2017 IEEE International Conference on
Autonomous Robot Systems and Competitions (ICARSC)
April 26-28, Coimbra, Portugal

Adaptive Non-Maximal Suppression Filtering for Online Exploration Learning with Cost-Regularized Kernel Regression

Carlos Cardoso and Alexandre Bernardino
Institute for Systems and Robotics, Instituto Superior Técnico, Lisboa, Portugal
Email: carlos.cardoso@tecnico.ulisboa.pt, alex@isr.ist.utl.pt

***Abstract*—In many robotics tasks involving impacts (e.g. grasping, hitting, kicking) the existence of complex interactions between the physics of the objects and of the robot makes it hard to create an analytical model of the interactions that can be used for prediction and planning. Exploration learning can enable a robot to autonomously learn such tasks and models simultaneously by trial and error. The Cost-Regularized Kernel Regression (CrKR) algorithm has been used successfully for learning robot skills such as table tennis, ball-in-a-cup, and dart-throwing. However, despite its effectiveness for fast exploration learning with few data samples, a matrix inversion is required for each new trial. This matrix inversion makes the method computationally expensive so that exploration must finish before the task is fully learned. We present a novel method that can learn continually, by selecting the best information for learning at each step. Using an Adaptive Non-Maximal Suppression (ANMS) filter, we select a sparse subset of points for the regression for each new trial. This enables the robot to continue acquiring samples without the need to discard potentially informative trials. We compare our algorithm with the original CrKR regression, showing a good trade-off between the performance and efficiency of the exploration.**

I. Introduction

The perception of robots as future helpers, capable of performing tasks with the same dexterity as a human, has been driving the research of robots with increasing mechanical complexity. Today robots take the place of humans in the most tedious and dangerous jobs. They can already be seen vacuuming people's homes and searching for life on mars. However, these are still relatively simple robots, with a high degree of specialization for a single task. More general robots with many degrees of freedom and a large number of sensors have become common in research (e.g. iCub, PR2) and are also showing signs of increasingly more industrial use (e.g. Baxter). However, this increase in mechanical sophistication demands for more advanced control methods. Many tasks that are relatively simple for humans, such as driving a car or repairing a broken object, today remain an immense challenge to robots. The increased complexity of the robots leads to an increase in the required human effort to model the robot and its environment for each new task.

An appealing alternative to extensive analytical modeling is to learn the task and model together by demonstration or by trial and error.

One practical way to allow robots to learn motor skills is through the "programming by demonstration" paradigm [1]. The main idea is that the motor skill is demonstrated to the robot (typically through kinesthetic teaching) and a representation of the movement is learned from the recorded data. One key issue is to find a proper representation for such movement primitives. Promising approaches draw from the theory of dynamical systems, giving rise to solutions such as Stable Estimator of Dynamical Systems (SEDS), sequenced Linear Dynamical Systems, Implicit Dynamical Systems [2] and, most notably, Dynamic Movement Primitives (DMPs [3]). This latter formulation, in particular, has proven to be a very effective tool for imitation learning, and has been therefore widely used in robotics and inspired many extensions to add velocity goals [4] and allow uncertainty and way-points in the execution [5]. After acquiring a set of primitives we can use learning to: a) refine the shape of the learned primitive using PoWER [6], or PI^2 [7] b) find a task specific mapping from perception to the goal for the primitive, which can be done offline in batch using Gaussian Process Regression (GPR) [8], [9] or by autonomous exploration using Cost-Regularized Kernel Regression (CrKR) [10], [4].

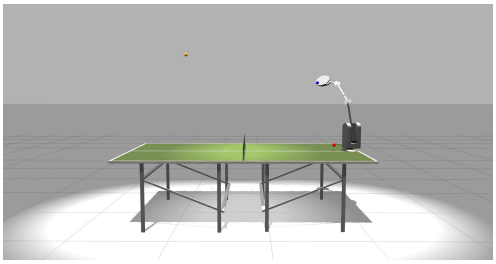

Fig. 1. A screenshot of our simulated robot learning table tennis by exploration.

To illustrate our methods, in this paper we consider the task of tennis play learning by a robot with 6 degrees-of-freedom (see Fig. 1). We represent the learning problem as a map between the ball state (cartesian position and velocities) when it passes at a certain distance of the robot, and the desired hit time and angular position and velocity of the robot end-effector (see Fig. 2). The task is to hit the ball and make it bounce on the table.

Our approach is based on CrKR. CrKR is a method that uses a heuristic cost measure to estimate the uncertainty of each robot trial. The estimated uncertainty is used to add variability to the exploration of the task space. If a trial has been very successful the cost is low and next trial in similar configurations will require small exploration. On the

contrary, if the task was not successful, the cost is large and more exploration is required. The algorithm requires a matrix inversion with $\mathcal{O}(n^3)$ computational complexity and $\mathcal{O}(n^2)$ space complexity with the number of trials n. This severely limits the learning of complex tasks, restricting the usefulness of the algorithm to tasks in low dimension spaces. This limitation in the number of data points that can be collected makes it extremely important to initialise CrKR with very good (i.e. low cost) initial demonstrations and to choose very carefully the kernel and its parameters, in order to quickly converge to good solutions.

In order to make online learning tractable, we need to approximate the matrix inversion operation. A possible strategy based on splits in the learning process was explored by Macedo et al. [11]. However, this approach appears sensitive to the input dimensionality and we could not apply it successfully to our table tennis task. Another strategy proposed by Nguyen et al [12] consists in keeping a fixed size dictionary of data-points. A new trial is only added to the regression when the data-point achieves a higher independence score than all the previously collected trials. Although this independence metric can be calculated by an incremental update it is still computationally expensive, again limiting the number of collected samples.

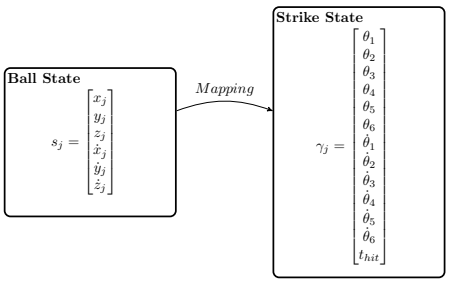

Fig. 2. In order to generalize a primitive in a 6 dof table tennis robot we need to find a mapping from the measured state of the ball (with Cartesian position and velocity) to a desired position and velocity of the robot in joint space and a desired hit time ($\mathbb{R}^6 \rightarrow \mathbb{R}^{13}$)

II. Reinforcement Learning of Motion Parameters

In this section, we introduce some background information and the mathematical notation used in the remainder of the manuscript. We start with Gaussian Processes, a simple class of models of functions suitable for probabilistic inference in both regression and classification problems. Next, we introduce CrKR, a kernel method for exploration learning of tasks. Finally, we present our proposed approach for learning beyond the point where the cubic time complexity with the number of samples becomes a limitation.

A. Gaussian Processes

A Gaussian Process (GP) is by definition any distribution over functions such that any finite set of function values $f(\boldsymbol{x_1}), f(\boldsymbol{x_2}), ...f(\boldsymbol{x_N})$ has a joint Gaussian distribution [13]. The GP is specified by its mean function

$$\mathbb{E}[f(\boldsymbol{x})] = \mu(\boldsymbol{x}) \tag{1}$$

and by its covariance function or *kernel*:

$$Cov[f(\boldsymbol{x}), f(\boldsymbol{x'})] = k(\boldsymbol{x}, \boldsymbol{x'}). \tag{2}$$

In practice, it is common to assume the mean function is zero everywhere. The choice of the covariance function is of great importance since it implicitly encodes assumptions about the underlying function to be modeled.

Gaussian Process Regression (GPR) is the problem of modeling a stochastic sampling process by a GP. Given n training points $(\boldsymbol{x}_i, y_i)_{i=1}^n$, a GPR models a latent function $f(\boldsymbol{x})$ that transforms the input $\boldsymbol{x}$ into a prediction $y = f(\boldsymbol{x}) + \boldsymbol{\varepsilon}$ where $\boldsymbol{\varepsilon}$ is Gaussian noise with zero mean and variance $\boldsymbol{\sigma}^2(\boldsymbol{x})$, i.e. $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \boldsymbol{\sigma}^2(\boldsymbol{x}))$.

The prediction of a GPR is found by calculating the mean and the variance at a new evaluation point $\boldsymbol{x}_*$:

$$\begin{aligned} \bar{f}(\boldsymbol{x}_*) &= \boldsymbol{k}_*^T (\boldsymbol{K} + \boldsymbol{\sigma}_n^2 \boldsymbol{I})^{-1} \boldsymbol{y} \\ \boldsymbol{\sigma}_f^2(\boldsymbol{x}_*) &= \boldsymbol{k}(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{k}_*^T (\boldsymbol{K} + \boldsymbol{\sigma}_n^2 \boldsymbol{I})^{-1} \boldsymbol{k}_* \end{aligned} \tag{3}$$

where $\boldsymbol{K}$ is the matrix where each entry corresponds to the covariance kernel evaluated on the input training data $\boldsymbol{K}_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $\boldsymbol{k}_*$ is a vector where each entry is the covariance function evaluated between the input point $\boldsymbol{x}_*$ and the remaining training data. The variance of the input data is $\boldsymbol{\sigma}_n^2$ and $\boldsymbol{y}$ is the vector of the training outputs.

The kernel defines how the model generalizes to new data. In this way, it provides a prior and specifies the kind of structure that can be captured by the GP. The specification of a kernel that represents the particular structure of the problem being modeled is one of the main challenges in applying GP for model learning. However, in practice, the squared exponential (SE) kernel has become the *de facto* standard when the structure of the function is unknown or hard to specify. It is defined by only two parameters that are easy to interpret. First, the *lengthscale* or *bandwidth* of the kernel l determines the smoothness of the function and how far away from the training data the GP can extrapolate. The output variance σ^2 determines the average distance of the function away from its mean. Intuitively it acts as a scale factor.

$$k_{SE}(x, x') = \sigma^2 \exp(-\frac{(x - x')^2}{2l^2}) \tag{4}$$

For problems with D dimension, we can multiply many SE kernels, each with a bandwidth adjusted to the corresponding dimension d. This multiplication of kernels with different bandwidths composes the SE-ARD kernel in eq. 5, illustrated graphically in Fig. 3.

$$k_{SE-ARD}(\boldsymbol{x}, \boldsymbol{x'}) = \sigma_f^2 \exp(-\frac{1}{2} \sum_{d=1}^{D} \frac{(x_d - x'_d)^2}{l_d^2}) \tag{5}$$

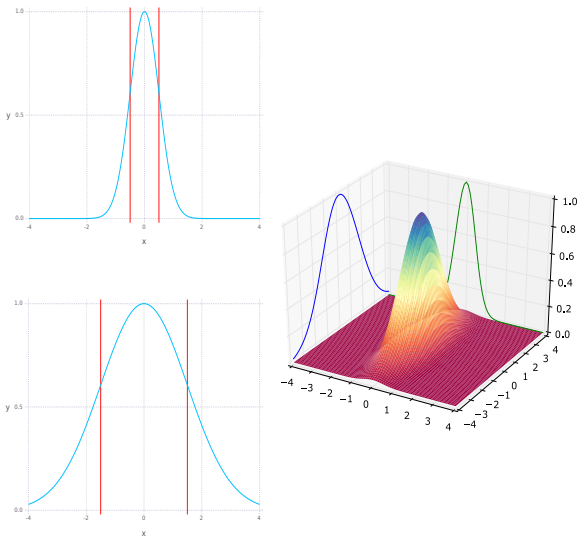

Fig. 3. SE-ARD kernel formed by multiplying two SE kernels with a different bandwidth.

B. Marginal Likelihood Maximization.

A very important feature of GPs is the ability to compute the *marginal likelihood* of a test data set given the model or *evidence*. Models can be compared using the marginal likelihood, which allows automatically tuning the parameters of a model and its fit to the data.

$$p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta}) = (2\pi)^{-\frac{n}{2}} \times |\boldsymbol{K}|^{-\frac{n}{2}} \times \exp\{-\frac{1}{2}(\boldsymbol{y}^T \boldsymbol{K}^{-1}\boldsymbol{y})\} \tag{6}$$

We can write in log form and note that it is differentiable.

$$\begin{aligned} \log p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta}) &= -\frac{1}{2}\boldsymbol{y}^T \boldsymbol{K}_y^{-1}\boldsymbol{y} - \frac{1}{2}\log|\boldsymbol{K}| - \frac{n}{2}\log 2\pi \\ \frac{\partial}{\partial\theta_j}\log p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta}) &= \frac{1}{2}\boldsymbol{y}^T \boldsymbol{K}^{-1}\frac{\partial \boldsymbol{K}}{\partial\theta_j}\boldsymbol{K}^{-1}\boldsymbol{y} - \frac{1}{2}tr(\boldsymbol{K}^{-1}\frac{\partial \boldsymbol{K}}{\partial\theta_j}) \\ &= \frac{1}{2}tr((\boldsymbol{\alpha}\boldsymbol{\alpha}^T - \boldsymbol{K}^{-1})\frac{\partial \boldsymbol{K}}{\theta_j}) \\ \textit{where } \boldsymbol{\alpha} &= \boldsymbol{K}^{-1}\boldsymbol{y} \end{aligned} \tag{7}$$

The free parameters of a GPR are the *hyper-parameters* of the kernel. For the SE kernel, these are the output variance σ^2 and the bandwidth l. The usual practice to determine the best parameters is to maximize the *log marginal likelihood* using common optimization tools such as quasi-Newton methods or take advantage of its differentiability (Eq. (7)) and use gradient methods [14].

C. Cost-regularized Kernel Regression

Gaussian processes assume that the training points are sampled from the underlying process with Gaussian noise. However, some tasks are interactive in nature and must be actively explored by the robot system. CrKR is a kernelized version of the reward-weighted regression [10], that is suitable for reinforcement learning (RL) of complex tasks with continuous input and output spaces [10]. A common application of CrKR is to learn a small set of continuous meta-parameters that can generalize a motor primitive [10], [4].

Algorithm 1 Cost-regularized Kernel Regression algorithm

Preparation steps:
determine initial state $\boldsymbol{s_0}$, output $\boldsymbol{\gamma_0}$ and cost $\boldsymbol{c_0}$ of the demonstration.
initialize the corresponding matrices $\boldsymbol{S}$, $\boldsymbol{\Gamma}$, $\boldsymbol{C}$.
choose a kernel $\boldsymbol{k}$, initialize $\boldsymbol{K}$.
set the exploration/exploitation trade-off λ.
for all iterations j **do**
Determine the state $\boldsymbol{s}^j$ specifying the situation.
Calculate the meta-parameters $\boldsymbol{\gamma}_j$ by:
Determine the mean of each meta-parameter i:
$\bar{\gamma}_i(\boldsymbol{s}^j) = \boldsymbol{k}(\boldsymbol{s}^j)^T(\boldsymbol{K}+\lambda\boldsymbol{C})^{-1}\boldsymbol{\Gamma}_i$,
Determine the variance:
$\sigma^2(\boldsymbol{s}^j) = \boldsymbol{k}(\boldsymbol{s}^j,\boldsymbol{s}^j) - \boldsymbol{k}(\boldsymbol{s}^j,\boldsymbol{S})^T(\boldsymbol{K}+\lambda\boldsymbol{C})^{-1}\boldsymbol{k}(\boldsymbol{s}^j,\boldsymbol{S})$,
Draw the meta-parameters from a Gaussian distribution
$\gamma^j \sim \mathcal{N}(0,\sigma^2)$
Execute the motor-primitive using the new meta-parameters
Calculate the cost c^j at the end pf the episode
Update the matrices $\boldsymbol{S}$,$\boldsymbol{\Gamma}$,$\boldsymbol{C}$ with the new sample
end for

CrKR has been successfully applied to many robot learning tasks such as dart throwing, ball throwing, and table tennis in many different robots, outperforming techniques like finite difference gradient and reward-weighted regression [15].

The inference is performed by estimating a *maximum a posteriori* (MAP) estimate under a zero mean Gaussian prior. This is the common approach in discriminative supervised learning methods like ridge regression, or the Gaussian Processes method, for its analytical and computational tractability. In this case, the inference expression is similar to a Gaussian noise process with input-dependent (heteroscedastic) variance. The major improvement, that allows autonomous exploration by a robot is the replacement in eq. 3, of the term $\sigma^2\boldsymbol{I}$ by $\lambda\boldsymbol{C}$ to make a cost weighted regression. The open parameter λ expresses an exploration-exploitation trade-off that scales the cost function. This is simply an adjustment to the predicted uncertainty of each sample given by our cost function. Because of its fundamental similarities, we can look to the approximate inference strategies employed with other discriminative methods like GPR (eq. 3) and adapt to CrKR in an exploration learning task. The CrKR algorithm and a graphical illustration are provided in Algorithm 1 and Fig. 4, respectively.

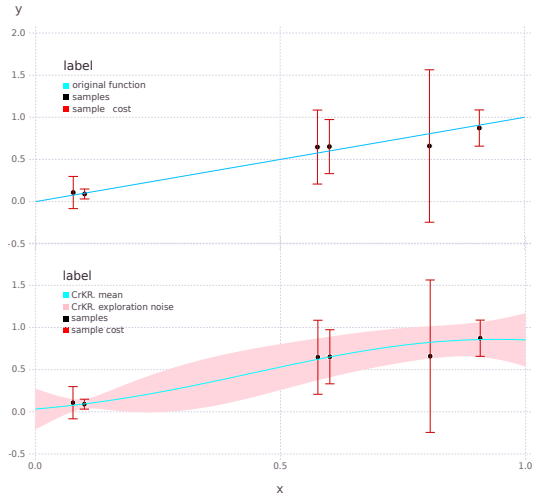

Fig. 4. CrKR after learning from an initial demonstration and performing five exploration iterations. The pink region shows magnitude of the exploration noise applied by CrKR for each region, the error bars show the cost of each tryout.

D. Selection of the Cost Function

Great care must be taken in the choice of a cost function appropriate to the task. Cost functions that represent some notion of closeness to the target tend to perform better than binary success/failure functions in robotic tasks. In the case of CrKR, the cost function is employed as a heuristic prediction of the uncertainty at each data-point and is used to guide the exploration. Another consideration is the possibility of including desirable properties such as low accelerations in the computation of the cost. Fortunately for many tasks these properties of the learned policy can be discovered by the autonomous agent without needing to be explicitly included in the cost function. A possible cost function that was shown mimic some properties of the human reasoning [16], is the saturated cost function in eq. 8.

$$c(\boldsymbol{x}) = 1 - exp(-\frac{1}{2a^2}\left\|\boldsymbol{x} - \boldsymbol{x}_{target}\right\|^2) \tag{8}$$

where a is a tuning parameter that regulates the interval of distances where the function is most sensitive. This cost function is locally quadratic but saturates for large deviations from the desired target. It is also very easy to reason with and was our choice for the table tennis task. For robot table tennis we can set $a^2 = 6$ cm, so that trials that miss the paddle will have almost maximum cost, while successful data points lie in the more sensitive range of the cost function.

III. ANMS-CrKR

In this section, we describe the main contribution of this paper, the ANMS-CrKR algorithm. As seen in the previous section the CrKR prediction step involves the inversion of a square matrix $(K+\lambda C)^{-1}$ of order n where n is the number of samples. The time complexity of most matrix inversion algorithms is $O(n^3)$ and the space for storing the matrix is $O(n^2)$. The high computational complexity becomes specially relevant when dealing with on-line learning, and real-time applications were the previous trial must be integrated quickly in the algorithm to generate the next one.

A possible approach to circumvent the cubic complexity of regression algorithms is to reduce the instance size by finding a representative subset of the data points. Since in CrKR each point has an associated cost we can formulate the problem as finding the sparsest set[1] of N low-cost points[2]. This is a similar problem to the selection of key points in images for computer vision applications. We decided to combine CrKR with ANMS (Adaptive Non-Maximal Suppression) a filter commonly used for spatial feature distribution [17], [18]. ANMS works by sorting the entire data set according to the weights. Then, starting from the lowest cost data point, all data points are added in sequence by cost order. For each point, the distance to the closest point already evaluated is saved and added to a list. The N points with highest distances in this list are selected for the regression. This promotes a good coverage of the state-space. We provide a pseudo-code implementation of the algorithm in listing 2.

Algorithm 2 Implementation of ANMS Filtering

Preparation steps:
Sort P by cost, let p_1,p_2,... denote the points in that order.
Radius List = [(∞, p_1)] //(radius,keypoint)
for all $p_i \in P$, $i \neq 1$ **do**
r_i = distance to nearest point already in Radius List.
Add (r_i,p_i) to Radius List
end for
Sort Radius List by radius, return first k entries in Radius List.

This method allows retaining a larger number of training samples from which a subset is used for the regression. Another advantage is that the filter can use any distance measure, allowing flexibility in the choice of the kernel for the regression. In our online regression case since we want to update our regression for a new trial, we can keep the P list of points sorted by cost ($\mathcal{O}(n)$). Although naively ordering by radius takes $\mathcal{O}(n^2)$, by using a euclidean distance and approximate nearest neighbors we can reduce this to $\mathcal{O}(n\log(n))$ [17], [19]. Getting the first k elements can be done by partial sorting $\mathcal{O}(k\log(n))$. However since in our case we do not need the selected data to be in a specific order the problem can be reduced to partition-based-selection and solved in $\mathcal{O}(k\log(n))$. As such the filter can be implemented

[1] sparsity is interpreted as the separation between datapoints
[2] low cost points are more important in guiding exploration to good regions in the state space.

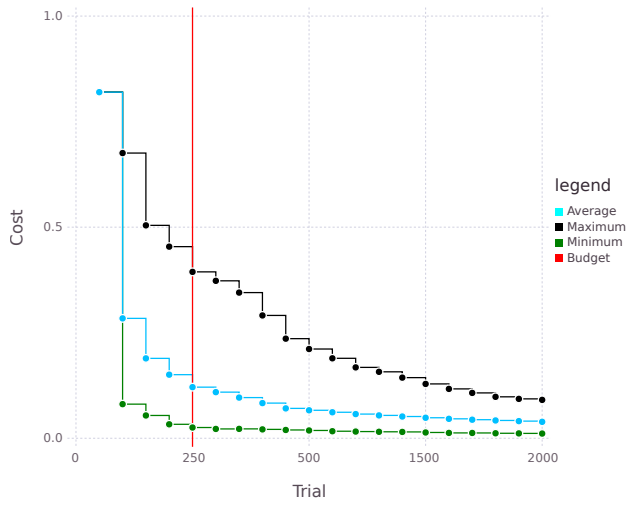

Fig. 5. In this figure we can see how CrKR can continue to learn beyond a fixed budget by selecting the data samples with an ANMS filter. The system is learning an analytical simulation of the ball trajectory and the robot kinematics of a table tennis task. The vertical line shows the trial in which the budget of 250 samples was filled, beyond this line ANMS was used to select the most representative of all collected samples for regression. Each training epoch consists of 50 training samples, the results shown are the average, the maximum and the minimum of 10 runs evaluated after each epoch.

with an overall complexity of $\mathcal{O}(n\log(n))$, allowing a much larger amount of samples to be collected.

IV. Experimental Setup and Simulation

In order to test the algorithms described in this work, we designed a robot table tennis experiment. In this experiment, the goal is to hit a table tennis ball by means of a serial manipulator with a paddle attached. The experiments were done in a simulated environment implemented in Gazebo a multi-robot simulator with dynamics simulation and advanced 3D graphics.

The trial is classified as successful if, after the hitting time t_{hit}, the ball is traveling away from the robot and bounces in the table. Only the successful trials were used as data points to update the regression. The position of the ball and of the paddle at time t_{hit} were used to compute the cost c_j associated with the trial according to the saturated cost function of Eq. 8. This sequence of steps during a trial can be seen in figure 6.

To learn the mapping between the state of the incoming ball and the goal for our robotic manipulator we applied the proposed ANMS-CrKR algorithm. In this case, the input to the system is the measured state of the ball's position and velocity in cartesian 3D space when it passes at a fixed distance from the robot. The output is the position and velocity of each joint of the robotic manipulator at a desired hit time. This goal is then passed to a motion primitive computed using the Quadratic Programming (QP) method of [20] to generate an appropriate trajectory. During the execution of the swing, a game state tracker stores information to evaluate the cost of the swing:

- The ball state at the goal time t_{hit}.
- The cartesian position of the paddle by computing the forward kinematics with the measured state of the joints at time t_{hit}.
- The state of the ball in the next bounce from ground or table after time t_{hit}.

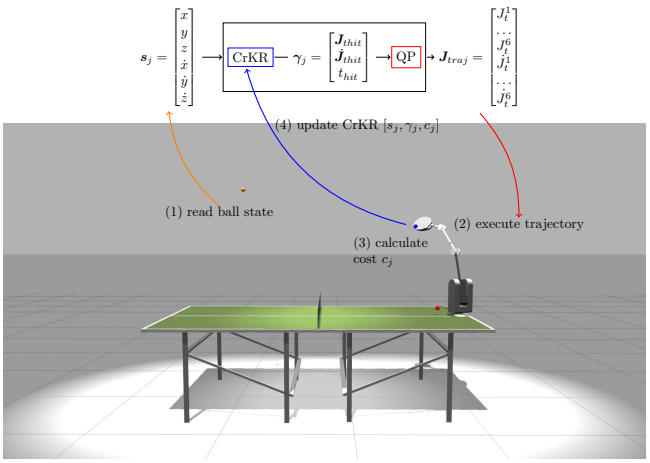

Fig. 6. The sequence of steps used when learning the motion parameters by trial and error. At a predetermined distance a snapshot of the ball state $\boldsymbol{s}_j$ is collected (1). The state is used to get a new prediction of the desired robot state, $\boldsymbol{\gamma}_j$, by regression with CrKR over the demonstrations and previous tryouts. The desired state is used to generate a trajectory $\boldsymbol{J}_{traj}$ based on a motion primitive. The Trajectory is then executed in the robot arm (2). Analyzing the resulting play, we calculate a cost c_j (3) that is used to update the matrices of samples in CrKR (4).

The table acts as a ground plane that significantly limits the configurations that the serial manipulator can achieve without collision. To ensure that no collision occurs with the table we sample the forward kinematics of the generated trajectory using *OpenRAVE* [21]. The table was included in the robot model so that any situation where the arm would collide with the table is detected internally as a self-collision.

A. Results

The simulation was run starting from an initial set of five demonstrations and a motion primitive. The system throws the balls starting with position and velocity sampled from a uniform distribution that was tuned to generate trajectories lying mostly within the bounds of the robots reachability. Regularly the learned policy is evaluated by running a fixed set of 50 trials with the same probability distribution of the training samples. The results obtained can be seen in Fig. 7. We achieved a success rate of 78% according to the criteria previously described. It should be noted that this rate is dependent on the trajectory of the balls thrown to the robot, some trajectories in the test set may be unreachable for the serial manipulator so a 100% success rate is not possible. The learned policy can be visualized by variating one of the input state dimensions and plotting the resulting policy for the

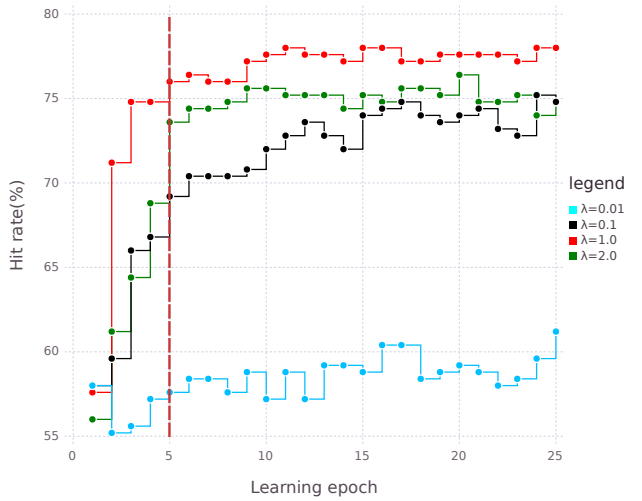

Fig. 7. The success rate by epoch while learning in the Gazebo simulator for different λ. After each training epoch of 50 random samples, an evaluation is performed on a fixed test set of 50 trials. A trial is considered successful if the ball is hit and bounces the table while moving away from the robot. The cost is calculated by the expression 8. The plots are an average over 5 different runs for each value of the CrKR exploration/exploitation trade-off λ. The vertical line marks the point beyond which the samples start to be chosen by ANMS for the regression.

joint positions in cartesian space by computing the forward kinematics. Another important consideration is that due to timings and errors in the physics of the simulation there is some variability in our test set. The same policy executed in the same test can result in a slightly different outcome.

V. Conclusions

In this work, expanding from the known methods used for RL in this class of robotics problems, we proposed the addition of a sparsification method based on ANMS filtering. This sparsification method is capable of choosing online a low cost and representative subset of the collected samples. By using this selected subset for regression the robot can continue learning past the number of samples beyond which the matrix inversion computation becomes prohibitively expensive. We test our method in a simulated robot, concluding that it can learn past conventional CrKR, given the same computational resources and without showing a significant degradation in the learning efficiency. Future work will focus on making large scale statistical tests to thoroughly characterize the ANMS-CrKR methods, as well as compare with other approximate kernelized regression methods.

Acknowledgment

This work was supported by the Portuguese Foundation for Science and Technology (FCT) through the project LARSyS-FCT [UID/EEA/5009/2013], and by the project AHA-Augmented Human Assistance [CMUP-ERI/HCI/0046/2013].

References

[1] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *Proceedings of the Fourteenth Int. Conference on Mach. Learning*, ser. ICML '97. Morgan Kaufmann Publishers Inc., 1997, pp. 12–20.

[2] R. Krug and D. Dimitrovz, "Representing movement primitives as implicit dynamical systems learned from multiple demonstrations," in *ICAR, 2013*, Nov 2013, pp. 1–8.

[3] A. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," *ICRA, 2002*, pp. 1398–1403, 2002.

[4] K. Mulling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *The Int. J. of Robotics Research*, vol. 32, no. 3, pp. 263–279, 2013.

[5] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Probabilistic movement primitives," in *NIPS*, 2013, pp. 2616–2624.

[6] J. Kober and J. Peters, "Policy search for motor primitives in robotics," *Machine Learning*, vol. 84, no. 1, pp. 171–203, 2011.

[7] F. Stulp, E. Theodorou, and S. Schaal, "Reinforcement learning with sequences of motion primitives for robust manipulation," *Robotics, IEEE Transactions ...*, vol. 28, no. 6, pp. 1360–1370, dec 2012.

[8] A. Ude, A. Gams, T. Asfour, and J. Morimoto, "Task-specific generalization of discrete and periodic dynamic movement primitives," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 800–815, 2010.

[9] B. Nemec and A. Ude, "Reinforcement learning of ball-in-a-cup playing robot," *2011 IEEE International Conference on Robotics and Biomimetics, ROBIO 2011*, pp. 2682–2687, 2011.

[10] J. Kober, A. Wilhelm, E. Oztop, and J. Peters, "Reinforcement learning to adjust parametrized motor primitives to new situations," *Autonomous Robots*, vol. 33, no. 4, pp. 361–379, Nov. 2012.

[11] J. Macedo, C. Santos, and L. Costa, "Using Cost-regularized Kernel Regression with a high number of samples," *2014 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pp. 199–204, may 2014.

[12] D. Nguyen-Tuong and J. Peters, "Incremental online sparsification for model learning in real-time robot control," *Neurocomputing*, vol. 74, no. 11, pp. 1859–1867, May 2011.

[13] C. E. Rasmussen, "Gaussian processes for machine learning," 2006.

[14] C. Plagemann, K. Kersting, P. Pfaff, and W. Burgard, "Heteroscedastic gaussian process regression for modeling range sensors in mobile robotics," in *Snowbird learning workshop*, 2007.

[15] J. Kober, "Learning motor skills: from algorithms to robot experiments," *it-Information Technology*, vol. 56, no. 3, pp. 141–146, 2014.

[16] K. P. Körding and D. M. Wolpert, "The loss function of sensorimotor learning," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 101, no. 26, pp. 9839–9842, 2004.

[17] S. Gauglitz, L. Foschini, M. Turk, and T. Höllerer, "Efficiently selecting spatially distributed keypoints for visual tracking," *ICIP*, pp. 1869–1872, 2011.

[18] M. Brown, R. Szeliski, and S. Winder, "Multi-Image Matching using Multi-Scale Oriented Patches," 2005.

[19] T. M. Chan, "A minimalists implementation of an approximate nearest neighbor algorithm in fixed dimensions," *See http://www. cs. uwaterloo. ca/~ tmchan/pub. html*, 2006.

[20] C. Cardoso, L. Jamone, and A. Bernardino, "A novel approach to dynamic movement imitation based on quadratic programming," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 906–911.

[21] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, Carnegie Mellon University, Robotics Institute, August 2010.