Abstract—Many strategies for controlling a robot require a detailed model of the robot’s kinematics or dynamics. However, such models can be dependent on unknown context variables. One such context variable is the mass of an object lifted by the robot, since for compliant robots this mass changes the model in a non-linear way. Models for specific contexts can be learned using Gaussian Processes (GPs), but have to be re-learned for each novel context. In this paper, multi-task GPs are used to learn a model that allows generalization between different contexts. Consequently, only a few new data points have to be generated when a new context is encountered. We employ learned task features to efficiently learn the multi-task model. The approach is evaluated on an object-lifting task with a continuum robot arm under several settings, and compared to standard GPs trained for each object separately. The learned model is shown to be usable for control. We show that multi-task GPs using learned feature representations for the tasks outperform standard GPs when few data points are available.

I. INTRODUCTION

Carrying and lifting grasped objects is a common task in household and industrial settings. Many strategies for controlling a robot in such situations require detailed dynamics and kinematics models of the robot [1], [2]. Hence, finding such models for robots is a typical problem in robotics. Classical industrial robots, for example, are designed to have analytically solvable kinematics and dynamics [2].

For more complex robots, e.g., continuum robot arms, an exact analytical model is often not available. Instead, there are model learning techniques that learn robot models from data using regression algorithms [1], [3]. Gaussian Process (GP) regression is an example of such a method [4], [5].

However, there are cases where the model is dependent on certain context variables [6], [7]. If one of the context variables changes, the model also changes. For example, a forward model will not be accurate if tested in environments with high viscosity (e.g., under water) or a different gravitational field (e.g., outer space). If we consider lifting and grasping unknown objects, the mass of the object is an unknown context. For compliant robots, this mass can have a major impact on the robot’s dynamics, as shown in Fig. 1.[1]

For industrial robots, inaccurate models are not a major problem, since they are mostly programmed for one specific task and high control gains can be used to largely sidestep this problem [8]. For robots that should work in the presence of humans, however, high control gains should be avoided since they result in the generation of dangerously high forces [9].

A naive way to address the problem of different contexts is to maintain a separate model for every context. However, in this case for each novel context the robot has to learn a model from scratch. Selecting the right model out of a library of learned models is, moreover, an open problem in itself. For robots lifting objects as part of household tasks, collecting data for every novel object to learn a new GP forward model from scratch is not practical. If the analytical form of the dynamics are known, the inertial parameters of the object can be inferred from data [7]. However, for continuum robot arms such as the one shown in Fig. 1, the analytical form of the dynamics is in general not known.

A more general approach is to regard prediction in different contexts as different tasks and generalize between those task using multi-task GPs [10]–[13]. In comparison to conventional Gaussian Processes, predictions consider not only the data of the current task, but also the data of related tasks weighted by the corresponding task-similarity [11]. Thus, experience from a limited number of tasks can be leveraged to enhance performance on tasks for which little data is available.

We aim to learn a non-linear model that generalizes between tasks. We propose addressing this problem using multi-task GPs. This generality of this model makes it applicable to a wide range of systems. In this paper, we will apply this method the specific example of lifting novel objects using a continuum robot arm, using as few data points from each novel task as possible. In contrast to previous work, we will use learned task features for fast inference and to generalize efficiently to novel tasks.
II. RELATED WORK

There is a very large body of work on modeling robot dynamics and kinematics. To keep our discussion concise, here, we will focus on two aspects of related work. First, we will discuss different approaches for obtaining models for the robot we are interested in, the Bionic Handling Assistant (BHA). Then, we will present an overview of multi-task GP models in dynamics modeling and other tasks. A broader discussion of learning robot models is presented in [1].

A. Modeling the Bionic Handling Assistant

The forward kinematics for the BHA can be modeled by using geometric approaches [14], [15]. Even if the robot lifts an object this model stays valid: for any robot state $q$, the position of the end-effector is always the same. The same is not true for the dynamics: with a heavier object, a larger pressure is needed to reach the same robot state.

Such dynamics can be learned for a specific object. Bischoff et al. [5] learned a forward dynamics GP model for a robot equipped with the BHA in an object pick-up task. In that paper, the lifted object was always the same, so that a single-task model sufficed. Rolf and Steil [3] developed a controller based on a learned model for an extended version of the BHA without holding an object.

However, picking up various objects is a common task for robots such as the BHA. Whenever an object is picked up, the learned dynamics change. In such cases, these learning approaches would have to re-learn the model. This requires gathering a new training set, which is time consuming.

B. Gaussian Processes for Multi-task learning

Once models have been learned for specific contexts, these could be leveraged to make better predictions for new contexts for which only a few data points are available. Learning models for different contexts can be considered a multi-task or transfer learning problem [16], [17]. Such a generalizable model can be learned by using multi-task Gaussian Processes [11]. Multi-task GPs do not just use the data gathered for that particular task to make predictions, but also the data of related tasks weighted by the task-similarity [10]–[13].

These task-similarities are stored in a task-similarity matrix. This matrix can be learned using different approaches. In some previous work [10], [11], the values of the task-similarity matrix are set directly. To optimize the matrix an EM-Algorithm is used. The drawback of this approach is that the matrix must be positive definite, since it represents a covariance matrix. Furthermore, there are many parameters that have to be optimized. In contrast, feature representations of the tasks can be defined [12], [13], [18]. Hence, task similarities can be calculated using a kernel function on these features, instead of searching directly for a similarity score for each task pair. These features can, for example, be defined manually for each task [12]. If learned, feature-based similarity scores reduce the number of parameters to be linear in the number of tasks rather than quadratic, as optimizing for each pairwise similarity would be.

Chai et al. [10] applied multi-task GPs to learn an inverse dynamics model for a rigid 6 DoF robot. Their method can be used to quickly adapt to objects of different mass lifted by the robot, but is limited to known rigid robots where only the inertia parameters of the last link (including the held object) are unknown. A different model for multi-task dynamics learning with Gaussian Processes is described by Zubizarreta-Rodriguez and Ramos [19]. That paper considered the prediction of various output variables to be different tasks and did not focus on the case of different task contexts.

In contrast to the work by Chai et al. [10], we use a compliant robot where we do not know the analytical form of the model. As an alternative to manually specifying task features [12], which reduces autonomy, or learning the similarity matrix directly [10], [11], which requires learning a quadratically increasing number of parameters, we will consider learning the task features. Furthermore, we investigate how to update the model more efficiently, as it should be updated on-line to improve control during a robot lifting task.

III. MULTI-TASK GAUSSIAN PROCESSES

Gaussian Processes (GP) are the method of choice for regression problems with small to intermediate datasets [20], as for large datasets computation costs become prohibitive. GPs specify a prior distribution over the function values at observed training points. Conditioning on training data allows a posterior mean and variance to be calculated at new points.

The prior covariance between any two input values $x_p, x_q$ is given by kernel function $k(x_p, x_q)$. We will use the squared exponential kernel

$$k(x_p, x_q) = \sigma_f^2 \exp \left(-\frac{||x_p - x_q||^2}{2l^2}\right),$$  \hspace{1cm} (1)

where $\sigma_f$ denotes the scale of the output values of the learned function, and $l$ is the length scale, which influences the range in which data points are considered to be similar. Both $\sigma_f$ and $l$ are hyper-parameters of the kernel function.

Multi-task Gaussian Processes are a generalization of GPs. In multi-task GPs, the aim is to learn several related functions called tasks. Predictions for tasks with few data can then be improved by also considering the weighted training points of related tasks.

We consider a set of $M$ tasks $T = \{f_1, f_2, \ldots, f_M\}$. To learn those tasks, we use a dataset consisting of a matrix $X = [x_1, x_2, \ldots, x_N]^T$ of $N$ $D$-dimensional training inputs with $x_i \in \mathbb{R}^D$ with corresponding observed outputs $y = (y_1, y_2, \ldots, y_N)^T$ and a list of corresponding task indices $t = (t_1, t_2, \ldots, t_N)^T$ with $t_i \in T$. Let $k^t(x_p, x_q)$ be the covariance function that returns the similarity of the data points $x_p$ and $x_q$ and $K^t$ be the data kernel matrix such that each element $K^t_{pq}$ is calculated by $k^t(x_p, x_q)$. Let $k^t(f_p, f_q)$ be the covariance function that returns the similarity of task $f_p$ and $f_q$ and thus $K^t$ be the task-similarity matrix where each entry is calculated by $k^t(t_p, t_q)$ analogous to $K^t$. How function $k^t$ is defined will be discussed in [II-A] and [II-B].
Further, let $K^\text{total} = K^f \circ K^s$ be the kernel matrix where each entry of the data kernel matrix is multiplied by the corresponding entry of the task-similarity matrix. The symbol $\circ$ denotes the Hadamard product (element-wise multiplication).

Applying the task-similarity to the mean and the covariance predictions yields the following equations for predictions at a matrix of $L \times D$-dimensional test inputs $X_* = [x_{*1}, x_{*2}, \ldots, x_{*L}]^T$ belonging to task $f_i$

$$
\bar{f}_i(x_*) = K^i_{\text{total}} \left[ K^\text{total} + \sigma_n I \right]^{-1} y,
$$

$$
cov(f_i) = K^i - K^i_{\text{total}} \left[ K^\text{total} + \sigma_n I \right]^{-1} K^i_{\text{total}}^T,
$$

where $K^s_i$ denotes the data kernel matrix of the test data and $K^i_{\text{total}} = K^i(f_i, t) \circ K^s(X_*, X)$ the total kernel matrix of the test inputs and the training inputs [11]. Here, $\sigma_n$ denotes another hyper-parameter, which corresponds to additive noise in the dataset and can be seen as a regularization factor.

These equations for the mean and covariance of the multi-task GPs are similar to the equations of the single GP formulation. The only difference is the kernel matrix, which not only depends on the similarity of the data but also on the similarity of the corresponding tasks.

Making good estimates using (multi-task) GPs depends on how well the hyper-parameters are adjusted to the dataset. For a single-task GP employing the squared-exponential kernel [11] there are three hyper-parameters: $l$, $\sigma_f$ and the noise $\sigma_n$. These hyper-parameters are optimized by maximizing the marginal log-likelihood of the training data $X$ using $k$-fold cross-validation as suggested in [20].

In multi-task GPs there are additional parameters that have to be learned for the task kernel function $k^f$. Which parameters these are depends on how function $k^f$ is defined and, therefore, how the entries of the task-similarity matrix $K^f$ are computed. Two ways to define this function are discussed in the following subsections.

A. Task-Similarity Approach

The first approach to set the task-similarity matrix $K^f$ is simply setting the values of the task-similarities directly. We first define a separate matrix $G \in \mathbb{R}^{M \times M}$ which is a symmetric matrix containing the similarities between each pair of tasks. The function $k^f$ is then simply defined as $k^f(f_p, f_q) = G_{pq}$. The additional parameters are in this case the entries of $G$. Since the matrix $G$ is symmetric, the number of free parameters is $(M(M-1))/2$, where $M$ is the number of tasks. The number of free parameters is therefore quadratic in the number of tasks.

This approach is used in [11], where the problem structure makes calculation of the task-similarity matrix easier. For example, all tasks have training data at the same input locations $X$, which does not hold in our setting as we generalize from known tasks with many data points to novel tasks where we might have only a few data points.

When optimizing the parameters the resulting matrix $G$ must be positive semi-definite (PSD), since $G$ is a covariance matrix. In every iteration of the optimization algorithm $G$ is checked for positive-definiteness. If it is not PSD, it is projected to the nearest PSD matrix using the method in [21] (due to the special problem structure, this procedure was not necessary in [11]).

B. Feature Based Approach

Instead of directly optimizing the task-similarities, each task can be represented by so-called task features. The task-similarity is then simply calculated by evaluating a kernel function on the task features [12], [13]. That is, a positive definite task kernel function $k^f$, like the squared exponential kernel [11], defines the covariance between two tasks based on their respective feature representations. The task covariance $K^f_{pq}$ is therefore calculated by evaluating the kernel function $k^f_{pq} = k^f(\phi(f_p), \phi(f_q))$ on the task features $\phi(f_p), \phi(f_q)$.

Since the true features (masses) of the objects are not known, we optimize the features $\phi(f_i)$ of each of the tasks $i \in 1, \ldots, M$ directly. These are specified by $MD_\phi$ parameters, where $M$ is the number of tasks and $D_\phi$ the dimensionality of the feature space. In addition, there is a constant number of hyper-parameters for the task kernel function, specifying e.g. its length-scale. Therefore, in this case the number of parameters is linear in the number of tasks, as opposed to quadratic in case we optimize the entries of the task similarity matrix directly. This procedure is similar to the optimization of latent values in GP-LVMs [22], with additional structure provided by knowledge about the task index for each data point.

In contrast to direct optimization of $G$, the resulting task-similarity matrix $K^f$ is guaranteed to be positive semidefinite, since its entries are calculated by a positive definite kernel function. Therefore, we do not need to perform the costly projection to the closest positive semi-definite matrix as was necessary for the direct optimization.

When optimizing the features in a one-dimensional space, the optimization is likely to get stuck in local optima. For example, if a light object would initially be assigned a feature value in between two heavy objects, moving the light object to either side would worsen the value of the objective function. Instead, we chose to optimize features in a two dimensional space. Even though this doubles the amount of free parameters, the additional freedom for the optimization algorithm makes the optimization less likely to get stuck in local optima.

Figure 2 shows the results of a preliminary experiment to compare the approaches introduced in III-A and III-B. The approaches are evaluated on a one dimensional data set. We run six trials, with each task being used once as the ‘new’ task and the others as known task, added in a randomized order that is the same for the different methods. Figure 2a shows the negative log-likelihood of data points of the new task using data from different numbers of known tasks. It compares the task-similarity and feature-based approach to a naive baseline fitting a single GP to the data points of all tasks thus far. The two generalizing approaches perform roughly similar and yield much better results than both using 0 previous tasks and the naive baseline. To avoid the computational cost of projecting to a positive-definite matrix,
The proposed method is evaluated on the Robotino XT continuum robot arm. It is a pneumatically actuated robot, that has a trunk-shaped manipulator, called the Bionic Handling Assistant (BHA), on a drivable platform. The BHA, shown in Fig. 1 has six chambers that can individually be filled with compressed air to set different pressures up to 1.5 bar. The control action $u$ specifies the desired pressures. The BHA has a gripper with which it can grab different loads up to 600 grams and six cable potentiometers to measure the actuator lengths of the BHA. The length of the cable potentiometers defines the robot state $q$.

Forward and inverse dynamics are normally specified as relations between the state of the robot denoted by $q, \dot{q}$, the desired acceleration $\ddot{q}^d$, and the applied control input $u$

$$q, \dot{q}, u \rightarrow \ddot{q}^d, \quad \dot{q}, \ddot{q}, \dddot{q} \rightarrow u.$$ 

However, in case of the pneumatically driven BHA, the dynamics can be simplified if we choose time steps long enough for the arm to converge for the current pressure. The same control input applied to the robot then always leads to the same state, independent of the robots current state. Thus, the equations for the forward and inverse dynamics simplify to $u \rightarrow q$, and $q \rightarrow u$, respectively. We will use this simplified formulation for learning the robot’s forward model.

B. Dataset

To gather a dataset, a sequence of control inputs is applied to the robot and the resulting potentiometer values (encoding the extension of the different arm compartments) are stored. We only varied the values in the lower 3 actuators to keep the amount of data manageable. The upper 3 actuators are always set to a pressure of 0 bar. Similarly, we only considered the actuator lengths of the 3 potentiometer along the whole length of the arm (rather than those covering just the first segment), since they are related more strongly to the Cartesian end-effector position.

As training data, we gather data using 6 different masses at the end effector. For each of these ‘known tasks’, a set of 216 control inputs is applied to the robot: the pressure in the lower three compartments is set to all possible combinations of pressures between 0 and 1.5 bar with a step size of 0.3. The robot lifts a cup filled to different degrees, to obtain object masses of 0 grams (no object), 78 grams (the empty cup), 200, 300, 430 and 600 grams.

For testing, a set of randomly generated control inputs is applied to the robot and the corresponding lengths of the cable potentiometers are recorded. For each tested object the same set of random inputs is used to be able to compare the resulting performance in a fair way. The test objects and their corresponding masses are shown in Fig. 3.

C. Learning Models for Control

We want to evaluate how well the robot can reach desired configurations for novel objects. That is, given a specified robot state $q$ (the lengths of the cable potentiometers), we want to obtain the pressures $u$ to set in the air chambers to reach that state. Since multiple commands can reach the same robot state, the inverse dynamics model is not a function and cannot be learned using ordinary regression methods.

Therefore, instead of learning an inverse dynamics model, we find a control action $u$ to reach the desired state $q^d$ by minimizing the distance between the desired configuration and the configuration predicted by the learned forward dynamics model. The result is a version of model predictive control with a control horizon of just one action. Although the inverse dynamics model is not a function, regression on the inverse data provides a rough first guess, which we used to initialize the optimization. This procedure leads to better results than random guesses and helps avoiding local optima.

D. Model Learning

To learn the model the whole data set of the known tasks is loaded and a multi-task GP is trained for this dataset. The model is learned by optimizing feature representations of the tasks and the hyper-parameter of the data kernel function, as mentioned in Sec. III To optimize these parameters the
negative log-likelihood of the data is minimized by using a 4-fold cross-validation objective. The resulting learned features for each of the training tasks encode the similarities between those tasks (tasks consisting of lifting similar weights are assigned similar features during the optimization). To avoid possible local optima, the optimization is repeated 10 times with different initializations of the task-features. The initial task features were chosen randomly from a normal distribution with a variance of 0.2. The optimized parameters that reached the highest likelihood value were subsequently fixed for the rest of the experiment.

For the test object (the new task), initially only 2 data points are given to the algorithm. We evaluate how data-efficient the model is by then adding data points to the model. After adding data points, the feature representation of the test task is optimized again, while all the other parameters stay fixed to allow faster training as explained in the next paragraph. These other parameters can stay fixed because they were trained on a relatively large data set, and therefore should hardly change when adding an additional task with relatively little data to the model. Again, this optimization is done by minimizing the negative log-likelihood with 4-fold cross-validation. These folds are only from the data of the new task, such that the training set contains all the data from the known tasks and 3 folds of the new task and the test set contains only data from the new task.

To save computation time, the inverse of the covariance matrix of the known tasks is only calculated once. During optimization this inverse can then be updated in an efficient manner by using a block-wise inversion

\[
\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = \begin{pmatrix} M \\ -D^{-1}CM - D^{-1}CMBD^{-1} \end{pmatrix},
\]

with \(M = (A - BD^{-1}C)^{-1}\).

In this equation, \(D\) is a matrix containing the data points from the training object that stay fixed during learning, such that \(D^{-1}\) can be pre-calculated. Only the matrix \(M\), which is a squared matrix of a size corresponding to the newly added data, needs to be inverted for every new learning step. Consequently, a much smaller matrix has to be inverted in each optimization step. This procedure speeds up the optimization approximately by a factor of 2 in this setting.

**E. Baseline and Evaluation Criterion**

To measure the quality of the learned model a set of 10 validation points is picked out of the gathered set of data points of the new task. The estimated control inputs of these validation points are applied to the robot and the resulting states are recorded. An error measure the Euclidean norm between the desired and recorded length of cable potentiometers is calculated.

We compare to training an individual GP for each new tasks as a baseline. In this case, no features or parameters have to be optimized for the task kernel, as only the data kernel is used. Otherwise, the same procedure is used as in the multi-task case.

### Table: Mean configuration error for different numbers of training data points

<table>
<thead>
<tr>
<th># training data points new task</th>
<th>single GP</th>
<th>multi-task</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.25</td>
<td>0.2</td>
</tr>
<tr>
<td>7</td>
<td>0.2</td>
<td>0.15</td>
</tr>
<tr>
<td>12</td>
<td>0.16</td>
<td>0.12</td>
</tr>
<tr>
<td>17</td>
<td>0.15</td>
<td>0.1</td>
</tr>
<tr>
<td>22</td>
<td>0.09</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Fig. 4: Squared Euclidean distances of the state the robot ended up using the estimated control input and the desired state. The solid line is the average error of 10 runs using different training data for the new task. The error bars show the standard deviation over the 10 runs. The error of each run is the average error of 10 validation points. The images at each plot show the corresponding test object used as the new task.

### V. Results

Figure 4 shows the average errors and the standard deviation for each test object over 10 runs. The error of each run is the average distance between the desired and actual joint state for 10 validation points. In each run the model is learned using different training data for the new task.

The error varies in each run, since the estimates for the validation points made by the GPs is dependent on the available training data. The error of the multi-task GP decreases much slower than the error of the single GP, since for the multi-task GP there is already a lot of information from other tasks. Therefore, adding a few data points influences the estimates less than in the single GP case.

The performance of the single GP decreases with increasing weight of the object. The decreased performance could be caused by the fact that for low pressures there is almost no movement of the robots trunk, since the weight of the objects pulls it to the ground. So the effect of applying low pressures to the robot is small. Again, the multi-task GP already has information about heavy objects and so the performance is stable over the different weights of the objects.

For the first two objects, the single GP outperforms the multi-task GP for seven or more data points in our experiment setup. This performance difference could be caused by hysteresis in the robot arm during recording of the validation points. It seems that the multi-task GP approach is more sensitive to hysteresis effects than the single GP.

Nevertheless, the results show that multi-task GPs yield good results for control even when just a very small amount of data points is used for training on a new task. Adding more data points to the training set has no major effect to the quality of the multi-task GP. The results also show that for increasing amounts of data points the benefit
of using multi-task GPs instead of single GPs decreases.

Figure 5 visualizes the advantage of the multi-task GP for a
example test point. The transparent state of the robot in
this figure is the desired state. This example shows the action
selected by the controller based on the multi-task GP already
reaches the desired state quite well for few data points. For a
high amount of training points the performance of the multi-
task GP and the single GP is similar.

VI. CONCLUSION AND FUTURE WORK

In this paper, multi-task Gaussian Processes are proposed
to tackle the problem of learning models that can generalize
between different contexts. Different approaches to model the
task-similarities are investigated, one approach optimizes the
similarities directly whereas the other approach learns task-
features and computes the similarities of the tasks in feature-
space. In the first approach, the number of parameters grows
faster and the resulting matrix has to be projected to the
nearest positive semi-definite matrix. The second approach
computes the task-similarity matrix using a positive definite
kernel function, which avoids this problem. As furthermore,
the second approach has fewer parameters to optimize, this
approach is concluded to be better suited to our task.

We evaluate this method on the specific problem of
learning a forward model for a compliant robot lifting objects
of different weights. We show that multi-task GPs can be
used to learn a forward model for robot control, by using a
model predictive control approach, considering sequences of
a single action. For a very small number of available training
data points for an unknown task, multi-task GPs outperform
single GPs. With increasing amount of available data points
for the unknown task the advantage of using multi-task
GPs compared to single GPs decreases. Eventually, for big
datasets single task GPs perform about equally well as multi-
task GPs. A possible disadvantage of any kind of GP model
is that these models are very powerful at interpolating, but
tend to not extrapolate very well. Therefore, it is beneficial
if the known tasks include extreme cases.

With more than seven data points, single-task GPs did
surprisingly well and sometimes outperformed multi-task
GPs. This could be explained by the small workspace of
the robot, which made learning a new model relatively easy,
or by hysteresis effects, which might make generalization
harder. For future work, applying this method to other
platforms with greater work spaces and less hysteresis would
be desirable. Furthermore, the model could be improved by
taking dynamic properties, such as velocity or acceleration,
to account. To avoid potential negative transfer, a Bayesian
model selection scheme could be used to switch to a single
GP whenever sufficient data becomes available for the current
task.

REFERENCES

tics on a bionic elephant trunk,” Trans. Neural Networks and Learning
local Gaussian process regression.” Advanced Robotics, vol. 23, no. 15,
2009.
mussen, A. Knoll, J. Peters, and M. Deisenroth, “Policy search for
learning robot control using sparse data,” in ICRA, 2014.
with applications to robotics,” Trans. Systems, Man and Cybernetics,
task Gaussian process learning of robot inverse dynamics.” in NIPS,
of multisection bionic manipulator: Application to RobotinoXT,” in
compliant pneumatically actuated redundant manipulator,” in ICIEA,
[16] S. Thrun, “Is learning the n-th thing any easier than learning the first?”
system dynamics with maximum information gain,” in ICRA, 2011.
[22] N. D. Lawrence, “Gaussian process latent variable models for visual-
isation of high dimensional data,” NIPS, vol. 16, no. 3, pp. 329–336,
2004.