Learning Velocity Kinematics: Experimental Comparison of On-line Regression Algorithms

Alain Droniou, Serena Ivaldi, Patrick Stalph, Martin Butz and Olivier Sigaud

***Abstract*—The increasing complexity of tasks addressed by humanoid robotics requires accurate mechanical models which are difficult to obtain in practice. One approach is to let the robot learn its own models. In [16], two algorithms were compared on learning the velocity kinematics model of iCub: XCSF and LWPR. This comparison was based on simulated data. In this paper, we propose to extend this study to data recorded from the iCub robot. We analyze the behavior of these algorithms in presence of large noise in real conditions of use. We also add the study of a third algorithm, iRFRLS. After a detailed study on the tuning of the three algorithms, we show that the results obtained in [16] are still valid on real data: XCSF converges more slowly, but to a lower error than LWPR. However, we show that iRFRLS outperforms these two algorithms.**

I. Introduction

For the few last decades, robot control was based on hard-coded mechanical models. Even if it can still be suitable for complex robots with many degrees of freedom [10], such methods usually do not take into account perturbations, like viscous and solid friction (which vary with mechanical wear), contacts with unknown objects, ... For instance, when interacting with new objects, autonomous robots require new mechanical models to adapt their behavior: playing with a bottle is different depending on its size, its material, whether it is filled or empty... Learning techniques are expected to provide efficient and fast generalization to transfer knowledge between related situations.

In [16], two algorithms, LWPR and XCSF, were compared on learning velocity kinematics of the humanoid robot iCub from visual inputs. In order to introduce uncertainties and address a situation where the model is unknown, the end-effector was modeled by a green ball at the top of a stick held by the robot. The study was carried on simulated data. In this paper, we refine the study with data recorded from the real robot. In fact, as soon as we work in real conditions, we must face multiple sources of noise. In these conditions, a good algorithm must be robust and its generalization capability is crucial.

Alain Droniou (PhD candidate in Robotics), Serena Ivaldi (postdoctoral researcher in Robotics), and Olivier Sigaud (Professor in Computer Science) are with: Université Pierre et Marie Curie, Institut des Systèmes Intelligents et de Robotique - CNRS UMR 7222, Pyramide Tour 55 - Boîte Courrier 173, 4 Place Jussieu, 75252 Paris CEDEX 5, France, Contact: firstname.name@isir.upmc.fr

Patrick Stalph (PhD candidate in Computer Science) and Martin Butz (Professor in Computer Science) are with: Universität Tübingen, Sand 14, 72076 Tbingen, Germany

Contact: name@informatik.uni-tuebingen.de

We first describe the learning task and algorithms in section II. In addition to XCSF and LWPR, we also study a third algorithm, iRFRLS, and we highlight its differences with respect to XCSF and LWPR. Then, we present the experimental set-up in section III. In section IV, we compare the performances of the three algorithms and show that iRFRLS converges faster and to a lower error than the other algorithms. We discuss this result in section V, as well as the influence of the parameters of the algorithms.

II. Methods

In this section, we first describe the learning task. Then, we present the three studied algorithms.

A. Learning velocity kinematics

Our approach to learning robot control is to learn the forward velocity kinematics model and then invert it for control. This provides more flexibility than directly learning an inverse model [13].

1) Forward kinematics: The simplest model of a robot is given by its kinematics, i.e. the correspondence between joint space coordinates (articular position of each joint) $\mathbf{q}$ and task space coordinates ξ, usually 3-D coordinates of the end-effector: $\xi = f(\mathbf{q})$.

When the joint space has more dimensions than the task space, the system is redundant and there is no simple method to span the set of solutions at the kinematics level. Therefore, the model is usually derived to obtain the velocity kinematics model through the expression of a Jacobian matrix $J(\mathbf{q}) = \frac{\partial f}{\partial \mathbf{q}}$:

$$\dot{\xi} = J(\mathbf{q})\dot{\mathbf{q}} \tag{1}$$

2) Inverse velocity kinematics: In order to be able to control the robot, we need to inverse (1) using

$$\dot{\mathbf{q}} = J^{\sharp}(\mathbf{q})\dot{\xi} \tag{2}$$

where $J^{\sharp}$ is a pseudo-inverse of J. With (2), we can compute joint velocities corresponding to the desired end-effector velocity.

However, in the redundant and non singular case, there is an infinite number of solutions $\dot{\mathbf{q}}$ providing the desired velocity $\dot{\xi}^{\star}$. Among all these possibilities, the Moore-Penrose pseudo-inverse J^{+} provides the minimum norm solution [4]. To avoid critical effects at singular configurations, regularization terms can be added, like in Damped Least Square Pseudo-inverse (DLS - PINV) [3]. Further information on control in our experiments can be found in [16].

B. Regression algorithms

We focus our study on three regression algorithms: LWPR [19], XCSF [21] and iRFRLS [6]. A survey of these three algorithms can be found in [17].

1) LWPR: The Locally Weighted Projection Regression (LWPR[1]) algorithm [19] is a recursive function approximator, which provides accurate approximation in very large spaces at low computational cost.

It uses a sum of linear models weighted by normalized Gaussians. These Gaussians, also called receptive fields (RFs), define the domain of influence of each corresponding linear model. The RFs and their linear models are both updated incrementally to match the training data. LWPR reduces the input dimensionality using the Partial Least Squares (PLS) algorithm [22]. The global algorithm provides as output $\hat{y}$ the weighted sum of all outputs $\hat{y}_k$ of each RF

$$\hat{y}\left(x\right) = \frac{\sum_{k=1}^{K} w_k \hat{y}_k\left(x\right)}{\sum_{k=1}^{K} w_k} \tag{3}$$

where K is the number of receptive fields.

We refer the reader to [15] or [19] for a further presentation of the incremental version of the algorithm.

2) XCSF: XCSF[2] [21] is another function approximator that shares some similarities with LWPR but comes from Learning Classifier Systems (LCSs) [7]. As any LCS, XCSF manages a population of rules, called *classifiers*. These classifiers contain a *condition part* and a *prediction part*. In XCSF, the condition part defines the domain $\phi_i(z)$ of a local model whereas the prediction part contains the local linear model β_i itself. From classifiers, XCSF predicts a local output vector y_i relative to an input vector x_i. The β_i are updated using the Recursive Least Squares (RLS) algorithm [8], the incremental version of the Least Squares method. The classifiers in XCSF form a population P that clusters the condition space into a set of overlapping prediction models. XCSF uses only a subset of the classifiers to generate an approximation. Indeed, at each learning iteration, XCSF generates a match set M that contains all classifiers in the population P whose condition part $\mathcal{Z}$ matches the input data z i.e., for which $\phi_i(z)$ is above a threshold ϕ_0.

In XCSF, the output $\hat{y}$ is given for a (x, z) pair as the sum of the linear models $\hat{y}_i$ of each matching classifier i weighted by its fitness F_i

$$\hat{y}\left(x, z\right) = \frac{\sum_{k=1}^{n_M} F_k\left(z\right) \hat{y}_k\left(x\right)}{\sum_{k=1}^{n_M} F_k\left(z\right)} \tag{4}$$

where n_M is the number of classifiers in the match set M. In all other respects, the genetic algorithm (GA) that drives the evolution of the population of classifiers is directly inherited from XCS [20].

An important process in the context of this study is *compaction*. At the end of a learning process, the final population is composed of highly overlapping classifiers. To reduce the size of the population, XCSF uses a Closest Classifier Matching (CCM) rule [2] to get a fixed size match set M.

[1]We use the *liblwpr* C/C++ library available on http://www.ipab.inf.ed.ac.uk/slmc/software/lwpr/index.html.

[2]We use the Java *XCSFServer* developed by M. Butz and P. Stalph [18].

3) iRFRLS: iRFRLS[3] is based on a regularized least square method with random features [6]. In this paper, we present this algorithm from a different point of view, based on Fourier transform.

Almost any usual function can be calculated by inversion of its Fourier transform. Approximating the Fourier transform of a function instead of the function itself allows to exploit some regularity properties to make the learning process more robust and simpler.

Since we work on real, continuous functions defined on finite intervals (and so virtually periodic), the partial sums of the Fourier series of f,

$$\begin{aligned} S_n(f(x)) = \frac{a_0(f)}{2} &+ \sum_{k=1}^{n} a_k(f) \cos\left(kx\frac{2\pi}{T}\right) \\ &+ \sum_{k=1}^{n} b_k(f) \sin\left(kx\frac{2\pi}{T}\right) \end{aligned} \tag{5}$$

converge towards f when $n \to \infty$, with

$$\begin{aligned} a_n(f) &= \frac{2}{T}\int_{-T/2}^{T/2} f(t)\cos\left(nt\frac{2\pi}{T}\right)\,\mathrm{d}t \\ b_n(f) &= \frac{2}{T}\int_{-T/2}^{T/2} f(t)\sin\left(nt\frac{2\pi}{T}\right)\,\mathrm{d}t \quad \forall n \geq 0 \end{aligned} \tag{6}$$

To approximate the function, we can set some (actually most of) coefficients to 0, and learn the others. The quality of predictions depends in that case on the number D of learned coefficients (features). Writting $x_1, ..., x_k$ the points from the training set and

$$\begin{aligned} f &= \left[f(x_1), ..., f(x_k)\right],\ w = \left[a_{i_0}, b_{i_0}, ..., a_{i_D}, b_{i_D}\right]^T \\ z(x) &= \left[cos(\omega_{i_0}x), sin(\omega_{i_0}x), ..., cos(\omega_{i_D}x), sin(\omega_{i_D}x)\right]^T \\ Z &= \left[z(x_1), ..., z(x_k)\right]^T \end{aligned}$$

the prediction can be written $\hat{f}(x) = w^T z(x)$ and a classical single vector regression method [5] can be used to learn the coefficients. The solution is then given by

$$w = (\lambda I + Z^T Z)^{-1} Z^T f \tag{7}$$

where λ is a regularization factor. This solution w can be computed incrementally and easily inverted with a Cholesky decomposition of $(\lambda I + Z^T Z)$ [1], [14], [6].

Finally, iRFRLS requires to tune only three parameters: the regularization factor λ, the number of features D and the distribution of pulsations ω. Because of a strong relationship with Gaussian kernel methods, one usually chooses normal distributions whose only parameter γ is the variance. In these conditions, iRFRLS behaves as a Gaussian kernel regression algorithm. Features $z(x)$ then approximate Gaussian kernel values [11].

[3]We use the C++ library developed by A. Gijsberts, which can be found in the official iCub repository https://robotcub.svn.sourceforge.net/svnroot/robotcub/trunk/iCub.

III. EXPERIMENTS

We use the 104cm high and 53 degrees of freedom humanoid robot iCub [9]. In [16], algorithms were tested on an iCub simulator. For this study, we use the real robot and hence face real conditions of use. As a consequence, we must address multiple sources of noise: imprecision of sensors, power supply noise, mechanical vibrations and backlashes, inaccurate vision algorithms, imperfect calibration or non-simultaneous measures.

A. Data

We tune and evaluate the algorithms on data recorded according to the setup of [16]. For those experiments, iCub is performing an asterisk reaching task. As we do not want to address vision process difficulties, the end-effector is represented by a green ball at the top of a stick held by the robot (see [16] for further details). As a results, the kinematics model is unknown. The task involves 6 degrees of freedom: 2 for the head and 4 for the arm. During these experiments, the explored region consists of a cube of about 10 cm side length. Compared with the 40 cm of the iCub arm, it represents roughly 20% of the useful reachable space[4]. This can bias some results because this emphasizes some linearities.

We recorded 4 million points $(\mathbf{q}, \dot{\mathbf{q}}, \xi, \dot{\xi})$ corresponding to 30 experiments, sampled at approximately 20Hz.

B. Tuning the algorithms

One of the main points when working with learning methods is the tuning of each algorithm. In order to perform a fair comparison, we use a grid-search method: for each parameter, a set of values is chosen (Tables I, II, III), and we test all possible combinations of values.

To get statistically significant results using paired Student's t-tests [12], each experiment is repeated on the 30 recorded datasets. These datasets are the same for the three algorithms.

We measure the performance as the mean squared error in the prediction $\hat{\dot{\xi}}$ of $\dot{\xi}$: $MSE = \frac{1}{3}\mathbb{E}\left[||\hat{\dot{\xi}} - \dot{\xi}||^2\right]$ where coefficient $\frac{1}{3}$ comes from the dimension of $\dot{\xi}$. Test sets consist of 1000 points, different from learning sets.

Thanks to the distinction between condition and prediction spaces, XCSF can directly learn the velocity kinematics Jacobian matrix. On the other hand, LWPR and iRFRLS cannot provide directly this matrix. We use them to learn the kinematics model, from which we can compute the velocity kinematics Jacobian matrix by derivation.

IV. RESULTS

In this section, we carry out experiments to study both the influence of each parameter and the influence of the data order on the algorithms. We focus on the convergence of the algorithms and their computation time.

[4]One might approximate the reachable space as a half sphere of radius 40 cm. This would lead to a ratio explored/reachable of 0.7%. Actually, due to mechanical constraints and joints limits, the reachable space is much smaller.

A. Influence of the parameters

In order to study the influence of parameters of each algorithm, we compute the average error obtained on all simulations by setting the value of one parameter at a time. The 28 XCSF parameters cannot be exhaustively tested. We select six of them, identical to those tested in [16]. The results are presented in Table I. We do the same for LWPR, in Table II. Since iRFRLS has only three parameters, it is possible to perform an exhaustive search (Table III).

TABLE I
INFLUENCE OF EACH PARAMETER FOR XCSF. THE NOTATION $v_1 \overset{c}{>} v_2$ MEANS THAT THE VALUE v_1 OF THE PARAMETER LEADS TO BETTER PERFORMANCE THAN THE VALUE v_2, WITH A CONFIDENCE OF $c\%$. VALUES IN BOLD (ITALIC) CORRESPOND TO THE BEST (WORST) SET OF PARAMETERS. GRID-SEARCH TIME CORRESPONDS TO THE TIME NEEDED TO TEST EVERY SET OF PARAMETERS ON 30 DATASETS, WITHOUT PARALLELIZATION.

XCSF		
Parameter	Ranking	Time
Δ	$0.01 \overset{76}{>} 0.05 \overset{99.2}{>} \mathbf{0.1} \overset{84}{>} 0.5$	constant
converConditionRange	$\mathbf{0.995} \overset{99.9}{>} 0.7$	constant
minConditionStretch	$0.001 \overset{98}{\approx} \mathbf{0.005} \overset{81}{\approx} 0.1$	constant
ϵ_0	$0.01 \overset{98}{>} \mathbf{0.005} \overset{99.4}{>} 0.001$	constant
maxPopulationSize	$\mathbf{1500} \overset{90}{>} 500 \overset{100}{>} 100$	increasing
startCondensation	$\mathbf{500} \overset{67}{\approx} 1000 \overset{99.9}{>} 2000 \overset{99.9}{>} 5000$	increasing
Grid-search time		9 days

TABLE II
INFLUENCE OF EACH PARAMETER FOR LWPR.

LWPR		
Parameter	Ranking	Time
useMeta	$\mathbf{true} \overset{89}{\approx}$ false	constant
updateD	false $\overset{99.7}{>}$ ***true***	*true* longer
penalty	$\mathbf{0.001} \overset{67}{>} 0.01 \overset{93}{\approx} 0.1$	constant
wGen	$0.9 \overset{92}{\approx} \mathbf{0.2} \overset{64}{>} 0.1 \overset{93}{>} 0.01$	increasing
setInitAlpha	$\mathbf{0.01} \overset{58}{>} 0.1 \overset{67}{>} 200 \overset{98}{\approx} 500$	constant
setInitD	$\mathbf{50} \overset{75}{>} 40 \overset{66}{>} 30 \overset{84}{>} 20$	increasing
Grid-search time		11 hours

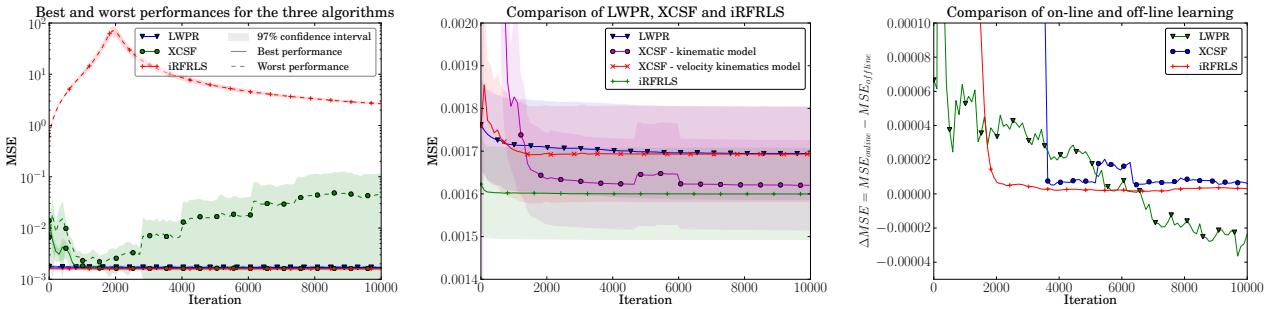
TABLE III
INFLUENCE OF EACH PARAMETER FOR IRFRLS.

iRFRLS		
Parameter	Ranking	Time
γ	$\mathbf{10^{-6}} \overset{99.9}{>} 10^{-12} \overset{96}{\approx} 10^{-3} \overset{100}{>} 1 \overset{100}{>} 10$	constant
λ	$10 \overset{99.9}{>} 1 \overset{99.9}{>} 10^{-3} \overset{97}{\approx} 10^{-6} \overset{96}{\approx} \mathbf{10^{-12}}$	constant
D	$50 \overset{100}{>} 500 \overset{99.9}{>} \mathbf{1000} \overset{99.9}{>} 2000$	$\mathcal{O}(D^2)$
Grid-search time		12 days

Plots in Fig. 1(a) show the worst and best performances obtained by each of the three algorithms.

B. Comparison of learning algorithms

In [16], parameters were optimized for learning a velocity kinematics model for which data were generated randomly



(a) Error plots obtained with the best and the worst set of parameters for LWPR, XCSF and iRFRLS. Note logarithmic scale.

(b) Comparison of the best sets of parameters for each algorithm. XCSF was tested on both kinematics and velocity kinematics models.

(c) Comparison between on-line and off-line learning. We represent the difference $\Delta MSE = MSE_{online} - MSE_{offline}$.

Fig. 1. Empirical study of the behavior of LWPR, XCSF and iRFRLS

(no temporal relationship between them, excluding any notion of trajectory). Therefore, we start with a similar approach, before testing the algorithms when the data is ordered along the trajectories of the robot.

1) Offline Learning: Initially, training and testing sets are generated by drawing random points from the database. Fig. 1(b) compares the performance of the three algorithms with the best sets of parameters.

First, we note that the final performance of XCSF is better when learning the velocity kinematics Jacobian matrix than when learning the kinematics model (with a confidence higher than 99.9%). In order to compare the best performance of the algorithms, we evaluate XCSF on learning the velocity kinematics model.

The convergence is almost instantaneous for LWPR and iRFRLS (models are correctly learned in less than 500 iterations). However, the convergence of XCSF is much longer. That confirms a result on simulated data in [16]: XCSF converges slower than LWPR, but the final performance is better (confidence of 98.8%). Similarly, iRFRLS converges as fast as LWPR with a better final performance (confidence higher than 99.9%). But if it converges faster than XCSF, the difference in final performance is significant up to only 84%.

2) Online Learning: We now use sets of consecutive points instead of randomly selected ones. We thus confront the algorithms in real conditions, where learning is constrained by the movements of the robot. Fig. 1(c) represents the difference between the error obtained with off-line and on-line training $\Delta MSE = MSE_{online} - MSE_{offline}$. Though the convergence is slower, the final performance is similar for iRFRLS and XCSF, while it is better for LWPR. However, the final ranking is identical to the one for off-line training: iRFRLS is always the fastest to converge and the best in terms of final performance.

V. Discussion

From the results presented in the previous section, iRFRLS appears superior in all respects. However, some points require further analysis.

A. Performance

The results in section IV-B induce two discussions: one on the interest of optimization, the other on the influence of the data order. Table IV provides a synthesis of these discussions.

1) Advantages of optimization: If the impact of parameter optimization is tangible with the three algorithms (Fig. 1(a)), its interest is limited for LWPR: neither the final performance (gain less than 2 %) nor the speed of convergence are impacted significantly. However, this optimization is the fastest of the three algorithms. In contrast, optimization is essential for iRFRLS (gain by a factor of 1000) as well as XCSF for which final performance, speed of convergence and stability are impacted.

2) Data order: Obviously, the convergence speed of the algorithms depends on the time needed to cover the space with enough training data. Hence, the convergence is slower when data order follows the trajectory, compared to a random order. Studying the impact on the final performance is more relevant.

Theory ensures that the final performance of iRFRLS does not depend on the data order, except for rounding errors that may accumulate around the calculation of the incremental Cholesky factorization. Similarly, the GA in XCSF reduces the dependence of the algorithm on the data order. Fig. 1(c) indeed shows a fast convergence of iRFRLS to the same level of error when learning off-line. XCSF is slower to converge, but also reaches the same level of error.

Unlike XCSF, LWPR does not have any mechanism to optimize the position of the RFs centers. Since the data order determines the RFs creation order and hence their positions, it changes the final performance, as shown in Fig. 1(c). However, contrary to expectations, it must be emphasized that the final performance of LWPR is better when data are presented along trajectories. This can be because it is easier to identify and properly filter noise when data correspond to consecutive points of the trajectory, building RF overlapping along these trajectories and whose combination gives a more reliable prediction. However, if the signal to learn is not as noisy, this mechanism may work against LWPR by smoothing inner variations of the estimated function. This remains to be studied.

TABLE IV
SYNTHESIS OF THE PERFORMANCE COMPARISON OF LWPR, XCSF AND IRFRLS.

Algorithm	LWPR	XCSF	iRFRLS
Convergence	Fast	Slow	Fast and unstable
Final performance	-	+	++
Sensitivity to tuning	Low	High	Extreme
Computation time	Fast	Variable, sometimes slow	Accuracy / time compromise
Main advantages	Fast and stable	Distinction condition / prediction	Constant cost, few parameters and good performances
Main drawbacks	Many parameters to optimize	Sometimes unstable and slow convergence, many parameters	Slow for large values of D, sensitive to tuning

B. Comparative study

In this section, we analyse the influence of the parameters with respect to the specificities of our experimental set-up, in order to highlight some rules which simplify tuning.

1) Noise and regularity of estimated functions: In our study, the function to learn is very regular, with strong linearities. Therefore this impacts the tuning of parameters and explains the fast convergence of the algorithms when learning the kinematics model. Because of its higher dimensionality, learning the Jacobian matrix with XCSF is slower (Fig. 1(b)).

However, learning the kinematics model to compute the velocity kinematics model increases the error variance (since it uses twice the model to calculate the velocity), which can justify the lower final performance of XCSF with the kinematics model compared to the velocity kinematics Jacobian.

Besides the regularity of the function to learn, data are very noisy. Hence, learning regular models is even more important to counter over-fitting. This favor larger areas of validity for XCSF classifiers, which is reflected in the value of *coverConditionrange* which hides *minConditionStretch* values. Similarly, a higher value for ϵ_0 smooths predictions, taking into account more classifiers for each prediction. To avoid over-fitting, the tuning of the GA is crucial: one must avoid to specialize the population on noise, and favor a high turnover. This is achieved through *startCondensation* and Δ parameters. Working in Fourier space, iRFRLS filters noise more efficiently, which may explain its superiority. Note that a choice of a too high γ for iRFRLS, i.e. an over-representation of high frequencies, degrades the performance.

On the contrary, LWPR seems to favor smaller RFs, through the high value of *setInitD*. This seems contradictory to the previous analysis, but unlike XCSF, LWPR does not optimize the center of its RFs, and smaller RFs then allow to achieve better accuracy and adaptivity thanks to a finer partitioning of the space.

2) Models complexity: LWPR and XCSF experiments highlight the existence of a minimum population size for both algorithms. For XCSF, it is between 100 and 500 classifiers, as shown by the large difference in performance for these two values of *maxPopulationSize*, while the gain is much more limited from 500 to 1500 classifiers. For LWPR, the effect is noticeable on *wGen*, the threshold that determines the creation of a new RF. The gain is much more important between 0.01 and 0.1, than with further increases.

The study of the features number D of iRFRLS is much more surprising at first sight. Indeed, this number is directly related to the quality of the Fourier transform approximation and a high value should always be preferred. But the study shows an average performance decreasing with the number of features. Because of the strong linearities of the kinematics model, a small number of frequencies is sufficient (and preferable, if one wants to avoid learning the discontinuities due to noise). In addition, the number of coefficients to be learned corresponds to the number of features and a small number of features thus facilitates learning. This explains why on average, with random values for the other parameters, it is easier to have better performances with only 50 features. Nevertheless, though the better performance on average is obtained with 50 features, the best performing model was obtained with 1000 features.

3) Computation Time:

C. Computation Time

In the robotics context, computation time is an important feature for learning algorithms, allowing or not real-time applications. Table V compares training and prediction time for each of the three algorithms[5].

TABLE V
MEAN (MAX) COMPUTATION TIME PER SAMPLE IN TRAINING AND PREDICTION.

Algorithm \ Time	Training	Prediction
LWPR	50μs (1ms)	50μs (1ms)
XCSF	300 μs (20ms)	300μs (20ms)
iRFRLS, D=50	20μs (25μs)	20μs (25μs)
iRFRLS, D=2000	50ms (50ms)	450μs (500μs)

LWPR is the fastest of the three algorithms, making it easily usable in real-time applications. The influence of the parameters (Table II) is thus not crucial.

For XCSF, the computation time depends mainly on the population size and on the starting time of condensation (table I). However, the latter effect is noticeable only in the learning phase: once the population is stable after condensation, the prediction time becomes constant. On average, training and prediction time are around 300μs, with a maximum of 20ms typically achieved just before condensation.

The computation time of iRFRLS depends on only the number D of features (Table III). This allows to easily search

[5]Experiments were carried out on an Intel Core i7 960 processor (4 cores, hyper-threading, 3.2GHz) with 6Gb RAM.

for a compromise between computation time and accuracy. One reaches 50 ms in learning process with $D = 2000$. Such computation time may require off-line training or significant computing resources for real-time applications.

D. Tuning cookbook

Through this discussion, we identified some rules thanks to which it is easier to tune the algorithms depending on the learning problem. They are presented on Table VI. The table must be read as follows: to deal with differences in the function regularity, for LWPR the best parameters to tune are those controlling RFs size and overlapping. To prevent over-fitting with LWPR, one should tune parameters controlling RFs optimization and meta-learning, etc.

TABLE VI
AN EASIER TUNING OF THE ALGORITHMS CAN BE ACHIEVED BY IDENTIFYING SOME RECURRENT ISSUES IN LEARNING PROBLEMS AND THEIR RELATED PARAMETERS.

Algorithm	Function regularity	Over-fitting	Time complexity
LWPR	RFs size and overlapping	RFs optimization and meta-learning	Always fast
XCSF	Classifiers size and number	Condensation and GA	Population size
iRFRLS	Sampling distribution		Number of features

VI. CONCLUSION AND PERSPECTIVES

In this work, we have learned models from iCub and shown that LWPR, XCSF and iRFRLS perform well in the presence of large noise sources. iRFRLS outperforms XCSF and LWPR, in terms of speed of convergence and performance. However, computation time can become prohibitive for iRFRLS for real-time applications. Nevertheless, it remains possible to ensure real-time computation at design level, thanks to a computation time / accuracy trade-off, by a relevant tuning of the number of features. We also highlighted that the optimization of the parameters is crucial for iRFRLS and XCSF, while the gain in performance is much lower for LWPR.

In addition, if on-line learning is slower, which is mainly due to the time required to cover the whole space, the impact on the final performance is minor for the three algorithms.

In the future, we want to validate our results on-line on iCub. This will provide two improvements to this study: it will be possible to test the algorithms on larger spaces and performances will be evaluated not only on the mean squared error of predictions, but also on the quality of the obtained trajectories. On a longer term, we also want to study the effects of the introduction of artificial curiosity mechanisms on the algorithms and learn the dynamics of the robot.

ACKNOWLEDGMENTS

This work is supported by the French ANR program (ANR 2010 BLAN 0216 01), more at http://macsi.isir.upmc.fr

REFERENCES

[1] Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, 1996.
[2] Martin V. Butz and Oliver Herbort. Context-dependent predictions and cognitive arm control with XCSF. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, pages 1357–1364, New York, NY, USA, 2008. ACM.
[3] Arati Deo and Ian Walker. Overview of damped least-squares methods for inverse kinematics of robot manipulators. *Journal of Intelligent and Robotic Systems*, 14:43–68, 1995. 10.1007/BF01254007.
[4] K. L. Doty, C. Melchiorri, and C. Bonivento. A theory of generalized inverses applied to robotics. *Int. J. Robotics Research*, 12(1), 1993.
[5] Harris Drucker, Chris Kaufman, Burges L., Alex Smola, and Vladimir Vapnik. Support vector regression machines. In *Advances in Neural Information Processing Systems 9*, volume 9, pages 155–161, 1997.
[6] Arjan Gijsberts and Giorgio Metta. Incremental learning of robot dynamics using random features. In *ICRA*, pages 951–956, 2011.
[7] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, April 1992.
[8] T. Kailath, A.H. Sayed, and B. Hassibi. *Linear estimation*. Prentice-Hall information and system sciences series. Prentice Hall, 2000.
[9] Giorgio Metta, Giulio Sandini, David Vernon, Lorenzo Natale, and Francesco Nori. The iCub humanoid robot: an open platform for research in embodied cognition. In *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, PerMIS '08, pages 50–56, New York, NY, USA, 2008. ACM.
[10] Ugo Pattacini, Francesco Nori, Lorenzo Natale, Giorgio Metta, and Giulio Sandini. An experimental evaluation of a novel minimum-jerk cartesian controller for humanoid robots. In *IROS*, pages 1668–1674. IEEE, 2010.
[11] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1177–1184. MIT Press, Cambridge, MA, 2008.
[12] Graeme D. Ruxton. The unequal variance T-test is an underused alternative to Student's T-test and the Mann-Whitney U test. *Behavioral Ecology*, 17(4):688–690, 2006.
[13] C. Salaun, V. Padois, and O. Sigaud. Control of redundant robots using learned models: an operational space control approach. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 878–885, Saint-Louis, USA, oct 2009. doi:10.1109/IROS.2009.5354438.
[14] Ali H. Sayed. *Adaptive Filters*. Wiley-IEEE Press, 2008.
[15] Stefan Schaal, Christopher G. Atkeson, and Sethu Vijayakumar. Scalable techniques from nonparametric statistics for real time robot learning. *Applied Intelligence*, 17:49–60, June 2002.
[16] G. Sicard, C. Salaun, S. Ivaldi, V. Padois, and O. Sigaud. Learning the velocity kinematics of iCub for model-based control: XCSF versus LWPR. In *Proceedings of the 11th IEEE-RAS International Conference on Humanoid Robots - Humanoids 2011*, Bled, Slovenia, 2011.
[17] Olivier Sigaud, Camille Salaün, and Vincent Padois. On-line regression algorithms for learning mechanical models of robots: a survey. *Robotics and Autonomous Systems*, 59:1115–1129, July 2011.
[18] Patrick O. Stalph and Martin V. Butz. Current XCSF capabilities and challenges. In *IWLCS 2008/2009*, LNCS (LNAI), vol. 6471, pages 57–69. Springer, 2010.
[19] Sethu Vijayakumar and Stefan Schaal. Locally weighted projection regression: An o(n) algorithm for incremental real time learning in high dimensional space. In *in Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000*, pages 1079–1086, 2000.
[20] S. W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.
[21] Stewart W. Wilson. Function approximation with a classifier system. *Genetic and Evolutionary Computation Conference, GECCO 2001*, pages 974–981, 2001.
[22] H. Wold. Soft Modeling by Latent Variables; the Nonlinear Iterative Partial Least Squares Approach. *Perspectives in Probability and Statistics. Papers in Honour of M. S. Bartlett*, 1975.