Generation and Exploitation of Local Models for Rapid Learning of a Pouring Task

Joshua D. Langsfeld[1] and Krishnanand N. Kaipa[1] and Satyandra K. Gupta[1]

***Abstract*— We present an approach that allows a robot to learn to perform a complex task instance in as few attempts as possible. Specifically, our approach allows a robot to bootstrap from initial exploratory experiments and identify task parameters to successfully perform a given task instance. This is achieved by first generating local models in the vicinity of previously conducted experiments that explain both task function behavior and estimated divergence of the generated model from the true model when moving within the neighborhood of each experiment. Later, these models are used to guide parameter selection given a desired task outcome and the models are updated based on the actual outcome. The local models are built within adaptively chosen neighborhoods, thereby allowing the algorithm to capture arbitrarily complex function landscapes. We validate our approach by testing it on both synthetic nonlinear functions and a physical robot performing a dynamic fluid pouring task. We show initial results comparing our approach with a simple greedy algorithm using Gaussian Process Regression. Real robot results reveal that the correct pouring parameters for a new pour volume can be learned quite rapidly, with sparse exploratory experiments.**

I. Introduction

Programming robots to carry out real-world tasks is challenging and time consuming. Manual construction of analytical models that accurately capture the dynamics of complex tasks (e.g., manipulation of deformable materials and/or fluids) is often not feasible. Additionally, compliant joints typically found in new robots such as Baxter make it difficult to specify the task based on purely geometric descriptions. Such scenarios can often be more naturally handled by having the robot build an internal model of the task, which is improved as more data is acquired. This theme is common in all robot learning, including imitation [1], reinforcement, and active [20] learning.

A central issue with learning algorithms is the exploration/exploitation trade-off. In this paper, we present an approach that is biased toward exploitation in order to allow a robot to learn to perform a complex task instance in as few attempts as possible. Specifically, our approach allows a robot to bootstrap from partial information that is available either from human demonstrations or robot's initial exploratory experiments (also called *motor babbling* [19]), and thereafter, identify task parameters to successfully perform a given task instance by generating local models in the vicinity of the previously conducted experiments. The algorithm selects a new set of task parameters by exploiting the local model information. We show that our approach allows the robot to conduct sparse experiments and adapt to task variations in few attempts.

We model the task abstractly as a set of parameters existing in a finite-dimensional space where each point in the space defines a policy to perform a single task variation. These parameters are arbitrary and might correspond to trajectory features that are input to a planner or possibly to the shaping parameters of a dynamic motion primitive [6].

Our approach has the goal of rapidly finding valid solution points in the parameter space corresponding to the problem of new task variations with sparse initial data. This is achieved by first generating local models in the vicinity of the previously conducted experiments that explain both the task function behavior and the estimated divergence of the generated model from the true model when moving within the neighborhood of each experiment. Later, these models are used to guide parameter selection given a desired task outcome and the models are updated based on the actual outcome. The local models are built within adaptively chosen neighborhoods, thereby allowing the algorithm to capture arbitrarily complex function landscapes.

Our approach mainly takes a two-fold inspiration from LWPR: (1) Similar to LWPR, we use linear regression models to capture the target function behavior. LWPR also projects the original input space into a relatively much lower dimensional space that has critical influence on the output (dependent variable). However, in this paper, we restrict our approach to maintaining full dimensionality during model generation. (2) LWPR incorporates a local feature by using kernel functions that explicitly emphasize the influence of samples in the vicinity of, while discounting the influence of samples far away from, a test point in the input space. We incorporate this by constructing a local model at each sample using other samples that reside only within a small adaptive neighborhood and using a divergence-model that provides estimates of the approximation error as a function of distance from the sample point. Note that while these two features are somewhat inspired from LWPR, we significantly differ in how they are realized in our approach.

We validate the approach by testing it both on synthetic nonlinear functions and on a physical robot. For physical experiments, we consider a dynamic pouring task, in which the robot is tasked to pour a certain volume of liquid from a bottle into a container placed on a rotating turntable. Moving the bottle to track the container while simultaneously pouring from it creates complex fluid dynamics. This makes the mapping from pouring-trajectory parameters to poured volume highly nonlinear and infeasible to model analytically.

[1]Department of Mechanical Engineering, University of Maryland, MD USA [jdlangs|kkrishna|skgupta]@umd.edu

Our results reveal that the correct pouring parameters for a new pour volume can be learned quite rapidly.

II. Related Work

Our problem, formulated abstractly, shares aspects with nonlinear function approximation from policy parameters to task description (score), and is commonly addressed in machine learning. In the robotics community, the two most-commonly used algorithms for approximating functions are Gaussian Process Regression (GPR) [16] and Locally Weighted Projection Regression (LWPR) [24]. GPR is a standard approach for estimating a nonlinear function from sparse samples and has been used in many motion learning tasks [14]. Whereas this approach requires the function to be learned globally in a batch process, we focus on developing an approach based on local models. We do, however, show initial results comparing our approach with a simple algorithm using GPR model. LWPR provides an efficient method of incrementally learning nonlinear functions through the use of linear models that have local influence only, similar to our approach. LWPR has the advantage of being nonparametric and able to generate new local models as needed. However, a gradient descent is required for each local model, which substantially increases the number of samples needed when searching for a target point.

Another related field, active learning [20], has been applied to a variety of problems in engineering and robotics where mathematical models are difficult to obtain. Some examples include nonlinear system identification [2], inferring robot's own morphology [3] and that of other robots in the environment [7], learning the inverse kinematics of highly redundant bionic arms [17], learning environment's degrees of freedom [15], and robot grasping [9].

The underlying theme of active learning approaches is to allow algorithms to rapidly learn internal models of target systems by formulating methods that pose informative queries in the parameter space. Some traditional query formulating strategies include query-by-committee [22], expected error reduction [18], and expected model change [21]. While we take inspiration from these approaches in terms of using existing information to guide parameter selection for a given task instance and updating of local models, we differ in that we exploit local information and global exploration is only a secondary effect.

In reinforcement learning, Kakade et. al. [8] show an approach similar to ours where the behavior of a continous state-action pair space is assumed to be approximated by local models surrounding a finite set of samples. They prove bounds on the optimality of such a model but the problem of defining the ideal finite covering set remains specific to the application. Other work includes a hybrid approach [12] where an agent undergoing reinforcement learning can request relocation to a different region of the state space and can actively select the most promising state to reduce its task cost. In a sense, our approach includes this feature, but we expand it by continually evaluating the benefit of relocation to any previous point for each task trial. Our approach also has similarities with Bayesian policy optimization [11], [10], which uses function approximation for mapping parameters to performance in a reinforcement learning setting with probabilistic interpretations. Other relevant work includes demonstration-guided motion planning [4], in which a sampling-based motion planner uses the learned cost metric to compute plans that avoid obstacles and satisfy task constraints.

Finally, our approach parallels the work in previous work in robot pouring [13], where a stationary pouring task is learned using reinforcement learning techniques with error approximation models and local movements in the parameter space. Our main conceptual difference is to allow for movements from any previously evaluated point in the space. Other related work involving robot pouring includes the work of Tamosiunaite et. al. [23] and Brandi et. al. [5], which we primarily distinguish ourselves from by addressing the problem of pouring specific volumes of liquid with high tolerances rather than the entire bottle.

III. Approach

The primary guiding principle of the proposed approach is to iteratively explore points in the task parameter space by creating local models of parameteric neighborhood and finding a movement within a variable-size neighborhood that has the minimum uncertainty. First, in an initialization phase, a set of training samples is obtained from robot executions of randomly selected points in the parameter space. Each sample is a tuple comprising a parameter vector and its mapping to a task score. Second, in a *model generation* phase, the algorithm builds local models based on the current training set. Third, in a *model exploration* phase, the algorithm finds a new point by using the current set of local models, evaluates its task score, and adds the newly found point to the training set. The cycle of model generation and model exploration repeats until a point is found whose task score is within the desired success tolerance.

A. Initialization

Let $\mathcal{S} = \{(x^{(i)}, y^{(i)}) : x^{(i)} \in \mathcal{X}, y^{(i)} \in \mathcal{Y}, i = 1, 2, \ldots, m\}$ be the initial set of training samples, where $\mathcal{X}$ is the set of sample points in the parameter space, $x^{(i)} \in \mathbb{R}^n$ is the i^{th} point in $\mathcal{X}$, $\mathcal{Y}$ is the set of corresponding task scores, $y^{(i)} \in \mathbb{R}$ is the i^{th} task score in $\mathcal{Y}$, and m is the number of training samples. Let $x_j^{(i)} \in \mathbb{R}$ represent the j^{th} element of $x^{(i)}$. Further, let $x \in \mathbb{R}^n$ represent a general point in the parameter space.

B. Model Generation

This phase is achieved in three steps: (1) adaptive neighborhood selection, (2) planar model approximation, and (3) error-divergence model approximation.

1) Adaptive neighborhood selection: For each point $x^{(i)} \in \mathcal{X}$, the algorithm builds a local linear model by using points $x^{(j)} \in \mathcal{X}$ residing within an adaptive box-neighborhood $\mathcal{N}^{(i)} \subset \mathcal{X}$:

$$\mathcal{N}^{(i)}(k) = \{x^{(j)} \in \mathcal{X} : ||x^{(i)} - x^{(j)}||_\infty \leq \delta_k^{(i)}\} \quad (1)$$

where k is the number of neighbors in the neighborhood and $\delta_k^{(i)}$ is the neighborhood size, which is given by

$$\delta_k^{(i)} = \max_{x^{(j)} \in \mathcal{N}^{(i)}(k)} ||x^{(i)} - x^{(j)}||_\infty \text{ s.t. } |\mathcal{N}^{(i)}(k)| = k. \quad (2)$$

According to (1) and (2), note that $\delta_k^{(i)}$ is assigned to the minimum possible size that results in $k-$nearest neighbors. Since the density of points in the data set can be highly variable, it is important to find a $\delta_k^{(i)}$ that results in a neighborhood with sufficient points to generate a relatively accurate local approximation, but not so many that the nonlinear behavior of the underlying function deteriorates the approximation. This is done by using a leave-one-out cross-validation technique to estimate the optimal neighborhood size $\delta_{k^*}^{(i)}$. In particular, for each $x^{(j)} \in \mathcal{N}^{(i)}(k)$, a plane is fit using least-squares on the set $\mathcal{N}^{(i)}(k)/x^{(j)}$ and the linear-fit error at $x^{(j)}$ is computed. Now, the mean of linear-fit errors η_k over all $x^{(j)} \in \mathcal{N}^{(i)}(k)$ is used as a fitness to evaluate the neighborhood size.

We consider $k = n + 2$ as the least number of desired points in $\mathcal{N}^{(i)}(k)$ since $n + 1$ points are needed for a unique plane fit, plus an additional point for cross-validation error measurement. Accordingly, the neighborhood size is initialized to $\delta_{n+2}^{(i)}$ and the corresponding η_k is computed. Next, k is incremented by one and η_{k+1} is computed. If $\eta_k < \eta_{k+1}$, then $\delta_k^{(i)}$ is reported as optimal. Otherwise, the search continues to find a better neighborhood-size.

In general, if sample density around $x^{(i)}$ is moderate, the successive error values will decrease as the plane fits are less sensitive to noise induced from few samples, but then increase again as the neighborhood begins to include the nonlinearity of the sampled function. Finding the neighborhood-size when the error first increases is used as a rough heuristic to find the balance between these two factors while also not evaluating more neighborhood sizes than necessary.

2) Planar model approximation: An affine hyperplane $\mathcal{F}^{(i)}(x)$ is fit to points in the neighborhood $\mathcal{N}^{(i)}(k^*)$:

$$\mathcal{F}^{(i)}(x) = A^{(i)}(x - x^{(i)}) + y^{(i)} \quad (3)$$

where $A^{(i)} = \begin{bmatrix} a_1^{(i)} & a_2^{(i)} & \cdots & a_n^{(i)} \end{bmatrix}$ is a row vector of planar model coefficients. If the plane is not uniquely determined (e.g., all the points are collinear), then the neighborhood size is incrementally expanded until a set of points is found that uniquely determines the plane.

3) Divergence-model approximation: The plane obtained using (3) is assumed to be an approximation of the tangent plane of the true task function in the vicinity of the point in question. This approximation is expected to diverge substantially as we move away from the fitted neighborhood. Therefore, we then estimate how quickly this divergence occurs by computing the absolute error $e^{(j)}$ between the predicted task score (using plane approximation at $x^{(i)}$) and the actual measured score for every point $x^{(j)}$ in an annular-box-neighborhood $\mathcal{M}^{(i)}$:

$$\mathcal{M}^{(i)}(\beta) = \{x^{(j)} : x^{(j)} \in \mathcal{X} / \mathcal{N}^{(i)} \wedge ||x^{(i)} - x^{(j)}||_\infty \leq \beta\} \quad (4)$$

where $\beta > \delta$ is the size of the neighborhood:

$$e^{(j)} = |y^{(j)} - \mathcal{F}^{(i)}(x^{(j)})| \ \forall \ x^{(j)} \in \mathcal{M}^{(i)} \quad (5)$$

where $\mathcal{F}^{(i)}$ is the affine hyperplane corresponding to $x^{(i)}$.

For each $x^{(i)}$, we then construct an error bound function $\mathcal{E}^{(i)}(\Delta x)$ that acts as an upper bound on these absolute error values. Our formulation uses a quadratic function with different weights $\omega_j^{(i)}$ for each parameter axis and whose minimum lies at $x^{(i)}$ where the plane is fit:

$$\mathcal{E}^{(i)}(\Delta x) = \sum_{j=1}^{n} \omega_j^{(i)} \Delta x_j^2 \quad (6)$$

where $\Delta x = x - x^{(i)}$. The weights $\omega_j^{(i)}$ are found by solving the following optimization problem:

$$\text{Minimize} \quad \sum_{j=1}^{n} \left(\omega_j^{(i)}\right)^2, \quad (7)$$

$$\text{subject to} \quad \mathcal{E}^{(i)}(\Delta x^{(j)}) \geq e_j \forall x^{(j)} \in \mathcal{M}^{(i)}. \quad (8)$$

At the end of the model generation phase, we have $\mathcal{N} = \{\mathcal{N}^{(i)} : i = 1, 2, \ldots, m\}$, $\mathcal{F} = \{\mathcal{F}^{(i)} : i = 1, 2, \ldots, m\}$ and $\mathcal{E} = \{\mathcal{E}^{(i)} : i = 1, 2, \ldots, m\}$ representing the sets of m adaptive neighborhoods, m hyperplanes, and m error bound functions, respectively, corresponding to each sample point in $\mathcal{S}$.

C. Model Exploration

Given a desired task score y_d, the sets $\mathcal{S}$, $\mathcal{F}$, and $\mathcal{E}$ are used to find a new point in the parameter space. As we want to explore the parameter space quite conservatively, we would like to query a new point that will provide the smallest uncertainty in task score with respect to a local error bound function. This is performed by selecting an existing sample point in $\mathcal{S}$ as a base point and by selecting only a single parameter for modification at that base point, which minimizes the possibility of error arising from unknown cross-effects between the parameters.

These two selections are made by conducting a search at each base point $x^{(i)}$ in the following way. For each parameter $x_j^{(i)}$, the desired corrective movement $\Delta x_j^{(i)}$ is calculated by finding a point in the direction parallel to that parameter axis whose task score based on the plane approximation is equal to the desired amount y_d. The parameter change is saturated if the corresponding error bound function rises above a given threshold e_{max} before reaching the new point. Accordingly, the parameter change is given by

$$\Delta x_j^{(i)} = \text{sgn}\left(\frac{y_d - y^{(i)}}{a_j^{(i)}}\right) \min\left(\left|\frac{y_d - y^{(i)}}{a_j^{(i)}}\right|, \sqrt{\frac{e_{max}}{\omega_j^{(i)}}}\right) \quad (9)$$

$\forall\ j = 1, 2, \ldots, n.$

The saturation limit on parameter change used in (9) prevents the system from testing points that have the potential for large error, possibly resulting in trials outside the proper operating range which would give no new information.

Note that explorations in the search space are always restricted to individual parameter directions. This results in generation of new sets of points called *line-sets*, where all points in a line-set $\mathcal{L}_j^{(i)}$ lie on a line parallel to single parameter axis j, $j = 1, 2, \ldots, n$:

$$\mathcal{L}_j^{(i)} = \{x^{(k)} \in \mathcal{X} : |x_\ell^{(i)} - x_\ell^{(k)}| \neq 0 \text{ only for } \ell = j\}. \quad (10)$$

Therefore, whenever such a line-set is available for a base point $x^{(i)}$, the algorithm makes use of a line-approximation over the points in the line-set, instead of using the planar approximation in (3), to compute the parameter change at that point. That is, for each parameter j where $|\mathcal{L}_j^{(i)}| \neq 0$, the algorithm computes $b_j^{(i)}$ as the slope of the best fit line through the points in $\{x^{(i)}, \mathcal{L}_j^{(i)}\}$. Accordingly, $b_j^{(i)}$ replaces $a_j^{(i)}$ in (9) during the computation of $\Delta x_j^{(i)}$.

Finally, all parameter changes from all initial data points are compared and the one with the smallest error function value is selected as optimal. Note that this optimal parameter change will depend both on how large it is, which depends on the model coefficients $a_j^{(i)}$ or $b_j^{(i)}$, and how quickly the error function rises, which is a function of the quadratic surface weight $\omega_j^{(i)}$.

The new point $\hat{x}$ is then sent to the trajectory generation module, which then provides the robot with a new trial. The robot performs the trial and the new task score $\hat{y}$ is recorded. Assuming the trial execution still results in failure, the new sample $(\hat{x}, \hat{y})$ is appended to $\mathcal{S}$ and the process is repeated.

Algorithm 1 Model generation and movement selection

Input: $\mathcal{S} = \{(x^{(i)}, y^{(i)}) : x^{(i)} \in \mathcal{X}, y^{(i)} \in \mathcal{Y}, i = 1, 2, \ldots, m\}$, target score y_d, success tolerance ϵ
while $(\nexists\ y^{(i)} \in \mathcal{Y}$ s.t. $|y_d - y^{(i)}| < \epsilon)$ **do**
  for $i = 1 : m$ **do**
    $\left[\mathcal{N}^{(i)}, k\right] \leftarrow$ AdaptiveNeighborhoodSelection$(i, \mathcal{S})$;
    $A^{(i)} \leftarrow$ FitHyperplane $\left(\{(x^{(j)}, y^{(j)}) : x^{(j)} \in \mathcal{N}^{(i)}(k)\}\right)$ using (3)
    $\mathcal{M}^{(i)}(\beta) \leftarrow \{x^{(j)} : x^{(j)} \in \mathcal{X} / \mathcal{N}^{(i)} \wedge ||x^{(i)} - x^{(j)}||_\infty \leq \beta\}$
    for $j = 1 : |\mathcal{M}^{(i)}|$ **do**
      $e^{(j)} \leftarrow |y^{(j)} - \mathcal{F}^{(i)}(x^{(j)})|$;
    end for
    $\omega^{(i)} \leftarrow$ FindWeights $\left(\mathcal{E}^{(i)}(\Delta x), f(\omega^{(i)}), \{e^{(j)} \in \mathcal{M}^{(i)}\}\right)$; using (6)−(8)
  end for
  for $i = 1 : m$ **do**
    for $j = 1 : n$ **do**
      $\mathcal{L}_j^{(i)} \leftarrow \{x^{(k)} \in \mathcal{X} : |x_\ell^{(i)} - x_\ell^{(k)}| \neq 0 \text{ only for } \ell = j\}$;
      $c \leftarrow a_j^{(i)}$;
      if $|\mathcal{L}_j^{(i)}| \neq 0$ **then**
        $c \leftarrow FitLine\left(\{x^{(i)}, \mathcal{L}_j^{(i)}\}\right)$;
      end if
      $\Delta x_j^{(i)} \leftarrow \text{sgn}\left(\frac{y_d - y^{(i)}}{c}\right) \min\left(\left|\frac{y_d - y^{(i)}}{c}\right|, \sqrt{\frac{e_{max}}{\omega_j^{(i)}}}\right)$;
    end for
  end for
  $\Delta x_{j^*}^{(i^*)} \leftarrow \min_i \left(\arg\min_j \mathcal{E}^{(i)}(\Delta x_j^{(i)})\right)$;
  $[i^*, j^*] \leftarrow \arg\ \Delta x_{j^*}^{(i^*)}$
  $\hat{x}_j \leftarrow \begin{cases} x_j^{(i^*)} + \Delta x_j^{(i^*)} \text{ if } j = j^* \\ x_j^{(i^*)}, \text{ otherwise} \end{cases}$
  $\hat{y} \leftarrow$ Evaluate$(\hat{x})$;
  $\mathcal{S} \leftarrow$ Append$(\hat{x}, \hat{y})$;
  if $|y_d - \hat{y}| < \epsilon$ **then**
    $success \leftarrow 1$; **break**;
  end if
  $t \leftarrow t + 1$;
end while

Algorithm 2 Adaptive neighborhood selection

Input: Sample set $\mathcal{S}$, index of base point i;
$k \leftarrow n + 2$;
$\mathcal{N}^{(i)}(k) \leftarrow$ GenerateNeighborhoodSet(k); using(1) & (2)
$\eta \leftarrow 0$;
for $j = 1 : k$ **do**
  $A \leftarrow$ FitHyperplane$(\mathcal{N}^{(i)}(k)/x^{(j)})$;
  $\eta \leftarrow \frac{\eta + \text{FindLinearFitError}(A, x^{(j)})}{k}$;
end for
$k^* \leftarrow k$;
$\eta_{prev} \leftarrow \eta$;
while (true) **do**
  $k \leftarrow k + 1$;
  $\mathcal{N}^{(i)}(k) \leftarrow$ GenerateNeighborhoodSet(k); using(1) & (2)
  $\eta \leftarrow 0$;
  for $j = 1 : k$ **do**
    $A \leftarrow$ FitHyperplane$(\mathcal{N}^{(i)}(k)/x^{(j)})$;
    $\eta \leftarrow \frac{\eta + \text{FindLinearFitError}(A, x^{(j)})}{k}$;
  end for
  if $\eta_{prev} < \eta$ **then**
    $k^* \leftarrow k - 1$; **break**;
  end if
  $\eta_{prev} \leftarrow \eta$;
end while
return $\left[\mathcal{N}^{(i)}(k^*), k^*\right]$

IV. Results

A. Function Approximation

Our approach was first tested by considering a synthetic task defined by an unknown nonlinear function and having the algorithm attempt to find points in the function domain with values equal to given targets. An N-dimensional variation of the Schwefel function was used as the test function, defined by

$$z_s(\mathbf{x}) = \frac{1}{2c} \sum_i^N cx_i \sqrt{\sin |cx_i|}. \quad (11)$$

For the results presented, both functions were tested in a five-dimensional parameter space.

The algorithm performance was evaluated by generating random samples of the function with each parameter in the range -1 to 1. A set of function value targets was defined uniformly over the range of values covered by the samples and the average number of iterations needed to find each target in the set was used as the performance metric. All results are shown with a set of 51 targets tested.

For comparison purposes, we also implemented a simple algorithm using GPR, designed to greedily select new points

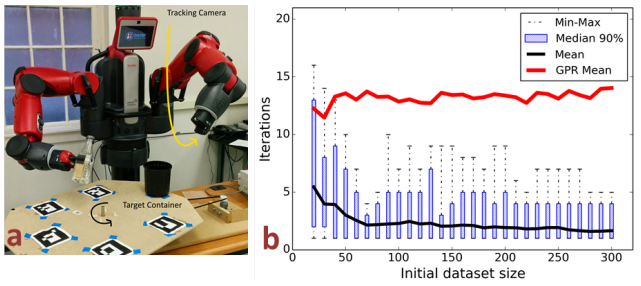

Fig. 1. (a) Experimental setup. (b) A comparison of the algorithm performance on the Schwefel test function, with varying sizes of the initial randomly sampled data set. All trials were performed with a tolerance of 0.005.

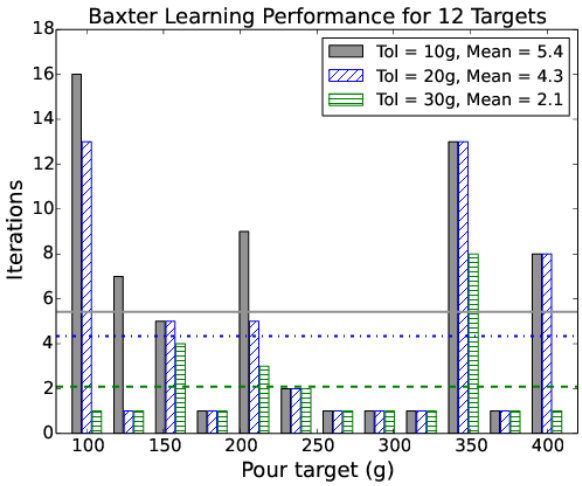

Fig. 2. Baxter learning performance for a uniformly distributed set of targets. Each group of three bars is the number of iterations needed for a single target, with the gray, blue, and green bars corresponding to tolerances of 10, 20, and 30 grams, respectively. All trials were done with the initial set of 37 random points.

that are expected to acheive task success, without regard to long-term improvement of the model quality. We trained a Gaussian Process globally on the full data set and then used a standard optimizer to find the point most likely to have task success. If the new point did not lead to success, the GP was fully retrained on the new data.

Figure 1(b) shows the performance as a function of the initial random data set size. As we expect to see, there is indication of asymptotic behavior where the algorithm is able to find a solution in a single iteration with sufficiently dense sampling of the parameter space. However, it is notable that the majority of the decrease in estimated number of iterations occurs at small data sizes (around 100), indicating good performance is attainable with only minor cost-intensive initial sampling. Notice that this trend does not occur with GPR, simply because we are searching over the entire parameter space and adding one sample at a time does not change the model greatly when the data set sizes are large.

B. Dynamic Pouring

To validate our approach on a physical robot, we used a dynamic pouring task (seen in Fig. 1(a)), where a Baxter robot is tasked to pour a specific amount of liquid into a container which is placed on a rotating table. This task is intended to be representative of a manufacturing scenario where the robot may be asked to perform new tasks or task variations with limited time available for learning. Additionally, the task of pouring liquid into a moving container is highly amenable to autonomous learning as it is extremely difficult to model accurately without an existing data set.

We selected the tilt trajectory of the bottle held by the robot as the learning target with the complementary motions of the robot's joints found using standard planning algorithms. In particular, the tilt profile robot's end-effector action consists of tilting the bottle in one direction (forward tilt) for some duration, keeping the tilt steady for some time, and untilting the bottle (reverse tilt). Accordingly, the tilt profile was parameterized by five real-valued parameters, which were manually defined as relevant physical features of the pouring action: (1) forward tilt time t_f, (2) forward tilt rate $\dot{\theta}_f$, (3) steady pouring time t_s, (4) reverse tilt time t_r, and (5) reverse tilt rate $\dot{\theta}_r$. A value for each of these parameters defines a point in the five-dimensional parameter space in which the learning algorithm operates.

An initial library of 40 points was generated by evaluating randomly generated points in the parameter space. Three were removed where either the entire volume of fluid (450 grams) or none was poured. With this initial data set, twelve targets were given uniformly from 100 to 400 grams. Figure 2 shows the performance on these targets for three different task success tolerance values. Notice the differing means for the different tolerance levels. Figure 3 shows a second experiment where the same targets were repeated but the initial data set used included the points tested in all previous targets. Again, paralleling the results from the previous section, there is a visible trend of a decreasing number of iterations needed as the initial library size grows. Also note that the all the means for the long-term learning in Fig. 3, using a 20 gram tolerance, are lower than the mean for the 20 gram tolerance case in Fig. 2.

V. Conclusions

We presented an approach that allowed a robot to bootstrap from sparse initial exploratory experiments and learn to perform a pouring task in very few attempts. Our initial results indicate that our local model exploitation is effective at both rapid and long-term learning for a physical task with complex dynamics. Our approach is mainly useful for motion planning problems in which model prediction by simulating the underlying physics involving the trajectory variables and task behavior is very difficult.

Future work will focus on extending the approach to be feasible for more complex problems. In particular, how the complexity and computational limits of the algorithm scale to higher dimensions is still an open question. Currently we update all local models as new data is acquired but the formulation naturally extends to an incremental learning

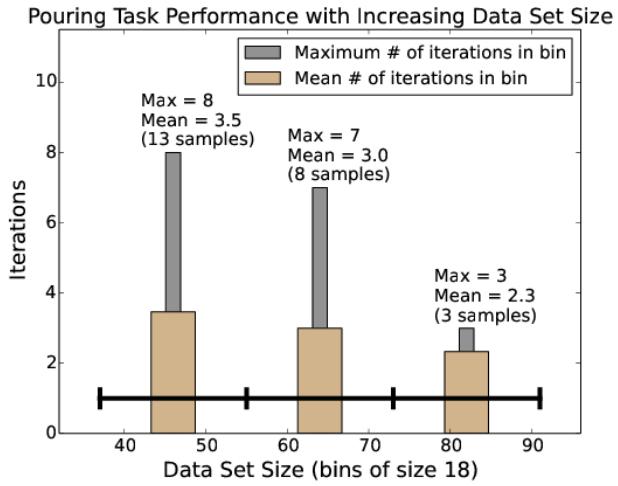

Fig. 3. Baxter learning performance depending on initial size of the data set when starting each new target. For this data, the robot was given the same initial set of 37 samples, but as it progressed through the 12 targets, all trials were saved in the data set so later targets had more samples to learn from. The full experiment was then repeated starting from the initial set once again to obtain 24 total target samples. As there was substantial variation in the samples, the results were divided into three bins so that a rough mean could be estimated. The tolerance was 20 grams for all targets.

scheme where only the models in the vicinity of the new data point are adjusted. Additionally, other modeling algorithms can also be used with the exploitation strategy effectively. This suggests we could expand the approach to become model-agnostic and even use a hybrid approach where different regions of the parameter space could be modeled by different algorithms, depending on the function behavior. The linear models could be used in regions of sparse data with high computational performance, a Gaussian process can be used to provide more accurate predictions in regions of high variability, and LWPR could be used when sufficient samples had been acquired to slow down the other algorithms. These improvements may enable the approach to be competitive for much higher dimensional problems as well.

Other important questions remain to be answered include whether we can provide theoretical guarantees for boundedness of model error and convergence of parameters. Currently, the pouring task is parameterized manually. The applicability of this approach can be enhanced by devising methods to automatically extract relevant task parameters. Another future direction worth exploring is to see if there are similarities between tasks and can a learned model be extrapolated from one task to solve another instead of learning the model again.

References

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

[2] J. C. Bongard and H. Lipson. Nonlinear system identification using coevolution of models and tests. *IEEE Transactions on Evolutionary Computation*, 9(4):361–384, Aug 2005.

[3] J. C. Bongard, V. Zykov, and H. Lipson. Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121, 2006.

[4] C. Bowen, Gu Ye, and R. Alterovitz. Asymptotically optimal motion planning for learned tasks using time-dependent cost maps. *Automation Science and Engineering, IEEE Transactions on*, 12(1):171–182, Jan 2015.

[5] Sascha Brandi, Oliver Kroemer, and Jan Peters. Generalizing Pouring Actions Between Objects using Warped Parameters. In *Humanoid Robots, 2014 14th IEEE-RAS International Conference on*, 2014.

[6] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal. Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors. *Neural computation*, 25:328–73, 2013.

[7] K. N. Kaipa, J. C. Bongard, and A. N. Meltzoff. Self discovery enables robot social cognition: Are you my teacher? *Neural Networks*, 23(89):1113 – 1124, 2010. Social Cognition: From Babies to Robots.

[8] S. M. Kakade, M. J. Kearns, and J. Langford. Exploration in metric state spaces. *Proceedings of the 20th International Conference on Machine Learning*, pages 306–312, 2003.

[9] O.B. Kroemer, R. Detry, J. Piater, and J. Peters. Combining active learning and reactive control for robot grasping. *Robotics and Autonomous Systems*, 58(9):1105 – 1116, 2010. Hybrid Control for Autonomous Systems.

[10] Scott Kuindersma, Roderic Grupen, and Andrew Barto. Variational bayesian optimization for runtime risk-sensitive control. *Robotics: Science and Systems*, page 201, 2013.

[11] Daniel Lizotte, Tao Wang, Michael Bowling, and Dale Schuurmans. Automatic gait optimization with gaussian process regression. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 944–949, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[12] L. Mihalkova and R. Mooney. Using active relocation to aid reinforcement learning. *Proceedings of the 19th International FLAIRS Conference*, pages 580–585, 2006.

[13] B. Nemec, D. Forte, R. Vuga, M. Tamosiunaite, F. Worgotter, and A. Ude. Applying statistical generalization to determine search direction for reinforcement learning of movement primitives. *IEEE-RAS International Conference on Humanoid Robots*, pages 65–70, 2012.

[14] Duy Nguyen-Tuong and Jan Peters. Model learning for robot control: a survey. *Cognitive Science*, 12(4):319–40, 2011.

[15] S. Otte, J. Kulick, M. Toussaint, and O. Brock. Entropy-based strategies for physical exploration of the environment's degrees of freedom. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 615–622, Sept 2014.

[16] C. E. Rasmussen and C. K. Williams. *Gaussian Processes for Machine Learning*. MIT Press, Boston, Massachusetts, United States, 2006.

[17] M. Rolf and J. J. Steil. Efficient exploratory learning of inverse kinematics on a bionic elephant trunk. *IEEE Transactions on Neural Networks and Learning Systems*, 25(6):1147–1160, 2014.

[18] N. Roy and A. McCallum. Toward optimal active learning through sampling estimation of error reduction. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 441–448, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[19] R. Saegusa, G. Metta, G. Sandini, and S. Sakka. Active motor babbling for sensorimotor learning. In *IEEE International Conference on Robotics and Biomimetics*, pages 794–799, Feb 2009.

[20] B. Settles. Active Learning Literature Survey. *Machine Learning*, 15:201–221, 2010.

[21] B. Settles, M. Craven, and S. Ray. Multiple-instance active learning. In J.C. Platt, D. Koller, Y. Singer, and S.T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1289–1296. Curran Associates, Inc., 2008.

[22] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 287–294, New York, NY, USA, 1992. ACM.

[23] Minija Tamosiunaite, Bojan Nemec, Aleš Ude, and Florentin Wörgötter. Learning to pour with a robot arm combining goal and shape learning for dynamic movement primitives. *Robotics and Autonomous Systems*, 59(11):910–922, 2011.

[24] S. Vijayakumar, A. D'Souza, and S. Schaal. Incremental online learning in high dimensions. *Neural computation*, 17:2602–2634, 2005.