by the Collaborative Center of Applied Research on Service Robotics (ZAFH Service Robotics).

II. GAUSSIAN PROCESS MODELS

A. Gaussian process review

In recent years, Gaussian processes became very popular in the context of machine learning for regression and classification problems [15]. They are powerful tools to solve non-linear problems while requiring only relatively simple linear algebra. The Gaussian process theory provides a very natural way to define a prior distribution over (regression) functions. In the standard non-linear regression problem, we try to estimate a latent function $f(\boldsymbol{x})$ given input variables $\boldsymbol{x} \in \mathbb{R}^D$ and noisy observed target values $y \in \mathbb{R}$ modeled as $y = f(\boldsymbol{x}) + \epsilon$, where $\epsilon \in \mathbb{R}$ is a random noise variable that is *independent, identically distributed* (i.i.d.) for each observation. The training data set comprising n input points together with the corresponding observations is denoted by $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)|_{i=1}^n\}$. The Gaussian process regression model tries to learn the predictive distribution $p(f^*|\boldsymbol{x}^*, \mathcal{D})$ of a new test output f^* given a test input $\boldsymbol{x}^*$. To simplify notation, all training inputs $\{\boldsymbol{x}_i\}_{i=1}^n$ are collected in a so called *design matrix* $\boldsymbol{X}$ of size $D \times n$ and we define the matrix $\boldsymbol{X}^*$ and the vectors $\boldsymbol{f}^*, \boldsymbol{y}$ in the same way. The key in Gaussian processes is to consider the training outputs $\boldsymbol{y}$ and the new testing points (prediction values) $\boldsymbol{f}^*$ as a sample from the same (zero mean) multivariate Gaussian distribution. The predictive distribution is then again a multivariate Gaussian

distribution $\mathcal{N}(\bar{\boldsymbol{f}}^*, \text{cov}[\boldsymbol{f}^*])$ conditioned on the training data with mean

$$\bar{\boldsymbol{f}}^* = \boldsymbol{K}^*(\boldsymbol{K} + \sigma_n^2 \boldsymbol{I})^{-1}\boldsymbol{y} \tag{1}$$

and covariance matrix $\text{cov}[\boldsymbol{f}^*]$

$$\text{cov}[\boldsymbol{f}^*] = \boldsymbol{K}^{**} - \boldsymbol{K}^*(\boldsymbol{K} + \sigma_n^2 \boldsymbol{I})^{-1}\boldsymbol{K}^{*T}, \tag{2}$$

where $\boldsymbol{I}$ is the identity matrix, σ_n^2 is the noise variance $\boldsymbol{K} \in \mathbb{R}^{n \times n}, \boldsymbol{K}_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $\boldsymbol{K}^* \in \mathbb{R}^{n^* \times n}, \boldsymbol{K}^*_{i,j} = k(\boldsymbol{x}^*_i, \boldsymbol{x}_j)$ and $\boldsymbol{K}^{**} \in \mathbb{R}^{n^* \times n^*}, \boldsymbol{K}^{**}_{i,j} = k(\boldsymbol{x}^*_i, \boldsymbol{x}^*_j)$. The function $k(\cdot, \cdot)$ is called covariance function, or kernel, and is used to construct the covariance matrices $\boldsymbol{K}, \boldsymbol{K}^*, \boldsymbol{K}^{**}$. The corresponding variance $\mathbb{V}[\boldsymbol{f}^*]$ is given by the diagonal elements of $\text{cov}[\boldsymbol{f}^*]$. We will write the Gaussian process as $\mathcal{GP}(\boldsymbol{0}, k(\cdot, \cdot))$ or simply $\mathcal{GP}$. Typically, the covariance function depends on parameters $\boldsymbol{\omega}$ which are called *hyper parameters* because they are determined in advance and not used to absorb the training data information. A widely used covariance function is given by the exponential of a quadratic form, namely the *Squared Exponential* (SE) to give

$$k(\boldsymbol{x}_p, \boldsymbol{x}_q) = \sigma_f^2 \exp\left(-\frac{\|\boldsymbol{x}_p - \boldsymbol{x}_q\|^2}{2l^2}\right), \tag{3}$$

with $\boldsymbol{\omega} = (\sigma_n, \sigma_f, l)^T$ defining the *noise level*, *signal variance* and *characteristic length scale*, respectively. We use $p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\omega})$ to obtain the *log marginal likelihood* given by

$$\log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\omega}) = -\frac{1}{2}\boldsymbol{y}^T \boldsymbol{K}_y^{-1}\boldsymbol{y} - \frac{1}{2}\log|\boldsymbol{K}_y| - \frac{n}{2}\log 2\pi, \tag{4}$$

where $\boldsymbol{K}_y$ substitutes $\boldsymbol{K} + \sigma_n^2 \boldsymbol{I}$ and $|\cdot|$ is the determinant of a matrix. We then use a *conjugate gradient* algorithm [12] to (locally) maximize the log marginal likelihood function with respect to the hyper parameters.

B. Heteroscedastic Gaussian Processes

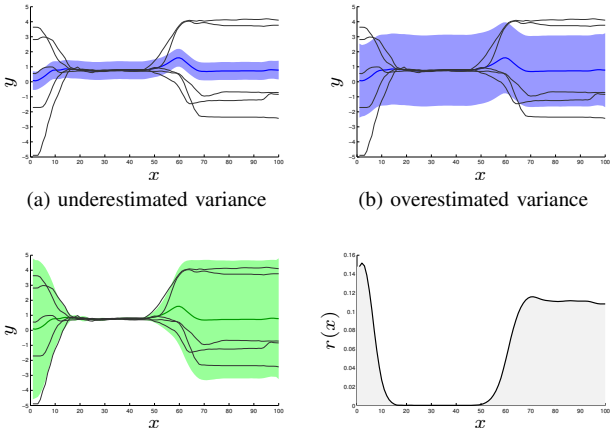
The Gaussian process model described so far assumes a constant noise level. Given a data set that requires a variable noise model as in Fig. 1 it would be able to correctly estimate the mean value, but it would fail to correctly estimate the variance. We therefore have to introduce a more flexible noise model called *heteroscedasticity*. Replacing the constant noise rate σ_n by an input dependent noise function $r(\boldsymbol{x})$, the mean and covariance function of the predictive distribution changes to

$$\bar{\boldsymbol{f}}^* = \boldsymbol{K}^*(\boldsymbol{K} + \boldsymbol{R})^{-1}\boldsymbol{y} \quad \text{and} \tag{5}$$

$$\text{cov}[\boldsymbol{f}^*] = \boldsymbol{K}^{**} + \boldsymbol{R}^* - \boldsymbol{K}^*(\boldsymbol{K} + \boldsymbol{R})^{-1}\boldsymbol{K}^{*T}, \tag{6}$$

respectively. $\boldsymbol{R}$ is defined as $\text{diag}(\boldsymbol{r})$, with $\boldsymbol{r} = (r(\boldsymbol{x}_1), \ldots, r(\boldsymbol{x}_n))^T$, and $\boldsymbol{R}^* = \text{diag}(\boldsymbol{r}^*)$, with $\boldsymbol{r}^* = (r(\boldsymbol{x}^*_1), \ldots, r(\boldsymbol{x}^*_{n^*}))^T$. We follow [6] and use a second independent Gaussian process (the z-process denoted by $\mathcal{GP}_z$) to model the log of the noise level giving $z(\boldsymbol{x}) = \log(r(\boldsymbol{x})) = (z_1, \ldots, z_n)^T$. The z-process maintains its own independent covariance function $k_z(\cdot, \cdot)$ and parameters ω_z. With $\boldsymbol{z}^*$ as the $\mathcal{GP}_z$ posterior prediction, we obtain the predictive distribution $p(\boldsymbol{f}^*|\boldsymbol{X}^*, \mathcal{D}) =$

$$\int\int p(\boldsymbol{f}^*|\boldsymbol{X}^*, \mathcal{D}, \boldsymbol{z}, \boldsymbol{z}^*) p(\boldsymbol{z}, \boldsymbol{z}^*|\boldsymbol{X}^*, \mathcal{D})\, d\boldsymbol{z}\, d\boldsymbol{z}^*, \tag{7}$$



(a) underestimated variance (b) overestimated variance

(c) heteroscedastic GP (d) latent local noise rate

Fig. 1: An example of a standard Gaussian process applied to a data set that requires a variable noise model. The data set shown in the panels (a), (b) and (c) consists of five trajectories (black lines) recorded with the sensors of a real physical robot arm. The bold solid line represents the Gaussian process mean and the shaded region denotes twice the standard deviation for the input. It can be clearly seen that the GP in (a) underestimates the variance from $x = 50$ to $x = 100$, whereas the GP in (b) overestimates the variance from $x = 10$ to $x = 50$. Panel (c) illustrates the resulting heteroscedastic Gaussian process after the EM run. Its internal noise model is illustrated in (d).

where the last term prevents an analytical solution of the integral. A common method is to approximate $p(\boldsymbol{f}^*|\boldsymbol{X}^*, \mathcal{D}) \approx p(\boldsymbol{f}^*|\boldsymbol{X}^*, \mathcal{D}, \tilde{\boldsymbol{z}}, \tilde{\boldsymbol{z}}^*)$, where $\tilde{\boldsymbol{z}}, \tilde{\boldsymbol{z}}_* = \arg\max_{\boldsymbol{z}, \boldsymbol{z}^*} p(\boldsymbol{z}, \boldsymbol{z}^*|\boldsymbol{X}^*, \mathcal{D})$. We use the expectation-maximization (EM) algorithm, introduced by [10], to iteratively estimate $p(\boldsymbol{f}^*|\boldsymbol{X}^*, \mathcal{D})$.

C. Local Gaussian Processes (LGP)

The heteroscedastic Gaussian processes described so far are one of the most flexible and accurate regression methods, but this comes at the costs of $\mathcal{O}(n^3)$ computing time to invert the training data covariance matrix $(\boldsymbol{K}+\boldsymbol{R})^{-1}$ in (5) and (6). Solutions to this problem can be divided into two groups: (i) *global (or sparse) GP approximations* [4], [5], [19] typically use a much smaller set of so called support points. The main problem is to find sufficient good support points that are representative for the whole input space. The other group contains (ii) *local GP approximations* or *mixture of experts (ME)* [11], [14], [20], where the input space is divided into smaller subspaces and only points "near" the test point are considered in prediction.

In this work we use a local GP approximation, where our input space clustering is similar to the procedure described in [11]. The idea in our approach is to assign the training inputs to the local model which fits best and then train an individual Gaussian process on each of these models. A training input $\boldsymbol{x}_i$ will be assigned to the model which center $\boldsymbol{c}_p$ has the

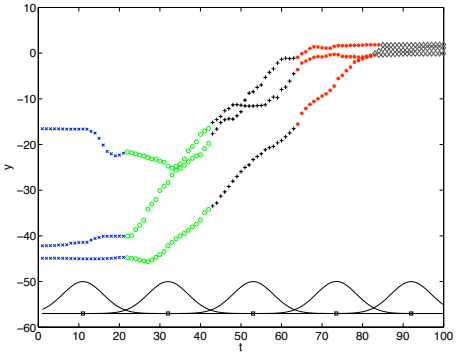


Fig. 2: The graph illustrates the input space clustering. The data set was split up into five individual subsets and assigned to local models indicated by the different markers/colors. The squares at the bottom of the figure and the Gaussian looking shape are the centers and the weighting functions of the local models, respectively.

highest covariance with the training input $w_k = k(\boldsymbol{x}_i, \boldsymbol{c}_p)$. This is more reasonable than a simple distance function, because it takes the properties of more complex covariance functions[1] into account. If no local model obtains a sufficient high covariance value, a new model will be created at the position of the training input. The clustering steps are summarized in Algorithm 1. This reduces the computation costs to $\mathcal{O}(n_l^3)$ for each of the L local models, where n_l presents the number of data points in a local model which is typically much smaller than the total number of training inputs n.

We calculate the prediction mean $\bar{\boldsymbol{f}}^*$ and covariance $\text{cov}[\boldsymbol{f}^*]$ as the Gaussian product of the local model predictions:

$$z \cdot \mathcal{N}(\bar{\boldsymbol{f}}^*, \text{cov}[\boldsymbol{f}^*]) = \prod_{l=1}^{L} \mathcal{N}(\bar{\boldsymbol{f}}_l^*, \boldsymbol{\Sigma}_l = diag(\mathbb{V}[\boldsymbol{f}^*]_l)), \quad (8)$$

where

$$\text{cov}[\boldsymbol{f}^*] = \left(\sum_{l=1}^{L} \boldsymbol{\Sigma}_l^{-1} \right)^{-1}, \bar{\boldsymbol{f}}^* = \text{cov}[\boldsymbol{f}^*] \left(\sum_{l=1}^{L} \boldsymbol{\Sigma}_l^{-1} \bar{\boldsymbol{f}}_l^*, \right)^{-1} \quad (9)$$

and z is a normalization constant. Note that we construct a diagonal matrix from the local variance prediction which is easy to invert, instead of taking the local covariance prediction $\text{cov}[\boldsymbol{f}^*]$ which would lead to costs of $\mathcal{O}(n^{*3})$ for predictions. The prediction procedure is summarized in Algorithm 2. A comparison of computation speed for the standard and local Gaussian process models is given in the appendix.

III. THE LEARNING FROM DEMONSTRATION FRAMEWORK

We use the same learning from demonstration framework as described in [2] to encode a task and summarize the used methods. During a demonstration, sensor information is collected and stored as a state/observation $\boldsymbol{\xi} = (\xi_t^{(i)}, \boldsymbol{\xi}_s^{(i)}) \in \mathbb{R}^D$. Here, $\xi_t^{(i)}$ denotes the i-th *temporal value* $\in \mathbb{R}$ and $\boldsymbol{\xi}_s^{(i)} \in \mathbb{R}^{(D-1)}$ is the i-th vector of *spatial values*. A mapping $\pi : \mathbb{R} \rightarrow \mathbb{R}^{D-1}, \xi_t^{(i)} \rightarrow \boldsymbol{\xi}_s^{(i)}$ is called a *policy*. In our experiments described in section IV we will use a simple timestamp t as the temporal value. The spatial value is represented through coordinates in *joint space* $(\theta_1, \ldots, \theta_k$ for the k motor encoders of the robot) or *task space* (Cartesian coordinates and orientation of the end effector). Suppose we have n demonstrations and each demonstration is resampled to a fixed size of T, then the data set $\mathcal{D}$ of all observations will be of length $N = nT$, formally $\mathcal{D} = \{(\xi_t^{(i)}, \boldsymbol{\xi}_s^{(i)}) | i = 1, \ldots, N\}$. We use an independent GP to model each dimension of $\boldsymbol{\xi}_s^{(i)}$ so π consists of $D-1$ Gaussian processes. This approach perfectly fits the requirements of an approximation in the context of robots, namely generating *continuous and smooth paths* and provide a *generalization over multiple demonstrations*. The Gaussian process covariance function controls the function shape and ensures a smooth path, whereas the fundamental GP algebra calculates the mean over demonstrations. This reduces noise (introduced by sensors and human jitter during the demonstration) and recovers the underlying trajectory.

The key idea is to use the variations and similarities

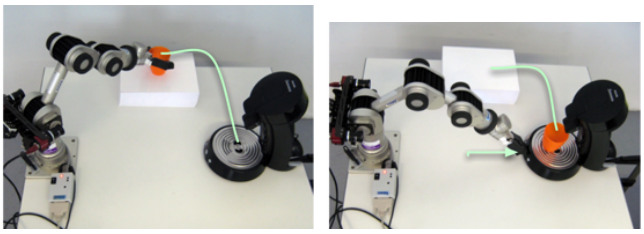
[1] As for example periodic covariance functions

Algorithm 1 Input Space Clustering & Learning

input: $\boldsymbol{X}$ (inputs), $\boldsymbol{y}$ (observations), $k(\cdot,\cdot)$ (the GP covariance function)
init number of local models to zero
$L = 0$
for each data point $(\boldsymbol{x}_i, y_i)$ **do**
  for $l = 1 \rightarrow L$ **do**
    Compute covariance between the model center and the input
    $w_l = k(\boldsymbol{x}_i, \boldsymbol{c}_l)$
  end for
  Choose the model with the highest covariance value
  $p = \arg\max_p w_p$
  if $w_p >$ min. required weight and $L > 0$ **then**
    insert $(\boldsymbol{x}_i, y_i)$ to the local model data set
    $\tilde{\boldsymbol{X}}_p = \tilde{\boldsymbol{X}}_p \cup \boldsymbol{x}_i, \tilde{\boldsymbol{y}}_p = \tilde{\boldsymbol{y}}_p \cup y_i$
    update model mean
    $\boldsymbol{c}_p = \text{MEAN}(\tilde{\boldsymbol{X}}_p)$
  else
    create a new local model
    $\boldsymbol{c}_{l+1} = \boldsymbol{x}_i, \tilde{\boldsymbol{X}}_{l+1} = \boldsymbol{x}_i, \tilde{\boldsymbol{y}}_{l+1} = y_i$
    $L = L + 1$
  end if
end for
for $l = 1 \rightarrow L$ **do**
  train l-th Gaussian process
  $\mathcal{GP}_l =$ Gaussian process posterior using $\tilde{\boldsymbol{X}}_l, \tilde{\boldsymbol{y}}_l$
end for
return: local models

Algorithm 2 Local Gaussian Process Prediction

input: $\boldsymbol{X^*}$ (test inputs)
for $l = 1 \rightarrow L$ **do**
  Compute local Gaussian process posterior prediction
  $[\bar{\boldsymbol{f}}_l^*, \mathbb{V}[\boldsymbol{f}^*]_l] = \mathcal{GP}_l(\boldsymbol{X^*})$
  $\boldsymbol{\Sigma}_l = diag(\mathbb{V}[\bar{\boldsymbol{f}}^*]_l)$
end for
$\mathcal{N}(\bar{\boldsymbol{f}}^*, \text{cov}[\boldsymbol{f}^*]) = \prod_{l=1}^{L} \mathcal{N}(\bar{\boldsymbol{f}}_l^*, \boldsymbol{\Sigma}_l)$
return: $\bar{\boldsymbol{f}}^*$ (prediction mean), $\text{cov}[\boldsymbol{f}^*]$ (prediction covariance)



(a) Subtask 1 (b) Subtask 2

Fig. 3: The *coffee machine* task. The first subgoal is to move the robot arm in proper position in order to grasp the orange cup. Subsequently, the robot has to move under the machine carefully , place the cup and move a little bit away in order to be able to press the small button at the front panel of the coffee machine.

between demonstrations to extract *what* is important to imitate. Assume a robot arm scenario where the goal is to reach a specific end state (e.g. given 3D coordinates) and the start state is chosen randomly. The trajectories will differ significantly at the beginning, whereas the position should be more or less the same at the end. Such a part of a trajectory with low variation is defined as a *constraint* because no discrepancy is desired. The *heteroscedastic* Gaussian process model, as discussed in section II, allows us to encode both the trajectory and the variation between the demonstrations (see also Fig. 1). The lower the variance of the Gaussian process prediction is, the stronger is the constraint.

In the training (encoding) phase, policies are created with respect to the (absolute) joint angle trajectories θ and relative to all $m \in M$ objects $\boldsymbol{o}^{(m)}$ detected in the environment. More precisely, the relative position of the robot's end effector to the initial position of each object (in Cartesian coordinates) is calculated.

In the reproduction phase, the task space predictions are transformed into joint angle predictions and then all predictions are weighted by their variance to form an overall controller. More precisely, a Cartesian trajectory estimation $\hat{\boldsymbol{x}}^{(m)}$ and a related covariance matrix $\hat{\boldsymbol{\Sigma}}^{x(m)}$ for each detected object[2] is calculated by the Gaussian process policies, giving M task space constraints. These constraints are used together with the joint angle estimation $\hat{\boldsymbol{\theta}}$ and related covariance matrix $\hat{\boldsymbol{\Sigma}}^{\theta}$ to create a final policy which considers constraints in both Cartesian space and joint space. The projection from Cartesian space to joint space is done with a pseudoinverse Jacobian algorithm that compensates the missing orientation information of the end effector. Algorithm 3 shows this procedure and is described in [2] in more detail.

IV. EXPERIMENTS

The experiment consists of two subtasks: First, grasp the cup somewhere on the table and place it under the coffee machine as in Fig. 3. Second, press the button on the machine. Especially the movement to reach the button is very complex, because the robot has to move a little bit away from the machine without hitting the cup or anything else.

[2]It is no problem if some of the objects from the training phase are not detected. Constraints with respect to these objects will be skipped.

Algorithm 3 Reproduction

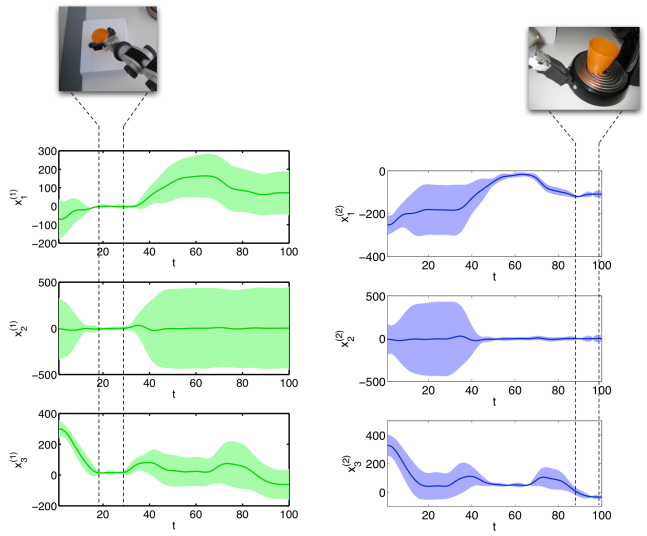
input: $\hat{\boldsymbol{\theta}}$ ($\mathcal{GP}$ estimate joint space), $\hat{\boldsymbol{\Sigma}}^{\theta}$ (joint space covariances), $\boldsymbol{o}^{(m)}$ (M objects), $\hat{\boldsymbol{x}}^{(m)}$ ($\mathcal{GP}$ estimate relative to objects), $\hat{\boldsymbol{\Sigma}}^{x(m)}$ (covariances relative to objects)
$\alpha = 0.5$ (weight factor for Jacobian null space optimization)
Set initial values by direct kinematics (DK)
$\boldsymbol{\theta}_0 = \hat{\boldsymbol{\theta}}_0, \quad \boldsymbol{x}_0 = \mathrm{DK}(\hat{\boldsymbol{\theta}}_0)$
for $t = 0 \rightarrow T$ **do**
  Compute Jacobian matrix, where k is the number of joints and p is the number of elements in the Cartesian coordinate vector:
  $\boldsymbol{J}(\boldsymbol{\theta}_t) = \left(\frac{\partial x_i}{\partial \theta_{t,j}} \right)_{i,j} \Big| \, i \in [1, \ldots, p], j \in [1, \ldots, k]$
  Compute the pseudoinverse of the Jacobian matrix
  $\boldsymbol{J}^{\dagger}(\boldsymbol{\theta}_t) = (\boldsymbol{J}(\boldsymbol{\theta}_t)^T \boldsymbol{J}(\boldsymbol{\theta}_t))^{-1} \boldsymbol{J}(\boldsymbol{\theta}_t)^T$
  Approximate dynamics with Δ-values for the M objects and apply the pseudoinverse Jacobian method.
  for $m = 1 \rightarrow M$ **do**
    $\Delta \boldsymbol{x}_{t+1}^{(m)} = \boldsymbol{o}^{(m)} + \hat{\boldsymbol{x}}_{t+1}^{(m)} - \boldsymbol{x}_t$
    $\Delta \boldsymbol{e}_{\theta} = \hat{\boldsymbol{\theta}}_{t+1} - \boldsymbol{\theta}_t$
    $\Delta \boldsymbol{\theta}_{t+1}^{(m)} = \boldsymbol{J}^{\dagger}(\boldsymbol{\theta}_t) \Delta \boldsymbol{x}_{t+1}^{(m)} + \alpha \left(\boldsymbol{I} - \boldsymbol{J}^{\dagger}(\boldsymbol{\theta}_t) \boldsymbol{J}(\boldsymbol{\theta}_t) \right) \Delta \boldsymbol{e}_{\theta}$
    $\boldsymbol{\Sigma}_{t+1}^{(m)} = \boldsymbol{J}^{\dagger}(\boldsymbol{\theta}_t) \hat{\boldsymbol{\Sigma}}_{t+1}^{x(m)} \left(\boldsymbol{J}^{\dagger}(\boldsymbol{\theta}_t) \right)^T$
  end for
  Compute posture in joint & task space by Gaussian product rule : $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \prod_{m=1}^{M} \mathcal{N}\left(\Delta \boldsymbol{\theta}_{t+1}^{(m)}, \boldsymbol{\Sigma}_{t+1}^{(m)} \right) \mathcal{N}\left(\hat{\boldsymbol{\theta}}_{t+1} - \boldsymbol{\theta}_t, \boldsymbol{\Sigma}_{t+1}^{\theta} \right)$
  Update new position by direct kinematic:
  $\boldsymbol{x}_{t+1} = \mathrm{DK}(\boldsymbol{\theta}_{t+1})$
end for
return: $\boldsymbol{\theta}$ (trajectory to reproduce)

The experiments were performed on a Katana 6M from Neuronics. This is a lightweight portable robot arm with six degrees of freedom (including one degree for the gripper). For the *kinesthetic teaching process* all motors are set to *passive mode* which allows the user to freely move the robot arm. During the movement, all motor encoder values are recorded at a sampling rate of 100 Hz. The data set for training consists of four trajectories with nine variables describing the five joint angles of the arm, the gripper and three Cartesian coordinates representing the position of the end effector. We resample each trajectory to $T = 2000$ data points in a preprocessing step. The initial positions of the cup and the coffee machine are gathered by the object detection prior to the beginning of each demonstration.

We use an independent local, heteroscedastic Gaussian process to encode each of the five joint angle trajectories and for each dimension of the path relative to the objects in Cartesian space. This yields to six Gaussian processes for the joint space, three for the cup and three for the coffee machine. To achieve a smooth trajectory, we use a squared exponential covariance function with a single length scale parameter and an additional parameter for the overall process variance as in Eq. 3. We also directly introduce a noise term into the covariance function to filter out small human jitter during demonstration giving

$$k(\boldsymbol{\xi}_t^{(p)}, \boldsymbol{\xi}_t^{(q)}) = \sigma_f^2 \exp\left(-\frac{\|\boldsymbol{\xi}_t^{(p)} - \boldsymbol{\xi}_t^{(q)}\|^2}{2l^2} \right) + \delta_{pq} \sigma_n^2, \quad (10)$$

where δ_{pq} is the *Kronecker delta*. This function is used for each of the local Gaussian processes and its noise level pro-



(a) Relative to cup (b) Relative to coffee machine

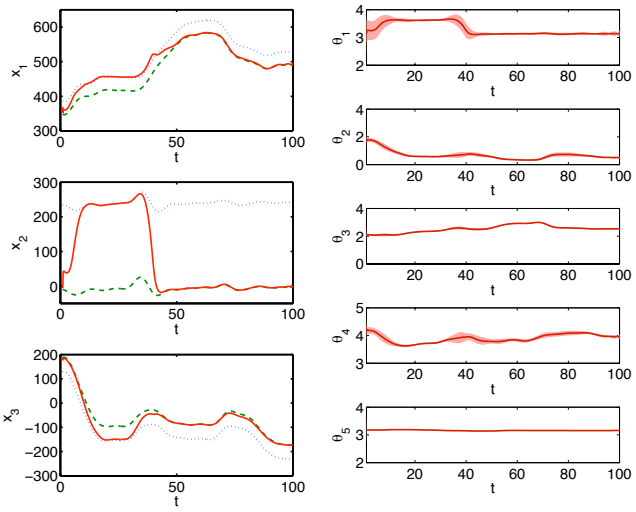
Fig. 4: The learned policies relative to the initial positions of the cup (a) and the coffee machine (b) in Cartesian coordinates. We can see that the movement relative to the cup is highly constrained between time steps 20 and 30. At this time, the end effector reaches the cup and grasps it. In contrast, the movement relative to the coffee machine is constrained when it places the cup (time steps 50 to 65) and when the button is pressed (time steps 85 to 100).

cess, but with individual parameterization and optimization.

The learned policies relative to the initial positions of the cup and the coffee machine in Cartesian coordinates can be seen in Fig. 4a and Fig. 4b, respectively. The reproduction trajectories generated by the overall policy for a new constellation of cup and coffee machine positions are illustrated in Fig. 5. In some cases the robot was not able to press the tiny button and stopped a few millimeters in front of it or don't pressed it deep enough. However, the robot always managed to grasp the cup and to place it under the coffee machine.

V. Performance

For the comparison of computation times between the LGP and the Standard GP we used a real data set from the *coffee task* (experiments section) and resampled it to different sizes. Fig. 6a, 6b shows the comparison between the two approaches for the encoding/training phase, whereas Fig. 6c, 6d illustrates the prediction time. The local Gaussian process clearly outperforms the original version of the Gaussian process regression model for data set sizes bigger than 100 in the training phase and 250 for predictions. The slightly worse performance at the beginning is caused by the additional overhead for clustering and and the Gaussian product calculations in training and prediction, respectively.



(a) Reproduction in task space (b) Reproduction in joint space

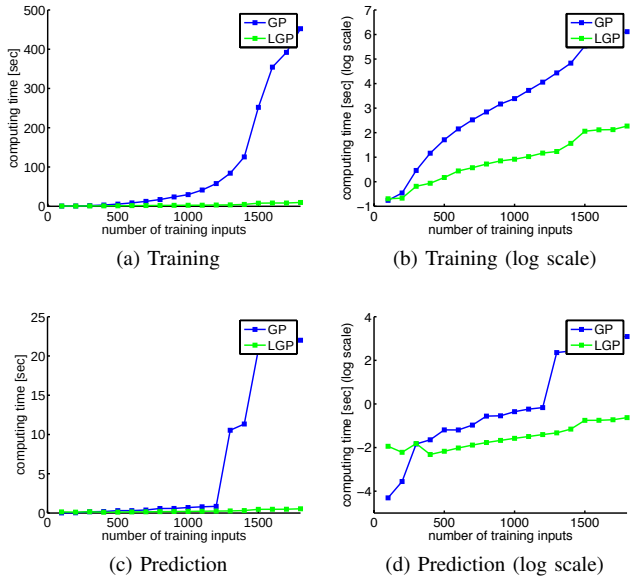
Fig. 5: Reproduction attempts for a new constellation of cup and coffee machine positions. The left panel shows the reproduction in task space, where the *dotted blue* line is the policy relative to the cup, the *dashed green* line is the policy relative to the coffee machine and the *solid red* line marks the resulting policy. We can see that the final controller first follows the constraint for grasping the cup (region between 20 and 50) and then follows the constraint for moving to the coffee machine (time steps 60 and 100). It can be also seen that the initial position of the cup is irrelevant to perform the "press-button" task. The reproduction in joint space is drawn in the right panel. The shaded region denotes the remaining variance after the reproduction algorithm.

VI. Conclusions and Future Works

A. Conclusions

In this paper we have presented how *local*, heteroscedastic Gaussian process regression can be used to *efficiently* extract the essential features of a task from a set of demonstrations and to form a reproduction policy. We also showed how this can be incorporated in a Learning from Demonstration framework. We demonstrated the convenience of this method in the application of learning to place a cup on a coffee machine and press the button with only a few kinesthetic demonstrations. These demonstrations had been sufficient for the robot to gather all relevant informations about the task to learn without the need of additional information from the human teacher. Furthermore, the robot was able to reproduce the task on an unknown setup of the environment in a robust way, which is one of the essential properties of a Learning from Demonstration system.

Gaussian processes are one of the most accurate regression methods, but they suffer from very high computational costs. To be specific the computing time to invert the training data covariance matrix $(\boldsymbol{K} + \boldsymbol{R})^{-1}$ in Eq. 5 and Eq. 6 is $\mathcal{O}(n^3)$.



(a) Training (b) Training (log scale)

(c) Prediction (d) Prediction (log scale)

Fig. 6: Comparison between the local and the standard Gaussian process. The upper and lower row show the computing time for training and prediction phase, respectively. In the right column, the computing time is plotted logarithmically against the different sizes of the training data set. The data had been averaged over 50 results.

We significantly reduce this computational cost by the use of smaller local models introducing a new type of local Gaussian process models. The inverse covariance matrix has to be calculated only during the demonstration/training phase which does not require a very fast computation and can be stored for later use.[3] The Gaussian process prediction part can then be performed very quickly, which can be advantageous for example to (online) modify the trajectory during reproduction.

The Gaussian process model perfectly fits into the framework introduced in [2]. The only open "parameters" are the choice of the GP covariance function and its hyper parameters. The first is very intuitive to set and defines the function shape properties. As illustrated in Fig. 4, the choice of the squared exponential covariance function leads to a smooth and continuous trajectory and therefore fits our preference of robot arm trajectories. We also did not have to distinguish between joint space, task space and gripper data because the Gaussian process model, even with this simple covariance function, was able to model all policies. The Gaussian process hyper parameters can be optimized using maximum likelihood if the initial guess is not extremely wrong.

B. Future Works

Future work will consider the use of *Reinforcement Learning* in combination with *Dynamic Movement Primitives* [9], [16] to equip the robot with self improvement capabilities. For example, this can be used in the *press-the-button* step (where the robot sometimes failed to press the button correctly) to adapt the movement. We also propose to combine the encoding at trajectory level with a symbolic approach. For example small variance regions (as *grasp-the-cup* and *press-the-button*) could be extracted and labeled. The use of active sensing would be advantageous, because then it is also possible to adapt the motion during reproduction if the positions of the objects are changing.

[3] Actually we store only the result of the covariance matrix Cholesky decomposition, $\boldsymbol{L}$, solving $(\boldsymbol{L}^T)^{-1}(\boldsymbol{L}^{-1}\boldsymbol{y})$

REFERENCES

[1] B. Argall, S. Chernova, M. M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[2] S. Calinon and A. Billard. A probabilistic programming by demonstration framework handling skill constraints in joint space and task space. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, September 2008.

[3] S. Calinon, F. Guenter, and A. Billard. On learning, representing and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man and Cybernetics, Part B. Special issue on robot learning by observation, demonstration and imitation*, 37(2):286–298, 2007.

[4] J. Q. Candela and C. E. Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959, 2005.

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

[6] P. W. Goldberg, C. K. I. Williams, and C. M. Bishop. Regression with input-dependent noise: A gaussian process treatment. In *Neural Information Processing Systems*. The MIT Press, 1997.

[7] D. B. Grimes and R. Chalodhorn. Dynamic imitation in a humanoid robot through nonparametric probabilistic inference. In *Robotics: Science and Systems*. MIT Press, 2006.

[8] D. H. Grollman and O. C. Jenkins. Dogged learning for robots. In *ICRA*, pages 2483–2488. IEEE, 2007.

[9] A. J. Ijspeert, J. Nakanishi, and S. Schaal. Learning attractor landscapes for learning motor primitives. In *Advances in Neural Information Processing Systems 15*, volume 15, pages 1547–1554, 2003.

[10] K. Kersting, C. Plagemann, P. Pfaff, and W. Burgard. Most likely heteroscedastic gaussian process regression. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvalis, Oregon, USA, June 20-24, 2007*, volume 227 of *ACM International Conference Proceeding Series*, pages 393–400, 2007.

[11] D. Nguyen-Tuong and J. Peters. Local gaussian process regression for real-time model-based robot control. In *Intl Conf. on Intelligent Robots and Systems (IROS)*, pages 380–385. IEEE, 2008.

[12] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 1999.

[13] C. Plagemann. *Gaussian Processes for Flexible Robot Learning*. PhD thesis, University of Freiburg, Department of Computer Science, December 2008.

[14] C. E. Rasmussen and Z. Ghahramani. Infinite mixtures of gaussian process experts. In *In Advances in Neural Information Processing Systems 14*, pages 881–888. MIT Press, 2002.

[15] C. E. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[16] S. Schaal. Dynamic movement primitives - a framework for motor control in humans and humanoid robotics. pages 261–280. 2006.

[17] A. P. Shon, K. Grochow, and R. P. N. Rao. Robotic imitation from human motion capture using gaussian processes. In *Humanoid Robots, 2005 5th IEEE-RAS International Conference on*, pages 129–134, 2005.

[18] B. Siciliano and O. Khatib, editors. *Springer Handbook of Robotics*. Springer, Berlin, Heidelberg, 2008.

[19] E. Snelson and Z. Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems 18*, pages 1259–1266, 2006.

[20] E. Snelson and Z. Ghahramani. Local and global sparse gaussian process approximations. In *Artificial Intelligence and Statistics*, 2007.