Journal of Intelligent & Robotic Systems (2021) 102: 5
https://doi.org/10.1007/s10846-021-01328-y

REGULAR PAPER

Analysis of Methods for Incremental Policy Refinement by Kinesthetic Guidance

Mihael Simonič[1] · Tadej Petrič[1] · Aleš Ude[1] · Bojan Nemec[1]

Received: 30 June 2020 / Accepted: 25 January 2021 / Published online: 15 April 2021

Abstract
Traditional robot programming is often not feasible in small-batch production, as it is time-consuming, inefficient, and expensive. To shorten the time necessary to deploy robot tasks, we need appropriate tools to enable efficient reuse of existing robot control policies. Incremental Learning from Demonstration (iLfD) and reversible Dynamic Movement Primitives (DMP) provide a framework for efficient policy demonstration and adaptation. In this paper, we extend our previously proposed framework with improvements that provide better performance and lower the algorithm's computational burden. Further, we analyse the learning stability and evaluate the proposed framework with a comprehensive user study. The proposed methods have been evaluated on two popular collaborative robots, Franka Emika Panda and Universal Robot UR10.

Keywords Incremental learning · Coaching · Stability analysis · User study

1 Introduction

Despite recent changes in the manufacturing paradigm, robots are still predominantly used in large-scale and large-batch production manufacturing plants such as the automotive and electronics industry [1]. The global competitive arena forces companies to react quickly to the demands of the market while maintaining economic feasibility. Consequently, the robotized production plants should adapt to handle a wide variety of products by frequent reconfiguration of both hardware sub-components and robot programs [2]. Another issue is to shorten the robot programming time and reduce the required skill level of the operators [3]. Traditional robot programming requires profound knowledge in computer science and understanding issues arising from robot kinematics and dynamics [4]. Learning from Demonstration (LfD), often referred to as Imitation Learning, is a promising and powerful technology that can among others solve the issues mentioned above [5, 6]. Within this framework, the robot tasks are demonstrated naturally rather than coding in a robot-oriented programming language or using expensive and complex robot simulation systems. It is important to emphasize that LfD concept goes beyond the simple showing and copying of a policy performed by a demonstrator. It includes many aspects such as generalization of the robot policy to different contexts, the ability to adapt to changes in the environment, or refinement of the captured policy [5, 7]. Within LfD, techniques like Reinforcement Learning (RL) [8] or Iterative Learning Control (ILC) [9] are often used for autonomous adaptation of the policy. ILC generally adapts the policy in a few iteration cycles, but it requires at least partial knowledge of the environment and robot dynamics. RL, on the other hand, is generally model-free but needs lengthy adaptation and is currently not yet considered as mature technology for real-word applications.

If the robot is not required to act autonomously, it is more efficient to adapt the existing policy through human-robot interaction, e.g., by utilizing kinesthetic guidance. During this skill transfer from human to robot, often referred as coaching, it is essential that we do not always demonstrate the policy over and over again from the beginning but only

✉ Mihael Simonič
mihael.simonic@ijs.si

Tadej Petrič
tadej.petric@ijs.si

Aleš Ude
ales.ude@ijs.si

Bojan Nemec
bojan.nemec@ijs.si

1 Department of Automatics, Biocybernetics and Robotics, Jožef Stefan Institute, Jamova 39, 1000, Ljubljana, Slovenia

gradually refine the existing policy. This issue is efficiently addressed by incremental learning.

1.1 Related Work

Incremental learning has been studied intensively in the past decade. During the incremental task demonstration, the demonstrator must have the ability to change only the parts of the trajectory that require correction, leaving the rest unchanged. Calinon and Billard [10] incrementally updated the policy encoded with Gaussian Mixture Models while learning gestures for a humanoid robot. Kulic et al. [11] dealt with the problem of incremental learning and simultaneous segmentation of human body motions using Hidden Markov Models. Gams et al. [12] studied how to incrementally refine periodic movements using either visual or force feedback. A framework for incremental learning with kinesthetic guidance was proposed by Lee and Ott [13]. Although kinesthetic guiding is intuitive, it is often impractical as the user can only move a limited number of joints simultaneously. In [13], they proposed incremental policy adaptation along the refinement tube, which allows to limit the motion of the joints during the teaching in a controlled way. Incremental adaptation of context-dependent robot skills was studied in [14], where they applied probabilistic Motion Primitives and Iterative Learning Control framework. Incremental learning can also be implemented using vocal commands, as it was demonstrated in [15]. Incremental learning by combining several demonstrations and using virtual tool dynamics has been proposed in [16]. For assisting kinesthetic modifications of the learned behavior in task variations, the concept of virtual fixtures [17] and penetrable virtual fixtures [18] was recently proposed.

A typical policy consists of both spatial and temporal parts; often, it is beneficial to treat them separately. Since complex movements can be more accurately demonstrated at a low-speed [19], it is also necessary to allow for a separate refinement of the spatial and temporal part of the task. This important aspect was not considered in the aforementioned approaches. In [20] we proposed first to refine the spatial part, i.e., the shape of the trajectory, with an arbitrary speed and introduced the concept of variable stiffness along Frenet-Serret (FS) frames. The desired speed profile was learned in the last stage of learning. Another essential feature of the proposed approach is the ability to incrementally refine the movement by moving back and forth along the trajectory during the demonstration, i.e., to update the path in multiple passes. In [21], batch regression was used to update the parameters of the existing policy incrementally. An alternative way, proposed in [22], applies recursive regression, which updates the policy parameters directly and avoids saving the entire set of coordinates captured during the refinement.

1.2 Main Contributions

This paper is an extension of work originally presented at the 19th International Conference on Advanced Robotics [22], which presented a framework for efficient incremental learning from demonstration aiming to directly modify the policy parameters by recursive regression technique.

In this paper, we further improve our previously presented incremental learning algorithms [20, 22] with several modifications, that enhance kinestethic feeling during policy corrections and improve the learning accuracy.

- We introduce Constant Speed Cartesian DMPs, a modified formulation of DMPs, that follows the idea of Arc Length DMPs (AL-DMPs) [23]. Using this formulation, we can completely decouple the spatial and temporal component of the task, which contributes to more efficient learning.
- Next, we refine the original DMP policy by learning offsets instead of modifying the DMP parameters, which diminishes the computational burden and improves long-term stability of the policy [24].
- Using sigmoid activation functions, we provided a soft start and stop when pushing along the trajectory, enhancing the kinestheitc feeling.
- The encoding accuracy during incremental learning has been improved by replacing entire sections of the trajectory and implementing a new phase detection algorithm.

Additionally, we compare different incremental learning schemes for admittance or impedance controlled robots with policy updated based on recursive or batch regression technique by providing the stability analyses of our incremental learning schemes. Finally, we conduct a user study to evaluate their performance statistically and discuss the benefits and weaknesses of proposed algorithms.

The paper is organized into five sections. In Section 2, we revisit Cartesian DMPs, introduce DMPs with constant speed profile, and provide extensions that enable the reversibility of the policies. Procedures for incremental refinement of an existing policy using kinesthetic guidance are given in Section 3. Section 4 analyses the stability of these procedures, describes the experimental verification of our framework and statistical evaluation of the user study. Finally, in Section 5 we highlight the contributions and benefits of the proposed methods with respect to the learning efficiency and ease of implementation.

2 Policy Encoding using Dynamic Movement Primitives

Initial step of every policy refinement procedure is to encode the original trajectory in a suitable parametric form, often referred as a Movement Primitives (MP). Our approach relies on Dynamic Movement Primitives (DMP) [25], but other representations could also be used.

We apply a modified version of DMPs, which handles non-uniform velocity scaling [26] and encoding Cartesian space policies using unit quaternions [27]. In this work, we propose DMPs with a constant speed profile to fully decouple the spatial and temporal part of the task. Next, we revisit and refine further extensions of the DMPs framework that handle reversible policies, i.e., policies that allow us to follow the same path both forwards and backward [20, 22].

2.1 Speed Scaled Cartesian DMPs

The original policy $\pi(t)$, where t is the time, can be obtained in various ways, e.g. by a motion tracker or kinesthetic guidance. In all cases, we acquire the Cartesian space tool center point (TCP) positions and orientations, angular velocities and accelerations at times t_k, $k = 0, \ldots, T$, in the form

$$\mathcal{G}_d = \{\mathbf{p}_k, \mathbf{q}_k, \dot{\mathbf{p}}_k, \boldsymbol{\omega}_k, \ddot{\mathbf{p}}_k, \dot{\boldsymbol{\omega}}_k, t_k\}_{k=0}^{\mathrm{T}}. \tag{1}$$

Positions and orientations are represented as a frame $\{\mathbf{p}, \mathbf{q}\}$, where $\mathbf{p} \in \mathbb{R}^3$ and $\mathbf{q} = v + \mathbf{u}$, $v \in \mathbb{R}$, $\mathbf{u} \in \mathbb{R}^3$, is a unit quaternion.

First, we briefly review the foundations of Cartesian dynamic movement primitives (CDMP) for discrete tasks. Within this framework, a frame $\{\mathbf{p}, \mathbf{q}\}$ is generated by integrating the nonlinear dynamic system, described with the following equations

$$\tau\dot{\mathbf{z}} = \nu(s)(\alpha_z(\beta_z(\mathbf{g}_p - \mathbf{p}) - \mathbf{z}) + \mathbf{f}_p(s)), \tag{2}$$
$$\tau\dot{\mathbf{p}} = \nu(s)\mathbf{z}, \tag{3}$$
$$\tau\dot{\boldsymbol{\eta}} = \nu(s)(\alpha_z\,(\beta_z 2\log(\mathbf{g}_o * \overline{\mathbf{q}}) - \boldsymbol{\eta}) + \mathbf{f}_o(s)), \tag{4}$$
$$\tau\dot{\mathbf{q}} = \frac{1}{2}\,\nu(s)\,\boldsymbol{\eta} * \mathbf{q}, \tag{5}$$
$$\tau\dot{s} = -\nu(s)\alpha_s s. \tag{6}$$

where $s \in \mathbb{R}$ is the phase variable, $\mathbf{z}$, $\boldsymbol{\eta} \in \mathbb{R}^3$ are auxiliary variables, $\tau \in \mathbb{R}$ is the duration of the policy, $\nu(s) \in \mathbb{R}$ is a nonlinear temporal scaling factor. Scalar $\alpha_z > 0$ sets the convergence rate of the underlying second order linear dynamic system. Setting $\beta_z = \alpha_z/4$ provides that it is critically damped. α_s sets the decay rate of the phase variable s. Operator $*$ denotes quaternion multiplication and $\overline{\mathbf{q}}$ conjugate of quaternion $\mathbf{q}$. An alternative formulation for (5) has been proposed recently to better account for nonlinearities in orientation space [28]. The quaternion logarithm log : $\mathrm{S}^3 \mapsto \mathbb{R}^3$ is defined as

$$\log(\mathbf{q}) = \log(v, \mathbf{u}) = \begin{cases} \arccos(v)\dfrac{\mathbf{u}}{\|\mathbf{u}\|}, & \mathbf{u} \neq 0 \\ [0,0,0]^{\mathrm{T}}, & \text{otherwise} \end{cases}. \tag{7}$$

S^3 denotes the unit sphere in $\mathbb{R}^4$. Quaternion logarithm maps the quaternion $\mathbf{q}$ to the angular velocity $\boldsymbol{\omega}$ that rotates the identity orientation to the current orientation defined with $\mathbf{q}$ within unit time. Its inverse transformation is quaternion exponent ($\exp : \mathbb{R}^3 \mapsto \mathrm{S}^3, \forall\boldsymbol{\omega} \in \mathbb{R}^3$, $\|\boldsymbol{\omega}\| < 2\pi$), defined as

$$\exp(\boldsymbol{\omega}) = \begin{cases} \cos(\|\boldsymbol{\omega}\|) + \sin(\|\boldsymbol{\omega}\|)\dfrac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|}, & \boldsymbol{\omega} \neq 0 \\ 1 + [0,0,0]^{\mathrm{T}}, & \text{otherwise} \end{cases}. \tag{8}$$

The nonlinear forcing terms $\mathbf{f}_p(s)$ and $\mathbf{f}_o(s)$ shape the response of the second-order differential equation system (2) – (6) for any smooth point-to-point trajectory from the initial position $\mathbf{p}_0$ and orientation $\mathbf{q}_0$ to the final position $\mathbf{g}_p \in \mathbb{R}^3$ and orientation $\mathbf{g}_o \in \mathbb{R}^4$. They can be represented by a sum of N radial basis functions Ψ (RBFs)

$$\mathbf{f}_p(s) = (\mathbf{x}(s)\mathbf{W}_p)^{\mathrm{T}}, \tag{9}$$
$$\mathbf{f}_o(s) = (\mathbf{x}(s)\mathbf{W}_o)^{\mathrm{T}}, \tag{10}$$
$$\mathbf{x}(s) = \frac{[\Psi_1(s), \ldots, \Psi_N(s)]}{\sum_{j=1}^{N}\Psi_j(s)}s, \tag{11}$$

$$\Psi_j(s) = \exp\left(-h_j\left(s - c_j\right)^2\right),\ j = 1, \ldots, N. \tag{12}$$

The matrices $\mathbf{W}_p, \mathbf{W}_o \in \mathbb{R}^{N\times 3}$, $\mathbf{W} = [\mathbf{W}_p, \mathbf{W}_o]$ contain free parameters with column vectors $\mathbf{w}_{p,i}$, $\mathbf{w}_{o,i} \in \mathbb{R}^N$ that determine the shape of the positional and orientational part of the trajectory, respectively. Index i denotes the i-th coordinate axis of the Cartesian position. c_j are the centers of RBFs and h_j their widths. Usually they are selected in such a way that RBFs 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 (9) and (10), it is encoded as a linear combination of RBFs

$$\nu(s) = 1 + \mathbf{x}(s)\mathbf{w}_\nu, \tag{13}$$

where $\mathbf{w}_\nu \in \mathbb{R}^N$ is a column vector containing the weights. For ν we initially set $\mathbf{w}_\nu = 0$ so that $\nu(s) = 1\ \forall s$, meaning that the speed profile remains as demonstrated.

While the goal pose $\mathbf{g}_p$, $\mathbf{g}_o$, initial pose $\mathbf{p}_0$, $\mathbf{q}_0$ needed for the integration of (3), and the trajectory duration τ can be directly retrieved from the demonstrated trajectory $\mathcal{G}_d$, the weights $\mathbf{W}_p$ and $\mathbf{W}_o$ need to be calculated by applying standard regression techniques [27]. By replacing the system of two first order equations for positions (2), (3) and orientations (4), (5) with one equation of the second order, respectively, and assuming $\nu(s) = 1$, we obtain

$$\mathbf{f}_{p,k} = \tau^2\ddot{\mathbf{p}}_k + \tau\alpha_z\dot{\mathbf{p}}_k - \alpha_z\beta_z\left(\mathbf{g}_p - \mathbf{p}_k\right), \tag{14}$$

$$\mathbf{f}_{o,k} = \tau^2\dot{\boldsymbol{\omega}}_k + \tau\alpha_z\boldsymbol{\omega}_k - \alpha_z\beta_z 2\log\left(\mathbf{g}_o * \overline{\mathbf{q}}_k\right), \tag{15}$$

where $\mathbf{p}_k, \mathbf{q}_k, \dot{\mathbf{p}}_k, \boldsymbol{\omega}_k, \ddot{\mathbf{p}}_k, \dot{\boldsymbol{\omega}}_k$ are the measurements gathered from the demonstrated trajectory (1). Writing

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}(s(t_0)) \\ \vdots \\ \mathbf{x}(s(t_T)) \end{bmatrix}, \ \mathcal{F}_p = \begin{bmatrix} \mathbf{f}_p^{\mathrm{T}}(t_0) \\ \vdots \\ \mathbf{f}_p^{\mathrm{T}}(t_T) \end{bmatrix}, \ \mathcal{F}_o = \begin{bmatrix} \mathbf{f}_o^{\mathrm{T}}(t_0) \\ \vdots \\ \mathbf{f}_o^{\mathrm{T}}(t_T) \end{bmatrix}, \tag{16}$$

we obtain the following system of linear equations

$$\mathbf{X}\mathbf{W}_p = \mathcal{F}_p, \ \mathbf{X}\mathbf{W}_o = \mathcal{F}_o. \tag{17}$$

In (16), $s(t_j)$ are computed by integrating (6) with the initial condition $s(0) = 1$ and t_j specified in (1). The sets of weights that solve linear equations (17) in a least square sense can then be calculated as follows

$$\mathbf{W}_p = \mathbf{X}^+\mathcal{F}_p, \ \mathbf{W}_o = \mathbf{X}^+\mathcal{F}_o, \tag{18}$$

where $\mathbf{X}^+$ denotes the Moore-Penrose pseudo-inverse of the system matrix $\mathbf{X}$.

2.2 Construction of Constant Speed Cartesian DMPs

For incremental kinesthetic guidance, it is more advantageous if the trajectory has a constant speed profile. See Section 3 for the explanation why this is useful. In this section, we provide a construction to transform the demonstrated trajectory into a Cartesian DMP with constant speed profile. For this purpose, we first re-sample the trajectory at constant arc length steps. A similar idea was exploited by Gašpar et al. [23] to develop arc length dynamic movement primitives (AL-DMPs), which are advantageous for statistical skill learning. Here instead of changing the DMP formulation, we replace the originally demonstrated trajectory with a new trajectory with constant speed profile. We refer to the resulting CDMP trajectory as Constant Speed Cartesian DMP (CS-CDMP).

The arc length parameter for a position and orientation trajectory is defined as

$$l(t) = \int_0^t \left(||\dot{\mathbf{p}}(u)|| + \zeta||\boldsymbol{\omega}(u)||\right) du. \tag{19}$$

The overall arc length of the Cartesian space DMP defined by data (1) is given by $L = l(t_T)$. Note that unlike in [23], l contains in addition to the standard arc length of position trajectory also the angle length of orientation trajectory. ζ is a positive scalar used to scale the angles, which must be done because of different units. To estimate the arc length of a Cartesian space DMP, we integrate the Cartesian space DMP at a high frequency (usually higher than the servo rate of the robot controller) and obtain a new set of discrete measurement $\{\widehat{\mathbf{p}}_k, \widehat{\mathbf{q}}_k\}$. The arc length can then be estimated as

$$l_k = l_{k-1} + \Delta l_k, \tag{20}$$

$$\Delta l_k = ||\widehat{\mathbf{p}}_k - \widehat{\mathbf{p}}_{k-1}|| + 2\zeta||\log(\widehat{\mathbf{q}}_k * \overline{\widehat{\mathbf{q}}}_{k-1})||. \tag{21}$$

We obtain the pairs $\{\widehat{t}_k, l_k\}_{k=1}^{\widehat{T}}$, where $l_{\widehat{T}} = L$ and $\widehat{T} > T$. Assuming that $l_k > l_{k-1}$, $\forall k$, we can interpolate time as the function of arc length $t(l)$, $0 \leq l \leq L$, e.g. using cubic spline interpolation.

Next we select the desired constant arc length sampling step $\Delta l > 0$ and an integer $\widetilde{T} > 0$ so that $L = \widetilde{T}\Delta l$. By using the previously calculated interpolation function $t(l)$, we can compute the times $\widetilde{t}_k = t(k\Delta l)$, $k = 0, \ldots, \widetilde{T}$, when the original DMP reaches the constant arc length steps. By integrating the original Cartesian space DMP to the newly calculated times $\widetilde{t}_k$, we obtain a new set of positions and orientations

$$\{\widetilde{\mathbf{p}}_k, \widetilde{\mathbf{q}}_k\}. \tag{22}$$

This new set of positions and orientations is sampled at constant arc length step

$$\Delta l = ||\widetilde{\mathbf{p}}_k - \widetilde{\mathbf{p}}_{k-1}|| + 2\zeta||\log(\widetilde{\mathbf{q}}_k * \overline{\widetilde{\mathbf{q}}}_{k-1})||, \forall k.$$

The next step is to replace the originally demonstrated speed profile $\dot{l}(t)$ with the constant speed profile $\dot{l}_d$. This can be achieved everywhere except at both ends of the trajectory where the velocities and accelerations must be zero to ensure that the robot is at rest at the beginning and the end of motion. We first determine the arc length $L_1 = \tilde{k}\Delta l$ at which the robot should reach the desired speed $\dot{l}_d$. The number of initial steps $\tilde{k}$ and the desired speed $\dot{l}_d$ must be specified by the user. We then specify the arc length parameterization for the initial interval using a fifth order polynomial $\widetilde{l}_b$ with the following boundary conditions

$$\begin{aligned} \widetilde{l}_b(0) &= \dot{\widetilde{l}}_b(0) = \ddot{\widetilde{l}}_b(0) = \ddot{\widetilde{l}}_b(T_1) = 0, \\ \widetilde{l}_b(T_1) &= L_1, \ \dot{\widetilde{l}}_b(T_1) = \dot{l}_d, \end{aligned} \tag{23}$$

where T_1 is a user provided time in which the acceleration phase of motion should finish. Similarly, the arc length for the final interval is parametrized with the fifth order polynomial $\widetilde{l}_e$ with the following boundary conditions

$$\begin{aligned} \dot{\widetilde{l}}_e(T_2) &= \ddot{\widetilde{l}}_e(T_2) = \ddot{\widetilde{l}}_e(T_2 - T_1) = 0, \\ \widetilde{l}_e(T_2) &= L, \\ \widetilde{l}_e(T_2 - T_1) &= L - L_1, \dot{\widetilde{l}}_e(T_2 - T_1) = \dot{l}_d, \end{aligned} \tag{24}$$

where $T_2 = 2T_1 + (L - 2L_1)/\dot{l}_d$. The complete arc length is parameterized as follows

$$\widetilde{l}(t) = \begin{cases} \widetilde{l}_b(t), & 0 \leq t \leq T_1 \\ L_1 + (t - T_1)\dot{l}_d, & T_1 \leq t \leq T_2 - T_1 \\ \widetilde{l}_e(t), & T_2 - T_1 \leq t \leq T_2 \end{cases} \tag{25}$$

Note that the above parametrization ensures that the speed profile is constant on the interval $T_1 \leq t \leq T_2 - T_1$ and that the robot continuously accelerates and decelerates to the desired speed and acceleration at the beginning and the end of motion.

In the final step we compute a new Cartesian space DMP with the provided speed profile. For this purpose we need to invert

$$\widetilde{t}_k = \widetilde{l}^{-1}(k\Delta l),\ k = 0, \ldots, \widetilde{T}. \tag{26}$$

It is trivial to invert (25) for $L_1 \leq k\Delta l \leq L - L_1$ since on this interval, the arc length is a linear function of time. For $0 \leq k\Delta l \leq L_1$ and $L - L_1 \leq k\Delta l \leq L$, we have to invert a fifth order polynomial. Bisection or Newton's method can be used for this purpose as it is known that the solution lies between $0 \leq t \leq T_1$ and $T_2 - T_1 \leq t \leq T_2$, respectively. The newly computed times $\widetilde{t}_k$ are added to (22) to form a new sampling set

$$\widetilde{\mathcal{G}}_d = \{\widetilde{\mathbf{p}}_k, \widetilde{\mathbf{q}}_k, \widetilde{t}_k\}_{k=1}^{\widetilde{T}}. \tag{27}$$

This set and numerically computed derivatives are then used to compute the Cartesian DMP with constant speed profile (except at both ends). For the sake of clarity, in the rest of this paper we omit the tilde modifier when referring to CS-CDMP computed from data (27).

2.3 Reversible Cartesian Space DMPs

Speed scaling factor $\nu(s)$ has an important impact on the trajectory generated by the DMP. Positive $\nu(s)$ scales the trajectory velocity in its original direction, setting $\nu(s) = 0$ stops the motion and negative $\nu(s)$ reverses the trajectory. However, reversing the trajectory in this way causes DMP stability issues. Namely, if we omit the nonlinear term[1] $\mathbf{f}_p(s)$ and assume $\nu(s) = 1$, the solution of the linear dynamic equation system (2)–(3) is given by

$$\begin{bmatrix} \mathbf{z} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{g} \end{bmatrix} + e^{t\mathbf{A}}\mathbf{c},\ \mathbf{A} = \frac{1}{\tau}\begin{bmatrix} -\alpha_z \mathbf{I}_3 & -\alpha_z\beta_z\mathbf{I}_3 \\ \mathbf{I}_3 & 0 \end{bmatrix},\ \mathbf{c} \in \mathbb{R}^6, \tag{28}$$

where $\mathbf{c}$ is a constant vector specified in such a way that the initial conditions $\mathbf{p}(s(0)) = \mathbf{p}_0$, $\mathbf{z}(s(0)) = 0$, are fulfilled. By setting $\alpha_z = 4\beta_z > 0$, the system becomes critically damped because the matrix $\mathbf{A}$ has six equal negative eigenvalues $\lambda_{1,\ldots,6} = -\alpha_z/2\tau < 0$. Thus the exponential part of the solution vanishes as $t \mapsto \infty$ and the system is stable. On the other hand, if we reverse the trajectory by setting $\nu(s) = -1$, the trajectory is given by

$$\begin{bmatrix} \mathbf{z} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{g} \end{bmatrix} + e^{-t\mathbf{A}}\mathbf{c}, \tag{29}$$

which means that the system is unstable because unlike $e^{t\mathbf{A}}$, $e^{-t\mathbf{A}}$ does not vanish as $t \mapsto \infty$. Consequently, even small disturbances in the DMP (e.g. the DMP integration noise) might result in unstable trajectory generation[2]. It follows that we need to keep $\nu(s)$ positive in order to ensure the stability of the system.

To avoid instabilities, we create two sets of DMPs, one for the forward direction and the other for the reverse direction as we proposed in [21]. The parameters of the reverse DMP, denoted as $\overline{\text{DMP}}$, are computed for policy $\overline{\pi}(t) = \pi(\tau - t)$, where $\overline{\pi}$ denotes the reverse policy and π the forward policy. In order to assure smooth switching between the two DMPs at phase s, we need to initialize the internal variables according to the following formulas

$$\overline{\mathbf{z}} = -\mathbf{z}, \tag{30}$$

$$\overline{\boldsymbol{\eta}} = -\boldsymbol{\eta}, \tag{31}$$

$$\overline{s} = \frac{e^{-\alpha_z}}{s}, \tag{32}$$

$$\overline{\nu}(\overline{s}) = \nu(s), \tag{33}$$

where overlined variables $\overline{\times}$ belong to the $\overline{\text{DMP}}$. Note that $\mathbf{p}$ and $\mathbf{q}$ remain unchanged at the time instance when we switch from DMP to $\overline{\text{DMP}}$ and vice versa. Iturrate et al. [29] proposed another approach for reversible DMPs, where the linear second-order differential equation system defining the fixed part of DMP was replaced with a nonlinear logistic differential equation. Such a system is stable in both directions within the initial and goal point, but becomes unstable outside this region. As a result, all points on the trajectory must lie between the initial and goal point, which limits the practical applicability of their approach.

3 Incremental Refinement of the Control Policy using Kinesthetic Guidance

In this section, we describe techniques applied to modify an existing robot policy incrementally. In doing so, the operator moves the robot at any speed forward and backward along the previously learned trajectory and changes only those parts of the path where changes are necessary. We assume that the desired trajectory has been encoded as a constant speed profile, reversible CS-CDMP, as presented in Sections 2.2 and 2.3.

[1]The nonlinear term acts as a feed-forward term and does not influence the closed-loop stability of the DMP.

[2]By using a more precise method to integrate (2) – (6), the reverse integration of the original DMP might remain stable for a longer time.

In our framework, we do not change the policy encoded with CS-CDMP directly. Rather, we learn an offset from the policy. This phase dependent offset, here denoted as $\mathbf{d}(s) \in \mathbb{R}^6 = [\mathbf{d}_p(s)^{\mathrm{T}}, \mathbf{d}_o(s)^{\mathrm{T}}]^{\mathrm{T}}$, will be encoded as a linear combination of RBFs as defined in Eqs. (11) – (12). In this way, we avoid recomputing CS-CDMP and CS-CDMP parameters in the iterations of the refining process. Further, we eliminate the influence of the DMP integration noise, which would accumulate during the iterative policy improvements.

The pose sent to the robot controller is thus specified as

$$\begin{bmatrix} \mathbf{p}_c \\ \mathbf{q}_c \end{bmatrix} = \begin{bmatrix} \mathbf{p}(s) + \mathbf{d}_p(s) \\ \exp\left(\frac{1}{2}\mathbf{d}_o(s)\right) * \mathbf{q}(s) \end{bmatrix}, \tag{34}$$

where $\mathbf{p}(s)$ and $\mathbf{q}(s)$ are obtained from CS-CDMP integration using (2), (3) and (6). Position and orientation offsets $\mathbf{d}_p(s)$ and $\mathbf{d}_o(s)$ are calculated as a weighted sum of radial basis function (RBFs)

$$\mathbf{d}_p(s) = (\mathbf{x}(s)\mathbf{W}'_p)^{\mathrm{T}}, \tag{35}$$

$$\mathbf{d}_o(s) = (\mathbf{x}(s)\mathbf{W}'_o)^{\mathrm{T}}, \tag{36}$$

similarly as in (9) and (10). Prior to learning we have to initialize the RBF weights $\mathbf{W}'_p$, $\mathbf{W}'_o$, to zero, meaning that no offsets are learned yet.

For clarity, we refer to the original demonstrated trajectory encoded with CS-CDMP and the corresponding offsets RBF as the nominal policy π in the remainder of this paper.

3.1 Mechanism for kinesthetic guidance along the policy

To allow the operator to move the robot forward and backward along a policy π in an intuitive way and change the speed of motion (see Fig. 1), we associate the speed scaling factor ν of CS-CDMPs with a force projected onto the tangential direction of the policy.

The tangential direction of the policy is given by

$$\mathbf{t}_p(s) = \begin{cases} \dfrac{\dot{\mathbf{p}}_c(s)}{\|\dot{\mathbf{p}}_c(s)\|}, & \|\dot{\mathbf{p}}_c(s)\| \neq 0 \\ [0,0,0]^{\mathrm{T}}, & \text{otherwise} \end{cases}, \tag{37}$$

where $\dot{\mathbf{p}}_c = \dot{\mathbf{p}} + \dot{\mathbf{d}}_p$ are the velocities obtained by integrating the CS-CDMP and RBF at the current phase s. The corresponding direction for the rotational motion is given by

$$\mathbf{t}_o(s) = \begin{cases} \dfrac{\boldsymbol{\omega}_c(s)}{\|\boldsymbol{\omega}_c(s)\|}, & \|\boldsymbol{\omega}_c(s)\| \neq 0 \\ [0,0,0]^{\mathrm{T}}, & \text{otherwise} \end{cases} \tag{38}$$

where $\boldsymbol{\omega}_r = \boldsymbol{\omega} + \boldsymbol{\omega}_d$ is the robot's end-effector angular velocity. CS-CDMP angular velocity $\boldsymbol{\omega}$ is calculated by

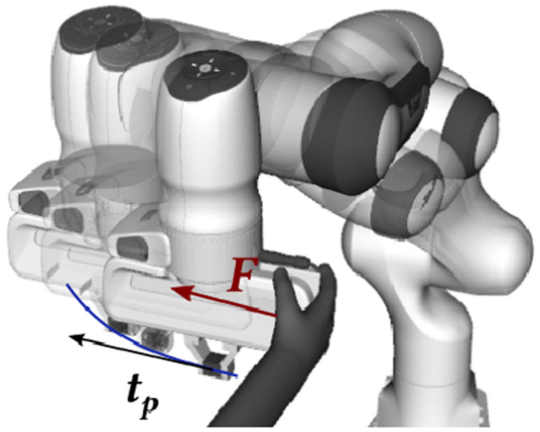


Fig. 1 Guiding along the policy (blue curve) is possible by pushing the robot with force $\mathbf{F}$ in the tangential direction of the policy (denoted with $\mathbf{t}_p$)

integrating (4) – (6) and taking into account the relation $\boldsymbol{\omega} = \boldsymbol{\eta}/\tau$. RBF angular velocity is $\boldsymbol{\omega}_d(s) = \mathbf{d}_o(s)/\tau$. Based on tangential directions and the measured forces and torques, a new speed scaling factor is calculated using

$$\tilde{\nu} = \mu_p \,\mathrm{sgm}(\|\mathbf{F}\|)\mathbf{F} \cdot \mathbf{t}_p(s) + \mu_o \,\mathrm{sgm}(\|\mathbf{M}\|)\mathbf{M} \cdot \mathbf{t}_o(s), \tag{39}$$

where $\cdot$ denotes dot product and μ_p, μ_o are positive scalars used to scale the velocities of the translational and rotational motion along the desired path. sgm() denotes a sigmoid function. In our experiments, we used logistic function given by

$$\mathrm{sgm}(x) = \frac{1}{1 + e^{-ax}}. \tag{40}$$

It ensures a smooth start and stop when pushing along the policy and effectively solves the jittery robot motion due to noisy input from force sensor. The constant scalar a was chosen empirically and determines the softness of the transition. Generally, it should be decreased with increased force-torque sensor noise.

$\mathbf{F} \in \mathbb{R}^3$ and $\mathbf{M} \in \mathbb{R}^3$ denote the Cartesian forces and torques in robot base coordinate system. Since forces and torques are usually measured at the end-effector, the following transformation is used to transform them in the robot base coordinate system

$$(0, \mathbf{F}) = \bar{\mathbf{q}} * (0, \mathbf{F}_m) * \mathbf{q}, \tag{41}$$

$$(0, \mathbf{M}) = \bar{\mathbf{q}} * (0, \mathbf{M}_m) * \mathbf{q} \tag{42}$$

where $\mathbf{F}_m \in \mathbb{R}^3$ and $\mathbf{M}_m \in \mathbb{R}^3$ are the measured forces and torques at the end-effector and $\mathbf{q}$ is the unit quaternion describing the current orientation of the robot's end-effector.

If $\tilde{\nu}$ becomes negative, we switch the direction of motion along the desired path specified by the policy π. Instead of following the policy specified by π, we start using the reverse policy $\overline{\pi}$ with the initialization provided by Eqs. (30) – (33). Thus we can always use a positive speed scaling factor for CS-CDMP integration, i. e. $\nu(s) = |\tilde{\nu}|$. By applying this procedure, the guiding process becomes natural and intuitive, as the operator simply pushes the robot forth and back in the tangential directions of the path. Namely, when both $\mathbf{F} \cdot \mathbf{t}_p(s)$ and $\mathbf{M} \cdot \mathbf{t}_o(s) \to 0$, $\nu(s) \to 0$, which slows down the robot motion and eventually stops it. Using $\nu(s)$ in (2) – (6), the robot moves along the policy $\pi(s)$ in the direction of the applied forces and torques, with the speed proportional to them.

Since robot motion is modified along the policy based on the measured forces and torques, the robot should be stiff in the tangential direction of the policy. If the robot is not equipped with a force-torque sensor but is compliant and we know its stiffness, we can still implement the proposed approach by computing virtual forces and torques as

$$\mathbf{F} = \mathbf{K}_p(\mathbf{p}_m - \mathbf{p}_c), \tag{43}$$

$$\mathbf{M} = \mathbf{K}_o 2\log(\mathbf{q}_m * \overline{\mathbf{q}_c}), \tag{44}$$

where $\mathbf{K}_p \in \mathbb{R}^{3\times3}$ $\mathbf{K}_o \in \mathbb{R}^{3\times3}$ are position and orientation stiffness matrices. $\mathbf{p}_m$, $\mathbf{q}_m$ respectively denote the actual (measured) robot position and orientation whereas $\mathbf{p}_c$ and $\mathbf{q}_c$ denote the commanded position and orientation as calculated in (34). The virtual forces and toques can be used in (39) to calculate the speed scaling factor. However, this method requires that the robot is compliant and can not be applied on many standard industrial robots that are typically stiff.

3.2 Changing the Desired Path

So far, we have proposed a method that allows the motion along the policy by kinesthetic guidance. To modify the desired path, the robot has to enable movements that deviate from the desired path. Two cases will be considered here: a) the robot is impedance-controlled and it is possible to set the stiffness and damping matrices, and b) it is not possible to adjust the stiffness and damping matrices but the robot is equipped with a force-torque sensor.

In the first case, the robot has to be compliant in the plane attached to the current robot position and orthogonal to the trajectory direction. In the continuation of the paper, we will refer to this plane as the *refinement* plane (see Fig. 2). A suitable choice to define such a plane is given by Frenet-Serret formulas [30]

$$\mathbf{R}_t = \begin{bmatrix} \mathbf{t} & \mathbf{n} & \mathbf{b} \end{bmatrix}, \tag{45}$$

where tangential $\mathbf{t}$, normal $\mathbf{n}$, and bi-normal coordinate axis $\mathbf{b}$ are determined as

$$\mathbf{t} = \frac{\dot{\mathbf{p}}_c}{\|\dot{\mathbf{p}}_c\|}, \ \mathbf{b} = \frac{\dot{\mathbf{p}}_c \times \ddot{\mathbf{p}}_c}{\|\dot{\mathbf{p}}_c \times \ddot{\mathbf{p}}_c\|}, \ \mathbf{n} = \mathbf{b} \times \mathbf{t}. \tag{46}$$

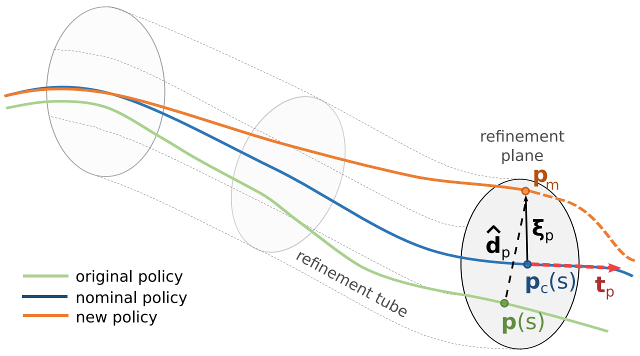


Fig. 2 Refinement tube is formed around the nominal policy shown by the blue curve. Position $\mathbf{p}_c(s)$, generated from this policy at the current phase s, already includes offset from the original position $\mathbf{p}(s)$ generated from the original policy shown by the green curve. Orange curve shows a policy with newly sampled offsets. Current displacement $\boldsymbol{\xi}_p$ from the nominal point is projected to refinement plane shown with a gray circle. Full offset to the original point $\widehat{d}_p$ is shown with a dashed line. t_p denotes tangential direction with respect to the nominal policy

The refinement plane is spanned by the normal and bi-normal axes. It is necessary to ensure that the Frenet-Serret (FS) frame does not change abruptly. Since linear velocity $\dot{\mathbf{p}}_c$ and acceleration $\ddot{\mathbf{p}}_c$ are obtained from CS-CDMP and RBF, they are guaranteed to be smooth. Note that the FS frame can not be calculated when the robot's velocity and/or acceleration are equal to zero. Therefore, we suspend the updating of $\mathbf{R}_t$ at low speeds and accelerations until the motion becomes faster again.

To arbitrary set the compliance along any axis of the FS frame, the stiffness matrices of the impedance control are calculated as [31]

$$\mathbf{K}_p = \mathbf{R}_t \mathbf{K}'_p \mathbf{R}_t^{\mathrm{T}}, \tag{47}$$

$$\mathbf{K}_o = \mathbf{R}_t \mathbf{K}'_o \mathbf{R}_t^{\mathrm{T}}. \tag{48}$$

$\mathbf{K}'_p$ and $\mathbf{K}'_o$ are diagonal matrices that define compliance along tangential, normal and binormal axis defined by the FS frame. To obtain a critically damped response of the robot, it is also necessary modify the damping matrices

$$\mathbf{D}_p = 2\sqrt{\mathbf{K}_p}, \tag{49}$$

$$\mathbf{D}_o = 2\sqrt{\mathbf{K}_o}. \tag{50}$$

In this way, the displacement from the nominal trajectory π in the refinement tube is proportional to the applied force projected to the refinement plane and chosen stiffness $\mathbf{K}'_p$ and $\mathbf{K}'_o$.

The offset $\widehat{\mathbf{d}}(s) \in \mathbb{R}^6$ from the originally demonstrated path is calculated as follows

$$\widehat{\mathbf{d}}(s) = \begin{bmatrix} \mathbf{p}_m - \mathbf{p}(s) \\ 2\log(\mathbf{q}_m * \bar{\mathbf{q}}(s)) \end{bmatrix}, \tag{51}$$

where $\mathbf{p}_m$, $\mathbf{q}_m$ is the measured robot pose and $\mathbf{p}(s)$, $\mathbf{q}(s)$ the CS-CDMP pose.

Incremental learning can be also applied to robots that are not compliant, but are equipped with a six-axis force-torque sensor at the end-effector. In this case, we first calculate a virtual displacement vector $\boldsymbol{\xi}$ from the applied forces and torques projected onto the refinement plane as

$$\boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{\xi}_p \\ \boldsymbol{\xi}_o \end{bmatrix} = \mathbf{K}_g \begin{bmatrix} \mathbf{F} - (\mathbf{F} \cdot \mathbf{t}_p(s))\mathbf{t}_p(s) \\ \mathbf{M} - (\mathbf{M} \cdot \mathbf{t}_o(s))\mathbf{t}_o(s) \end{bmatrix}, \tag{52}$$

where $\mathbf{K}_g \in \mathbb{R}^{6\times 6}$ is a diagonal gain matrix with positive coefficients. In compassion with (34), the virtual displacement vector is added to the pose sent to the robot controller

$$\begin{bmatrix} \mathbf{p}_c \\ \mathbf{q}_c \end{bmatrix} = \begin{bmatrix} \mathbf{p}(s) + \mathbf{d}_p(s) + \boldsymbol{\xi}_p \\ \exp(\frac{1}{2}\boldsymbol{\xi}_o) * \exp(\frac{1}{2}\mathbf{d}_o(s)) * \mathbf{q}(s) \end{bmatrix}. \tag{53}$$

As the robot is commanded to also account for the virtual displacement calculated in (52), we can still sample $\widehat{\mathbf{d}}(s)$ using (51).

By adjusting the compliance parameters in the refinement plane, we can prevent the robot from deviating too much from the existing trajectory. With increasing displacement from the existing trajectory, the operator feels increasing force directed towards the reference trajectory. This force prevents the operator to unintentionally modify the path when guiding the robot along the existing trajectory. Thus, major trajectory changes can be learned gradually over several iterations of moving back and forth and updating offsets trajectory.

Papageorgiu et al. [18] proposed an alternative method where a dedicated controller was used to realize a penetrable virtual wall around the trajectory. In their approach, the virtual wall pushes the robot towards the reference trajectory, but the robot motion is not constrained when the operator penetrates the virtual wall. In this way, the operator can achieve major trajectory changes even in a single pass. In our framework, a similar behavior can be achieved by setting the robot's stiffness $\mathbf{K}'_p$ (47) for normal and bi-normal directions in the refinement plane as follows:

$$\mathbf{K}'_p = \begin{bmatrix} K'_t & 0 & 0 \\ 0 & K'_p & 0 \\ 0 & 0 & K'_p \end{bmatrix},$$
$$K'_p = \frac{K'_{p,1}}{1 + e^{|\mathbf{d}_p|a-b}} + K'_{p,0}, \tag{54}$$

where $K'_{p,1}$ and $K'_{p,0}$ are positive scalars that define the upper and lower bound of the refinement tube, respectively. Scalars a and b set the shape of the tube. K'_t sets the desired stiffness in the tangential direction. An example of the refinement tube and the operator's corresponding forces are shown in Fig. 3. Note that when updating the compliance gains (47) and (48), the corresponding damping coefficients must also be set according to (49) and (50), respectively.

The above described procedure allows to change the desired path only on segments where changes are necessary. The set of sampled displacements is used to compute a new displacement RBF. For this, we consider two approaches: batch regression and recursive regression.

3.3 Batch Regression Based Policy Update

This method is very similar to the classic trajectory learning by kinesthetic guidance except that here we sample the offsets caused by human guidance while the robot attempts to track the policy π. The resulting deviations from the desired trajectory are influenced by the compliance settings in the normal and binormal direction of the FS frame.

We denote by $\mathcal{D}$ a set of offsets (51) sampled along the CS-CDMP with the constant arc length step, just like the set (27) (for the sake of clarity, we omit the tildes)

$$\mathcal{D} = \{\mathbf{d}_{p,j}, \mathbf{d}_{o,j}, s_j\}_{j=1}^{T}. \tag{55}$$

$\mathbf{d}_{p,j}, \mathbf{d}_{o,j} \in \mathbb{R}^3$ are the position and orientation offsets, respectively. Initially we set $\mathbf{d}_{p,j} = \mathbf{d}_{o,j} = 0, \forall j$. We encode the offsets as the function of phase s along the desired CS-CDMP, as specified in Eqs. (35) – (36). The commanded (desired) trajectory is then given by Eq. (34).

We start collecting new offsets once the robot motion deviates from the commanded trajectory (34). We update the displacement set (55) whenever we change the direction of motion. Note that the demonstrator can change only a part of the trajectory. Let

$$\widehat{\mathcal{D}} = \{\widehat{\mathbf{d}}_j\}_{j=1}^{h}, \; \widehat{\mathbf{d}}_j = [\widehat{\mathbf{d}}_{p,j}, \; \widehat{\mathbf{d}}_{o,j}] \tag{56}$$

denote the new offsets collected from the time when the robot started to deviate from the nominal trajectory until the robot rejoined it. Just like (27), they are sampled with constant arc length steps, which enables us to match the data from $\mathcal{D}$ and $\widehat{\mathcal{D}}$ along the arc length instead of phase. Next we determine which offsets from (55) should be replaced with offsets from (56). The first offset $\mathbf{d}_s = [\mathbf{d}_{p,s}, \mathbf{d}_{o,s}]$ from $\mathcal{D}$ to be replaced is known because we constantly monitor if the robot deviates from the commanded trajectory. The final offset $\mathbf{d}_e = [\mathbf{d}_{p,e}, \mathbf{d}_{o,e}]$ is determined by finding a pose in $\mathcal{D}$, which is closest to the last pose from $\widehat{\mathcal{D}}$, denoted as $\widehat{\mathbf{d}}_h = [\widehat{\mathbf{d}}_{p,h}, \; \widehat{\mathbf{d}}_{o,h}]$. To calculate the nearest displacement, we minimize

$$e = \arg\min_{j=s,\dots,T}\{|(\widehat{\mathbf{d}}_{p,h} - \mathbf{d}_{p,j}) \cdot \mathbf{t}_{p,j}| + \zeta |(\widehat{\mathbf{d}}_{o,h} - \mathbf{d}_{o,j}) \cdot \mathbf{t}_{o,j}|\}, \tag{57}$$

where $\mathbf{t}_{p,j}$ and $\mathbf{t}_{o,j}$ are the tangents calculated at the j-th sample using (37) and (38). Once e is small enough, we stop accumulating corrected offsets since the actual robot motion rejoined the nominal trajectory.

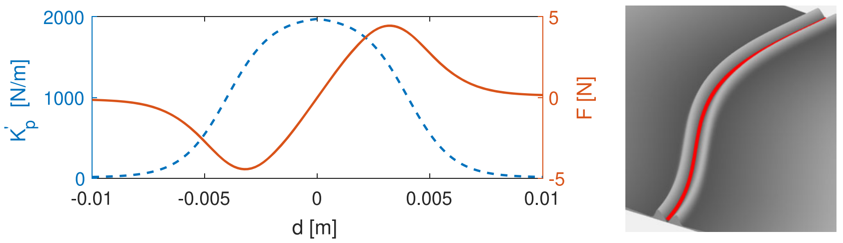


Fig. 3 Left: Refinement tube for the following settings: $K'_{p,1} = 2000$, $K'_{p,0} = 10$, $a = 1000$, $b = 4$. Blue dotted curve shows the control gain and red curve the forces felt by the operator. d is the deviation from the existing trajectory. Right: Virtual wall around the path (red curve). The force F felt by the operator is proportional to the wall height (dark shades – lower forces, light shades – higher forces)

The data between $\mathbf{d}_s = [\mathbf{d}_{p,s}, \ \mathbf{d}_{o,s}]^{\mathrm{T}}$ and $\mathbf{d}_e = [\mathbf{d}_{p,e}, \ \mathbf{d}_{o,e}]^{\mathrm{T}}$ in the existing set $\mathcal{D}$ can now be replaced with the data $\widehat{\mathcal{D}}$, as illustrated in Fig. 4. Similarly to (18), the final step is to calculate the new RBF parameters for the offset trajectory (35) – (36) using the updated data $\mathcal{D}$ and batch regression technique.

The proposed approach enables the operator to demonstrate parts of the trajectory multiple times. If a part of the trajectory has not been executed correctly, the operator can change the direction of motion by pushing the robot in the opposite direction along the trajectory. Note that the speed scaling factor $\tilde{\nu}$ defined in (39) is positive as long as the user pushes the robot forwards along the trajectory. We switch between the two CS-CDMPs forming the reverse CS-CDMP whenever $\tilde{\nu}$ becomes negative. The offset trajectory encoded with RBFs can be reversed with a negative speed scaling factor $\tilde{\nu}$, which does not cause any stability problems because there is no integration.

Pseudo code for BR-based incremental learning is given as Algorithm 1. In this algorithm, function join$(\cdot)$ denotes insertion of sampled set of poses $\widehat{\mathcal{D}}$ into $\mathcal{D}$, as described above.

The sampling procedure presented above differs from the approach in [20], where we used the phase to match the poses on the existing trajectory with the current robot poses. Due to pushing along the trajectory, a tracking error can occur, which causes that the phase of the reference trajectory is not aligned with the current robot motion. Consequently, the sampled poses might be inserted to $\mathcal{G}_d$ with wrong indices, which can significantly corrupt the modified trajectory, as illustrated in Fig. 5. In this paper, we solved this problem by sampling the offsets with a constant arc-length and matching the poses along the arc length. Papageorgiou et al. [18] also addressed this issue by finding the phase where the existing trajectory is closest to the current robot pose. However, unlike our approach, their procedure does not guarantee that the points are equidistant.

Algorithm 1: Algorithm for BR based incremental learning.

```
1  input: CDMP described with a set of parameters
      ${\mathbf{g}, \mathbf{w}, \tau}$, desired arc length $\Delta l$ between two samples
2  calculate CS-CDMP and $\overline{\text{CS-CDMP}}$ parameters
3  initialize offsets RBF, $\mathcal{D}$ and $\widehat{\mathcal{D}}$
4  while refining do
5      calculate FS frame for current phase using (45)
6      apply new stiffness and damping matrices using
         (47)–(50)
7      if impedance mode then calculate virtual force
         using (43)
8      calculate speed scaling factor $\tilde{\nu}$ using (39)
9      calculate $\Delta l_k$ using (21)
10     if $\Delta l_k \geq \Delta l$ then
11         sample displacement $\widehat{\mathbf{d}}_j$ as (51)
12         $\widehat{\mathcal{D}} \leftarrow \widehat{\mathcal{D}} \cup \mathbf{d}_j$
13     if $\tilde{\nu}$ changes sign then
14         $\mathcal{D} = \text{join}(\mathcal{D}, \widehat{\mathcal{D}})$ and reset $\widehat{\mathcal{D}}$
15         calculate RBF from $\mathcal{D}$
16         calculate CS-CDMP or $\overline{\text{CS-CDMP}}$ states
             using (30)–(33)
17     integrate CS-CDMP to get $[\mathbf{p}, \mathbf{q}]$ and RBF to get
         $[\mathbf{d}_p, \mathbf{d}_o]$
18     if admittance mode then calculate $[\mathbf{p}_c, \mathbf{q}_c]$ using
         (53)
19     else calculate $[\mathbf{p}_c, \mathbf{q}_c]$ using (34)
20     send new reference $[\mathbf{p}_c, \mathbf{q}_c]$ to the robot controller
```

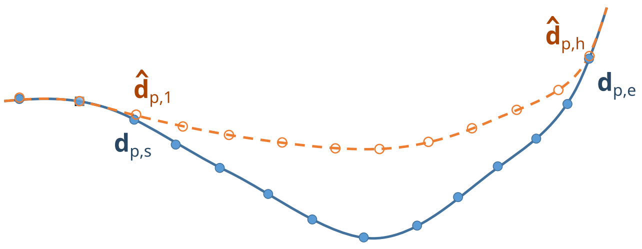


Fig. 4 The blue curve shows current offsets from the original policy, with $\mathbf{d}_{p,s}$ and $\mathbf{d}_{p,e}$ marking the boundaries of the segment to be modified. The orange line shows the modified part of the offset trajectory segment

3.4 Recursive Regression Based Policy Update

Instead of updating the entire offset RBF at each change of the direction, we can concurrently modify weights of the offset RBF, $\mathbf{W}_i = [\mathbf{W}'_{p,i}\ \mathbf{W}'_{o,i}] \in \mathbb{R}^{N\times 6}$, using recursive regression formulas

$$\mathbf{W}_i = \mathbf{W}_{i-1} + \mathbf{P}_i \mathbf{x}(s_j)^{\mathrm{T}} \widehat{\mathbf{e}}(s_j)^{\mathrm{T}}, \tag{58}$$

$$\mathbf{P}_i = \frac{1}{\lambda}\left(\mathrm{P}_{i-1} - \frac{\mathrm{P}_{i-1}\mathbf{x}(s_j)^{\mathrm{T}}\mathbf{x}(s_j)\mathrm{P}_{i-1}}{\lambda + \mathbf{x}(s_j)\mathrm{P}_{i-1}\mathbf{x}^{\mathrm{T}}(s_j)}\right), \tag{59}$$

where i is the iteration index of recursive regression, $\mathbf{P}_i \in \mathbb{R}^{N\times N}$ is the covariance matrix of parameters $\mathbf{W}_i$, $\mathbf{x}(s_j) \in \mathbb{R}^{1\times N}$ is a vector of Gaussian kernel functions (11), N is the number of kernel functions, and λ is the forgetting factor, which is usually set to a value close but smaller or equal to 1. s_j, s_{j-1} denote the phase at the current and previous iteration step, respectively. $\widehat{\mathbf{e}}(s_j)$ denotes the displacement between the robot commanded pose and measured robot pose at phase s_j. Here, it is crucial to estimate the phase s_j, which corresponds to the measured current robot pose. Due to the robot compliance, this phase varies from the CS-CDMP and RBF phase (6) used to calculate the desired robot pose. It is calculated by projecting the measured robot pose to the existing robot trajectory and solving the following optimization problem

$$s_j = \arg\min_{s_k} |(\mathbf{p}_m - \mathbf{p}_r(s_k))\cdot \mathbf{t}_p(s_k)| + 2\zeta\, |\log(\mathbf{q}_m * \bar{\mathbf{q}}_r(s_k))\cdot \mathbf{t}_o(s_k)|, \tag{60}$$

where $s_k = s_i e^{-\frac{\alpha_s k \delta t}{\tau}}, \{k = 0 \ldots, h\}$, s_i is the current trajectory phase, δt is suitably chosen time interval and h denotes the search horizon. Once solving this optimization problem, we calculate the displacement vector as

$$\mathbf{e}(s_j) = \begin{bmatrix} \mathbf{p}_m - \mathbf{p}_c(s_j) \\ 2\log(\mathbf{q}_m * \bar{\mathbf{q}}_c(s_j)) \end{bmatrix}. \tag{61}$$

With a smaller λ the previous measurements are quickly forgotten, or in other words, more recent samples have a greater impact on the estimated trajectory shape. On the other hand, with $\lambda = 1$ the covariance matrix $\mathbf{P}_i$ becomes monotonically decreasing and the recent trajectory updates have less and less influence on the estimated forcing term weights. To prevent this, it is necessary to reset the covariance error matrix to the default value, $\mathbf{P}_i = \gamma \mathbf{I}$, $\gamma > 0$, whenever the factor $\tilde{\nu}$ changes its sign. γ is usually set to a large value, e. g. 100.

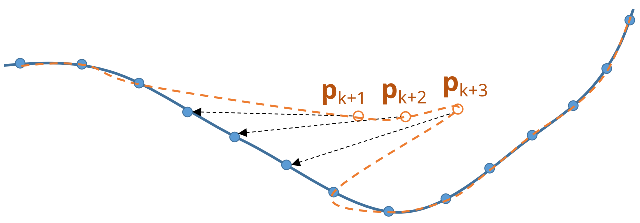


Fig. 5 In our previous method [20], we directly replaced the existing positions with the newly sampled positions. This led to incorrect matches between the positions on the edited path and original path, which caused deformations

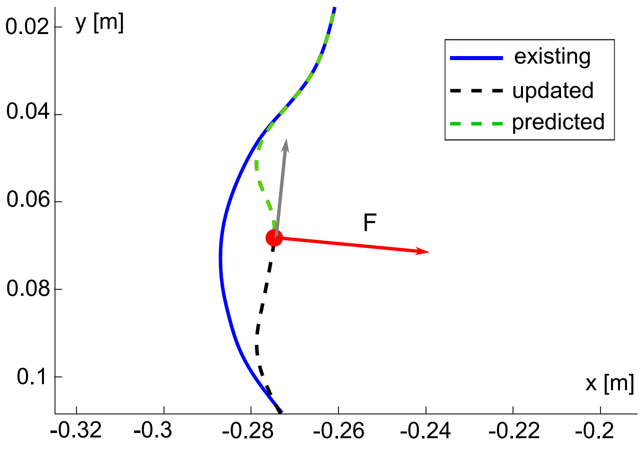


Fig. 6 A 2D plot of the concurrent update of an existing path (blue). The dashed black curve shows the updated path and demonstrates how RR predicts beyond the demonstrated part (green). The operator pushes the robot in the direction of the gray arrow. Red arrow indicates the force needed to displace the robot from the existing trajectory

When changing the reference trajectory by recursive regression, every new displacement vector $\mathbf{e}$ immediately affects the shape of the reference trajectory (see Fig. 6). The dynamics of change is determined by the equations of recursive regression (58) and (59), offset calculation using RBFs (35) and (36), and the robot dynamics. In our previous work [32] we showed that we can approximate this dynamics with a second-order filter. The overall scheme of such a learning process[3] can therefore be described as a feedback loop through a second-order filter, as shown in Fig. 7.

This filter is a low-pass filter whose parameters depend on the RBF parameters. Thus, this filter suppresses only the fast movements of the operator during trajectory refinement. On contrary, the system becomes fully compliant at slow movements.

Note that with recursive regression it is not possible to implement a penetrable virtual wall because the reference trajectory is constantly being adjusted. Instead, we can apply the sigmoid function (40) to the displacement vector $\mathbf{e}(s_j)$ vector, which makes the system stiff at low forces applied in the refinement plane. In this way we prevent unintentional changes of the path. Pseudo code for RR-based incremental learning is given as Algorithm 2.

3.5 Speed Profile Learning

The speed profile is demonstrated by pushing the robot along the trajectory with the desired speed. Using either the

[3]Here we assume that the nonlinear robot dynamics is decoupled and linearized using the impedance control law.

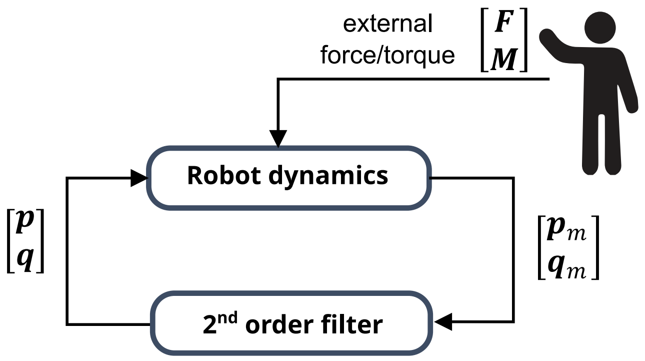

external force/torque
$\begin{bmatrix} F \\ M \end{bmatrix}$
Robot dynamics
$\begin{bmatrix} p \\ q \end{bmatrix}$
$\begin{bmatrix} p_m \\ q_m \end{bmatrix}$
2nd order filter

Fig. 7 The dynamics of learning is governed by the robot system dynamics and the recursive regression (RR) dynamics. Both subsystems can be approximated by a linear second order system

measured forces and torques or their virtual counterparts as defined by (43), ν can be completed computed according to (39) and sampled along with the corresponding phase. The weights $\mathbf{w}_\nu$ of the temporal scaling function defined in (13) are computed using regression techniques once the demonstration stops. This step can be repeated multiple times until the desired speed profile has been obtained. Usually, speed profile learning is the last phase in incremental learning, executed after the spacial part of the policy is refined.

Algorithm 2: Algorithm for RR based incremental learning.

```
1  input: CDMP described with a set of parameters
      ${\mathbf{g}, \mathbf{w}, \tau}$, desired arc length $\Delta l$ between two samples
2  calculate CS-CDMP and $\overline{\text{CS-CDMP}}$ parameters
3  initialize offsets RBF, $\mathcal{D}$ and $\widehat{\mathcal{D}}$
4  while refining do
5      calculate FS frame for current phase using (45)
6      apply new stiffness and damping matrices using
         (47)–(50)
7      if impedance mode then calculate virtual force
         using (43)
8      calculate speed scaling factor $\tilde{\nu}$ using (39)
9      calculate phase (60) and displacement vector (61)
10     update RBF weights using (58)
11     if $\tilde{\nu}$ changes sign then
12         calculate CS-CDMP or $\overline{\text{CS-CDMP}}$ states
             using (30)–(33)
13         reset covariance matrix $\mathbf{P} = \gamma\mathbf{I}$
14     integrate CS-CDMP to get $[\mathbf{p}, \mathbf{q}]$ and RBF to get
         $[\mathbf{d}_p, \mathbf{d}_o]$
15     if admittance mode then calculate $[\mathbf{p}_c, \mathbf{q}_c]$ using
         (53)
16     else calculate $[\mathbf{p}_c, \mathbf{q}_c]$ using (34)
17     send new reference $[\mathbf{p}_c, \mathbf{q}_c]$ to the robot controller
```

4 Evaluation

4.1 Stability of Learning

In the implementation of incremental learning, it is necessary to consider the closed-loop stability of the overall control scheme and how the choice of free parameters influences the behavior of the learning system. We consider four different cases, which differ according to whether the robot is compliant in normal and bi-normal direction of the FS frame or the robot is admittance-controlled and the virtual displacement vector is computed by (52) and also whether the parameters of offset RBF are computed using batch regression (BR) or recursive regression (RR). These four different cases are shown in Fig. 8. For the sake of simplicity, we limit our stability analysis to one positional degree of freedom. In this case, the decoupled robot dynamics is described by a second-order system [33].

Let's first assume that the robot is compliant, i.e. impedance-controlled. The block scheme for the compliant robot and BR (Fig.8a) shows that this is an open-loop control scheme where the dynamics is determined by the dynamics of the robot. $\mathbf{F}_e$ denotes an external force by which the operator guides the robot. The control scheme for the compliant robot and RR (Fig. 8b) is similar. However, here the RR is inside the feedback loop together with the robot dynamics. As previously mentioned in Section 3.4, the RR dynamics can be approximated with a low pass second-order filter [32]. Consequently, the overall dynamics can be approximated with a stable fourth-order linear system. In both arrangements, $\mathbf{F}_e$ changes the robot position $\mathbf{p}_m$ but does not affect the stability.

The external force $\mathbf{F}_e$ changes the robot motion also in the block diagrams of Fig. 8c and 8d, but here the robot is admittance controlled. As long as the robot is not in contact with the environment, the admittance control is in an open-loop and does not affect the stability of the system. However, the contact force $\mathbf{F}_c$ arises once the robot establishes contact with the environment. The admittance control forms feedback through the environment stiffness matrix $\mathbf{K}_s$, as shown in block diagrams of Fig. 8c and 8d. In these diagrams, vector $\mathbf{p}_o$ denotes the environment contact position. Thus, the overall system is non-linear, where the saturation block and the stiffness matrix model the contact forces $\mathbf{F}_c$.

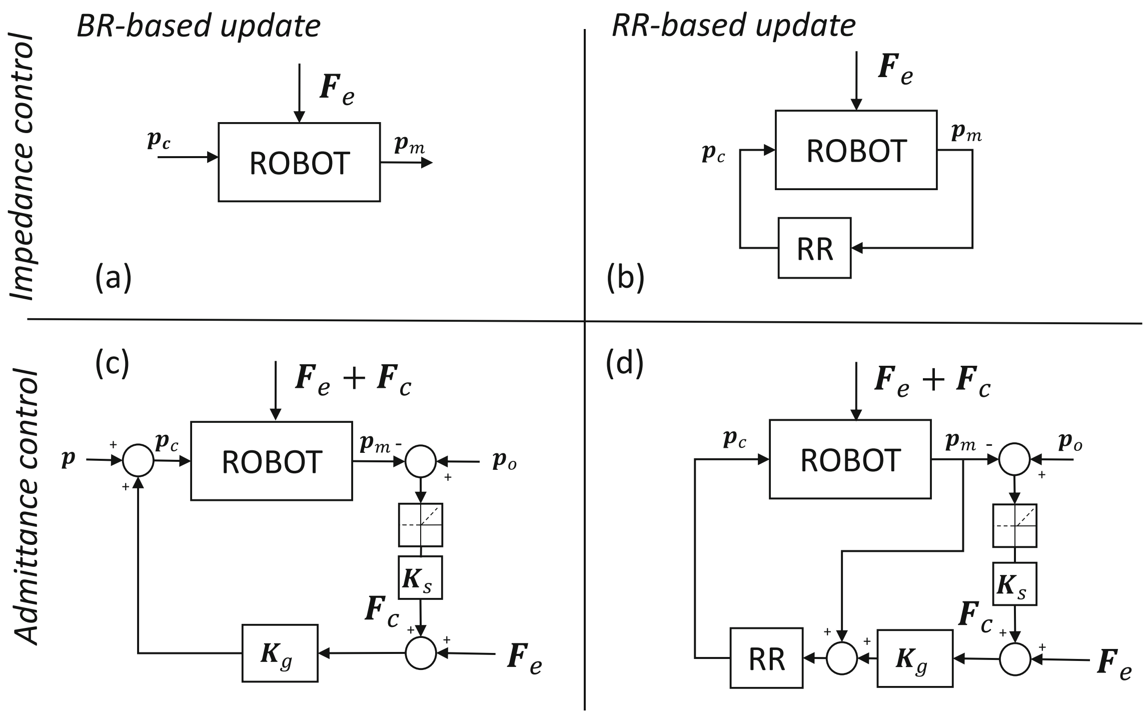


Fig. 8 Block diagrams of four different ways to calculate the displacement vector on a compliant robot with impedance control and **a** BR or **b** RR, and on a stiff robot with admittance control and **c** BR or **d** RR. $\mathbf{F}_e$ denotes external force by the operator, $\mathbf{F}_c$ the contact force, $\mathbf{p}$ is the desired robot position, obtained from CS-CDMP integration, $\mathbf{p}_c$ is the position sent to the controller, $\mathbf{p}_m$ measured robot position, and $\mathbf{p}_o$ the position of contact with the environment

Next, we analyze the behavior of the latter two non-linear systems in simulation.[4] The left side of Fig. 9a shows force and position of one positional degree of freedom for the BR admittance setup with the following parameters: external force $F_e = -10\,\text{N}$, environment stiffness $K_s = 2000\,\text{N/m}$, robot control gain $K_p = 1000\,\text{N/m}$, admittance gain $K_g = 0.002\,\text{m/N}$, and control sampling frequency 100 Hz. The force plot shows that the robot starts oscillating with constant frequency and amplitude after touching the environment. The phase plot on the right side of Fig. 9a shows that the system exhibits a stable limit cycle. If we increase the sampling frequency to 300 Hz, the limit cycle vanishes, as evident from Fig. 9b. If we also increase the admittance gain to $K_g = 0.005\,\text{m/N}$, the system starts oscillating, but oscillations are damped, and the system does not enter into the limit cycle, as evident from Fig. 9c. The above simulation experiments demonstrate that high sampling frequency is necessary for proper operation of the admittance BR setup.

A similar behavior can be noticed in the RR-based admittance scheme shown in Fig. 8d, except that the RR transfer function additionally damps the system. For this reason, the RR-based admittance scheme does not enter the stable limit cycle at sampling frequency 100 Hz and admittance gain $K_g = 0.002\,\text{m/N}$ as seen in Fig. 9d. However, if we further increase the admittance gain to $K_g = 0.005\,\text{m/N}$, we can notice strong vibrations in stable limit cycle as seen in Fig. 9e.

Given the above analysis, we can conclude that a high sampling frequency (500 Hz or more) is required for incremental learning with admittance control. Lowering admittance gain K_g stabilizes the system, but it also makes it less responsive, i.e., higher forces are needed to modify the robot trajectory. The RR-based algorithm is generally more resistant to the vibration caused by the limit cycle. On the other hand, the vibrations are more pronounced at excessive admittance gain K_g when the RR-based algorithm is applied.

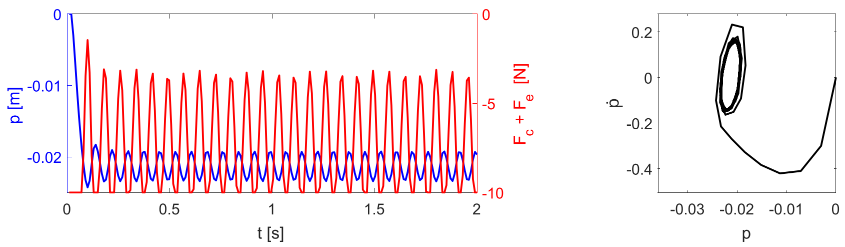
4.2 Experimental Verification

We implemented and evaluated the proposed framework on two collaborative robots: Franka Emika Panda and Universal robot UR10.

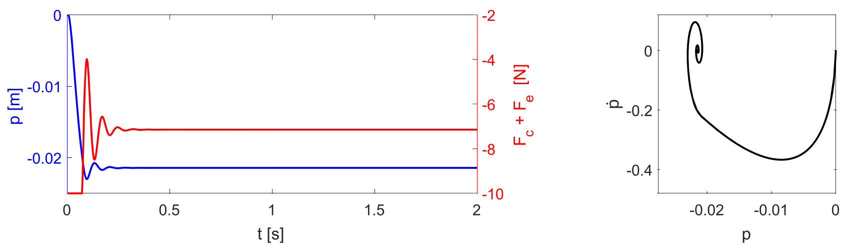
Implementation on Franka Emika Panda On the 7 DOF collaborative robot arm Franka Emika Panda, the control algorithm was implemented as a ros_control plug-in in C++ using libfranka library. The LfD framework was implemented

[4] A more concise analysis of this non-linear system involves the introduction of non-symmetrical describing function, which is beyond the scope of this paper.

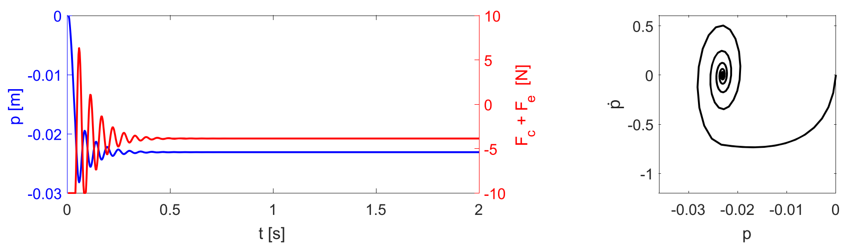
Fig. 9 Simulation results for admittance control and different incremental learning algorithms with varying sampling frequency f and admittance gain K_g. Time plots of position $\mathbf{p}$ and overall force $\mathbf{F}_c + \mathbf{F}_e$, are shown on the left. In the right column the corresponding phase portraits are given



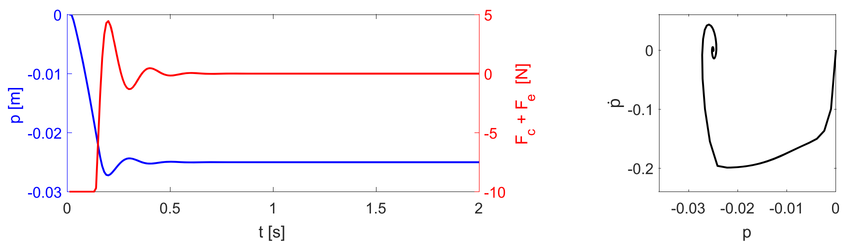
(a) BR-based update, $f = 100\,\mathrm{Hz}$, $K_g = 0.002\,\mathrm{m/N}$



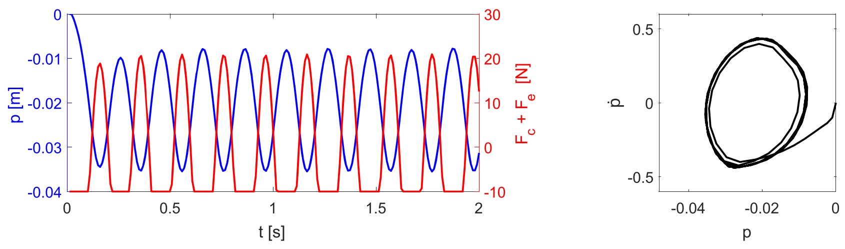
(b) BR-based update, $f = 300\,\mathrm{Hz}$, $K_g = 0.002\,\mathrm{m/N}$



(c) BR-based learning, $f = 300\,\mathrm{Hz}$, $K_g = 0.005\,\mathrm{m/N}$



(d) RR-based learning, $f = 100\,\mathrm{Hz}$, $K_g = 0.002\,\mathrm{m/N}$



(e) RR-based learning, $f = 100\,\mathrm{Hz}$, $K_g = 0.005\,\mathrm{m/N}$

in Matlab as a ROS node. Since the Panda robot supports variable compliance around any axis, we implemented RR- and BR-based algorithms in impedance mode. In the case of the BR-based scheme, we set high positional stiffness (3000 N/m) in the tangential direction and lower stiffness (500 N/m) in the normal and binormal directions of the FS frame. Accordingly, the stiffnesses of the rotational axes were set high (90 Nm/rad) for rotation around the tangent and low (10 Nm/rad) for other rotations. For the RR-based scheme, we set high stiffness in all directions.

Implementation on Universal Robot UR10 On the 6 DOF Universal Robot UR10, the control algorithm was implemented in Simulink using XPC Target platform, while the LfD framework was implemented in Python as a ROS node [2]. Wrist-mounted force-torque sensor ATI Delta was applied in this setup to implement admittance-based incremental learning. To preserve stability during the assembly operation that involves physical contact with the environment, it was necessary to select rather low admittance gains K_g (0.00005 m/N). However, since low admittance gains result in sluggish responses to the guiding force $\mathbf{F}_e$, we provided the user with the ability to increase K_g during the free space motion with a button at the robot wrist. In both BR and RR modes, the robot was stiff and the virtual displacement vector was calculated using Eq. (52).

Refinement of a policy for peg-in-hole insertion task The algorithms were experimentally verified on refining policy for peg-in-hole (PiH) insertion task. In this task, unavoidable positioning errors and tight tolerances between the objects require adaptation of the desired policies [24]. The existing policy resulted in a failure during insertion. Therefore, the insertion path was adapted using BR-based algorithm described in Section 3.3 and RR-based algorithm as described in Section 3.4.

A snapshot of the adaptation process for the PiH task using the Panda robot in BR-impedance mode is shown in Fig. 10, while Fig. 11 shows the original and refined positional part of the policy. Here, the policy had to be shifted due to positional displacement. The operator guided the robot to the start and demonstrated the required offsets in one pass. After the adaptation, the robot was able to perform the peg insertion successfully.

Figures 12 and 13 show a snapshot of the adaptation process for the PiH task using the UR10 robot in RR-admittance and the original and refined policy, respectively. As this system is less responsive, higher forces are needed to modify the robot trajectory compared to impedance-based control scheme. In this case, robot was able to successfully perform the task after three refinement passes.

4.3 User Study

We assessed the performance of the proposed algorithms by conducting a user study, where the subjects were teaching the robot a specific movement needed in shoe sole grinding. We chose this task as it requires many modifications depending on the shape and size of the shoe. Besides, it requires a precise speed profile, which is conditioned by the grinding technology. Similar are the tasks of applying glue, arc welding, polishing, etc. The evaluation was performed on a Franka Emika Panda robot using the impedance-based scheme and the same setup as described in Section 4.2.

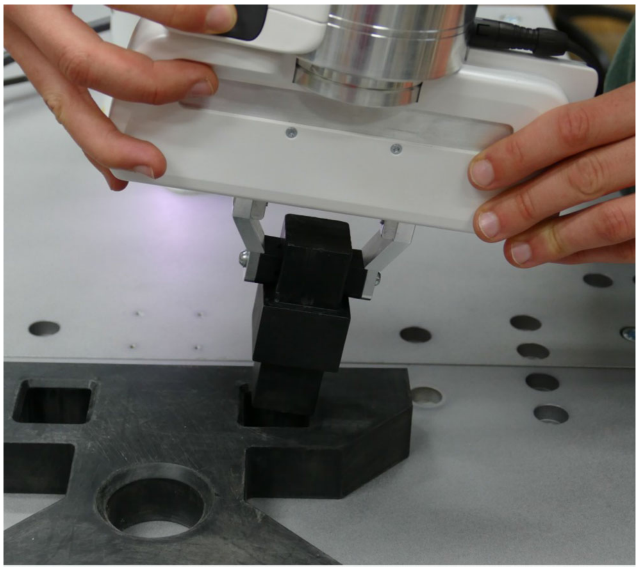


Fig. 10 Adaptation of a PiH policy using the Panda robot

During the experiment, the participants stood next to a small table with a shoe model mounted as depicted in Fig. 14 and were instructed to move the robot in such a way that the tip of the stick moves along the edge of the shoe sole from a start to the goal point which were both marked visually on the shoe. There were three experimental conditions: classical kinesthetic guidance (KG), BR-based incremental learning, and RR-based incremental learning. The sequence of conditions was randomized. Each condition consisted of an initial familiarization phase and a subsequent test phase.

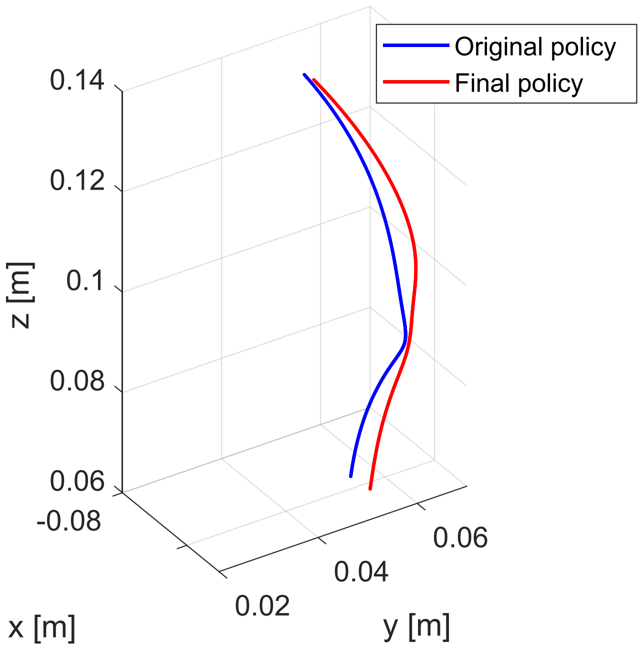


Fig. 11 Original and refined path of the PiH policy after adaptation using the Panda robot

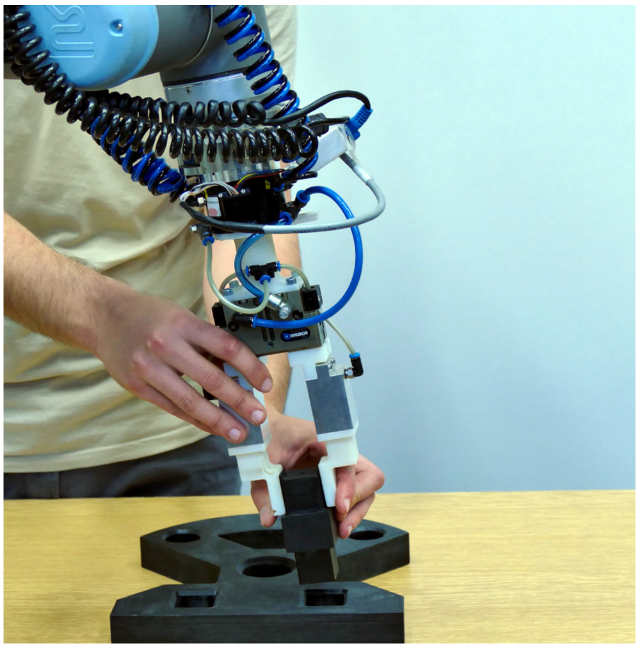


Fig. 12 Adaptation of a PiH policy using the UR10 robot with a wrist-mounted FT sensor

For the two variants of incremental learning, we initialized the adaptation process with a trajectory that corresponds to another shoe size. For classical KG, there is no initial trajectory. In the classical KG condition, the subjects had to demonstrate the movement in one pass and

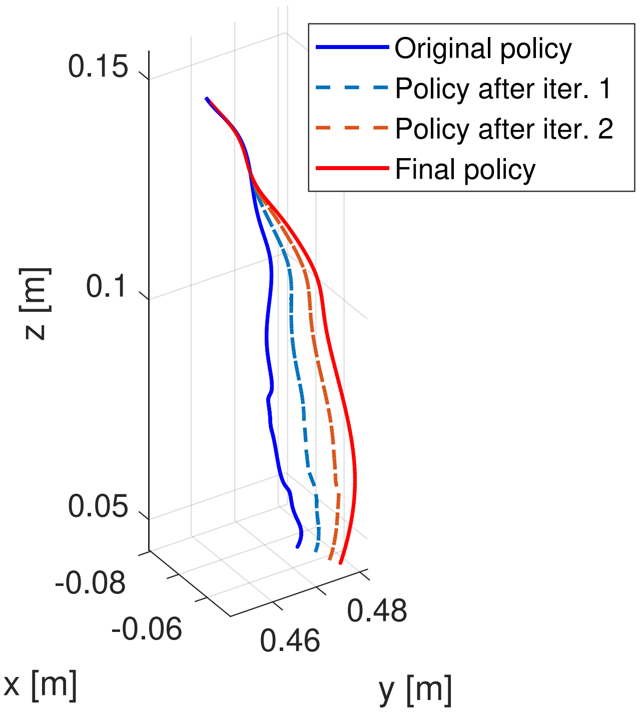


Fig. 13 Original and refined path of the PiH policy after adaptation using the UR10 robot

were allowed to repeat the procedure until they considered that the learned policy is good enough or felt that they could not improve it further. The first pass counted as the familiarization phase and was not considered in statistical evaluation. In both incremental learning conditions, the subjects got acquainted with the specific operation mode in the familiarisation phase. They received verbal instructions on how to move along an existing path and apply the corrections by pushing or pulling the robot. In the test phase, the subjects performed just one learning attempt but were able to correct eventual deviations of the learned path incrementally. Finally, they also demonstrated the speed profile. They were verbally instructed that the movement should take around 10 seconds. After each experiment, we collected the subjective assessment of the overall user experience (ease of use, kinesthetic feeling) of the learning procedure on a 1-5 rating scale.

A snapshot of the BR-based refinement process of the shoe grinding policies and a plot showing one representative operator's movement while adapting the spatial part of the policy is shown in Figs. 14 and 15a, respectively. Fig. 15b shows the corresponding intermediate policies throughout the iterations of the BR-based refinement process by this operator. The update occurs when the operator reverses the direction of teaching. Most participants needed 4-5 back and forth iterations to correct the policy. Note that there are no update iterations for the RR-based process as the policy is updated continuously.

Nineteen healthy subjects participated in the study (13 male, 6 female, age: 34.63 ± 14.59 years). Prior to their participation, the subjects were informed about the experimental procedure, potential risks, the aim of the study and gave their written informed consent in accordance with the code for ethical conduct in research at Jožef Stefan Institute. This study was approved by the National Medical Ethics Committee (No. 0120-228/2020-3).

To evaluate the time-efficiency of the proposed methods, we compared the total time of learning elapsed in the test phase. To evaluate the quality of learning, we calculated the deviation of both spatial and temporal course of the learned policy from the reference ground truth trajectory, which was carefully captured using a passive mechanical digitizer MicroScribe G2LX6. The position and velocity error measure the local accuracy of the trajectory and was calculated using the evaluation metrics proposed in [34].

We investigated the effects with One-way Repeated Measures ANOVA. The level of statistical significance used was 0.05 for all tests. Learning method had significant effect on position error [$F_{(2,18)} = 16.67, p < 0.01$], velocity error [$F_{(2,18)} = 82.36, p < 0.01$], learning time [$F_{(2,18)} = 27.67, p < 0.01$] and user experience [$F_{(2,18)} = 28.37, p < 0.01$]. Post-hoc t-tests showed that incremental learning (BR, RR) is more successful in all aspects compared to

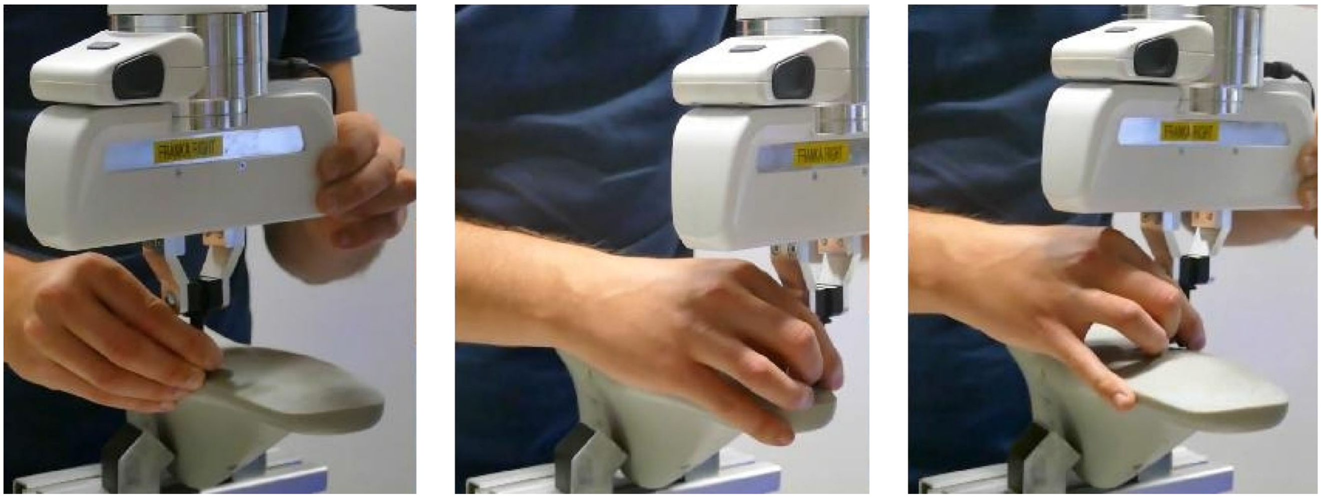


Fig. 14 An operator performs refinement of the shoe grinding policy. Pushing moves the robot along the nominal path, while pulling allows modifying it

the classical approach (KG). Both differences between KG vs. BR $[-6.3921 < t(9) < 9.91, p < 0.01]$ and KG vs. RR $[-6.6478 < t(9) < 12.02, p < 0.01]$ are statistically significant. On the other hand, no statistically significant differences were found between the two variants of incremental learning algorithms BR vs RR $[-0.27 < t(9) < 2.10, p > 0.05]$.

The plots in Fig. 16 show the means and standard errors (SEM) for different aspects for the three learning methods.

Based on subject's previous experience with operating robots using kinesthetic teaching, we split subjects in two groups. However, a separate two-way ANOVA shows no difference between groups of expert and novice users for any of the aspects $[0.1 < F_{(1,56)} < 2.74, p > 0.1]$.

The main result of the user study confirms that incremental policy learning by kinestheic guidance can effectively tackle limitations of the classical kinesthetic guidance. As the user can only control a limited number of joints simultaneously during the kinesthehic guidance, it is difficult to ensure the accuracy of the demonstrations [10]. Constraining the robot's movement along the nominal trajectory and allowing gradual refinements improves not

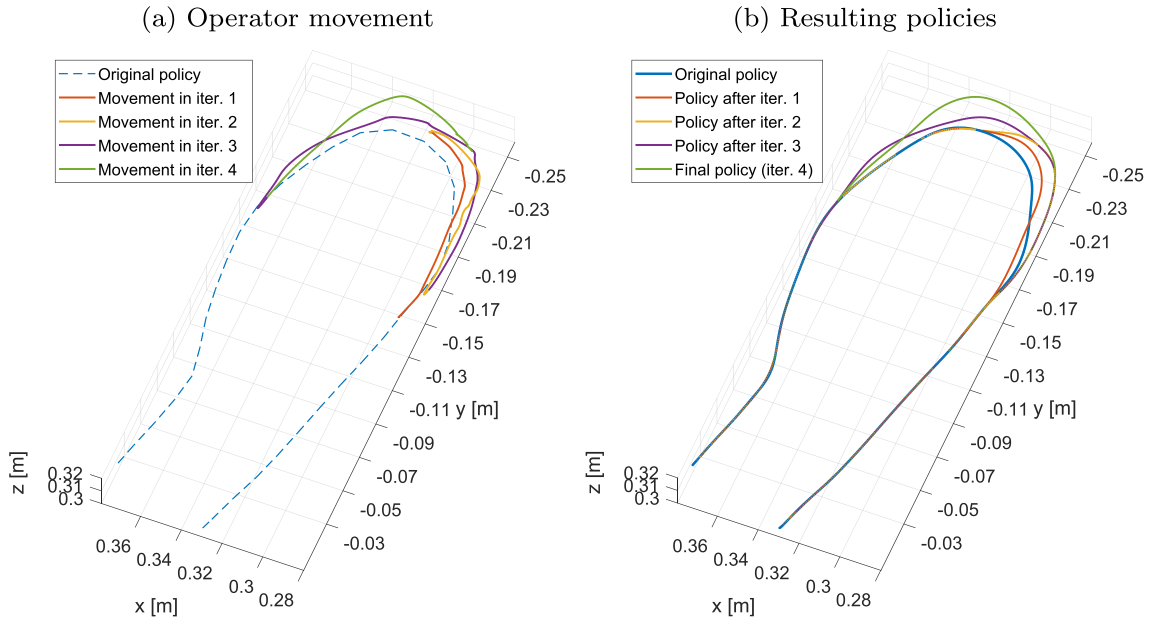


Fig. 15 Movement of the operator throughout the entire procedure of adapting the spatial part of the policy (on the left) and the corresponding intermediate policies (on the right). Different colors denote different iterations, which were triggered each time the operator changed movement direction

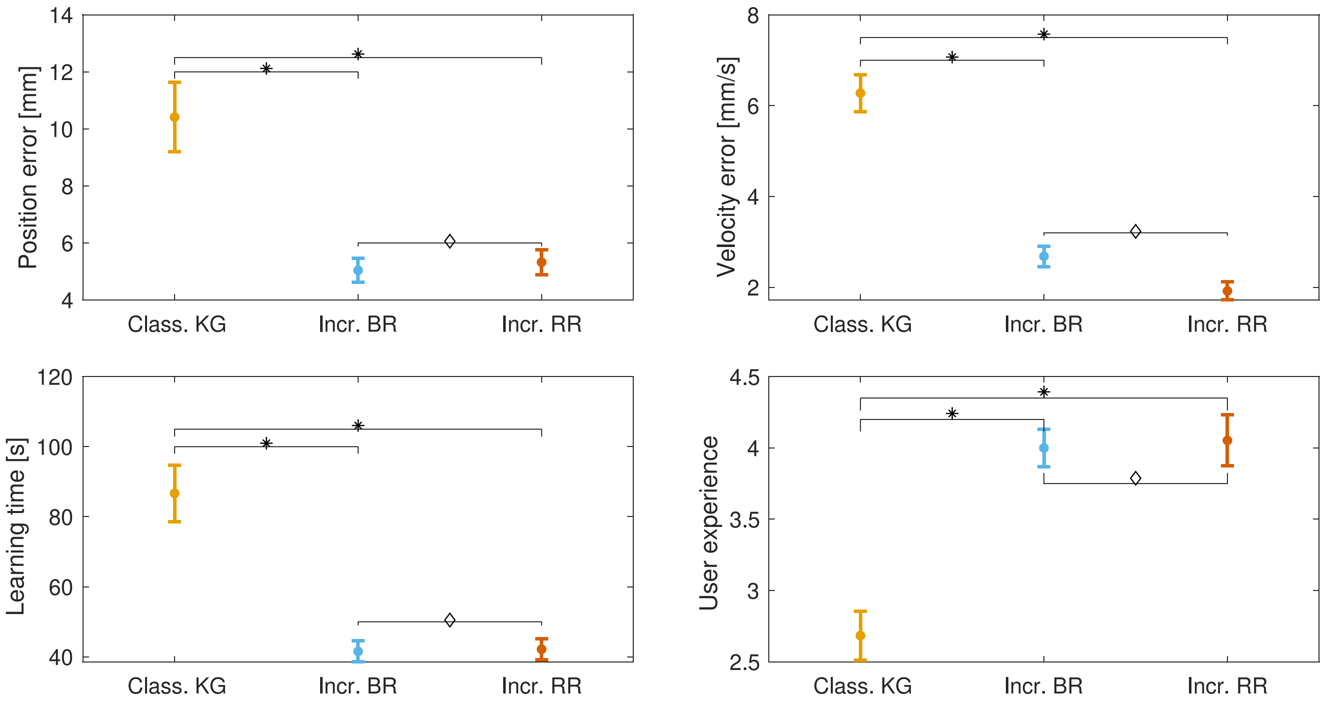


Fig. 16 Plots show four quality and efficiency measures (position error, velocity error, learning time, and user experience rating) for all participants when using classical kinesthetic guidance (KG), BR- or RR-based method for policy refinement. Stars denote pairs with statistically significant differences at $p < 0.05$. Diamonds show pairs without statistically significant differences

only the accuracy, but also shortens the learning time especially for complex policies. The statistically non-significant differences between BR - and RR-based learning in the user study can be justified by the moderate changes required to adapt to the new shoe sole as BR and RR-based learning behave similarly when changes are applied incrementally. Further, no statistical difference between groups of novice and expert users can be accounted to the intuitiveness of kinesthetic teaching paradigm and the familiarisation phase.

5 Conclusion

In the paper, we presented a framework for incremental adaptation of complex robot trajectories using kinesthetic guidance. It is based on speed scaled, constant arc-length, reversible Cartesian space DMPs that enable integration forward and backward along the trajectory. The update mechanism is based on two techniques. Batch regression temporarily stores modified trajectory segments and updates policy parameters when we reverse the direction of kinesthetic teaching. On contrary, recursive regression technique updates policy parameters directly within the control loop, which avoids storing modified trajectory.

There are several novelties in the presented framework. Instead of modifying the original DMP policy in learning iterations, we are learning offsets to the originally demonstrated policy and encode them separately using RBF. This improves the long-term stability of the learning algorithm and diminishes the computational burden of the learning algorithm. Next, we introduced Constant Speed Cartesian DMPs (CS-CDMPs), that encode the spatial component of the task with a constant speed profile. This allows proper decoupling of spatial and temporal part of the policy, which contributes both to more robust and more efficient learning. Among the other improvements, we enhanced the kinestheitc feeling during learning using soft activation functions and improved encoding accuracy with a new phase detection algorithm.

The proposed approach can be applied both to impedance and admittance controlled robots which was experimentally verified on two robot platforms. Stability analysis revealed that a high sampling frequency and low gains are required in the incremental learning with admittance control. Therefore, higher forces are needed to modify the robot trajectory compared to impedance-based control scheme. In practice this means, that more subsequent passes are needed to modify the existing trajectory

We performed a user study to evaluate the performance of proposed incremental learning algorithms and compared them with classical kinesthetic guidance. The user study confirmed that the concept of incremental updates is more effective and user-friendlier than the classical kinesthetic guidance for both experienced and novice operators. User study showed that the differences in performance and user-experience between the BR-based and RR-based algorithms are small. In our opinion, BR-based algorithm is still favorable, since it allows to implement the concept of penetrable virtual wall and allows for correcting the trajectory with larger deviations from the existing trajectory in a single step. On the other hand, the RR algorithm is less complex and easier to implement.

The proposed incremental learning can be used for learning many robotic tasks that involve contact with the environment. There are also some exceptions where our approach cannot be directly applied, e. g. when we need to modify the trajectory in the tangential direction. Changing the depth of peg insertion is such a case. Extensions that allow coping with such issues remain for our future work.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10846-021-01328-y.

Author Contributions Conceptualization, B.N. and M.S.; methodology, B.N., M.S., A.U. and T.P.; software, M.S. and B.N.; formal analysis, B.N., M.S., A.U. and T.P.; investigation, M.S.; data curation, M.S. and T.P.; writing–original draft preparation, B.N.; writing–review and editing, M.S., B.N. and A.U.; visualization, M.S. and B.N.; supervision, B.N. and A.U.; funding acquisition, B.N. and A.U. All authors read and approved the final manuscript.

Funding The research leading to these results has received funding from the Horizon 2020 RIA Programme grant 820767 CoLLaboratE and from the program group P2-0076 Automation, robotics, and biocybernetics funded by the Slovenian Research Agency.

Materials Availability The data that support the findings of this study are available from the corresponding author, B.N., upon reasonable request.

Declarations

Competing interests The authors have no conflict of interest to declare.

References

1. Molina, E., Lazaro, O., Sepulcre, M., Gozalvez, J., Passarella, A., Raptis, T.P., Ude, A., Nemec, B., Rooker, M., Kirstein, F., Mooij, E.: The AUTOWARE framework and requirements for the cognitive digital automation. In: Camarinha-Matos, L., Afsarmanesh, H., Fornasiero, R. (eds.) IFIP Advances in Information and Communication Technology: Volume 506. Springer International Publishing, Cham (2017)
2. Gašpar, T., Deniša, M., Radanovič, P., Ridge, B., Savarimuthu, T.R., Kramberger, A., Priggemeyer, M., Rossmann, J., Wörgötter, F., Ivanovska, T., Parizi, S., Gosar, Z., Kovač, I., Ude, A.: Smart hardware integration with advanced robot programming technologies for efficient reconfiguration of robot workcells. Robot. Comput. Integr. Manuf. **66**, 101979 (2020)
3. Dean-Leon, E., Ramirez-Amaro, K., Bergner, F., Dianov, I., Lanillos, P., Cheng, G.: Robotic technologies for fast deployment of industrial robot systems. IECON Proceedings (Industrial Electronics Conference), pp. 6900–6907 (2016)
4. Dillmann, R.: Teaching and learning of robot tasks via observation of human performance. Robot. Auton. Syst. **47**(2-3), 109–116 (2004)
5. Billard, A., Calinon, S., Dillmann, R., Schaal, S.: Robot Programming by Demonstration. In: Siciliano, B., Khatib, O. (eds.) Springer handbook of robotics, pp. 1371–1394. Springer, Berlin, Heidelberg (2008)
6. Argall, B., Chernova, S., Veloso, M., Browning, B.: A survey of robot learning from demonstration. Robot. Auton. Syst. **57**(5), 469–483 (2009)
7. Pastor, P., Kalakrishnan, M., Chitta, S., Theodorou, E., Schaal, S.: Skill learning and task outcome prediction for manipulation. IEEE International Conference on Robotics and Automation (ICRA), pp. 3828–3834 (2011)
8. Peters, J., Mülling, K., Kober, J.: Towards motor skill learning for robotics. In: Pradalier, C., Siegwart, R., Hirzinger, G. (eds.) Robotics research, pp. 469–482. Springer Verlag, Berlin, Heidelberg (2011)
9. Bristow, D.A., Tharayil, M., Alleyne, A.G.: A Survey of Iterative Learning Control - A learning-based method for high-performance tracking control. IEEE control systems magazine **26**(3), 96–114 (2006)
10. Calinon, S., Billard, A.: Incremental learning of gestures by imitation in a humanoid robot. In: 2nd ACM/IEEE International conference on human-robot interaction (HRI), pp. 255–262 (2007)
11. Kulic, D., Takano, W., Nakamura, Y.: Combining automated on-line segmentation and incremental clustering for whole body motions. In: IEEE International conference on robotics and automation (ICRA), pp. 2591–2598. Pasadena, CA (2008)
12. Gams, A., Petrič, T., Do, M., Nemec, B., Morimoto, J., Asfour, T., Ude, A.: Adaptation and coaching of periodic motion primitives through physical and visual interaction. Robot. Auton. Syst. **75**, 340–351 (2016)
13. Lee, D., Ott, C.: Incremental motion primitive learning by physical coaching using impedance control. In: IEEE/RSJ International conference on intelligent robots and systems (IROS), pp. 4133–4140, Taipei, Taiwan (2010)
14. Ewerton, M., Maeda, G., Kollegger, G., Wiemeyer, J., Peters, J.: Incremental imitation learning of context-dependent motor skills. In: IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids), pp. 351–358. Cancun, Mexico (2016)
15. Pardowitz, M., Knoop, S., Dillmann, R., Zollner, R.D.: Incremental Learning of Tasks From User Demonstrations, Past Experiences, and Vocal Comments. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **37**(2), 322–332 (2007)

16. Tykal, M., Montebelli, A., Kyrki, V.: Incrementally assisted kinesthetic teaching for programming by demonstration. In: 11th ACM/IEEE International conference on human-robot interaction (HRI), pp. 205–212 (2016)
17. Žlajpah, L., Petrič, T.: Unified Virtual Guides Framework for Path Tracking Tasks. Robotica **38**(10), 1807–1823 (2020)
18. Papageorgiou, D., Kastritsi, T., Doulgeri, Z.: A passive robot controller aiding human coaching for kinematic behavior modifications. Robot. Comput. Integr. Manuf. **61**, 101824 (2020)
19. Fitts, P.M.: The information capacity of the human motor system in controlling the amplitude of movement. J. Exp. Psychol. **47**(6), 381–391 (1954)
20. Nemec, B., Likar, N., Gams, A., Ude, A.: Human robot cooperation with compliance adaptation along the motion trajectory. Auton. Robot. **42**(5), 1023–1035 (2018)
21. Nemec, B., Žlajpah, L., Šlajpah, S., Piškur, J., Ude, A.: An efficient PbD framework for fast deployment of bi-manual assembly tasks. In: IEEE-RAS International conference on humanoid robots (Humanoids), pp. 166–173. Beijing, China (2018)
22. Nemec, B., Simonič, M., Petrič, T., Ude, A.: Incremental policy refinement by recursive regression and kinesthetic guidance. In: 19th International conference on advanced robotics (ICAR), pp. 344–349. Belo Horizonte, Brazil (2019)
23. Gašpar, T., Nemec, B., Morimoto, J., Ude, A.: Skill learning and action recognition by arc-length dynamic movement primitives. Robot. Auton. Syst. **100**, 225–235 (2018)
24. Abu-Dakka, F.J., Nemec, B., Jorgensen, J.A., Savarimuthu, T.R., Krüger, N., Ude, A.: Adaptation of Manipulation Skills in Physical Contact with the Environment to Reference Force Profiles. Auton. Robot. **39**(2), 199–217 (2015)
25. Ijspeert, A.J., Nakanishi, J., Hoffmann, H., Pastor, P., Schaal, S.: Dynamical movement primitives: Learning attractor models for motor behaviors. Neural Comput. **25**(2), 328–73 (2013)
26. Vuga, R., Nemec, B., Ude, A.: Speed adaptation for self-improvement of skills learned from user demonstrations. Robotica **34**(12), 2806–2822 (2016)
27. Ude, A., Nemec, B., Petrič, T., Morimoto, J.: Orientation in Cartesian space dynamic movement primitives. In: IEEE International conference on robotics and automation (ICRA), pp. 2997–3004. Hong Kong, China (2014)
28. Koutras, L., Doulgeri, Z.: A correct formulation for the orientation dynamic movement primitives for robot control in the cartesian space. In: Proc. conference on robot learning (CoRL), pp. 293–302. Osaka, Japan (2019)
29. Iturrate, I., Sloth, C., Kramberger, A., Petersen, H.G., Østergaard, E.H., Savarimuthu, T.R.: Towards reversible dynamic movement primitives. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 5063–5070. Macau, China (2019)
30. Ravani, R., Meghdari, A.: Velocity distribution profile for robot arm motion using rational Frenet-Serret curves. Informatica **17**(1), 69–84 (2006)
31. Khatib, O.: Augmented object and reduced effective inertia in robot systems. In: American control conference, pp. 2140–2147. Atlanta, GA (1988)
32. Nemec, B., Petrič, T., Ude, A.: Force adaptation with recursive regression Iterative Learning Controller. In: IEEE International Conference on Intelligent Robots and Systems (IROS), pp. 2835–2841. Hamburg, Germany (2015)
33. Khatib, O.: A unified approach for motion and force control of robot manipulators: The operational space formulation. IEEE J. Robot. Autom. **3**, 43–53 (1987)
34. Sturm, J., Engelhard, N., Endres, F., Burgard, W., Cremers, D.: A benchmark for the evaluation of RGB-D SLAM systems. In: IEEE/RSJ International conference on intelligent robots and systems (IROS), pp. 573–580 (2012)

Mihael Simonič received B.Sc. and M.Sc. degree in cognitive science from the University of Tübingen, Germany. As of 2018 he started his Ph.D. study at Faculty of electrical engineering of the University of Ljubljana, Slovenia and works at the Department of Automatics, Biocybernetics and Robotics, Jožef Stefan Institute, Ljubljana, Slovenia. His current research work focuses on the intersection between robot learning and human-robot collaboration.

Tadej Petrič received his M.Sc. in Electrical Engineering from the University of Maribor, Slovenia, in 2008. In 2013, he received a D.Sc. in robotics from the Faculty of Electrical Engineering at the University of Ljubljana. He conducted part of his doctoral research at the Department of Robotic Systems for Dynamic Control from Legged Humanoid Robots at German Aerospace Centre (DLR) in Oberpfaffenhofen. In 2013 and 2015 he was a visiting researcher at ATR Computational Neuroscience Laboratories in Japan and at Biorobotics Laboratory at Swiss Federal Institute of Technology (EPFL) in Lausanne, Switzerland. He is currently the head of the Laboratory for the Advancement of Collaborative Robot Behaviour in Physical Human-Robot Interaction (CoBoTaT), a Senior Research Associate in the Department of Automation, Biocybernetics and Robotics at Jožef Stefan Institute and Assistant Professor at Jožef Stefan International Postgraduate School, Slovenia. His current research focuses on the design of biologically plausible robotic controllers that achieve robustness and adaptation to changing environments comparable to humans.

Aleš Ude received the diploma degree in applied mathematics from the University of Ljubljana, Ljubljana, Slovenia, and the Dr.Eng. degree from the University of Karlsruhe, Karlsruhe, Germany, in 1990 and 1995, respectively. He is currently the Head of the Department of Automatics, Biocybernetics, and Robotics, Jožef Stefan Institute, and a profesor at the Faculty of Electrical Engineering, Ljubljana. He is also associated with the ATR Computational Neuroscience Laboratory, Kyoto, Japan. His research interests include robot learning, programming by demonstration, reconfigurable robotic systems, and humanoid robotics.

Bojan Nemec received the B.Sc and the M.Sc. degree in electrical engineering and the M.Sc. and Ph.D. degrees in robotics from the University of Ljubljana, Slovenia. He is currently the head of the Humanoid and Cognitive Robotics Lab and Research Councillor with the Department of Automatics, Biocybernetics, and Robotics, Jožef Stefan Institute, and a professor at Jožef Stefan International Postgraduate School, Ljubljana. His research interests include robot control, robot learning, service robotics, and sports biomechanics.