Accepted to 2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)

The publisher's version can be accessed through DOI: 10.1109/Humanoids43949.2019.9035052

Autonomous Learning of Assembly Tasks from the Corresponding Disassembly Tasks

Mihael Simonič, Leon Žlajpah, Aleš Ude, and Bojan Nemec

***Abstract*— An assembly task is in many cases just a reverse execution of the corresponding disassembly task. During the assembly, the object being assembled passes consecutively from state to state until completed, and the set of possible movements become more and more constrained. Based on the observation that autonomous learning of physically constrained tasks can be advantageous, we use information obtained during learning of disassembly in assembly. For autonomous learning of a disassembly policy we propose to use hierarchical reinforcement learning, where learning is decomposed into a high-level decision-making and underlying lower-level intelligent compliant controller, which exploits the natural motion in a constrained environment. During the reverse execution of disassembly policy, the motion is further optimized by means of an iterative learning controller. The proposed approach was verified on two challenging tasks - a maze learning problem and autonomous learning of inserting a car bulb into the casing.**

I. Introduction

Assembly is one of the most common, yet demanding applications in contemporary robotics. Assembly skills are needed not only in production plants but will also be important for the future generation of home and service robots, including humanoid robots. Applications for home robots are characterized by a wide variety of different assembly tasks. Hence, it is very important to shorten the programming time and to increase the autonomy of learning as in home environments we cannot relay on skilled operators. Ideally, a robot would be able to program itself autonomously. Various learning techniques, such as Reinforcement learning (RL) and Iterative Learning Control (ILC) [1] were successfully applied for policy improvement, where the initial task was previously demonstrated by a human [2], [3], [4]. There were very few successful attempts of completely autonomous learning of assembly tasks in robotics [5], [6]. Applied RL algorithms are required to efficiently scale to high dimensional learning problems encountered in robotics assembly [7]. Techniques like deep learning, learning in latent spaces, learning of meta parameters, which more efficiently describe the learning problem, or covariance matrix adaptation and statistical generalization techniques can dramatically reduce the search space in RL. However, they all require at least a partial knowledge of a model of the process, which can be either given apriori in an explicit form or inherited from previous experiments.

All authors are with Department of Automatics, Biocybernetics and Robotics, Jožef Stefan Institute, Ljubljana, Slovenia
mihael.simonic@ijs.si, leon.zlajpah@ijs.si, ales.ude@ijs.si, bojan.nemec@ijs.si

In the robotics community, tasks that involve interaction with the environment are considered as extremely hard to learn due to the unknown and possibly changing environment. On the other hand, interacting with the environment can be advantageous to accelerate the learning process. Namely, learning of physically constrained tasks is easier than the learning of tasks, where a robot can move completely freely in space. The reason is that the environment constrains the admissible movement directions. Consequently, the number of parameters that need to be learned can be greatly reduced. To implement this type of learning, we need to make use of the natural robot motion along the constraints imposed by the environment. A suitable framework for implementing such strategy is provided by the compliant robot control. This concept was used in our previous work, where we studied the autonomous learning for opening of doors and drawers [8].

In this paper, we extend this methodology to autonomous learning of assembly operations. For this purpose, we propose to first learn the reverse action – disassembly of an object. In an assembled object, the set of possible motions is constrained and typically only a single motion or operation is possible. During the disassembly, the object passes consecutively from state to state and the set of possible motion becomes less and less constrained until completely disassembled, where individual parts are no more constrained by the environment. In an assembly task the situation is opposite; the movement of individual parts changes from completely unconstrained to constrained. Given no previous knowledge about the nature of the task, learning of disassembly is therefore easier than learning of the assembly task, because of the advantages of physically constrained tasks for learning.

This idea is in accordance with cognitive and developmental studies, which show that human infants learn object manipulation in the same sequence: they will learn to insert a block into a container after a preceding period of merely taking the block out of the container [9]. Analysis of how parts interact/couple is crucial to enable the generation of assemblies in design for assembly. Transfer of knowledge obtained from disassembly was previously studied in the context of engineering education [10], but to our best knowledge has previously not been directly used to autonomously generate assembly procedures.

For learning of the disassembly policy, we propose a novel algorithm for hierarchical RL, where learning is decomposed

into a high-level decision-making and underlying lower-level intelligent compliant controller, which exploits the natural motion in a constrained environment. On the example of maze learning, we show that the proposed hierarchical RL is more efficient than classical RL in a constrained environment.

In assembly, the learned disassembly policy is reversed and further optimized by means of ILC using a method based on force profiles [4]. The proposed approach is suitable for the cases, where the assembly task is reversible. Most assembly tasks are directly or indirectly reversible [11], except for tasks including operations involving structural deformations (e.g. riveting) or activation of external equipment (e.g. for glueing). For the purposes of autonomous learning of disassembly policy, unreversible operations can be omitted and manually added to the assembly policy. On the other hand, operations such as putting/placing, or screwing are reversible [11]. Tasks including only reversible operations include generic peg-in-hole task, or more applied tasks such as electric motor assembly [12] and bayonet bulb insertion [13]. We verified the proposed approach on autonomous learning of inserting a model of a car bulb into the casing.

A general scheme for disassembly learning is presented in Section II. It is based on a lower level intelligent controller, introduced in Section III. Assembly policy is derived from the learned disassembly policy and needs refinement for reliable execution. The corresponding algorithms are given in Section IV. Section V shows the results of the experimental evaluation of the proposed framework. We conclude with a critical evaluation and possible extensions of the proposed algorithms.

II. Disassembly Learning

In this section we present the basics of our approach to autonomous disassembly policy learning. We assume that an object is composed of two parts. If the assembled object consists of more parts, it is necessary to apply disassembly learning on remaining parts again and again, until the object is fully disassembled. Furthermore, we assume that the part to be manipulated is grasped, whereas the other part is fixed.

For learning of the disassembly policy, we propose to apply hierarchical RL [14], rather than applying classical Q-learning or SARSA [15] to every possible discretized state and discrete action for this problem. In this way we can greatly reduce the number of states and obtain continuous policies.

The proposed hierarchical RL algorithm is based on the observation that disassembly processes usually consists of multiple stages – units of environmentally constrained motion. If we think in terms of these units, the learning of disassembly can be represented with a directed graph such as the one in Fig. 1, where nodes represent various key stages of the disassembly process and edges represent constrained motions. The goal of disassembly is to find a direct path from start to the target node.

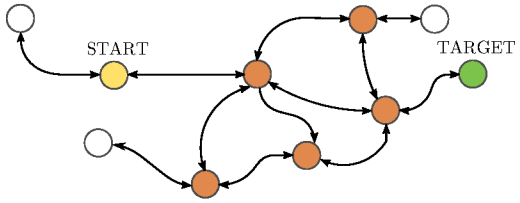

Fig. 1. Disassembly process illustrated as a directed graph. Edges represent motions between several key stages of the disassembly represented with nodes. We distinguish between different types of nodes. Start node (colored yellow) denotes the beginning of disassembly. Decision nodes (colored orange) correspond to the stages of disassembly where there are multiple ways how the disassembly should be continued. Pendant nodes (colored white) represent stages of disassembly where the motion cannot be continued without returning back. Target node represents fully disassembled stage.

Within the RL framework, the nodes correspond to the states, whereas edges represent actions (c.f. Table I). The states for reinforcement learning are specified with poses of the robot's end-effector in the local coordinate system of the object to be disassembled. A state s is a tuple:

$$s = (\boldsymbol{p}, \boldsymbol{q}), \tag{1}$$

where s is a state, $\boldsymbol{p} \in \mathbf{R}^3$ the position vector, and $\boldsymbol{q} \in \mathbb{R}^4$ the unit quaternion representation of orientation. Likewise, actions are given as Cartesian space dynamic movement primitives (DMPs) [16] encoding explored trajectories from one state to another together with captured forces and torques, which are encoded as radial basis functions (RBFs). (c.f. Sec. III).

Our hierarchical RL algorithm is decomposed into two levels. The lower level intelligent compliant controller moves in the admissible directions defined by physical constraints of the environment and simultaneously identifies key states for the upper level. The upper level then merely learns the policy, which action should be taken in each state, but not the trajectory itself.

TABLE I
Duality between the graph representation and HRL

Graph	Upper level RL	Lower level controller
Start node	Start state (disassembly begins)	Searching for admissible directions
Edge	Action (specified by DMP+RBF)	Moving along natural constraints
Decision node	Decision state (multiple actions can be taken)	Searching for admissible directions
Pendant node	Penalty state (movement cannot be continued)	Robot turns back
Target node	Target state (where disassembly is finished)	Robot stops

We exploit the duality between the graph representation and hierarchical RL to first graphically describe the learning

process and gradually formalize it within reinforcement learning framework. When we talk about graph, we operate with nodes and edges, whereas when we talk about reinforcement learning, we talk about states and actions.

Initially, we don't know the graph representation of the disassembly. The initial graph has only one node that corresponds to the robot pose at the beginning of the disassembly. Using the intelligent compliant controller the robot autonomously follows the environmental constraints in the only admissible direction, until it the movement is fully constrained. Trajectory of this movement is encoded as an action and represented in the graph as an edge. When the motion is not fully constrained anymore, there are multiple options to continue disassembly. This means that the robot either found a new state or came to an existing one. In the graph, this is represented as a decision node with multiple edges connected to it. If the movement cannot be continued in the same direction (represented with pendant node), the robot turns back. The disassembly is complete when the motion is unconstrained in all desired d.o.f. The disassembled state is represented as the target node in the graph.

During learning, a positive reward is given when the robot has disassembled the object. Negative reward is assigned when the robot arrived in a state where the motion could not be continued. When the robot explores state s_k, the action-value function $\mathcal{Q}(s_k, a_k)$ is updated according to the SARSA algorithm

$$\mathcal{Q}(s_k, a_k) \leftarrow \mathcal{Q}(s_k, a_k) + \alpha(r_k + \gamma Q(s_{k+1}, a_{k+1}) - \mathcal{Q}(s_k, a_k)), \quad (2)$$

where s_k is the label of the k-th state, a_k is the label of the action taken in s_k, r_k is the reward obtained in state s_k, $0 < \alpha < 1$ is the learning gain and $0 < \gamma < 1$ is the discount factor, which gives recent rewards higher importance. The optimal policy can be obtained by applying ϵ-greedy strategy in the form

$$\pi(s) = \begin{cases} \arg\max_a \ Q(s,a), & \text{with probability } 1-\epsilon, \\ \text{random action}, & \text{with probability } \epsilon, \end{cases} \quad (3)$$

where parameter ϵ is the ratio between the exploration and exploitation [15].

Note that in general there is a different set of actions for each state. The set of possible actions in state s_k is given by an action list $\mathcal{A}_k$. Actions become fully known only after the robot explores the entire trajectory and arrives in another state. This can be viewed as adding an edge in the graph. The entire proposed learning procedure is summarized in Algorithm 1.

Algorithm 1: Hierarchical learning algorithm

Input: Initial robot pose
Output: Learned policy $\pi(s)$

initialize Q_1 with start state s_1 from initial pose and the list of admissible actions $\mathcal{A}_1$
repeat
  while *not in target state* **do**
    greedy selection of action $a_k \in \mathcal{A}_k$ in state s_k
    if a_k *is not explored* **then**
      while *not in state* **do**
        follow the natural constraints, and search for *state* (Sec. III)
      encode travelled trajectory as DMP+RBF
      mark a_k as explored
    else
      execute a_k
    if *in a new state* **then**
      enumerate new state as s_{k+1}
      use detected admissible directions to create action list $\mathcal{A}_{k+1}$
    if *state = target state* **then**
      award positive reward r_k
    else if *state = penalty state* **then**
      award negative reward r_k
      turn back
    else
      no reward
    update action-value matrix Q using Eq. (2)
  compute policy $\pi(s)$ using Eq. (3)
until *last episode*

III. Controller Based Policy Search

In the previous section, a hierarchical RL was proposed for learning disassembly, where the task of the lower hierarchical level was the movement along the boundaries and search for possible states. In this section, we propose to use an intelligent controller for this purpose, which utilizes a compliant control framework. The main advantage of utilizing the controller is that it can generate continuous policy.

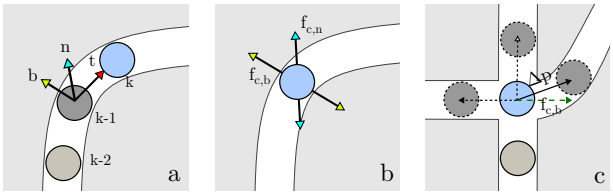

Fig. 2. Searching path and possible states in restricted environment. (a) shows Frenet-Serret frame attached to the robot in time sample $k-1$. (b) shows search forces applied in the normal and binormal direction. (c) shows an instance, when the controller discovers a new state.

In general, we do not know in advance where are the natural constraints of the system. To find a feasible motion direction, we apply a random force in a random direction. If this force results in a movement, we use compliant control to continue the motion in the direction initiated by the random force. The controller acts in tool coordinates and the control parameters make the robot compliant in all directions orthogonal to the direction of the motion. These directions can be estimated by applying Frenet-Serret frames [17] to the resulting motion trajectory. Whenever the initiated robot mo-

tion stops, we assume that this is due to the task constraints and we try to find a new feasible motion by applying again a random force in a random direction. Following this strategy, the robot eventually learns how to perform the task in a form of a parametrized policy. Whenever multiple possibilities how to continue are identified, the controller stops and waits for the decision of RL, in which direction it should move.

Let us first define a rotation matrix $\mathbf{R}_p$, where the coordinate frame with x coordinate specified in the desired direction of motion, i.e., $\dot{\boldsymbol{p}}$, and the other two coordinates orthogonal to it, as illustrated in Fig. 2 a. This matrix can be obtained by forming the Frenet-Serret frame at each sampling time. The Frenet-Serret frame consists of three orthogonal directions defined by the path's tangent (direction of motion), normal, and binormal. We obtain the following expression for $\mathbf{R}_p = \begin{bmatrix} \boldsymbol{t}_p & \boldsymbol{n}_p & \boldsymbol{b}_p \end{bmatrix}$ with

$$\boldsymbol{t}_p = \frac{\dot{\boldsymbol{p}}}{\|\dot{\boldsymbol{p}}\|},\ \boldsymbol{b}_p = \frac{\dot{\boldsymbol{p}} \times \ddot{\boldsymbol{p}}}{\|\dot{\boldsymbol{p}} \times \ddot{\boldsymbol{p}}\|}\ ,\ \boldsymbol{n}_p = \boldsymbol{b}_p \times \boldsymbol{t}_p, \tag{4}$$

where $\boldsymbol{p} \in \mathbb{R}^3$ are the measured robot end-effector positions. The original equation requires accelerations, which are very low and therefore very noisy during operations like assembly and disassembly. On the other hand, for our purpose, it is not important how the normal and bi-normal axis is chosen, since the robot is equally compliant in both directions. Therefore, we choose the bi-normal vector $\boldsymbol{b}$ as the arbitrary vector that satisfies the equation $\boldsymbol{t}_p \cdot \boldsymbol{b}_p = 0$. Using this simplification, the only parameter needed is the velocity $\dot{\boldsymbol{p}}$. For robust estimation of velocity vector $\dot{\boldsymbol{p}}$ we applied spatial filtering, as proposed in [18], which smooths the noisy estimates using a first order filter and assures, that the filtering does not affect the normalization. A discrete time implementation of the spatial filter is

$$\dot{\boldsymbol{p}}(k) = \dot{\boldsymbol{p}}(k\text{-}1) + \lambda(1 - \dot{\boldsymbol{p}}(k\text{-}1)\dot{\boldsymbol{p}}^T(k\text{-}1))(\boldsymbol{p}(k) - \boldsymbol{p}(k\text{-}1)), \tag{5}$$

where λ is the filter bandwidth and k denotes the k-th time sample. The above equations are used to set the rotation frame attached to the positional trajectory. The corresponding frame $\mathbf{R}_o$ is needed also for the orientation part of the trajectory. For the specification of the robot orientation, unit quaternions are used. We denote them as $\boldsymbol{q} = \{\eta, \boldsymbol{\epsilon}\} \in \mathbb{R}^4$, where η and $\boldsymbol{\epsilon}$ are the corresponding scalar and vector part of the quaternion, respectively. Angular velocities can be calculated from two subsequent quaternions as

$$\boldsymbol{\omega}(k) = 2\log(\boldsymbol{q}(k) * \bar{\boldsymbol{q}}(k-1)), \tag{6}$$

where $*$ denotes the quaternion multiplication and the quaternion logarithm is calculated as

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

Similar as for the positional part of the trajectory, the smoothed angular velocity $\boldsymbol{\omega}_s$ can be calculated as

$$\boldsymbol{\omega}_s(k) = \boldsymbol{\omega}_s(k-1) + dT\lambda(1 - \boldsymbol{\omega}_s(k-1)\boldsymbol{\omega}_s^T(k-1))\boldsymbol{\omega}(k), \tag{8}$$

and the corresponding $\mathbf{R}_o = \begin{bmatrix} \boldsymbol{t}_o & \boldsymbol{n}_o & \boldsymbol{b}_o \end{bmatrix}$ with

$$\boldsymbol{t}_o = \frac{\boldsymbol{\omega}_s}{\|\boldsymbol{\omega}_s\|},\ \boldsymbol{b}_o = \frac{\boldsymbol{\omega} \times \dot{\boldsymbol{\omega}}}{\|\boldsymbol{\omega} \times \dot{\boldsymbol{\omega}}\|}\ ,\ \boldsymbol{n}_o = \boldsymbol{b}_o \times \boldsymbol{t}_o, \tag{9}$$

where dT in (8) denotes the sampling frequency.

Next, we will define control law, which enables the robot to follow the operational space path, defined with the environmental boundaries. For this purpose, we utilized a variant of passivity-based impedance control for manipulators with flexible joints [19] and provided a modification, which enables to set the compliance along the operational space trajectory. The torque, which is passed to the robot motors, is calculated as

$$\boldsymbol{\rho_c} = \mathbf{B}\mathbf{B}_\Theta^{-1}\boldsymbol{u} + (\mathbf{I} - \mathbf{B}\mathbf{B}_\Theta^{-1})\boldsymbol{\rho} \tag{10}$$

$$\boldsymbol{u} = \mathbf{J}^{\mathrm{T}}(\boldsymbol{\theta})(\ddot{\mathcal{X}}_c + \boldsymbol{f}_c) + \bar{\mathbf{g}}(\boldsymbol{\theta}) + \mathbf{N}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}}_0 \tag{11}$$

where $\boldsymbol{\rho_c} \in \mathbb{R}^N$ is the control torque input for the motors, N is the number of robot joints, $\boldsymbol{\theta} \in \mathbb{R}^N$ is the joint position measured at the motor side, $\mathbf{J} \in \mathbb{R}^{N\times 6}$ is the manipulator Jacobian, $\mathbf{B}$ and $\mathbf{B}_\Theta \in \mathbb{R}^{6\times 6}$ denote the positive definite diagonal matrix of joint and desired joint inertia, respectively. $\boldsymbol{\rho}$ are measured joint torques and $\bar{\mathbf{g}}(\boldsymbol{\theta})$ is the gravity vector estimated in such a way, that it provides exact gravity compensation in the static case using the signals measured at the motor side [20]. $\mathbf{N}(\boldsymbol{\theta}) = (\mathbf{I} - \mathbf{J}(\boldsymbol{\theta})\mathbf{J}^{+}(\boldsymbol{\theta})) \in \mathbb{R}^{N\times N}$ is the null space projection operator, $\mathbf{J}^{+}(\boldsymbol{\theta})$ denotes Moore-Penrose pseudo-inverse of the Jacobian and the $\dot{\boldsymbol{\theta}}_0 \in \mathbb{R}^N$ is the corresponding null-space velocity vector. $\boldsymbol{f}_c$ is an additional force-torque vector in task coordinates. Basically, the motor torque controller (10) reduces the motor inertia and compensates for the robot non-linear dynamics, while (11) provides for the desired impedance and damping, additional task force and null space motion. The task command input $\ddot{\mathcal{X}}_c = [\ddot{\boldsymbol{p}}_c^{\mathrm{T}}, \dot{\boldsymbol{\omega}}_c^{\mathrm{T}}]^{\mathrm{T}}$ is chosen as

$$\ddot{\boldsymbol{p}}_c = -\mathbf{R}_p\mathbf{D}_p\mathbf{R}_p^{\mathrm{T}}\dot{\boldsymbol{p}} + \mathbf{R}_p\mathbf{K}_p\mathbf{R}_p^{\mathrm{T}}\boldsymbol{e}_p, \tag{12}$$

$$\dot{\boldsymbol{\omega}}_c = -\mathbf{R}_o\mathbf{D}_o\mathbf{R}_o^{\mathrm{T}}\boldsymbol{\omega} + \mathbf{R}_o\mathbf{K}_o\mathbf{R}_o^{\mathrm{T}}\boldsymbol{e}_q, \tag{13}$$

where position and orientation tracking errors are defined as $\boldsymbol{e}_p = \boldsymbol{p}_d - \boldsymbol{p}$ and $\boldsymbol{e}_o = 2\log(\overline{\boldsymbol{q}}_p * \boldsymbol{q}_d)$. $\mathbf{K}_p$ and $\mathbf{K}_o \in \mathbb{R}^{3\times 3}$ are the diagonal matrices, which define the positional and rotational stiffness along and around x, y, z axes, respectively. $\mathbf{D}_p$ and $\mathbf{D}_o \in \mathbb{R}^{3\times 3}$ are diagonal damping matrices, which are set to $\mathbf{D} = 2 * \sqrt{\mathbf{K}}$ for critically damped system.

With this control applied, the robot is able to autonomously move along the environmental boundaries, given that we apply high positional gain in the direction of movement, which is actually x axis and low gains in the orthogonal direction, which are y and z axes defined in the global coordinate system. We denote this trajectory as operational space trajectory. However, this control law alone can not discover new states. For this purpose, we applied short step force signals in the positive and negative directions of the normal and bi-normal, $\boldsymbol{f}_{c,n} = \mathbf{R}_p f_0 [0\ 1\ 0]^T$ for the

force in normal and $\boldsymbol{f}_{c,b} = \mathbf{R}_p f_0 [0\ 0\ 1]^T$ for the force in bi-normal direction. f_o is suitably chosen scalar, which changes sign in each test position along the operational trajectory (see Fig. 2 b). Test positions are positions, where the algorithm tests for possible states and are equally spaced along the operational space trajectory. If the application of this test signals results in motion, i.e. if $|\Delta \boldsymbol{p} \cdot \boldsymbol{n}|$ or $|\Delta \boldsymbol{p} \cdot \boldsymbol{b}|$ is above the predefined threshold, and if the motion along the tangential direction is also possible, the controller has found new state. $\Delta \boldsymbol{p}$ is the position displacement after applying each test signal. In a new state, action list is generated from admissible further movement directions, which can be calculated from $\Delta \boldsymbol{p}$. Note that the admissible movement directions do not have to be exactly aligned with the normal or bi-normal, as the robot is compliant (see Fig. 2 c). In order to improve the robustness of this search procedure, we can further lower the $\mathbf{K}$ gain in the direction of the x axis. The described search procedure as illustrated in Fig.2 applies to searching positional actions. As there are also actions, that correspond to different robot orientations, a similar search procedure has to be performed also for the rotations. Depending on the number of admissible actions, the state is categorized. If there are multiple actions, state is labeled as a decision, if there is just one as a penalty and if the motion is unrestricted as the target state.

The resulting operational space trajectory is encoded with Cartesian DMPs [21], [16] together with tangential vectors $\boldsymbol{t}_p, \boldsymbol{t}_o$, captured forces and torques during disassembly, which are encoded as RBFs, sharing common phase [21] with DMPs. The benefit of such encoding is twofold: It allows compact, smooth and scalable representation of a learned policy, and, it removes the explicit time dependency of signals, which enables to slow down or speed up the learned assembly policy according to the specific situation as it arises [22]. These trajectories are passed to the upper-level hierarchical RL together with newly discovered sates. Admissible motion continuations are then used to label new actions in a new state. At the same time, DMP and RBF containing the information of traveled trajectory are used to replace the corresponding action in the previous state. Doing so, state search is performed only once for each state.

IV. Assembly Learning

Once we successfully learn the disassembly policy, we merely reverse it to perform assembly. However, assembling is a demanding operation even for humans and there is no guarantee, that it will be successful even if the operation is reversible, as very small deviations in part geometry, grasping, material, etc. can result in failure. For this, we have to apply appropriate control together with the exception strategies, which mimic human behavior during the assembly. All these measures are successful when the source of error is stochastic. Errors of deterministic origin can be eliminated by policy improvement using ILC.

Disassembly policy is encoded with DMPs. DMPs are dynamical system and as such become unstable when they are executed with reverse time. Therefore, it is necessary to learn the time reverse policy with another DMP [23]. This is, however, not necessary for tangential vectors $\boldsymbol{t}_p$ and $\boldsymbol{t}_o$ and forces and torques, encoded with RBFs. They are used for on-line calculation of the rotation matrices $\mathbf{R}_p$ and $\mathbf{R}_o$, needed for the compliant control. For assembly, we use identical compliance settings as for disassembly with one exception; when the manipulated part is not constrained by the environment anymore, which happens at the end of the disassembly graph, we set high stiffness in all spatial directions. This assures precise path tracking during e.g. an approach motion in assembly. Additional force vector $\boldsymbol{f}_c$ is set to zero during the assembly. During the assembly, we also observe the measured contact forces and torques and compare them with measured forces and torques during disassembly. Note that the forces and torques during assembly have the opposite sign in relation to those measured at disassembly. When a high deviation occurs, we slow down the DMP integration, as described in [22], [4]. If, however, the forces/torques are still increasing, we carry out a trajectory in the opposite direction for some time and then try again, as suggested in [11]. Moving in the reverse direction, we have to switch between the two DMPs as described in [23].

Finally, we apply ILC in order to improve the policy obtained with the disassembly learning and to tune it to eventual change in part geometry and materials, whose origin is deterministic. The aim of this adaptation is to bring forces/torques during assembly close to those detected during disassembly by commanding the desired positions and orientations [24].

V. Experimental Evaluation

We first verified the proposed architecture on the example of maze learning in simulation. Maze learning was selected as it has many similarities with the disassembly process. In both cases, the robot has to find an optimal policy from the start state, which corresponds to starting position in the maze and the fully assembled object, to the target state, which corresponds to exiting the maze and to the disassembled object. As in disassembly, the robot only has to decide how to continue when multiple actions are possible. We created a 6×9 maze with corridors that restrict the robot movement, as shown in the upper part of Fig. 3. In this case state for learning is fully defined by robot position in the maze and actions are defined by sequences of movement directions. We applied both classical SARSA learning and the proposed hierarchical RL algorithm for exiting the maze. The lower part of Fig. 3 shows learning statistics of 100 runs of each setup.

This simple example clearly shows the benefit of the proposed hierarchical learning, since the optimal policy is learned in very few roll-outs. Furthermore, by connecting

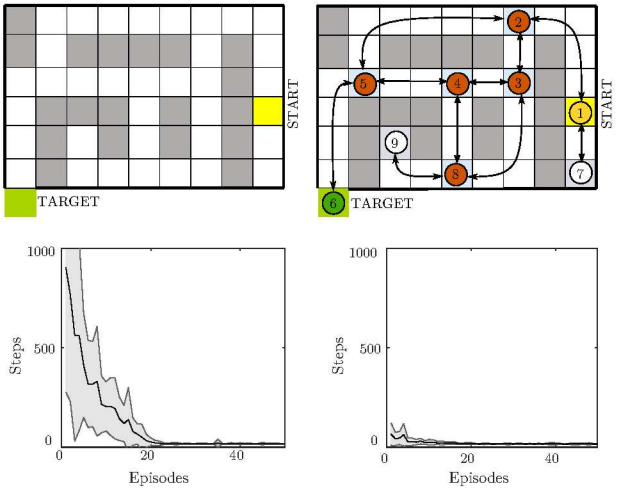

Fig. 3. The upper part shows two identical mazes with corridors (white cells). On the left side a classical RL based on SARSA is applied and on the right the proposed hierarchical RL algorithm. In both cases the simulated robot should learn a policy to arrive from the start (yellow) to the target state (green). For SARSA, all cells are taken as states, whereas for hierarchical RL states are assigned only in key cells. Start state can be anywhere in the maze, decision states (light blue) are in the cross-ways. Reward is given only in the dead-ends (penalty state, gray) and when exiting the maze (target state). States are assigned dynamically and might vary from run to run (for better visibility only indices are given). The bottom part shows for 100 runs the average number of actions needed to reach target in consequent episodes using SARSA (left) and hierarchical RL (right). Shaded areas show standard deviation.

the states, the duality between graph representation of the key stages of the learned process and reinforcement learning is explicitly revealed.

Experimental evaluation of the proposed approach including intelligent compliant controller was carried out on Franka Emika Panda robot with 7 d.o.f. The control algorithm was implemented as a ROS_control plug-in in C++ using libfranka (https://frankaemika.github.io/), while the learning algorithm was implemented in Matlab as a ROS node. The same algorithm was used to learn two tasks: 1) maze learning and 2) dismounting of the car bulb from the casing.

The maze used in our experiments is shown in Fig. 4 and shares topology with the one used in simulation (c.f Fig. 3). We kinestetically guided the robot to the arbitrary place within the corridors of the maze and the robot had to find the optimal way out. The only information passed to the robot was that this was a planar case. This limited the search to global x and y axes only. The robot autonomously detected the target as a state, where the motion was not constrained by the environment boundaries anymore. The number of states found during the exploration varied from 4 to 7. In average, the robot has learned the optimal path in 4 roll-outs. The learning statistics coincides with the learning statistics in Fig. 3.

Next experiment was learning to insert a car bulb into the casing. Car bulb and the casing (Fig. 5) were 3D printed in a tight fit. First, we learned the disassembly policy. The

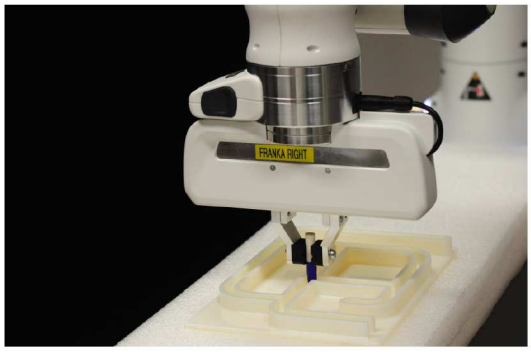

Fig. 4. Robot is autonomously learning how to exit a 3D printed maze.

robot was manually guided to the position, where the gripper could grasp the bulb. The only additional information given to the robot was, that the search should be performed in two d.o.f: z coordinate and rotation around z as the task requires both linear and angular exploration.

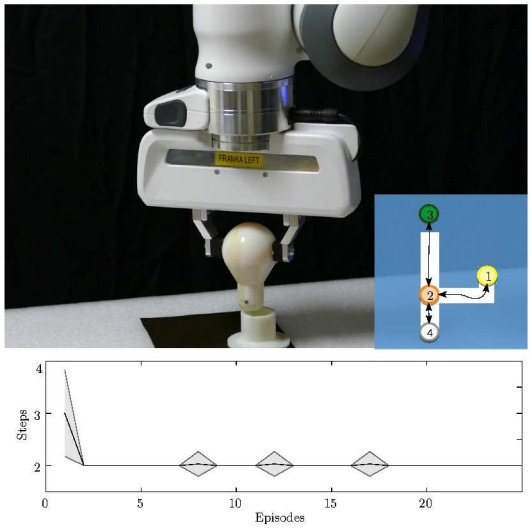

Fig. 5. The upper part shows the robot just after a successful disassembly. The states automatically discovered by the controller are shown on a projection of the casing gap to the plane. Start state (1) is shown in yellow, decision (2) in orange, target (3) in green and state with penalty (4) in white. The bottom part shows learning statistics for bulb disassembly using hierarchical RL. On average, the robot learns to disassemble in two steps already after the second roll-out. Due to 5% exploration rate, it can sometimes happen that the robot still goes in the penalty state.

From the projection of the casing gap to the plane it becomes evident that this problem can be described with only 4 states, as illustrated in Fig. 5. During the disassembly, the robot started in the state 1 and the only decision it had to make was in the state 2 to arrive in the final state 3, when the bulb is separated from the casing. There is also a penalty state 4. The graph representation reveals another aspect. If the proposed hierarchical RL would be applied directly to learning of bulb insertion, it would be impossible to autonomously determine whether the insertion has to finish in state 1 or 4. Contrarily, the terminating condition is well-defined for the reverse operation - when the motion gets unconstrained (in state 3), the disassembly is completed.

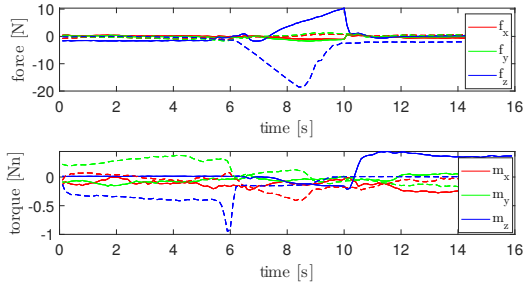

Fig. 6. Mean forces and torques of 20 disassembly and assembly runs. Dotted lines denote disassembly forces and torques. Solid lines denote optimized assembly forces and torques after 5 cycles of ILC

After learning the disassembly policy, we generated the corresponding assembly policy as described in Section IV and applied ILC, which additionally diminished the forces and torques during the disassembly. We obtained 100% disassembly and assembly success rate in 20 experiments.

VI. Conclusions

In this paper, we presented a novel approach, which autonomously learns an assembly task from initial disassembly. The disassembly-assembly process was represented as a directed graph, where the aim was to find the optimal path from the start node to the target node. To this end we propose to apply hierarchical RL, where the upper level is composed of standard SARSA algorithm and the lower level is assigned to the intelligent controller. The output of the learning process is smooth time-continuous policy, appropriate for precise tasks such as assembly.

We first verified the proposed approach on the well-known problem of maze learning, which has many similarities with disassembly-assembly learning. Final experiment showed that the autonomous learning of the bulb insertion can be successfully accomplished and simplified using information gained during disassembly.

The proposed approach has some similarities with the learning of door opening [8], where statistical RL algorithm was applied. However, in [8] force based policy was generated, which is less appropriate for precise tasks. Additionally, the approach proposed in this paper is able to generate compliant and scalable assembly primitives, which can be reused in similar cases. Our future research will focus on autonomous assembly learning for objects, composed of multiple parts.

References

[1] D. Bristow, M. Tharayil, and A. Alleyne, "A survey of iterative learning control," *IEEE Control Systems Magazine*, vol. 26, no. 3, pp. 96–114, jun 2006.

[2] J. Buchli, F. Stulp, E. Theodorou, and S. Schaal, "Learning variable impedance control," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 820–833, 2011.

[3] S. Calinon, L. Rozo, P. Kormushev, I. Sardellitti, P. Jimenez, C. Torras, and D. G. Caldwell, "Compliant skills transfer through kinesthetic teaching interaction," *Transactions on Haptics*, 2011.

[4] F. J. Abu-Dakka, B. Nemec, J. A. Jørgensen, T. R. Savarimuthu, N. Krüger, and A. Ude, "Adaptation of manipulation skills in physical contact with the environment to reference force profiles," *Autonomous Robots*, vol. 39, no. 2, pp. 199–217, 2015.

[5] S. Levine, N. Wagener, and P. Abbeel, "Learning Contact-Rich Manipulation Skills with Guided Policy Search," *International Conference on Robotics and Automation*, pp. 156–163, 2015.

[6] T. Inoue, G. D. Magistris, A. Munawar, T. Yokoya, and R. Tachibana, "Deep reinforcement learning for high precision assembly tasks," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 819–825.

[7] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[8] B. Nemec, L. Zlajpah, and A. Ude, "Door opening by joining reinforcement learning and intelligent control," in *18th International Conference on Advanced Robotics (ICAR)*, 2017, pp. 222–228.

[9] M. Hayashi, H. Takeshita, and T. Matsuzawa, "Cognitive development in apes and humans assessed by object manipulation," in *Cognitive development in chimpanzees*. Springer, 2006, pp. 395–410.

[10] D. Axinte, "An inverse method of teaching specialised manufacturing subjects: decomposing a focal representative product to sustain analysis and interaction of details," *European Journal of Engineering Education*, vol. 33, no. 1, pp. 67–84, 2008.

[11] J. S. Laursen, L.-P. Ellekilde, and U. P. Schultz, "Modelling reversible execution of robotic assembly," *Robotica*, vol. 36, no. 5, pp. 625–654, 2018.

[12] A. Kuehl, S. Furlan, J. Gutmann, M. Meyer, and J. Franke, "Technologies and processes for the flexible robotic assembly of electric motor stators," in *2017 IEEE International Electric Machines and Drives Conference (IEMDC)*, 2017, pp. 1–6.

[13] U. Thomas, B. Finkemeyer, T. Kroger, and F. M. Wahl, "Error-tolerant execution of complex robot tasks based on skill primitives," in *2003 IEEE International Conference on Robotics and Automation*, vol. 3, 2003, pp. 3069–3075.

[14] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *Journal of Artificial Intelligence Research*, vol. 13, pp. 227–303, 2000.

[15] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction, Second edition*. The MIT Press, Cambridge, London, 2015.

[16] A. Ude, B. Nemec, T. Petrič, and J. Morimoto, "Orientation in Cartesian space dynamic movement primitives," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2997–3004, 2014.

[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] G. Niemeyer and J.-j. E. Slotine, "A simple strategy for opening an unknown door," *Proceedings of the 1997 IEEE International Conference on Control Applications*, pp. 1448–1453, 1997.

[19] A. Albu-Schaffer, C. Ott, and G. Hirzinger, "A Unified Passivity-based Control Framework for Position, Torque and Impedance Control of Flexible Joint Robots," *The International Journal of Robotics Research*, vol. 26, no. 1, pp. 23–39, 2007.

[20] C. Ott, A. Albu-Schaffer, A. Kugi, S. Stramigioli, and G. Hirzinger, "A passivity based cartesian impedance controller for flexible joint robots-part I: Torque feedback and gravity compensation," *IEEE International Conference on Robotics & Automation*, pp. 2659–2665, 2004.

[21] 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.

[22] 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*, 2013, pp. 423–428.

[23] 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, pp. 166–173.

[24] B. Nemec, M. Simonic, L. Zlajpah, and A. Ude, "Enhancing the performance of adaptive iterative learning control with reinforcement learning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 2192 – 2199.