2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Open and Closed-Loop Task Space Trajectory Control of Redundant Robots Using Learned Models

Bruno Damas Lorenzo Jamone José Santos-Victor

Abstract— **This paper presents a comparison of open-loop and closed-loop control strategies for tracking a task space trajectory, using redundant robots. We do not assume any knowledge of the analytical forward and inverse kinematics, relying instead on learning these models online, while executing a desired task. Specifically, we employ a recent learning algorithm that allows to learn a probabilistic model from which both the forward and inverse solutions can be obtained, as well as the Jacobian of the kinematics map. Such learned model can then be used to implement both types of control. Moreover, the multi-valued solutions provided by the learned model can be applied to redundant systems in which an infinite number of inverse solutions may exist. We present experiments with a simulated version of the iCub, a highly redundant humanoid robot, in which this learned model is employed to execute both open-loop and closed-loop trajectory control. We show the advantages and drawbacks of both control strategies, and we propose a way to combine them to deal with sensor noise and failures, showing the benefits of using a learning algorithm that can simultaneously provide forward and inverse predictions.**

I. Introduction and Related Work

In this work we consider the general problem of trajectory tracking for redundant robots based on learned models: for a given sequence of desired task space points $\boldsymbol{x}_d^i$, together with a corresponding sequence of task space velocities $\dot{\boldsymbol{x}}_d^i$, we want to control the joints of the robot in such a way that the end-effector will visit each of the desired task space points in sequence, reaching its vicinity with the desired velocity. Moreover, we want to follow the trajectory without any knowledge of the analytical structure of the robot kinematics, relying only on models learned using data collected from the robot movement. The need for learned models arise from several different situations: while accurate models for inverse dynamics may be difficult to obtain due to a lack of knowledge of certain hard to measure physical parameters and unmodeled non-linear interactions, the kinematics analytical model may have inaccuracies due to backlash, actuator unmodeled nonlinearities, time drifts in the kinematics characteristics, produced by material wear, and, for humanoid robots, calibration errors in cameras used for end-effector tracking.

Given a learned model of a robot kinematics, trajectory control methods fall broadly into two major categories: open-loop controllers try to accomplish the desired trajectory following task relying solely on the learned model, while closed-loop methods also use sensory data to perform the trajectory, many times in the form of the current end-effector position feedback.

Resolved Motion Rate Control (RMRC) is a widely used class of closed-loop trajectory control methods, that has seen many adaptations and variants. In its original form, it uses the inversion of the kinematics Jacobian, that relates joint to task velocities, to obtain the joint velocities that will drive the end-effector in the target direction. Ideally, when a perfect kinematic model is available, straight line trajectories in the operational space are obtained when using RMRC schemes [1]. When using learned models the analytical Jacobian is replaced by its estimate, as calculated using the learning algorithm; some recent works [2], [3], [4] can successfully track desired task space points using the Locally Weighted Projection Regression (LWPR) algorithm [5] to predict the Jacobian during online operation.

Open-loop controllers rely on obtaining accurate solutions for the inverse kinematics problem. Learning inverse kinematics, however, is by far a much more demanding learning problem than Jacobian learning, due to the multi-valued nature of the inverse map, where many joints values typically map to the same end-effector position. The inverse kinematics map learning scheme of [6], for instance, spatially localizes the learning task, but this is ineffective for trajectory planning, where a global set of inverse solutions should be available. Another approach takes unsupervised learning schemes and uses the learned density of the joint input-output space to obtain the inverse kinematics map, by conditioning such density on the output query points [7], [8]. Such unsupervised learning methods, however, typically result in convergence to sub-optimal solutions that do not take the problem structure into account, as they ignore that the full data, apart from noise corruption, lies in a lower dimensional manifold. This is particularly problematic as the kinematics map dimension increases. Finally, another work, related to the approach proposed in this paper, uses a Mixture Density Network (MDN) [9] to estimate the robot inverse kinematics map [10]. This latter work, however, relies on an offline training of the model, where the number of components of the mixture must be defined beforehand. Additionally, the learned model cannot be used for Jacobian or forward prediction.

Closed-loop methods are known for their robustness towards external perturbations and model uncertainties: RMRC, for instance, works reasonably well even when the Jacobian estimate is not accurate, as long as the actuated joint velocities, calculated using the estimated Jacobian,

Bruno Damas, Lorenzo Jamone and José Santos-Victor are with the Instituto de Sistemas e Robótica, Instituto Superior Técnico, Lisboa, Portugal. Bruno Damas is also with Escola Superior de Tecnologia de Setúbal, Portugal. {bdamas, jasv}@isr.ist.utl.pt. This work was partially funded by EU Project POETICON++ (FP7-ICT-288382) and FCT project [PEst-OE/EEI/LA0009/2011].

move the end-effector closer to the desired task space point. However, since such methods need sensory feedback, they suffer, in their original form, from excessive sensor noise levels, many times resulting in robot jerky motions. Also, they are condemned to failure whenever signals from the sensors cease to be available, for instance when the end-effector is occluded in a vision-based tracking system. Open-loop controllers prove to be insensitive to these problems, as they do not use such sensory information for control. Still, they are not robust as closed-loop controllers with respect to unexpected movement perturbations and inaccurate learned models.

In this paper we propose the use of the Infinite Mixture of Linear Experts algorithm (IMLE), a recent online learning algorithm [11], [12], to provide a learned model that can be simultaneously used for open and closed-loop control. While several learning algorithms were used for both of these control schemes, to our knowledge IMLE has the unique feature of being able to provide single and multi-valued forward, inverse and Jacobian estimates from the same learned model. In closed-loop mode, IMLE provides Jacobian estimates that are used to control the robot joints, using a RMRC scheme. If an open-loop control is desired, the multiple solutions resulting from the inverse prediction queries are used to plan a trajectory in the joint space, as described with more detail in Section IV. This results in a general, learning based control scheme that can be switched from closed to open-loop at any time, either due to a high noise level in the sensors, occlusions in visual feedback or simple sensor failure. In fact, IMLE can be used to compare sensor readings to their expected values, as predicted from its forward model, in this way enabling an automatic switch to open-loop mode whenever there is a large discrepancy in these values.

We first introduce the IMLE algorithm in Section II, describing its most important features. After that, we explain how this machine learning method is used for closed and open-loop control, in Sections III and IV respectively. After that, we proceed to several experiments comparing both approaches in Section V. Section VI concludes the paper, providing the final remarks.

II. The IMLE Model

The IMLE algorithm is an online probabilistic algorithm that uses a generalized expectation-maximization (EM) procedure to update its parameters, fitting an infinite mixture of linear experts to an online stream of training data [11], [12]. Its probabilistic model assumes the training data to be described by a collection of linear models: a training point consisting of an input part $z_i \in \mathbb{R}^d$ and a corresponding output $x_i \in \mathbb{R}^D$ have its probability described by the generative model

$$p(x_i|z_i, w_{ij}; \Theta) \sim \mathcal{N}\left(\mu_j + \Lambda_j(z_i - \nu_j), \Psi_j\right) \ , \quad (1)$$

$$p(z_i|w_{ij}; \Theta) \sim \mathcal{N}\left(\nu_j, \Sigma_j\right) \ , \quad (2)$$

where the mean ν_j and covariance Σ_j define a Gaussian input region for each expert j. Parameter μ_j is the output mean for each expert, while Λ_j defines the linear relation from input to output. Ψ_j is a diagonal matrix that represents the uncorrelated noise at each of the output dimensions. The unobserved, latent indicator variable w_{ij} assigns training points to experts, while the parameter vector Θ gathers all the parameters to be learned. A set of priors for each of the mixture parameters is also defined, in order to perform some regularization and to enforce the principle of localized learning, thus avoiding the interference of experts across different regions of the input space.

Online training of the model is done using an online EM algorithm: in the expectation step (E-Step) responsibilities are assigned to experts for a new point (z_i, x_i), according to:

$$h_{ij} \equiv E[w_{ij}|x_i, z_i; \hat{\Theta}] = p(w_{ij}|x_i, z_i; \hat{\Theta}) \ , \quad (3)$$

where $\hat{\Theta}$ is the most recent estimate for the mixture parameters being learned. Maximization step (M-Step) then updates the parameters in $\hat{\Theta}$ according to the responsibilities h_{ij} previously obtained. Based on a model for outlier points, the mixture can grow by automatically adding new experts whenever the perceived data points are not well explained by the mixture. This and other aspects of the IMLE algorithm are thoroughly detailed in [11]: for details, please refer to the original paper.

Given a current set of mixture parameters, a conditional mean prediction method will typically mixture the individual linear models predictions according to some weighted average scheme, using weights $w_j^x(z_q)$ that depend on how strong each model is activated given only the input query point z_q. This works well for single-valued regression — and this is the approach followed by IMLE single-valued forward prediction mode — but for multi-valued regression, as typically required for prediction of inverse kinematics maps, this is unacceptable, as different solutions are then merged together.

Given an output query x_q, each of the linear models that constitute the current mixture can assign an inverse point estimate, together with an uncertainty, by conditioning the full input-output probability density $p(x, z|w_j)$ to the query x_q. For multi-valued inverse prediction, the IMLE algorithm tries to find, for such query x_q, a set of estimated predictions $\hat{\bar{z}}_k$, by grouping and clustering the linear models point estimates into a minimal set of predictions. It uses a probabilistic model that relates linear models point predictions $\hat{z}_j$ to the unknown set of true multi-valued predictions $\bar{z}_k$, also taking into account the weights $w_j^z(x_q)$ and the estimation variances R_j provided by each linear model. This process then has to group linear models point estimates into a set of N_{pred} coherent predictions, choosing the appropriate number of such predictions during the estimation process. The clustering problem is solved using another EM procedure, by assuming some latent variables s_{jk} exist that assign models point estimates $\hat{z}_j$ to the unknown multi-valued predictions $\bar{z}_k$. After the EM procedure is carried through, a statistical hypothesis test is performed, with a given significance level α_{multi}, to assess the fit of the resulting set of multi-valued

predictions: if the test rejects the goodness of fit hypothesis then it is assumed that the number of predictions N_{pred} is insufficient. For a query point x_q the IMLE algorithm starts with the single-valued prediction: if the test finds evidence to reject the hypothesis that the models point estimates are distributed according to a single-valued prediction, the value of N_{pred} is increased to 2 and the EM clustering procedure is carried on; if the goodness of fit hypothesis is again rejected the number of predictions N_{pred} is again increased, until the test fails to reject the hypothesis and a final set of N_{pred} multi-valued predictions is obtained. Note that this clustering process automatically deals with predictions coming from non-relevant experts with low weights $w_j^z(x_q)$, without the need to filter such spurious predictions.

If the input space dimension d is greater than the output dimension D, as is the case with redundant robots, the set of solutions that the IMLE algorithm provides for a inverse prediction query can be interpreted as a kind of sampling of the full continuous solution space. Depending on the value of α_{multi}, this sampling can be more or less coarse: in the limit, IMLE can provide a set of M inverse solutions, where M is the current number of linear models of the mixture, or, in the opposite direction, a single global solution, merging all the experts inverse predictions together, can be found. However, we empirically found that the parameter α_{multi} does not influence much the trajectories obtained using an open-loop controller based on the inverse kinematics prediction, as described in the following sections, as long as its value is not high enough to only produce a single-valued estimate.

The IMLE algorithm features a very low computational complexity: for every new training point presented the learning algorithm is $\mathcal{O}(Md(d+D))$, *i.e.*, linear in the number of active experts M and output dimensions D and quadratic in the number of input dimensions d, thus making it directly comparable to current state-of-the-art online learning algorithms in terms of computational complexity per training point. For inverse prediction, IMLE computational time also grows linearly with N_{pred}, the number of multi-valued solutions found for each query.

Finally, note that IMLE also provides Jacobian estimates at a given query point z_q, by calculating the derivatives of the forward solution(s) with respect to the input vector. This makes the IMLE algorithm one of the most versatile state of the art online learning algorithms, that can obtain forward, inverse and Jacobian predictions from the same learned model.

III. Closed Loop Position Tracking

To control the end-effector position in task space we follow the approach originally proposed in [1], a RMRC scheme where, due to the redundancy of the robot kinematics, the null space of the main task can be used to keep the joints values as far as possible from their physical limits. Motor velocity commands $\dot{\boldsymbol{q}}_{act}(t)$ are thus computed as follows:

$$\dot{\boldsymbol{q}}_{act}(t) = \boldsymbol{K}_m \boldsymbol{J}^{\dagger}(\boldsymbol{x}_d - \boldsymbol{x}(t)) - \boldsymbol{K}_s(\boldsymbol{I} - \boldsymbol{J}^{\dagger}\boldsymbol{J})\nabla M(\boldsymbol{q}(t)) \;, \tag{4}$$

where the Jacobian matrix $\boldsymbol{J}$, evaluated at the current joints value $\boldsymbol{q}(t)$, maps from motor velocities to task velocities, $\boldsymbol{J}^{\dagger}$ is its Moore-Penrose pseudo-inverse, $(\boldsymbol{I} - \boldsymbol{J}^{\dagger}\boldsymbol{J})$ is the null-space projector and $\nabla M(\boldsymbol{q}(t))$ is the gradient of the cost function that penalizes joints values closed to their limits. $\boldsymbol{K}_m$ and $\boldsymbol{K}_s$ are gains, represented by positive diagonal matrices. When the system is close to singularities we introduce a regularization term in the pseudo-inverse, similar to the damped least squares solution of [13], to avoid numerical instabilities. The Jacobian $\boldsymbol{J}(\boldsymbol{q}(t))$ is obtained, at each time step, from the current IMLE model, taking the estimate of the local slope, for input query point $\boldsymbol{q}(t)$, of the learned map from joint to task space at this point.

Following a trajectory in the task space, represented by a sequence of task space points $\boldsymbol{x}_d^i$, can be accomplished, using this closed-loop controller, by simply feeding the desired task space points in sequence to the controller, switching to the next available point whenever the current point $\boldsymbol{x}_d$ is reached. If additionally we wish to define the desired velocities $\dot{\boldsymbol{x}}_d^i$ at the trajectory points, Eq. 4 can be easily modified to accommodate these velocities, changing for instance its first term to $\boldsymbol{J}^{\dagger}(\dot{\boldsymbol{x}}_d + \boldsymbol{K}_m(\boldsymbol{x}_d - \boldsymbol{x}(t)))$.

IV. Open Loop Trajectory Planning

Given a trajectory in the task space, represented by a sequence of task space points $\boldsymbol{x}_d^i$, together with corresponding desired velocities $\dot{\boldsymbol{x}}_d^i$, the open loop trajectory planning problem consists of finding joints positions and velocity profiles, $\boldsymbol{q}_d(t)$ and $\dot{\boldsymbol{q}}_d(t)$ respectively, that will generate a task space trajectory satisfying the desired position and velocity constraints. The problem, however, is ill-posed, as redundant robots typically will have, for each desired task space positions, a continuum of inverse solutions; in fact, it is a well known result that even for non-redundant robots the inverse kinematics model will usually generate more than a single solution. To generate a feasible joint space trajectory, the approach taken in this paper first uses IMLE to obtain, for each task space point $\boldsymbol{x}_d^i$, a set of inverse kinematics solutions $\boldsymbol{q}_j^i$. After that we obtain a single solution for each of the inverse kinematics sets of solutions, by imposing a penalty on the overall joint space displacement and on the predicted forward error for the whole trajectory, choosing the joint space trajectory that minimizes such cost. Finally, position and velocity profiles are generated for each joint that respect the velocity and acceleration constraints for the joints. These steps are further detailed in the following sections.

A. Inverse Prediction

The IMLE model can directly provide inverse predictions for a given query $\boldsymbol{x}_d^i$ from the probabilistic model learned so far. In general, as discussed in Section II, inverse predictions may be not as accurate as predictions taken from the forward model, due to the multi-valued nature of the inverse kinematics map. However, since forward and Jacobian estimates are also readily available from the IMLE model, we can use these estimates to improve each inverse kinematics solution. This

is achieved through the use of the Jacobian pseudo-inverse:

$$\boldsymbol{q}_j^i = \hat{\boldsymbol{q}}_j^i + \boldsymbol{J}^\dagger(\hat{\boldsymbol{q}}_j^i)\left(\boldsymbol{x}_d^i - \hat{\boldsymbol{x}}(\hat{\boldsymbol{q}}_j^i)\right) \tag{5}$$

where $\hat{\boldsymbol{q}}_j^i$ is an inverse solution provided by IMLE and $\boldsymbol{J}(\hat{\boldsymbol{q}}_j^i)$ and $\hat{\boldsymbol{x}}(\hat{\boldsymbol{q}}_j^i)$ are respectively the Jacobian and forward predictions evaluated at $\hat{\boldsymbol{q}}_j^i$, as predicted by IMLE. We can think of Eq. (5) as performing a correction on the joints positions vector that will drive the error $(\boldsymbol{x}_d^i - \hat{\boldsymbol{x}}(\hat{\boldsymbol{q}}_j^i))$ to zero, following a first order approximation to the estimated forward kinematics map.

B. Trajectory Optimization

Having a set of candidate inverse solutions for each desired task space point, $\boldsymbol{q}_j^i$, the main issue is then how to choose an appropriate solution from each of these sets. A sensible approach is to pick the inverse solutions in a way that the overall joints displacement is minimized, avoiding large jumps in the joint space, and also to prefer inverse solutions with a low forward error, as estimated from the learned model.

In this paper we closely follow the approach proposed in [10], where a joint space trajectory is obtained, from the full set of inverse solutions, by minimization of a global penalty of the form

$$\sum_{i=1}^{N} \|\boldsymbol{q}^i - \boldsymbol{q}^{i-1}\| + \lambda \sum_{i=1}^{N} \|\boldsymbol{x}_d^i - \hat{\boldsymbol{f}}(\boldsymbol{q}^i)\|^2 \,, \tag{6}$$

for $\lambda \geq 0$. The first term penalizes joint space discontinuities, encouraging short trajectories, while the second term imposes a penalty on inaccurate solutions given by inverse prediction. Note that, contrary to the work of [10], we use the IMLE model to get the predictions $\hat{\boldsymbol{f}}(\boldsymbol{q}^i)$ — in the cited work the Mixture Density Network used for learning the inverse kinematics map cannot generate forward predictions, and so the forward error must be calculated either using an analytical model for the direct kinematics or an independent learning algorithm. Also, another fundamental difference is the online nature of the IMLE algorithm: in [10] the inverse kinematics model must be learned offline, before being used for control.

Minimization of (6) is computationally very cheap, using Dijkstra's algorithm over a directed acyclic graph; compared to the computations involved in obtaining forward and inverse predictions, its cost is negligible.

C. Generation of motor commands

After the calculation of the joint space trajectory corresponding to the desired trajectory in the task space, we need to generate temporal positions and velocity profiles for each of the joints. Additionally, we may wish to traverse each of the task space points with a given desired velocity $\dot{\boldsymbol{x}}_d^i$. Fortunately, these velocities can easily be mapped back to the joint space, once again using the pseudo-inverse of the estimated Jacobian $\boldsymbol{J}^\dagger$, making

$$\dot{\boldsymbol{q}}_d^i = \boldsymbol{J}^\dagger(\boldsymbol{q}_d^i)\, \dot{\boldsymbol{x}}_d^i \,. \tag{7}$$

Position, velocity and acceleration temporal profiles that achieve such task, described by $\boldsymbol{q}_d^i(t)$, $\dot{\boldsymbol{q}}_d^i(t)$ and $\ddot{\boldsymbol{q}}_d^i(t)$ respectively, can then be obtained for each joint, by resorting to classic joint trajectory generation based on, for instance, cubic polynomials or Bang-Bang acceleration policies [14] that take the acceleration and velocity limitations of each joint into account. These profiles can then be fed into the low level joint controllers: for velocity control, for instance, a possible control law is

$$\dot{\boldsymbol{q}}_{act}(t) = \dot{\boldsymbol{q}}_d(t) + K_p(\boldsymbol{q}_d(t) - \boldsymbol{q}(t)) \,, \tag{8}$$

where $\dot{\boldsymbol{q}}_{act}(t)$ is the motor command sent to the low level controllers at each time step and $\dot{\boldsymbol{q}}_d(t)$ and $\boldsymbol{q}_d(t)$ are the desired position and velocity temporal profiles, as calculated by the joint trajectory generator, and K_p is a positive gain.

V. Experimental Evaluation

We perform our experiments on iCub, a 53 degrees of freedom humanoid robot for research in embodied cognition. We use the iCub simulator [15], a realistic software that uses ODE (Open Dynamic Engine) for simulating rigid bodies and collision detection algorithms to compute the physical interaction with objects: a snapshot of this simulator is displayed in Figure 1.

Fig. 1: A snapshot of the iCub Simulator used in the experiments.

The right arm and the waist of the robot are actuated to control the end-effector position in the 3D Cartesian space, using the controllers presented in the previous sections. The joint space vector used in the experiments has 7 degrees of freedom, corresponding to the shoulder yaw, pitch and roll rotations (elevation/depression, adduction/abduction and rotation of the arm), the elbow flexion/extension, and the waist yaw, roll and pitch rotations (rotation, adduction/abduction, elevation/depression of the trunk). The corresponding joint limits are defined in Table I. The end-effector 3D position is taken with respect to a fixed reference frame placed on the ground, between the robot feet.

	arm				*waist*		
$\boldsymbol{q}^{min}$	$-80°$	$0°$	$0°$	$20°$	$-30°$	$-30°$	$-10°$
$\boldsymbol{q}^{max}$	$0°$	$80°$	$80°$	$80°$	$30°$	$30°$	$30°$

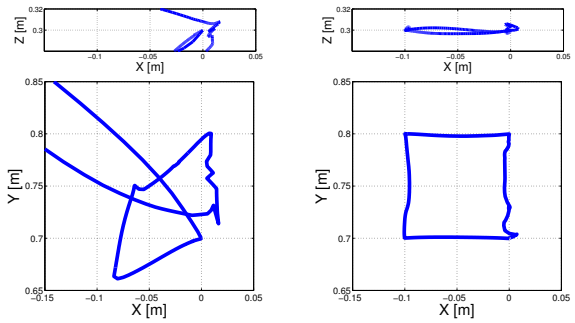
TABLE I: Joints limits of the iCub robot simulator.

A. *Open and Closed-Loop Trajectory Following*

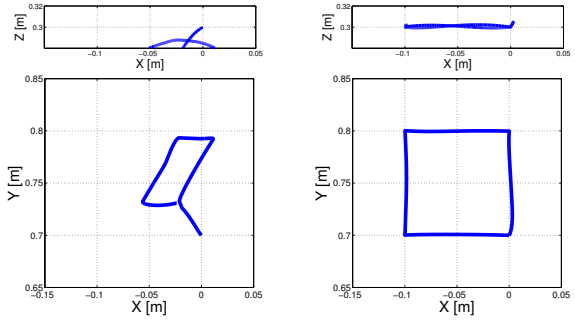
The first experiment compares the performance of the proposed open and closed-loop control schemes when following a desired trajectory in the task space. In this and the following experiments, the robot was asked to perform a square-like pattern in a X-Y plane in the task space, with side length equal to 0.1m, using its end-effector. One of the main advantages of the IMLE algorithm is the ability to work in an online manner, learning and updating its internal parameters while using the same model to make the predictions needed to execute the desired actions. Figure 2 shows the results obtained when trying to draw the square-like pattern, both for open-loop (in the left) and closed-loop control (in the right). In this experiment, the robot tried to perform the trajectory from scratch, using an untrained IMLE probabilistic model[1], and updated the IMLE parameters on the fly, presenting it with the training points as they where being acquired during the robot movement. The robot then repeated the movement for several iterations, trying to follow the desired square-like trajectory in each iteration.

To follow the desired trajectory, the open-loop controller planned, at each iteration, the joint space velocity and position profiles that would take the end-effector to each of the edges of the square, in sequence, stopping at each of them. Additionally, some task space via points were defined between the square edges, to guarantee a reasonably straight trajectory in the task space, between the edges of the square. As for the closed-loop controller, it sufficed to set its end-effector position reference to each of the edges of the desired square-like trajectory: each time the end-effector was close enough to the target point or a time limit was reached the reference in the task space would change to the next edge. There was no need to set via points for the closed loop controller, since the RMRC schemes are known to generate straight line task space trajectories when the model being used to obtain the Jacobian is accurate enough.

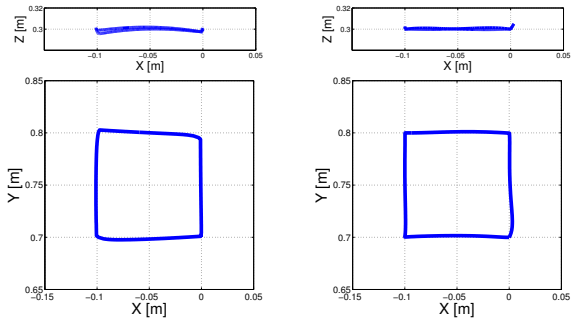
As expected, a poor controller performance was observed during the first iterations of the movement, due to a not yet properly learned model. However, as shown in Figure 2, after about 10 iterations the kinematics model had been learned and the robot could then properly follow the desired trajectory, for both open and closed-loop control. Note that in the first iterations the closed-loop controller was more successful at performing the task: this is a consequence of the online learning setup, since the closed-loop controller Jacobian estimation at each time step used a IMLE model that was also constantly being updated and improved. In contrast, the open-loop controller planned the whole trajectory in the beginning of the movement, and would only use the updated IMLE model in the next movement iteration. Alternatively, the open-loop controller could be easily adapted to re-plan the trajectory at a given time rate, to exploit the online update of the kinematics model.



(a) First Iteration.



(b) Second Iteration.



(c) Tenth Iteration.

Fig. 2: Online Learning and Trajectory Following: open-loop (left) and closed-loop (right).

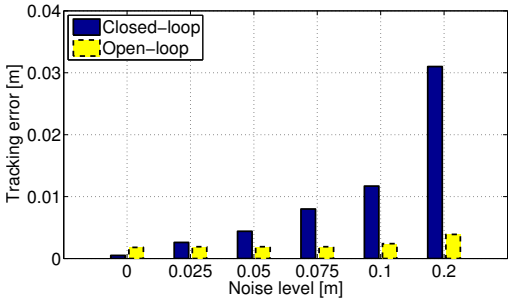
It can be argued that controlling the robot while simultaneously learning the model in an online fashion avoids the potential problems that can arise when estimating the inverse kinematics of a redundant robot, as the training data being fed to the learning algorithm specializes in a single joint space trajectory, corresponding to a specific solution of the inverse kinematics. To evaluate the IMLE inverse prediction capabilities in a more general setting, we performed a random exploration in the joint space of the robot; this motor babbling was executed until 100,000 training points were acquired and processed by the IMLE algorithm, in order to cover the whole workspace. After that, the task of the

[1]Actually some negligible babbling was performed by the robot before the experiments, but only to acquire a couple of training points required to initialize the mixture with a single expert.

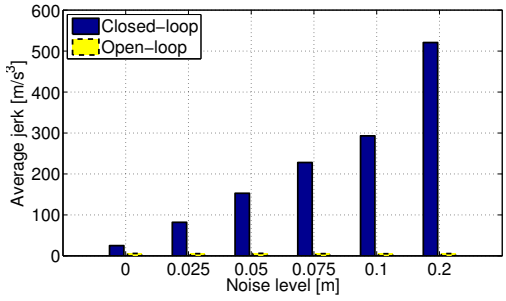
previous experiment was executed. We didn't observe any noticeable changes on the final open-loop trajectory, and the tracking error remained in the same level: this shows that, despite the redundancy in the kinematics map, the open-loop controller based on the inverse predictions provided by IMLE could still achieve a good accuracy while performing the desired task.

B. Sensitivity to Sensor Noise

Next, we studied the performance of both open and closed-loop controllers under different sensor noise levels. While a performance drop was to be expected for both controllers when the noise increased, as a consequence of less precise learned IMLE models, a larger sensitivity of the closed-loop controller to the noise was expected, as this controller relied on sensor readings to calculate the actuation at each control step. This is confirmed in Figure 3, where RMSE is depicted for both open and closed-loop controllers as a function of the noise level, for the square-like target trajectory, after the kinematics model has been properly learned[2]. The jerkiness of the motion is also depicted in this figure: once more, as expected, we can see that the jerk for the closed-loop controller quickly increases for high output noise levels. Figure 4 shows the attained trajectories, for open and closed-loop, for three different noise levels. Also, we show in Figure 5 a comparison of the first joint position profile under severe sensor noise, for both types of controllers; the obtained results for the other joints were similar.



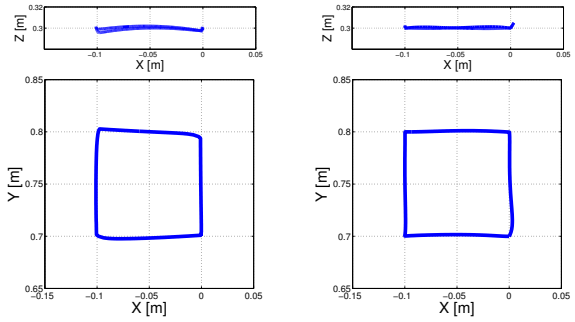
(a) Average error (RMSE).



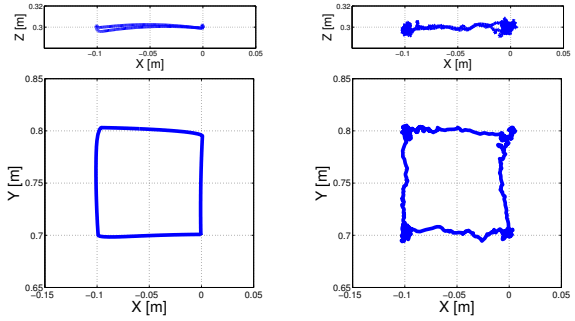
(b) Average jerk (m/s^3).

Fig. 3: Average error and jerk over the square-like trajectory, for different sensor noise levels.

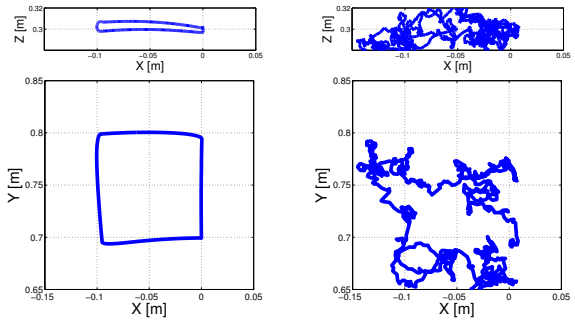
[2]A noise level of X in this paper means that the noise is uniformly distributed between -X and X. We also tried a Gaussian distributed noise: this didn't visibly change the results.



(a) No noise.



(b) Moderate noise level.

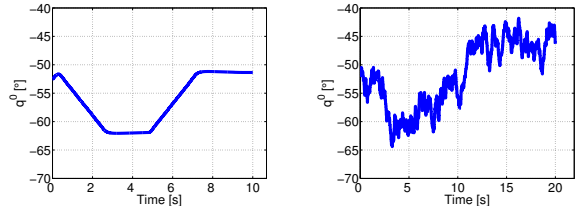


(c) High noise level.

Fig. 4: Trajectory following after the learning phase, for several noise levels: open-loop (left) and closed-loop (right).

C. Sensor Failure

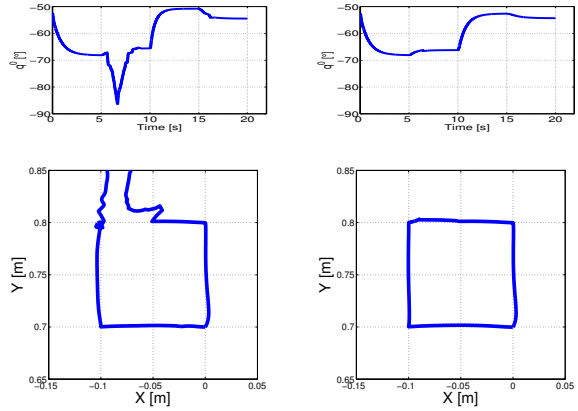
Sometimes the sensors reading the end-effector task space position may fail: this may be due to a sensor malfunction or, for vision based systems, simply a consequence of some kind of end-effector occlusion. When such kind of situation occurs we must resort to open-loop control, as the sensory feedback is no longer available. The IMLE learning algorithm can then be used for both types of control: it can use the Jacobian based RMRC scheme during normal operation, and when the end-effector position feedback fails a seamless switch to open-loop control, based on the same learned model, can be applied. Figure 6 depicts this situation: in the left image we show the task trajectory achieved by the closed loop controller when, at a specific time, we simulated a sensor malfunction. The figure to the right shows the recovery using the open-loop controller: as soon as the failure was



(a) Open-loop. (b) Closed-loop.

Fig. 5: Position profiles for the first joint, as the robot executes the task with a high level of sensor noise.

detected, the controller switched to open-loop mode and a new trajectory was planned and executed. As shown in the figure, for the composite controller based on both open and closed-loop control, there is no noticeable degradation on the task performance when the failure occurs. Above each end-effector trajectory is also depicted a position profile for the first joint of the arm. Note that even when a sensor failure is not properly communicated to the controlling algorithm, resulting in a situation where the sensory feedback simply returns erroneous values, the control system can detect such situation by permanently comparing the sensor readings to the corresponding predicted values, taken from forward predictions given by the IMLE model.



(a) Closed-loop controller. (b) Composite controller, changing to open-loop when the fault is detected.

Fig. 6: Simulating an end-effector position sensor fault.

VI. Conclusion and Final Remarks

We presented an application of IMLE, an online machine learning method, to the trajectory control of the end-effector in the task space of a robot. This learning algorithm is computationally very efficient, being extremely well suited for real-time operation, and is able to provide forward, inverse and Jacobian predictions from the same probabilistic model. Its multi-valued prediction capabilities allow the method to recover multiple solutions for the inverse kinematics problem, corresponding to different branches of the inverse map, or a sample of the continuum space of inverse solutions when redundant robots are considered. This characteristic, combined with its forward and Jacobian prediction, makes IMLE an invaluable algorithm for task space trajectory control. We showed how this algorithm can be used in a closed-loop RMRC scheme, updating its internal parameters while executing a desired task, and also how its inverse kinematics capabilities can be used for open-loop trajectory planning. This suggests using IMLE to support a composite controller that activates a closed or open-loop controller whenever needed. As demonstrated in the previous sections, the closed-loop scheme may be better suitable for normal operation of the robot, but the existence of severe sensor noise or delay, for instance, can be detected by the IMLE algorithm, leading to an automatically switch to open-loop trajectory planning. In the ultimate case of sensor failure, where resolved motion rate controllers are simply ineffective, the controller can seamless switch to open-loop, without any considerable degradation of the task being performed.

References

[1] A. Ligeois, "Automatic supervisory control of the configuration and behavior of multibody mechanisms," *Transactions on System, Man and Cybernetics*, no. 7, pp. 868–871, 1977.

[2] C. Salaün, V. Padois, and O. Sigaud, "Learning forward models for the operational space control of redundant robots," *From Motor Learning to Interaction Learning in Robots*, vol. 264, pp. 169–192, 2010.

[3] L. Jamone, L. Natale, M. Fumagalli, F. Nori, G. Metta, and G. Sandini, "Machine-learning based control of a human-like tendon driven neck," in *IEEE International Conference on Robotics and Automation*, Anchorage, AK, USA, 2010.

[4] L. Jamone, L. Natale, K. Hashimoto, G. Sandini, and A. Takanishi, "Learning task space control through goal directed exploration," in *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Phucket, Thailand, Dec. 2011, pp. 702–708.

[5] S. Vijayakumar, A. D'Souza, and S. Schaal, "Incremental Online Learning in High Dimensions," *Neural Computation*, vol. 17, no. 12, pp. 2602–2634, 2005.

[6] A. D'Souza, S. Vijayakumar, and S. Schaal, "Learning inverse kinematics," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Diego, CA, USA, pp. 298–303.

[7] Z. Ghahramani and M. Jordan, "Supervised Learning from Incomplete Data via an EM approach," *Advances in Neural Information Processing Systems 6*, 1994.

[8] M. Lopes and B. Damas, "A learning framework for generic sensory-motor maps," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Diego, CA, USA, Oct. 2007, pp. 1533–1538.

[9] C. Bishop, *Neural Networks for Pattern Recognition*. Oxford University Press, USA, 1995.

[10] C. Qin and M. A. Carreira-Perpinán, "Trajectory inverse kinematics by conditional density modes," in *IEEE International Conference on Robotics and Automation*, Pasadena, CA, USA, May 2008, pp. 1979–1986.

[11] B. Damas and J. Santos-Victor, "An online algorithm for simultaneously learning forward and inverse kinematics," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura, Portugal, Oct. 2012, pp. 1499–1506.

[12] B. Damas and J. Santos-Victor, "Online Learning of Single and Multivalued Functions with an Infinite Mixture of Linear Experts," Accepted for publication in *Neural Computation*, 2013.

[13] Y. Nakamura and H. Hanafusa, "Inverse kinematic solutions with singularity robustness for robot manipulator control," *Transactions of the ASME Journal of Dynamic Systems, Measurement and Control*, no. 108, pp. 163–171, 1986.

[14] J. Craig, *Introduction to Robotics: Mechanics and Control*. Addison-Wesley Longman Publishing Co., Inc., 1989.

[15] V. Tikhanoff, P. Fitzpatrick, G. Metta, L. Natale, F. Nori, and A. Cangelosi, "An open source simulator for cognitive robotics research: The prototype of the icub humanoid robot simulator," in *Workshop on Performance Metrics for Intelligent Systems*, National Institute of Standards and Technology, Washington DC, August 19-21 2008.