This paper is a Post-Print version (i.e. final draft post-refereeing).
For access to Publisher's version (IEEE), please access
proceedings of 2019 19th International Conference on Advanced Robotics (ICAR), pp. 344-349
DOI: 10.1109/ICAR46387.2019.8981606

Incremental Policy Refinement by Recursive Regression and Kinesthetic Guidance

Bojan Nemec, Mihael Simonič, Tadej Petrič and Aleš Ude

***Abstract*— Fast deployment of robot tasks requires appropriate tools that enable efficient reuse of existing robot control policies. Learning from Demonstration (LfD) is a popular tool for the intuitive generation of robot policies, but the issue of how to address the adaptation of existing policies has not been properly addressed yet. In this work, we propose an incremental LfD framework that efficiently solves the above-mentioned issue. It has been implemented and tested on a number of popular collaborative robots, including Franka Emika Panda, Universal Robot UR10, and KUKA LWR 4.**

I. Introduction

Nowadays, robots are still dominating in large-scale automotive and electronic industries, characterized by large-volume, large-batches productions. One of the issues of contemporary robotics is the introduction of flexible and collaborative robots in craft and small-scale SME production. These production plants are characterized by a wide variety of different assembly tasks, where it is essential to shorten the programming time and to reduce the required skill level of the operators [1]. One of the promising technologies capable of solving the above-mentioned issues is Learning from Demonstration (LfD). Using LfD the robot tasks are demonstrated in a natural way rather than coding in a robot-oriented programming language or using the expensive and complex CAD system, which require profound knowledge in computer science, robot programming and understanding issues arising from robot kinematics and dynamics [2]. LfD concept goes beyond the simple showing and copying a skill performed by a demonstrator. Rather, it encompasses many aspects such as possibly autonomous generalization to different contexts, understanding contexts, adaptation and refinement of the demonstrated policy, etc. [3], [4]. However, efficient demonstration of the various skills remains a key point for successful application of LfD framework. Efficient demonstration is needed not only in production plants but will also be necessary for the upcoming generation of home and service robots that will perform a variety of domestic tasks.

In robotics, we often face the problem that we have to adapt the existing policy to the new context, imposed by the changes in the environment. The robot can autonomously adapt to these changes using techniques like Reinforcement Learning (RL) [5] or Iterative Learning Control (ILC) [6]. However, it requires either at least partial knowledge of the environmental dynamics for ILC or lengthy refinement procedures required by RL. In many cases, it is more efficient to adapt the existing policy utilizing kinesthetic guidance. It is essential that we do not demonstrate the policy again from the beginning, but only gradually correct the existing trajectory. A suitable framework was proposed by Lee and Ott [7], where they introduced an incremental adaptation of the robot policy along the refinement tube and applied to humanoid upper-body robot Justin. Incremental adaptation was also studied in [8] where the main focus was on context-dependent robot skills. However, for the successful completion of the task, it is necessary to adapt both spatial and temporal part of the task. For complex policies, it is beneficial to demonstrate them separately. This important aspect was not considered in the aforementioned frameworks. In [9] we proposed to define the refinement tube regarding the Frenet-Serret (FS) frame and decouple the spatial and temporal part of the trajectory. In this paper, we propose an improved algorithm, which applies the recursive regression technique instead of batch regression used in [9]. Moreover, it enables incremental refinement by moving back and forth along the trajectory, i.e., to update the path in multiple passes.

The paper is organized into five sections. In the following section, we briefly review Cartesian Dynamic Motion Primitives and propose extensions, which enable the generation of movement back and forth along the parametrized trajectory. Procedures for incremental refinement of an existing policy using kinesthetic guidance are given in Section III. They rely on an appropriate control policy, which allows to set the compliance around an arbitrary axis and are briefly outlined in the Appendix. Section IV describes the experimental verification of our framework in learning different tasks. We conclude with critical discussion and plans on how to improve the proposed framework in Section V.

II. Speed Scaled Cartesian Dynamic Motion Primitives

Most contemporary LfD schemes rely on an appropriate Movement Primitive (MP) that allows efficient parametrization of the robot control policy. An MP aims to allow temporal and spatial scaling of the robot policy and easy adaptation to a new situation as it arises. Many previously presented MP schemes fulfill these requirements, i.e. Dynamic Motion Primitives (DMP) [10], Gaussian Mixture Models with Gaussian Mixture regression (GMM-GPR) [11], Probabilistic Motion Primitives (PMP), [12], Hidden Markov Models [7], [2], Interaction Primitives (IP) [13], etc.

Humanoid & Cognitive Robotics Lab, Department of Automatics, Biocybernetics and Robotics, Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia, (bojan.nemec, mihael.simonic,tadej.petric,ales.ude)@ijs.si

Our approach is based on DMPs, which were further extended to allow non-uniform velocity scaling [14] and encoding policies in Cartesian space using unit quaternions [15]. In this work, we propose modifications to encode bi-directional policies, i.e. policies that allow to roll back to a previous state following the same trajectory as in forwarding direction.

Here, we briefly review the foundations of Cartesian dynamic motion primitives (CDMP) for discrete tasks. For periodic tasks, please refer to [15]. Positions $\mathbf{p} \in \mathbb{R}^3$ and orientations $\mathbf{q} \in \mathbb{R}^4$ are generated by the following system of nonlinear differential equations

$$\tau \dot{\boldsymbol{z}} = \nu(s)(\alpha_z(\beta_z(\boldsymbol{g}_p - \mathbf{p}) - \boldsymbol{z}) + \mathbf{f}_p(s)), \quad (1)$$

$$\tau \dot{\mathbf{p}} = \nu(s)\boldsymbol{z}, \quad (2)$$

$$\tau \dot{\boldsymbol{\eta}} = \nu(s)(\alpha_z\left(\beta_z 2\log\left(\boldsymbol{g}_o * \overline{\mathbf{q}}\right) - \boldsymbol{\eta}\right) + \mathbf{f}_o(s)), \quad (3)$$

$$\tau \dot{\mathbf{q}} = \frac{1}{2}\nu(s)\,\boldsymbol{\eta} * \mathbf{q}, \quad (4)$$

$$\tau \dot{s} = -\nu(s)\alpha_s s. \quad (5)$$

where s is the phase variable, $\boldsymbol{z}$, $\boldsymbol{\eta} \in \mathbb{R}^3$ are auxiliary variables, τ is the duration of the trajectory, $\nu(s)$ is a potentially nonlinear temporal scaling factor, and α_z, β_z, $\alpha_s > 0$ are constants chosen such that the underlying second order linear dynamic system is critically damped. Operator $*$ denotes quaternion multiplication and $\overline{\mathbf{q}}$ conjugate of quaternion $\mathbf{q}$.

The quaternion logarithm $\log : \mathbf{S}^3 \mapsto \mathbb{R}^3$ is defined as

$$\log(\mathbf{q}) = \log(v, \mathbf{u}) = \begin{cases} \arccos(v)\dfrac{\mathbf{u}}{\|\mathbf{u}\|}, & \mathbf{u} \neq 0 \\ [0,0,0]^{\mathrm{T}}, & \text{otherwise} \end{cases}. \quad (6)$$

It maps the quaternion describing orientation to the angular velocity that rotates the identity orientation to the current orientation within unit time. Its inverse transformation ($\forall \mathbf{r} \in \mathbb{R}^3$, $\|\mathbf{r}\| < 2\pi$) is defined as

$$\exp(\mathbf{r}) = \begin{cases} \cos\left(\|\mathbf{r}\|\right) + \sin\left(\|\mathbf{r}\|\right)\dfrac{\mathbf{r}}{\|\mathbf{r}\|}, & \mathbf{r} \neq 0 \\ 1 + [0,0,0]^{\mathrm{T}}, & \text{otherwise} \end{cases}. \quad (7)$$

The nonlinear forcing terms $\mathbf{f}_p(s)$ and $\mathbf{f}_o(s)$ are formed in such a way that the response of the second-order differential equation system (1) – (5) can approximate any smooth point-to-point trajectory from the initial position $\boldsymbol{p}_0$ and orientation $\boldsymbol{q}_0$ to the final position $\boldsymbol{g}_p \in \mathbb{R}^3$ and orientation $\boldsymbol{g}_o \in \mathbb{R}^4$ (represented by a unit quaternion). They are defined as linear combinations of radial basis functions (RBFs)

$$\mathbf{f}_p(s) = (\mathbf{x}(s)\mathbf{W}_p)^{\mathrm{T}}, \quad (8)$$

$$\mathbf{f}_o(s) = (\mathbf{x}(s)\mathbf{W}_o)^{\mathrm{T}}, \quad (9)$$

$$\mathbf{x}(s) = \frac{\left[\ \Psi_1(s), \ldots, \Psi_N(s)\ \right]}{\sum_{j=1}^{N}\Psi_j(s)}s, \quad (10)$$

$$\Psi_j(s) = \exp\left(-h_j\left(s - c_j\right)^2\right),\ j = 1, \ldots, N,$$

where the matrices composed of free parameters $\mathbf{W}_p, \mathbf{W}_o \in \mathbb{R}^{N\times 3}$ determine the shape of the position and orientation trajectory, respectively. Centers c_j of RBFs with widths h_j are evenly distributed along the trajectory.

The temporal scaling function $\nu(s)$ determines variations from the demonstrated speed profile and allows to specify non-uniform speed changes along the demonstrated trajectory. Similarly to the forcing terms (8) and (9), it is encoded as a linear combination of RBFs

$$\nu(s) = \mathbf{x}(s)\mathbf{w}_\nu, \quad (11)$$

where $\mathbf{w}_\tau \in \mathbb{R}^N$ is a column vector with the corresponding weights.

In order to parameterize the demonstrated control policy with a CDMP, the weights $\mathbf{W}_p$ and $\mathbf{W}_o$ need to be calculated by applying standard regression techniques [15] using the demonstrated trajectory π_d. For ν we initially set such $\mathbf{w}_\nu$ that $\nu(s) = 1\ \forall s$, meaning that the speed profile remains as demonstrated.

In this paper we introduced slightly modified formulation of $\nu(s)$, which is actually the inverse function of the one introduced in [14]. This minor modification has important properties: it allows to completely stop the evolution of the CDMP by setting ν to 0, velocity scale the policy for $\nu > 0$ and generate time-reversed policy for $\nu < 0$. Please note that CDMP is a dynamical system and reversing the time results in unstable behavior. In [16], we proposed to create another CDMP, computed for policy $\pi_r(t) = \pi_d(\tau - t)$. Variable t denotes the time. Here, we set also the necessary condition to switch between the CDMP which parametrizes π_d and $\overline{\text{CDMP}}$ which parametrizes π_r. Smooth switching between CDMP and $\overline{\text{CDMP}}$ at a phase s is obtained with

$$\overline{s} = \frac{e^{-\alpha_z}}{s} \quad (12)$$

$$\overline{\mathbf{z}} = -\mathbf{z} \quad (13)$$

$$\overline{\boldsymbol{\eta}} = -\boldsymbol{\eta} \quad (14)$$

$$\overline{\nu}(\overline{s}) = |\nu(s)|, \quad (15)$$

where over-lined variables $\overline{(.)}$ belong to the $\overline{\text{CDMP}}$. Thus, whenever $\nu(s)$ changes the sign, we map CDMP to $\overline{\text{CDMP}}$ variables (or vice versa) using the above equations and start integrating CDMP or $\overline{\text{CDMP}}$ depending of the sign of the $\nu(s)$, using Eqs. (1) – (5).

III. Incremental Refinement of the Control Policy using Kinesthetic Guidance

In this section, we describe our main idea - how to intuitively and incrementally modify an existing robot policy. Our framework should provide the operator to freely move the robot forward and backward along the existing policy at any speed and change only those parts, where necessary. For this purpose, we applied our previously developed method [16] based on kinesthetic guiding within a refinement tube [7]. In this method, actions are parameterized with CDMP, presented in Section II. The desired robot poses denoted as tuple $[\mathbf{p}_d^T\ \mathbf{q}_d^T]^T$, are passed as references to the robot controller (see Appendix). They are calculated using a set of Eqs. (1) – (5).

To allow the operator to move the robot forward and backward along the existing trajectory π_a, we associate speed scaling factor τ of CDMPs with force projected to the tangential direction of the Frenet-Serret frame [17] computed at the current position on the desired path. The tangential direction of the Frenet-Serret frame (32) for translational motion is calculated as

$$\mathbf{t}_p(s) = \frac{\dot{\mathbf{p}}_d(s)}{\|\dot{\mathbf{p}}_d(s)\|}, \tag{16}$$

where $\dot{\mathbf{p}}_d \in \mathbb{R}^3$ are commanded (demonstrated) velocities obtained from Eq. (1) and s denotes the phase. The corresponding direction for the rotational motion is

$$\mathbf{t}_r(x) = \frac{\boldsymbol{\omega}_d(s)}{\|\boldsymbol{\omega}_d(s)\|}, \tag{17}$$

where $\boldsymbol{\omega}_d \in \mathbb{R}^3$ is the commanded robot end-effector angular velocity in Cartesian space calculated integrating Eq. (3) and taking into account the relation $\boldsymbol{\omega}_d = \boldsymbol{\eta}/\tau$. In order to move along the trajectory according to the applied forces and torques to the robot end effector, we calculate a new speed scaling factor using

$$\nu(s) = k_1 \mathbf{F} \cdot \mathbf{t}_p(s) + k_2 \mathbf{M} \cdot \mathbf{t}_r(s), \tag{18}$$

where $(\cdot)$ denotes dot product and k_1, k_2 are positive scalars, used to scale the velocities of the translational and rotational motion along the $\pi_a(s)$. $\mathbf{F} \in \mathbb{R}^3$ and $\mathbf{M} \in \mathbb{R}^3$ are the measured vectors of forces and torques in Cartesian coordinate system. As forces and torques are usually measured in the tool coordinate system, they need to be mapped to the Cartesian coordinates with

$$(0, \mathbf{F}) = \bar{\mathbf{q}} * (0, \mathbf{F}_t) * \mathbf{q} \tag{19}$$

$$(0, \mathbf{M}) = \bar{\mathbf{q}} * (0, \mathbf{M}_t) * \mathbf{q}, \tag{20}$$

where $\mathbf{F}_t \in \mathbb{R}^3$ and $\mathbf{M}_t \in \mathbb{R}^3$ are the measured vectors of forces and torques at the robot tool and $\mathbf{q}$ is the unit quaternion of the measured robot rotation. When both $\mathbf{F} \cdot \mathbf{t}_p(s)$ and $\mathbf{M} \cdot \mathbf{t}_r(s) \to 0$, $\tau(s) \to 0$, which actually stops the CDMP integration. Applying this ν to Eqs. (1) – (5), the robot moves along the trajectory $\pi_a(s)$ in the direction of the applied forces and torques, with the speed proportional to them. This algorithm makes the guiding process extremely intuitive. An operator pushes the robot along the tangent of the trajectory. To prevent undesired robot moves along the trajectory due to the force/torque sensor noise, a threshold is usually applied to the measured forces and torques.

As we control the motion along the trajectory with forces and torques, the robot should be stiff in the tangential direction of the path. By making the robot compliant in the directions of normal and binormal of the Frenet-Serret frame, we can also modify the trajectory. An appropriate control law provides such behavior (see Appendix). By this, we can displace the robot only in a plane along with these two directions. Namely, any force in the tangential direction immediately moves the robot along the refinement tube defined around the trajectory and changes this plane. From sampled robot poses $\mathbf{p}, \mathbf{q}$ we calculate position and orientation displacements in the form

$$\mathbf{d}(s) = \begin{bmatrix} \mathbf{d}_p(s) \\ \mathbf{d}_r(s) \end{bmatrix} = \begin{bmatrix} \mathbf{p}(s) - \mathbf{p}_d(s) \\ \log(\mathbf{q}(s) * \overline{\mathbf{q}_d}(s)) \end{bmatrix}. \tag{21}$$

These displacements are used to update the trajectory, as it will be shown later. Similar algorithm can be applied also to

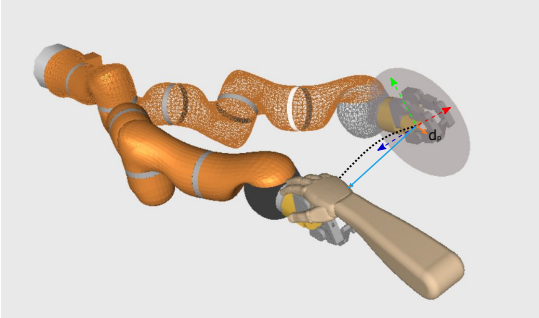

Fig. 1. FS coordinate frame and errors projected to the refinement plane (orange arrow).

the robots, which are not compliant, but provide force/torque measurement. In this case, another displacement vector can be expressed in the form

$$\mathbf{d}'(s) = \begin{bmatrix} \mathbf{d}'_p(s) \\ \mathbf{d}'_r(s) \end{bmatrix} = \begin{bmatrix} \mathbf{F} - (\mathbf{F} \cdot \mathbf{t}_p(s))\mathbf{t}_p(s) \\ \mathbf{M} - (\mathbf{M} \cdot \mathbf{t}_r(s))\mathbf{t}_r(s) \end{bmatrix}. \tag{22}$$

The intuition behind the above equation is straightforward; from measured forces/torques we subtract the projection to the tangential direction, what remains are the forces/torques in the plane defined with the normal and bi-normal of the FS frame, as shown in Fig. 1.

Yet another possibility is to apply this algorithm to the robots that are compliant in all directions and do not provide force/torque measurements. In this case, Eq. (18) changes to the

$$\nu(s) = k_3 \mathbf{d}_p(s) \cdot \mathbf{t}_p(s) + k_4 \mathbf{d}_q(s) \cdot \mathbf{t}_r(s), \tag{23}$$

where $\mathbf{d}_p(s)$ and $\mathbf{d}_q(s)$ are calculated from Eq. (21) and constants k_3 and k_4 are used to scale the position and orientation displacements. Using this equation, the operator drives the robot along the trajectory with a speed proportional to the position/orientation error projected to the tangential direction of the FS frame. The displacement used for the trajectory update is calculated by projecting $\mathbf{d}(s)$ to the plane defined with the normal and bi-normal component of the FS frame.

$$\mathbf{d}''(s) = \begin{bmatrix} \mathbf{d}_p(s) - (\mathbf{d}_p(s) \cdot \mathbf{t}_p(s))\mathbf{t}_p(s) \\ \mathbf{d}_q(s) - (\mathbf{d}_q(s) \cdot \mathbf{t}_r(s))\mathbf{t}_r(s) \end{bmatrix}. \tag{24}$$

From calculated $\mathbf{d}(s)$ (or $\mathbf{d}'(s)$ or $\mathbf{d}''(s)$) we have to calculate the CDMP update. In [16] we proposed to sample robot poses during the above described kinesthetic guiding process and calculate the new nonlinear forcing term of the CDMP whenever we change the sign of $\nu(s)$, using batch regression. The modified robot positions and orientations have to be sampled at exactly the same phase s as the original trajectory, which means, that we have to exactly

determine the phase of the modified trajectory. Even minor phase deviations of s can cause the captured trajectory poses not to follow in the correct order, which can significantly corrupt the modified trajectory. In this work, we propose an alternative solution that does not suffer from this limitation. Instead of capturing the complete trajectory and updating the nonlinear forcing term at the change of the direction, we concurrently modify weights of the CDMP nonlinear forcing terms, $\mathbf{W}(s) = [\mathbf{W}(s)_p \ \mathbf{W}(s)_o] \in \mathbb{R}^{N\times 6}$, using recursive regression formulas

$$\mathbf{W}(s) = \mathbf{W}(s_{-1}) + \mathbf{P}(s)\mathbf{x}^T(s)\mathbf{d}(s)^T\mathbf{K}_l, \quad (25)$$

$$\mathbf{P}(s) = \frac{1}{\lambda}\left(\mathbf{P}(s_{-1}) - \frac{\mathbf{P}(s_{-1})\mathbf{x}(s)^T\mathbf{x}(s)\mathbf{P}(s_{-1})}{\lambda + \mathbf{x}(s)\mathbf{P}(s_{-1})\mathbf{x}^T(s)}\right), \quad (26)$$

where $\mathbf{P}(s) \in \mathbb{R}^{N\times N}$ is the error covariance matrix and $\mathbf{x}(s) \in \mathbb{R}^{1\times N}$ is a vector of Gaussian kernel functions (10). N is the number of kernel functions and s_{-1} denotes the phase in the previous step. λ is forgetting factor that is kept close to 1. Recursive updating is governed by the displacement vector $\mathbf{d}(s)$ and the diagonal estimation gain matrix $\mathbf{K}_l \in \mathbb{R}^{6\times 6}$, which set the magnitude and thus the speed of the adaptation. Our algorithm resets the covariance error matrix to the default value, $\mathbf{P}(s) = \gamma\mathbf{I}$ whenever the factor ν changes its sign. $\mathbf{I}$ denotes identity matrix and γ is a suitably chosen scalar. This is necessary because the recursive algorithm becomes less and less sensitive to the new trajectory updates. Namely, from (26) it follows that the error covariance matrix $\mathbf{P}(s)$ is monotonically decreasing and consequently estimation updates (25) are also decreasing. Note that also a suitable choice of λ prevents the algorithm of decreasing the sensitivity to the new trajectory updates.

The last stage of our procedure is to learn the velocity profile. In this stage, we set the learning gain matrix $\mathbf{K}_l$ to 0 to prevent displacement of the adjusted trajectory and set the stiffness in the directions of normal and bi-normal of the Frenet-Serret frame to the maximal value, when applicable. Next, we are guiding the robot along the trajectory. As this selection of the gains restricts the robot to stay on the trajectory, we concentrate only to demonstrate the desired velocity profile and sample $\nu(s)$. Also, this procedure can be done in multiple repetitions until the required velocity profile is obtained. Finally, we compute the weights of $\mathbf{w}_\nu$ from Eq. (11) applying regression.

Now recall that we can implement our algorithm on three different types of robots: a) stiff in the tangential direction and compliant in the plane perpendicular to it, b) stiff in all directions and c) compliant in all directions. Regarding this, we pass to the recursive update algorithm values $\mathbf{d}(s)$,$\mathbf{d}'(s)$ or $\mathbf{d}''(s)$, respectively. While the environment does not constrain the robot motion, there are no substantial differences in them. However, algorithms act differently when in contact with the environment. The algorithm b) forms a feedback loop between the robot controller and the environment. Environmental stiffness thus directly affects the gain of the resulting admittance control, which might cause closed-loop stability problems. Therefore, with algorithm b), update gains in the diagonal matrix $\mathbf{K}_l$ need to be smaller, which consequently results in slower learning. A similar issue is with algorithm a) in the tangential direction of motion. This motion is, however, not constrained in most applications.

IV. Experimental evaluation

Experimental evaluation of the proposed framework was carried out on three platforms: a) Franka Emika Panda robot, b) Universal robot UR10 and c) bi-manual setup composed of two KUKA LWR 4 robot arms.

Franka Emika Panda is a 7 d.o.f collaborative robot arm, where the control algorithm was implemented as a ROS control plug-in in C++ using libfranka (https://frankaemika.github.io/), while the LfD framework was implemented in Matlab as a ROS node. As the control algorithm enables to implement variable compliance around any axes, we made the robot stiff in the tangential direction of the trajectory and compliant in the plane perpendicular to this direction. We set the stiffness to 3000 N/m in the tangential direction and 500 N/m in the direction of the normal and bi-normal of the FS frame. The displacement vector was calculated using Eq. (21). We executed two experiments, 1) refinement of the policy for the square peg in a hole and 2) adaptation of the shoe grinding trajectory from one to another shoe shape. In experiment 1, we learned an appropriate policy to insert the peg in the hole, as shown in Fig. 2. Next, we displaced the gripping position resulting in a failure to insert the peg. When the robot stopped, as it exceeded the maximal forces, we automatically triggered our policy refinement. We adapted the policy incrementally in subsequent passes, the last pass being the velocity demonstration. Fig. 3 shows the original and the refined positional parts of the trajectories.

Fig. 2. Refinement of the PiH operation.

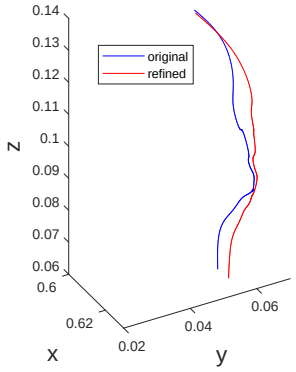

Fig. 3. Original and refined PiH trajectories.

Experiment 2 involves the transfer of trajectories for shoe grinding process from one type of shoe to another. Therefore, trajectories need to be adapted. As in the previous example, we modified the grinding trajectory in multiple passes by pushing the robot tool forward and backward and pressing it against the new shoe shape, until it fitted new shape. Also here, the last pass was used to define the velocity profile. A

snapshot of learning and 3D plots of the original and adapted trajectories are shown in Figs. 4 and 5, respectively.

Fig. 4. Refinement of the shoe grinding policy.

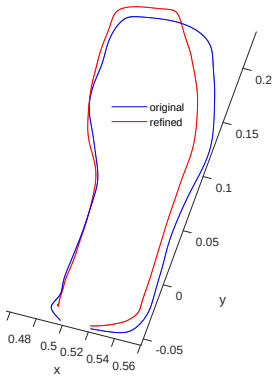

Fig. 5. Original and refined shoe grinding trajectories.

Next experiment involved the assembly of actuator elements for electronically adjustable furniture using Universal Robot UR10, shown in Fig. 6. Although it is possible to teach the UR10 robot in the zero-gravity mode, we applied our LfD framework, which used forces and torques from the wrist-mounted ATI force sensor. Namely, due to the insufficient friction compensation of UR10, it is difficult to demonstrate continues policy in zero gravity mode. During the refinement of the assembly policy, the robot was stiff in all directions, and the displacement vector was calculated using Eq. (22). The LfD framework was implemented in Python as a ROS node, while the control algorithm was implemented in Matlab/Simulink using XPC Target platform [18]. To preserve the stability during the assembly operation which involves physical contacts with the environment, it was necessary to select lower update gains $\mathbf{K}_l$ regarding one selected for experiment 1. Lowering gains resulted in slow movements and slow update of the control policy. In the current implementation, we control the magnitude of $\mathbf{K}_l$ and scalars k_1 and k_2 with a button at the robot wrist, which enabled to improve the performance during the free motion by increasing their values.

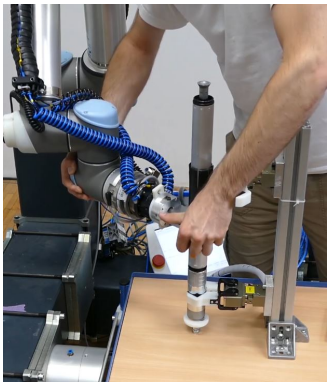

Fig. 6. Learning of the assembly policy for furniture actuator.

The last experiment was carried out on a bi-manual setup composed of two KUKA LWR 4 robot arms and applied to the assembly of long poles of the vacuum cleaner. The same experiment was previously accomplished with the incremental refinement procedure using batch regression in [16]. In this paper, we repeated the experiment with newly proposed recursive learning, where the displacement vector was calculated using Eq. (24). The robot was equally stiff in all directions with positional and rotational stiffness set to $500N/m$ and $50Nm/rad$, respectively. The bi-manual LfD algorithm [16] was implemented in Matlab/Simulink, that generated references for the KUKA LWR 4 controller. Note that this implementation controls directly relative and absolute coordinates of the bi-manual system [16]. Consequently, by moving one arm, the other moves in such a way that it preserves the relative coordinates. Regarding the previous implementation, the new algorithm features greater accuracy, and ease of implementation.

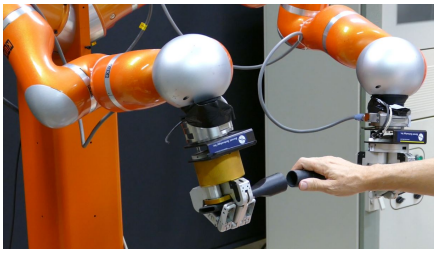

Fig. 7. Refinement of a bi-manual assembly policy.

V. Conclusions

In the paper, we presented an improved algorithm for incremental adaptation of complex robot trajectories using kinesthetic guidance. It is based on speed scaled DMPs, that allow moving forward and backward along the trajectory. It allows decoupling from learning of the spatial and temporal component of the task. The update mechanism is based on recursive regression, which improves the performance, does not require to store the modified trajectory, and is easier to implement. We have provided modifications that are necessary for the implementation of the algorithm also on robots that do not allow changing compliance or are inherently rigid. Implementation on a rigid robot, however, results in reduced performance in terms of a kinesthetic sense during learning, precision, and operational stability.

Our future work includes implementation of the proposed framework as an integrated application in the controller of the Franka Emika robot, which will additionally increase the performance and eliminate the need for an external computer.

Appendix

The dynamics of a robot arm interacting with the environment is described by

$$\boldsymbol{\rho} = \mathbf{H}\ddot{\boldsymbol{\theta}} + \mathbf{h} + \mathbf{J}^{\mathrm{T}} \left[\mathbf{F}^{\mathrm{T}}, \mathbf{M}^{\mathrm{T}}\right]^{\mathrm{T}} \tag{27}$$

where $\boldsymbol{\rho}$ is a vector of joint torques, $\mathbf{H} \in \mathbb{R}^{n\times n}$ is a symmetric, positive definite inertia matrix, $\mathbf{h} \in \mathbb{R}^n$ is vector of the centrifugal, Coriolis, friction and gravity forces, $\mathbf{J} \in \mathbb{R}^{6\times n}$ is the robot Jacobian, and $\mathbf{F}, \mathbf{M} \in \mathbb{R}^3$ are the vectors of environment contact forces and torques acting on the robot's end-effector. $\boldsymbol{\theta} \in \mathbb{R}^n$ denotes the joint angles.

Joint and Cartesian space accelerations of a kinematically redundant robot arm are related by [19]

$$\ddot{\boldsymbol{\theta}} = \mathbf{J}_{\mathbf{H}}^{+}\left(\begin{bmatrix} \ddot{\mathbf{p}} \\ \dot{\boldsymbol{\omega}} \end{bmatrix} - \dot{\mathbf{J}}\dot{\boldsymbol{\theta}}\right) + \mathbf{N}\boldsymbol{\xi}, \tag{28}$$

where $\mathbf{J}_{\mathbf{H}}^{+} = \mathbf{H}^{-1}\mathbf{J}^{\mathrm{T}}(\mathbf{J}\mathbf{H}^{-1}\mathbf{J}^{\mathrm{T}})^{+} = \mathbf{H}^{-1/2}(\mathbf{J}\mathbf{H}^{-1/2})^{+}$ denotes the inertia weighted pseudo-inverse of $\mathbf{J}$ [20]. $\boldsymbol{\xi} \in \mathbb{R}^n$ is arbitrary chosen vector that defines the null space motion. The control law is obtained by inserting (28) into (27)

$$\boldsymbol{\rho} = \mathbf{H}\mathbf{J}_{\mathbf{H}}^{+}\left(\begin{bmatrix} \ddot{\mathbf{p}} \\ \dot{\boldsymbol{\omega}} \end{bmatrix} - \dot{\mathbf{J}}\dot{\boldsymbol{\theta}}\right) + \mathbf{H}\mathbf{N}\boldsymbol{\xi} + \mathbf{h} + \mathbf{J}^{\mathrm{T}}\begin{bmatrix} \mathbf{F} \\ \mathbf{M} \end{bmatrix}, \tag{29}$$

where $\mathbf{N}$ is the projection matrix onto the null space of inertia weighted Jacobian, defined as $\mathbf{N} = \mathbf{I} - \mathbf{J}_{\mathbf{H}}^{+}\mathbf{J}$. The first term in Eq. (29) is the task controller, the second term is the null space controller and the third and the fourth term compensate for the non-linear robot dynamics and external forces, respectively.

Choosing the task command inputs $\ddot{\mathbf{p}}_c$, $\dot{\boldsymbol{\omega}}_c$ as

$$\ddot{\mathbf{p}}_c = \ddot{\mathbf{p}}_d + \mathbf{D}_p(\dot{\mathbf{p}}_d - \dot{\mathbf{p}}) + \mathbf{K}_p(\mathbf{p}_d - \mathbf{p}), \tag{30}$$

$$\dot{\boldsymbol{\omega}}_c = \dot{\boldsymbol{\omega}}_d + \mathbf{D}_q(\boldsymbol{\omega}_d - \boldsymbol{\omega}) + \mathbf{K}_q \log\left(\mathbf{q}_d * \overline{\mathbf{q}}\right), \tag{31}$$

we obtain the well known impedance control law [21] Subscript $_d$ denotes the desired values and variables without the subscript the current values as measured from the robot sensors. $\mathbf{D}_p$, $\mathbf{D}_q$, $\mathbf{K}_p$ and $\mathbf{K}_q$ are positional and rotational damping and stiffness positive definite, but not necessary diagonal matrices.

Let $\mathbf{R}_t$ define the FS frame, attached to the robot trajectory. It is calculated using

$$\mathbf{R}_t = \begin{bmatrix} \mathbf{t} & \mathbf{n} & \mathbf{b} \end{bmatrix}, \tag{32}$$

$$\mathbf{t} = \frac{\dot{\mathbf{p}}_l}{\|\dot{\mathbf{p}}_l\|},\ \mathbf{b} = \frac{\dot{\mathbf{p}}_l \times \ddot{\mathbf{p}}_l}{\|\dot{\mathbf{p}}_l \times \ddot{\mathbf{p}}_l\|},\ \mathbf{n} = \mathbf{b} \times \mathbf{t}.$$

Note that the absolute velocity $\dot{\mathbf{p}}_l$ and acceleration $\ddot{\mathbf{p}}_l$ are provided by CDMP integration at every phase s, which ensures smoothness. At low speed and low accelerations, we suspend the updating of $\mathbf{R}_t$ until the motion becomes faster again. Now choose $\mathbf{K}_p = \mathbf{R}_t\mathbf{K}'_p\mathbf{R}_t^T$ and $\mathbf{K}_q = \mathbf{R}_t\mathbf{K}'_q\mathbf{R}_t^T$. Then, $\mathbf{K}'_p$ and $\mathbf{K}'_q$ are diagonal matrices, that define compliance in tangential, normal and bi-normal axis defined with the FS frame. Damping matrices have to be defined accordingly. In order to obtain critically damped response of the robot, they are chosen as $\mathbf{D}_p = 2\sqrt{\mathbf{K}_p}$ and $\mathbf{D}_q = 2\sqrt{\mathbf{K}_q}$.

Null space motion of the robot is controlled with the null space command input $\boldsymbol{\xi}_c$. Simply setting $\boldsymbol{\xi}_c$ to 0 is not appropriate as the robot will continuously move in null space, minimizing its kinetic energy. To damp this motion, we set the desired null space velocities to zero, and the command null space motion vector is calculated as $\boldsymbol{\xi}_c = -\mathbf{K}_n\dot{\boldsymbol{\theta}}$. The commanded torque $\boldsymbol{\rho}_c$ is finally calculated by inserting $\ddot{\boldsymbol{p}}_c$, $\dot{\boldsymbol{\omega}}_c$, $\boldsymbol{\xi}_c$ into (29), whereas the current values are used for all other variables in (29).

ACKNOWLEDGMENT

The research leading to these results has received funding from the Horizon 2020 RIA Programme grant 820767 CoLLaboratE.

REFERENCES

[1] E. Dean-Leon, K. Ramirez-Amaro, F. Bergner, I. Dianov, P. Lanillos, and G. Cheng, "Robotic technologies for fast deployment of industrial robot systems," *IECON Proceedings (Industrial Electronics Conference)*, pp. 6900–6907, 2016.

[2] R. Dillmann, "Teaching and learning of robot tasks via observation of human performance," *Robotics and Autonomous Systems*, vol. 47, no. 2-3, pp. 109–116, jun 2004.

[3] A. Billard and S. Calinon, "Chapter 59: Robot Programming by Demonstration," in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Springer Berlin Heidelberg, 2008.

[4] P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou, and S. Schaal, "Skill learning and task outcome prediction for manipulation," *2011 IEEE International Conference on Robotics and Automation*, pp. 3828–3834, 2011.

[5] J. Peters, K. Mülling, and J. Kober, "Towards motor skill learning for robotics," *Robotics Research*, pp. 1–14, 2011.

[6] D. A. Bristow and M. Tharayil, "A Survey of Iterative Learning Control - A learning-based method for high-performance tracking control," *IEEE Control Systems Magazine*, no. June, pp. 96–114, 2006.

[7] D. Lee and C. Ott, "Incremental Motion Primitive Learning by Physical Coaching Using Impedance Control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010, pp. 4133–4140.

[8] M. Ewerton, G. Maeda, G. Kollegger, J. Wiemeyer, and J. Peters, "Incremental imitation learning of context-dependent motor skills," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, 2016, pp. 351–358.

[9] B. Nemec, N. Likar, A. Gams, and A. Ude, "Human robot cooperation with compliance adaptation along the motion trajectory," *Autonomous Robots*, vol. 42, no. 5, pp. 1023–1035, 2018.

[10] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural Computation*, vol. 25, no. 2, pp. 328–73, 2013.

[11] S. M. Khansari-Zadeh and A. Billard, "Learning stable nonlinear dynamical systems with gaussian mixture models," *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 943–957, 2011.

[12] A. Paraschos, E. Rueckert, J. Peters, and G. Neumann, "Model-Free Probabilistic Movement Primitives for Physical Interaction," *IROS*, pp. 2860–2866, 2015.

[13] H. B. Amor, G. Neumann, S. Kamthe, O. Kroemer, and J. Peters, "Interaction primitives for human-robot cooperation tasks," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 2831–2837.

[14] B. Nemec, A. Gams, and A. Ude, "Velocity adaptation for self-improvement of skills learned from user demonstrations," in *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, Atlanta, USA, 2013, pp. 423–428.

[15] A. Ude, B. Nemec, T. Petrič, and J. Morimoto, "Orientation in cartesian space dynamic movement primitives," in *IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China, 2014, pp. 2997–3004.

[16] B. Nemec, L. Zlajpah, S. Slajpah, J. Piskur, and A. Ude, "An Efficient PbD Framework for Fast Deployment of Bi-manual Assembly Tasks," in *18th IEEERAS International Conference on Humanoid Robots*, 2018.

[17] R. Ravani and A. Meghdari, "Velocity distribution profile for robot arm motion using rational Frenet-Serret curves," *Informatica*, vol. 17, no. 1, pp. 69–84, 2006.

[18] T. Gaspar, B. Ridge, R. Bevec, M. Bem, I. Kovač, A. Ude, and Ž. Gosar, "Rapid hardware and software reconfiguration in a robotic workcell," in *2017 18th International Conference on Advanced Robotics (ICAR)*, 2017, pp. 229–236.

[19] J. Nakanishi, R. Cory, M. Mistry, J. Peters, and S. Schaal, "Operational space control: A theoretical and empirical comparison," *The International Journal of Robotics Research*, vol. 27, pp. 737–757, 2008.

[20] Y. Nakamura, *Advanced Robotics: Redundancy and Optimization*. Boston, MA: Addison-Wesley, 1991.

[21] N. Hogan, "Impedance control: An approach to manipulation: Part I - theory," *Journal of dynamic systems, measurement, and control*, vol. 107, no. 1, pp. 1–7, 1985.