ADVANCED INTELLIGENT SYSTEMS Open Access
www.advintellsyst.com

Learning Assembly Tasks in a Few Minutes by Combining Impedance Control and Residual Recurrent Reinforcement Learning

Padmaja Kulkarni, Jens Kober, Robert Babuška, and Cosimo Della Santina*

Adapting to uncertainties is essential yet challenging for robots while conducting assembly tasks in real-world scenarios. Reinforcement learning (RL) methods provide a promising solution for these cases. However, training robots with RL can be a data-extensive, time-consuming, and potentially unsafe process. In contrast, classical control strategies can have near-optimal performance without training and be certifiably safe. However, this is achieved at the cost of assuming that the environment is known up to small uncertainties. Herein, an architecture aiming at getting the best out of the two worlds, by combining RL and classical strategies so that each one deals with the right portion of the assembly problem, is proposed. A time-varying weighted sum combines a recurrent RL method with a nominal strategy. The output serves as the reference for a task space impedance controller. The proposed approach can learn to insert an object in a frame within a few minutes of real-world training. A success rate of 94% in the presence of considerable uncertainties is observed. Furthermore, the approach is robust to changes in the experimental setup and task, even when no retrain is performed. For example, the same policy achieves a success rate of 85% when the object properties change.

1. Introduction

Assembly, hanging, and insertion are examples of essential tasks in robotics, which are common in industrial settings and in day-to-day activities. Using robots for these tasks has reduced process costs and improved the efficiency of the manufacturing process.[1] However, dealing with changing home environments or uncertainties in the industrial setting is still challenging.[2] Adaptable control approaches that can handle object pose and goal pose uncertainties are needed for the robust real-world application of robots.[3–6]

Adapting to uncertainties while conducting a task involving environment–robot dynamic interaction is challenging for conventional control methods. It is often difficult to model the environment, while accurately taking into account friction, object properties, and contacts.[7,8] Moreover, the physical parameters of such dynamic interactions are hard to identify. Thus, it is difficult to find a robust and generalized solution for these problems without spending significant effort in fine-tuning the controller.[9] In contrast to conventional methods, reinforcement learning (RL) offers promising solutions for such cases.[10,11] Instead of explicit problem formulation, RL methods allow for specifying the task reward. Furthermore, they can adapt to new situations that have not been experienced.

RL has shown promise in learning control tasks requiring robot–object interaction, for example, dexterous in-hand object manipulation[12] or opening doors.[13] However, using RL for real-world applications is still challenging as training robots with RL can be a data-extensive and time-consuming process. A typical RL approach can take millions of time steps to learn a pick and place task even when little or no uncertainties are present.[14] Moreover, the RL agent's initial exploratory actions can be unsafe for the robot and the environment.[15,16] To use an RL method directly in real-world settings, the method has to learn the task within a few minutes of training. In addition, the method should ensure the robot's safety while dealing with uncertainties in the environment.

One way to reduce learning time on a real-world robotic platform is to train the robot first in a simulation and then conduct fine tuning on the real robot. This paradigm is referred to as sim-

P. Kulkarni, J. Kober, R. Babuška, C. Della Santina
Department of Cognitive Robotics
Delft University of Technology
Delft 2628 CD, The Netherlands
E-mail: P.V.Kulkarni@tudelft.nl

R. Babuška
Czech Institute of Informatics Robotics and Cybernetics
Czech Technical University in Prague
Prague 160 00, Czech Republic

C. Della Santina
Institute of Robotics and Mechatronics
German Aerospace Center (DLR)
Oberpfaffenhofen 82234, Germany

The ORCID identification number(s) for the author(s) of this article can be found under https://doi.org/10.1002/aisy.202100095.

DOI: 10.1002/aisy.202100095

to-real in literature. Beltran-Hernandez et al.[17] used the soft actor-critic (SAC) algorithm to insert USB and LAN cables into ports. They train the robot in simulation for 50 000 time steps and in the real world for 15 000 time steps (20 min). However, when the object's properties, for example, deformability or size, change from the interaction with the environment, developing a simulation environment that resembles a real-world scenario can be quite difficult and more time-consuming than hand-engineering a solution on the real robot. Moreover, to apply such a technique, an expert must anticipate situations and design the simulator beforehand.

Inoue et al.[18] discretized the action space of the robot to enable fast learning for real-world robotic applications. They used recurrent deep neural networks with the *Q*-learning algorithm to learn the peg-in-hole task. However, the learnt RL model was used only for a single peg, hole, and clearance size. Different models are needed for different object sizes, and hence the method does not adapt to object sizes. Another approach to reducing learning time is to use sample-efficient model-based RL algorithms.[16] Fan et al.[19] combined model-based guided policy search (GPS) with the deep deterministic policy gradient (DDPG) algorithm to conduct high-precision assemblies. Luo et al.[20] used a variant of GPS for the robotic assembly of a rigid peg and deformable ring. However, these methods require manual tuning of force control gains and do not cope well with environmental variations.[17] Thomas et al.[21] combined trajectory planning with GPS to solve the robotic assembly problem when the object geometry is precisely known. However, they focus on following a precise trajectory instead of learning the assembly task itself. Thus, this method does not adapt to environment changes.

Reward shaping or learning from demonstration is also used to reduce the training time required for an RL algorithm. Wu et al.[22] used human demonstration and generative adversarial networks (GANs) to shape state-action potential function offline. This potential function is used as a reward in conjunction with the peg-in-hole task's sparse reward function while training the RL algorithm online. A few works used human demonstrations to learn peg-in-hole or assembly tasks.[23–27] However, they aim to learn the task assuming complete knowledge of the environment and cannot generalize to new situations.

In summary, training RL algorithms can be a data-extensive and time-consuming process. Furthermore, it can lead to unsafe robot actions that can potentially harm the robot and the environment while training. Thus, using RL directly for real-world training is not be efficient or safe.[16] Existing methods rely on pretraining RL algorithms using simulation environments or use human intervention while learning to reduce data requirements. Moreover, most of these methods learn the task under the assumption of correctly known object poses without considering the extreme uncertainties of real-world settings. In contrast, we propose an RL approach to handle uncertainties in contact-rich tasks via direct interaction with the environment without using any simulation environment. This is achieved by combining state-of-the-art impedance control with RL to alleviate data and safety concerns.

Impedance control is a generalization of stiffness and damping control, and it models robot–environment interaction as a mass–spring–damper system.[28] It provides an elegant way to enable robot–environment contact with stable motions during contact-state transitions.[29] Using Cartesian impedance control (IC) we ensure the robot's safety while interacting with the environment directly. However, although the impedance control can adapt to minor uncertainties, it cannot handle more extreme or discontinuous uncertainties.[30] Thus, using only the IC alone is not sufficient in real-world scenarios. In our article, we combine the IC with RL methods to deal with such uncertainties. This combination reduces RL's data and time requirement considerably, adapts to extreme position uncertainties, and safeguards the robot during direct training, as can be seen from the experiments later.

Johannink et al.[30] investigated residual RL with a sim-to-real approach to insert a block between two other blocks. However, they used a dense reward function that utilizes the knowledge of the actual goal pose and poses of all the blocks with a carefully crafted vision setup. For practical applications, this assumption might not hold due to the robot hand obstructing the object. In contrast, we do not make such assumptions and use a sparse reward while learning, wherein a robot is rewarded only when the object is at the goal position. Moreover, we aim at direct real-world robot training without the need for using a simulator beforehand.

In our approach, we formulate a nominal strategy to complete assembly tasks. A weighted sum of a nominal control strategy and an RL method generates a reference configuration for the IC. This combination safeguards the robot while interacting with the environment and can adapt to extreme environmental uncertainties. For the RL part, we extend twin-delayed DDPG (TD3),[31] with a recurrent neural network (RNN) architecture, namely, long short-term memory (LSTM). Furthermore, we introduce a decay factor that reduces the output of the nominal controller over time. The weighted sum of both controllers' actions is then passed to the IC, which generates control torques for the robot. Our approach is evaluated in a real-world setting for the task of inserting and hanging objects in a frame. During evaluation, we attempt to answer the following questions: 1) Does our approach outperform conventional feedback control methods, such as impedance control, in the presence of position uncertainties? 2) Can this method adapt to changes in object size and deformability?

The rest of the article is organized as follows. We explain the preliminaries of RL in Section 2. Section 3 defines the problem statement and discusses our approach along with the proposed algorithm. Section 4 describes our experimental setup, experiments, and results. Section 5 draws the conclusions for this work.

2. Preliminaries: RL

In RL algorithms, in an environment E, at a discrete time t, an agent observes state $s_t \in \mathbb{R}^n$, takes an action $a_t \in \mathbb{R}^m$, observes a scalar reward $r_t(s_t, a_t)$, and transitions to a state s_{t+1} for a deterministic case. The agent takes actions according to a deterministic policy μ. This policy is a mapping from states to actions, $\mu : \mathcal{S} \rightarrow \mathcal{A}$, where $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space. The return from a state s_i is defined as the sum of discounted future rewards from that particular state, and it is given as $R_t = \sum_{i=t}^{i=\infty} \gamma^{(i-t)} r(s_i, a_i)$. Here, $\gamma \in [0, 1]$ is a discount factor, which allows a trade-off to give a higher priority to the rewards received in the short term. The goal of the agent is to learn a policy that maximizes the expected return from the given start distribution. The expected return is given as $J = \mathbb{E}_{r_i, s_i \sim E, a_i \sim \mu}[R_0]$. This article considers episodic tasks that terminate after a discrete time or at an episode length T.

This RL formulation assumes that the underlying process is a fully observable Markov decision process (MDP). This is often not possible in practice, for example, due to noise in the sensors.[32] Most real-world systems are partially observed MDPs (POMDPs). The agent can observe its state at time t, s_t, in MDPs. The agent cannot observe this state directly in POMDPs. At a time t, the agent receives an observation $o_t \in \mathcal{O}$ that is conditioned on the state $p(o_t|s_t)$.[33] We model our problem as a POMDP and use observations instead of states. We extend the state-of-the-art algorithm TD3 using RNNs to handle the POMDP better. A detailed motivation behind this extension is given in Section 3.2.4. The following section explains the base algorithm TD3 and the rationale for our choice.

2.1. TD3

TD3 is the actor–critic algorithm that is stable, efficient, and needs less manual effort for parameter tuning than other policy-based methods.[30] It was proposed as an improvement over DDPG. Traditional Q-learning methods tend to overestimate Q-values and lead to suboptimal policies. This problem persists in actor–critic algorithms like DDPG. TD3 addresses this issue using double-Q-learning and outperforms purely Q-learning-based methods along with other actor–critic and policy gradient algorithms. It outperformed state-of-the-art algorithms like DDPG, proximal policy optimization (PPO), and SAC in challenging environments with continuous action space. TD3 showed better performance than these algorithms for tasks such as reaching goal positions with a robot arm, double-inverted pendulum, and contact-rich tasks such as walking.[31] Hence, we use TD3 as our base algorithm.

In actor–critic algorithms, the actor is the agent's policy μ, and the critic is a Q-value function that estimates the expected discounted return for taking an action a_t in state s_t, while following a policy μ henceforth. Both are represented as deep neural networks (DNNs). TD3 uses clipped double-Q learning, which learns two target Q-value functions. Let us assume that the policy is denoted as μ_θ and Q-value functions as Q_{ϕ_1} and Q_{ϕ_2}—with θ, ϕ_1, and ϕ_2 as the DNN parameters for the actor and two critic networks, respectively. At the start of the algorithm, TD3 initializes target networks ϕ'_1, ϕ'_2, and θ' with the same weights as ϕ_1, ϕ_2, and θ, respectively. The trick is to compute a target y, which is a value that the main Q-value function should be, using these target networks and update them only after a fixed number of time steps by copying values from the main network to achieve stable updates. At each time step t, the transition (s_t, a_t, r_t, s_{t+1}) is stored in a reply buffer $\mathcal{B}$. TD3 samples a minibatch of random transitions to compute a target for Q-value functions at each time step. It computes a single target for both Q-value functions by choosing the Q-value function with the least value. For a single transition (s_i, a_i, r_i, s_{i+1}) sampled from the replay buffer $\mathcal{B}$, the target is computed as follows.

$$y_i = r_i + \gamma \min_{j=1,2} Q_{\phi'_j}(s_{i+1}, a_{i+1}) \tag{1}$$

Here, a_{i+1} is the action obtained by adding a small random noise $(\varepsilon \sim \mathcal{N}(0, \sigma))$ to the action output by the target policy $\mu_{\theta'}$ and clipping it within a valid range such that $a_{i+1} = \mu_{\theta'}(s_{i+1}) + \varepsilon = \text{clip}(\mu_{\theta'}(s_{i+1}) + \text{clip}(\varepsilon, -c, c), a_{\text{low}}, a_{\text{high}})$. Here, c is a small constant and ε has the dimensions of the action vector. By adding this noise, TD3 ensures that the policy is robust to the Q-value function inaccuracies. Both Q-value functions regress to minimize the following expected squared loss using sampled transitions from the replay buffer, $\mathcal{B}$, to update ϕ_1 and ϕ_2.

$$\mathcal{L}(\phi, \mathcal{B}) = \mathop{\mathbb{E}}_{(s_i, a_i, r_i, s_{i+1}) \sim \mathcal{B}}[(y_i - Q_\phi(s_i, a_i))^2] \tag{2}$$

Here, $\phi = \{\phi_1, \phi_2\}$ and the operator $\mathbb{E}$ compute the probabilistic expectation of a variable, in this case, under the hypothesis that the samples (s_i, a_i, r_i, s_{i+1}) are drawn from the replay buffer $\mathcal{B}$. Finally, the agent's policy is learnt just by maximizing the expected Q-value function, Q_{ϕ_1}.

$$\max_\theta \mathop{\mathbb{E}}_{s_i \sim \mathcal{B}}[Q_{\phi_1}(s_i, \mu_\theta(s_i))] \tag{3}$$

The updates for θ and $\phi = \{\phi_1, \phi_2\}$ are computed at every time step using Equation (2) and (3). The updates for the targets θ' and $\phi' = \{\phi'_1, \phi'_2\}$ are then computed using a smoothing factor $\eta \ll 1$, such that, $\theta' = \eta\theta + (1 - \eta)\theta'$, and $\phi' = \eta\phi + (1 - \eta)\phi'$. Note that we do not provide a complete explanation of the algorithm TD3 and mention only the factors relevant for our approach. For in-depth information about this algorithm, readers can refer to the original paper.[31]

3. Proposed Solution

This section presents our approach. We formulate and provide a solution for an assembly problem in nominal conditions, that is, when no uncertainties are present in Section 3.1. Section 3.2 describes our algorithm and approach when uncertainties are present.

3.1. Solution of the Assembly Problem in Nominal Conditions

In this work, we consider assembly tasks involving two elements. Without the loss of generality, we consider one of the two elements at rest (frame or hanger hereinafter) and the other firmly grasped by a robotic manipulator in its end-effector (object hereinafter). With a stable grasp, the object does not move in the robot hand even when disturbed by a small external force.[34] Thus, a rigid coupling exists between the grasped object and the robot. As the robot moves, the object moves with it, keeping the same relative pose with respect to the robot end-effector. Let $\{B\}$, $\{EE\}$, $\{O\}$, and $\{H\}$ be the frames attached to the robot base, end-effector, object, and hanger, respectively. An ideal assembly problem with a known object geometry is: move the object $\{O\}$ based on its current state such that the object is inside the hanger. For simplicity, let us choose the object and hanger reference frames such that they coincide when the object is inside the hanger. **Figure 1**a shows a start state of the hanger, robot, and object system, and Figure 1c shows a goal configuration.

Let ${}^{H}_{B}T$, ${}^{O}_{B}T$, ${}^{EE}_{B}T$ be the transformation matrices of the hanger, object, and end-effector in the robot's base frame. The transformation matrix of the object with respect to the hanger frame is obtained as ${}^{O}_{H}T = {}^{O}_{B}T\,{}^{H}_{B}T^{-1}$. Then, we can define the assembly problem for the traditional control method as finding a trajectory such that the object and the hanger frame coincide, ${}^{H}_{O}T_{\text{end}} = I$. Note that each end-effector trajectory point ${}^{EE}_{B}T$ can be computed using the corresponding object trajectory point ${}^{H}_{O}T$ as ${}^{EE}_{B}T = {}^{EE}_{O}T\,{}^{H}_{O}T^{-1}\,{}^{H}_{B}T$. Figure 1 shows an example of such a scenario.

In an ideal case, when the object's exact geometry is known, its trajectory is computed using the exact geometry and the assembly constraints. This section explains such a procedure for the example in Figure 1. The direction approaching the goal is a straight line in this figure and all the other examples in Section 4. Nevertheless, the architecture proposed in the next section may also be applied to more complex assembly tasks with minimum changes. However, it is beyond the scope of the present article to discuss this possibility.

We consider a case where the direction of approach is free of obstructions in the nominal conditions. We choose to place the object such that one of its axes is aligned to the direction of insertion and the other two axes are parallel to the respective frame/hanger axes. This makes the mathematical formulation simpler. An example of this is shown in Figure 1b. Here, the object frame is chosen such that its transformation matrix with respect to the end-effector is

$$ {}^{EE}_{O}T = \begin{bmatrix} 1 & 0 & 0 & {}^{EE}_{O}p_X \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & {}^{EE}_{O}p_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4} $$

Here, ${}^{EE}_{O}p_X$ and ${}^{EE}_{O}p_Z$ is the distance between the end-effector and the object along the X and Z-axis, respectively. Here, if the object's goal position is (x_o, y_o, z_g), then we choose its new intermediate position on top of the goal (along the Z-axis), as (x_o, y_o, z_o). By virtue of Equation (4) and the object–robot rigid coupling assumption, the control problem for the robot becomes reaching the goal position $(x_o, y_o + {}^{EE}_{O}p_X, z_g + {}^{EE}_{O}p_Z)$ from position $(x_o, y_o + {}^{EE}_{O}p_X, z_o + {}^{EE}_{O}p_Z)$. Thus, the robot only needs to move along the negative Z-axis until the goal position is reached. This can be solved using a simple position-based nominal controller when all poses are perfectly known. The solution approach when goal position uncertainties are present is described in Section 3.2.

3.2. Combining Impedance Control and Residual Recurrent TD3 with a Decaying Nominal Controller Policy

The following challenges exist for the assembly task described earlier in real-world settings. 1) The object's position in the robot hand is not fully known and can be difficult to predict due to occlusions. 2) The object can move in the robot hand when subject to forces from the environment.

These uncertainties can result in incorrect goal position prediction, as shown in Figure 1d. In this scenario, the object is obstructed by the hanger completely, and a simple position-based nominal controller cannot be used to reach the actual goal. For

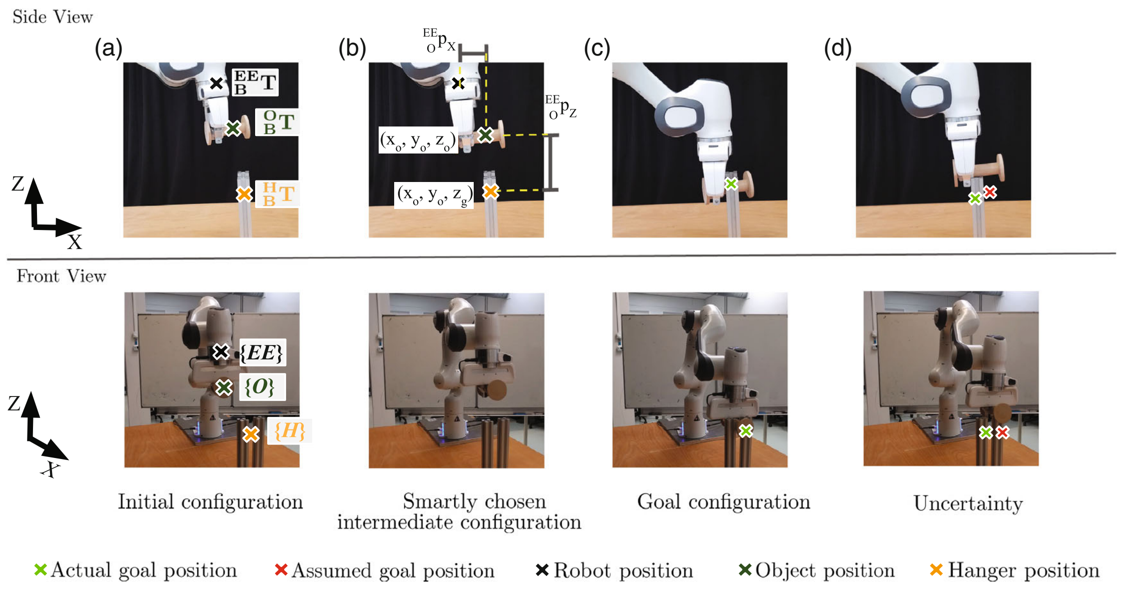


Figure 1. The colored crosses represent the frame attached to the object, hanger/frame, and robot end-effector. The aim is to move the object from an a) initial configuration in the image to the c) goal configuration in the image. By choosing an b) intermediate configuration in the image, we can reduce the insertion problem by simply moving along the Z-axis to reach the goal. In this article, we consider uncertainty, where the correct object or goal position is not known, and hence the controller's goal position can differ from the actual goal position, as shown in d) the image. Here, using transitional control approaches to find a solution can be difficult.

these situations, the control solution needs to incorporate the interaction dynamics between the object and the robot. Conventional control methods often divide the assembly task into contact detection and compliant control to solve this problem.[35] However, most approaches are optimized for one of the stages, and less research is focused on optimizing the whole task.[36] Moreover, compliant control strategies like impedance control or force-based compliant control can adapt to only minor uncertainties. They cannot handle more extreme and discontinuous uncertainties.[30]

In this article, we augment the combination of the nominal controller proposed in Section 3.1 and the IC with RL to learn assembly tasks in the presence of goal position uncertainties. We use RL in the task space rather than in joint space to make the trained policy invariant to the hanger position and orientation changes. The proposed RL algorithm, along with our control scheme, is explained below. **Figure 2** shows the control architecture of our approach, which comprises four distinct components: the IC, the time-varying averaging, the nominal controller μ_c, and the RL policy μ_θ. We describe in detail each of the blocks in the rest of the section.

3.2.1. Cartesian IC

We use a Cartesian IC without inertia shaping[37] to control the robot, as shown in Figure 2. The task space of the robot is the full configuration (position and orientation) of the end-effector. We use quaternions here to represent rotations. The joint torques u are computed using the following equation.[38]

$$u = -J(q)^T \left(S \begin{bmatrix} a' - P \\ \delta(1w + 0i + 0j + 0k, Q) \end{bmatrix} + DJ(q)\dot{q} \right) + C(q, \dot{q})\dot{q} + G(q) \tag{5}$$

Here, $q, \dot{q} \in \mathbb{R}^n$ are the robot joint angles and their time derivatives, respectively. $P \in \mathbb{R}^3$ and $Q \in \mathbb{Q}$ are the position and orientation of the end-effector in the object frame. δ implements a difference in quaternion space. $J \in \mathbb{R}^{7\times n}$ is the Jacobian matrix mapping $\dot{q}$ into velocities at the end-effector. $S, D \in \mathbb{R}^{7\times 7}$ are the desired stiffness and damping matrices, which we consider here to be diagonal. $C(q, \dot{q})\dot{q} \in \mathbb{R}^n$ is the torque applied to compensate for the Coriolis and centrifugal effect, and $G(q) \in \mathbb{R}^n$ is the gravity compensation torque. The closed loop generated by Equation (5) is such that the end-effector dynamics is equivalent to a spring–damper system with a configuration-dependent inertia tensor. For the position part, this is

$$\Lambda(q)\ddot{P} + D_P\dot{P} + S_P(a' - P) = 0 \tag{6}$$

where $\Lambda(q) \in \mathbb{R}^{3\times 3}$ is the mass as perceived at the end-effector, and $S_P, D_P \in \mathbb{R}^{3\times 3}$ are the top-left principal minors of S, D. Note that a' is then specified by the higher levels of the control architecture, as discussed later.

3.2.2. Weighted Sum

This simple yet fundamental component combines the a priori knowledge provided by the nominal algorithm with the learnt component. It computes a weighted sum of the nominal controller and RL algorithm actions. We formulate the assembly task as an episodic RL problem terminating at a discrete time horizon or an episode length T. At any discrete time $t \leq T$ this weighted sum is evaluated

$$a'_t = (1 - \lambda_t)a_t + \lambda_t a_{ct}, \quad \lambda_t = \max\left(0, \frac{T-t}{T}\right). \tag{7}$$

Here, a_t and a_{ct} are the actions' output by the RL algorithm (explained in Section 3.2.4) and the nominal controller (explained in Section 3.2.3), respectively, at a discrete time t. $\lambda \in [0, 1]$ is a decay parameter. The value of λ is one at the beginning of the task and is reduced at each time step linearly until it reaches zero. Thus, this formulation assigns more importance to the nominal controller at the beginning of the task and increases the weight of the RL algorithm over time. Using the time-varying weights, we ensure that the nominal controller can take the robot well within the goal's vicinity while gradually transferring control to the RL

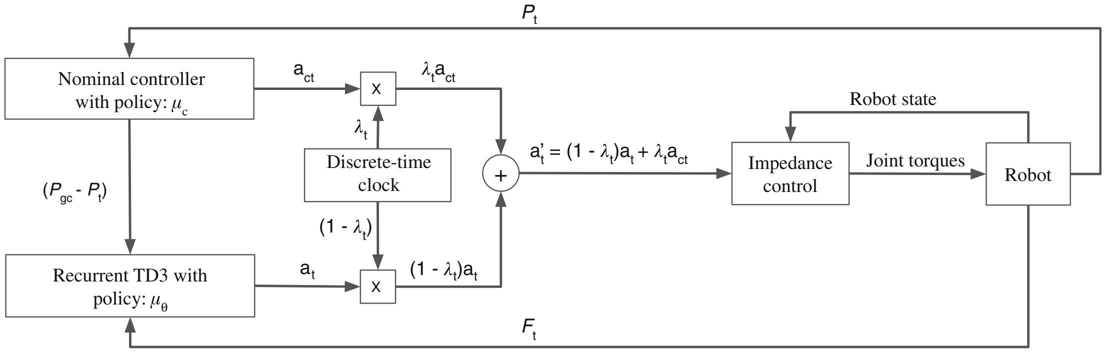


Figure 2. The control scheme for our approach. The nominal controller computes a goal position for the end-effector P_{gc} and outputs an action a_{ct} at a discrete time t. The RL block outputs an action a_t considering observed forces acting on the end-effector F_t and the difference between the assumed goal and current end-effector position $(P_{gc} - P_t)$. The final action to be executed is the weighted summation of the actions of the nominal controller and recurrent TD3 policy. This action is sent to the IC, which outputs the required joint torques to command the robot. Here, a_{ct} is the action output by the nominal controller at time t, and a_t is the action output by the recurrent TD3 policy at time t. λ_t is the time-dependent decay factor, and a'_t is the action (reference position) sent to the IC.

algorithm. Also, for our chosen class of assembly problems, uncertainties can be detected only when the robot is close to the goal and interacts with the frame.

The choice of T is clearly a critical hyperparameter to be tuned. It should be considered that good values should be higher than the settling time of Equation (6). In this way, if the nominal strategy can solve the task, then the RL will not have to intervene at all. As a rule of thumb, T can be selected to be two or three times the average settling time.

3.2.3. Nominal Strategy

This block computes the feed-forward nominal action proposed in Section 3.1, to be then used as a reference configuration of the virtual spring generated by the IC. Note that we output here only a 3D reference position as the reference rotation is constantly equivalent to the identity, as shown in Equation (4). This nominal control strategy is denoted by μ_c henceforth. The Cartesian IC discussed earlier will deal with minor orientation uncertainties arising from in-hand object rotation. Input to the Cartesian IC is the goal position of the end-effector. As we do not know the exact goal position, an assumed goal position P_{gc} is used as an input action.

To deter the robot from moving with high accelerations and generating very high interaction forces if the nominal policy fails, we pass the feed-forward action P_{gc} through a linear algorithm where at any discrete time t the policy μ_c outputs an action a_{ct} such that

$$a_{ct} = P_t + \mathrm{clip}(P_{gc} - P_t) \tag{8}$$

where P_t is the current end-effector positions expressed in the object frame and a_{ct} is the action output of the nominal controller's policy at time t. The reference position is clipped (i.e., saturated) within a suitable range. As a result, Equation (6) becomes $\Lambda(q)\ddot{P} + D_P\dot{P} + S_P(\mathrm{clip}(a' - P)) = 0$.

3.2.4. Learning the Residual Action: Recurrent TD3

In this block, we explain our extension of the TD3 algorithm. TD3 is introduced, assuming that the underlying process is a fully observable MDP. This is often not possible in practice. Hence, we model our system as a POMDP. In POMDPs, the agent can observe the underlying state only indirectly, based on past observations.

Existing approaches using POMDP formulation can be categorized as probability-based and memory-based approaches.[39] A naive approach to dealing with POMDPs is learning reactive stochastic policies that map observations to action probabilities.[40] It assumes that taking random actions can prevent the agent from getting stuck in the loop due to ambiguous or partial observations. Unfortunately, this approach can produce suboptimal policies.[39,41] Memory-based approaches learn from past observations with RNNs, recently, using LSTM units. For example, other studies[32,33,42] used LSTMs to approximate Q-value functions, policies, and actor–critic networks, respectively. These works showed that using LSTMs can substantially improve performance for high-dimensional continuous tasks with partial observability compared with their nonrecurrent counterparts. Therefore, using a recurrent network to extend TD3 is beneficial for solving real-world robotic applications which fall under the POMDP category. We use the following RL formulation.

States: Let the forces and the moments acting on the robot end-effector expressed in the object frame at a discrete time t be $F_t = (f_{xt}, f_{yt}, f_{zt}, n_{xt}, n_{yt}, n_{zt})$. Let the end-effector's current and the assumed goal positions in the object's frame be $P_t = (x_t, y_t, z_t)$ and $P_{gc} = (x_{gc}, y_{gc}, z_{gc})$, respectively. Then, the state of the system is given as $s_t = (F_t, P_{gc} - P_t)$.

RL Policy and Actions: We use a continuous 3D action space. An RL policy μ_θ at time t outputs an action a_t such that $a_t = (\Delta x_t, \Delta y_t, \Delta z_t)$. This action states how much the end-effector needs to move along each axis in the object frame. The policy parameters are denoted by θ.

Rewards: If the goal is not reached, the robot receives a reward of -1 for that time step. When the goal is reached, a reward of 100 is received.

In our extension of recurrent TD3, 1) instead of sampling a minibatch of single transitions (s_i, a_i, r_i, s_{i+1}) from the replay buffer $\mathcal{B}$ with an actual state s, a minibatch of trajectories τ of such transitions with observations is sampled. This trajectory is then used for computing the loss in Equation (2). An example of such a trajectory of length (time window) w is $\tau_i = ((o_i, a_i, r_i, o_{i+1}), (o_{i-1}, a_{i-1}, r_{i-1}, o_i), \ldots, (o_{i-w+1}, a_{i-w+1}, r_{i-w+1}, o_{i-w+2}))$. 2) Consequently, instead of a single state, a sequence of w consecutive observation–action pairs is sampled while updating the policy parameters in Equation (3). 3) An RNN, namely, LSTM, is used to represent both actor and value function networks. Note that RNNs can summarize this transition in their recurrent state without explicitly using the trajectories. However, in a practical setting, we observed that faster learning is achieved using such trajectories.

Let us suppose that the observation–action transition history for a sampled trajectory τ_i is $h_i = ((o_i, a_i), (o_{i-1}, a_{i-1}), \ldots, (o_{i-w+1}, a_{i-w+1}))$, and the reward history is $r_i^h = (r_i, r_{i-1}, \ldots, r_{i-w+1})$. Similarly, o_i^h and a_i^h are the action and observation histories, respectively.

The time step history at index $(i+1)$ then is denoted as h_{i+1}. The same subscript notation applies to the action, reward, and observation histories. We compute the target vector from Equation (1) for recurrent TD3 as

$$y_i^h = r_i^h + \gamma \min_{j=1,2} Q_{\phi'_j}(o_{i+1}^h, \mu_{\theta'}(o_{i+1}^h) + \varepsilon) \tag{9}$$

Here, both actor and critic output an array of Q-values and actions along the time axis. The critic then regresses to minimize the expected loss for a trajectory τ_i.

$$\mathcal{L}(\phi, \mathcal{B}) = \mathop{\mathbb{E}}_{\tau_i \sim \mathcal{B}}\left[\|(y_i^h - Q_\phi(h_i))\|_2^2\right] \tag{10}$$

Here, $\|.\|_2$ is the Euclidean norm. Finally, the agent's policy is learnt just by maximizing the expected Q-value function Q_{ϕ_1}.

$$\max_{\theta} \mathop{\mathbb{E}}_{\tau_i \sim \mathcal{B}}[|Q_{\phi_1}(o_i^h, \mu_\theta(o_i^h))|] \tag{11}$$

Here, $|.|$ is the absolute-value norm. The final learnt recurrent policy is then used to compute action a_t at time t, as shown in Figure 2.

Algorithm 1 explains the training procedure for residual recurrent TD3 with a decaying nominal controller. We first initialize the replay buffer using a random policy for the first b time steps. Later, we reset the robot's initial state, s_0, for each episode based on the assumed goal position and reset the number of episode time steps t, to 0. Until the task is finished or the number of time steps exceeds the episode's maximum time steps, T, we compute the action by weighted summation of the RL policy and nominal controller actions. The resulting action is then passed to the IC. A transition consisting of the current observation, action, next observation, and reward is added to the reply buffer. By sampling the sequence w of such transitions, we optimize the RL policy using recurrent TD3. Note that our method is algorithm agnostic, meaning that any algorithm that supports continuous action space can be used as the base algorithm instead of TD3.

We assume that while conducting most assembly tasks (especially tasks considered in this article), our approach does not require RL to adapt to uncertainties in orientations. This

Algorithm 1. Proposed RL algorithm and residual recurrent TD3 with a decaying nominal controller policy. Here, T is time horizon T, w is time sequence length, b is the number of initial data collection steps, M is the maximum number of time steps, and P_t is the current end-effector position. The pop function (line 9) removes the value at index 0 and returns the rest of the list. The append function (line 15, 16) appends the latest values to the list. τ is a minibatch of n transition trajectories each of length w sampled from the replay buffer $\mathcal{B}$.

```
1 Required: Nominal controller $\mu_c$, random data collection policy $\mu_r$, recurrent TD3 policy $\mu_\theta$.
2 $\mathcal{B}\leftarrow$InitializeReplayBuffer($\mu_r$)
3 for Time Steps = 1, M do
4:   $s_0 \sim E$
5    $t\leftarrow 0$
6    $h_t\leftarrow\{\}$, $o_t^h\leftarrow\{\}$
7    for t =[0, T] and Task != Done do
8       if length($h_t$) $\geq w$ then
9          $h_t\leftarrow$Pop($h_t$, 0), $o_t^h\leftarrow$Pop($o_t^h$, 0)
10      end
11      $\lambda = 1 - \frac{t}{T}$
12      $(a_t, \_) = \mu_\theta(o_t^h) + \varepsilon$
13      $a'_t = (1-\lambda)a_t + \lambda\mu_c(P_t)$
14      $(o_{t+1}, r_t)\leftarrow$Execute($a'_t$)
15      $h_t\leftarrow$Append($h_t$, $(o_t, a_t)$))
16      $\mathcal{B}\leftarrow$Append($\mathcal{B}$, $(o_t, a_t, r_t, o_{t+1})$)
17      $\tau : \tau_1, \tau_2, \ldots, \tau_n \sim \mathcal{B}$
18      $\theta\leftarrow$RecurrentTD3.Optimize($\theta, \tau$)
19   end
20 end
```

assumption is based on the ability of the impedance comptroller to adapt to minor orientation uncertainties and that of RL to handle extreme position uncertainties arising from real-world interactions. A validation of this assumption can be seen later in the experiments. Nevertheless, the proposed framework can extend straightforwardly to learning orientations by changing the RL algorithm's state space and action space to include orientations. This RL reformulation would enable the robot to conduct assembly tasks requiring learning orientation uncertainty, such as assembling mechanical parts with negative or tight tolerances.

4. Experimental Evaluation

In this section, we present real-world validation of the proposed technique in insertion and hanging tasks. Our experimental setup is described in Section 4.1. Section 4.2 presents the experiments and evaluation results for the insertion and hanging task. Furthermore, we conduct experiments to validate our approach in different real-world settings like hanging glasses and stacking discs on a wooden peg. The results of these validation experiments are presented in Section 4.3.

4.1. Experimental Section

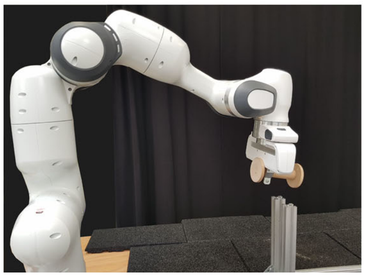
The experimental setups for the hanging and insertion scenarios and object variation are shown in **Figure 3**. We used a 7 degrees-of-freedom Franka Emika Panda arm with a parallel gripper.[43] The robot was controlled at 15 Hz using a Cartesian IC, which was tolerant of the contacts and environmental collisions. We assumed that the object was already grasped during experiments, and it was $\approx$10 cm in front of the assumed goal position. There was a tolerance of 3 mm between the object diameter at the insertion point and the frame width.

We manually inserted the object grasped by the robot in the frame and recorded the goal position of the robotic end-effector with respect to the frame. Note that this information is not included in the RL's state representation and is only used to give a sparse reward to the robot if the robot reaches the goal state. A time sequence of 20 states was used as input to our RL framework. The actor and critic networks of the algorithm were represented by an RNN of size $(200, 100, 40, 200, 100)$. In this, 40 was the number of LSTM units. The rest was fully connected hidden-layer sizes. We used the TD3 implementation from the Tensorflow library[44] for implementing our algorithm. An episode was an attempt of the robot to hang/insert the object at the desired location. The episode ended when the robot executed a maximum of 100 control actions (steps), or the goal was reached. These actions were sampled at 15 Hz. We do not make any assumptions about object interaction dynamics, contacts, and friction throughout the task execution.

4.2. Results

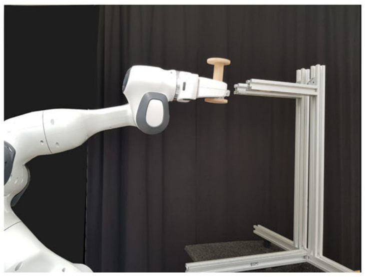
The experiments aim to answer whether our approach can handle goal position uncertainties and object property changes. Moreover, we show that the learnt policy is agnostic of the frame's position and orientation by evaluating the policy learnt in the insertion setup on the hanging setup. Finally, we validate

(a)



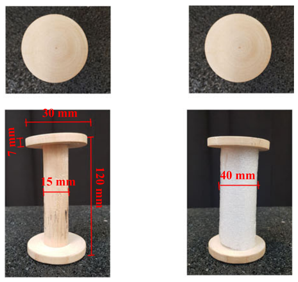
Setup to insert the object in the frame.

(b)



Setup to hang the object in the frame.

(c)



Objects used during experimentation.

Figure 3. The experimental setup with the Panda arm, the object, and the frame. The task of the robot is to insert or hang the object into the frame. a,b) Panels show the insertion and hanging setup, respectively, with the original object without any foam. c) Panel shows the original object to the left and the object covered with 5 mm-thick foam along its axis to the right. The foam is used to change the object size and surface properties. The experiments evaluate the robustness of the learnt policy for these property variations.

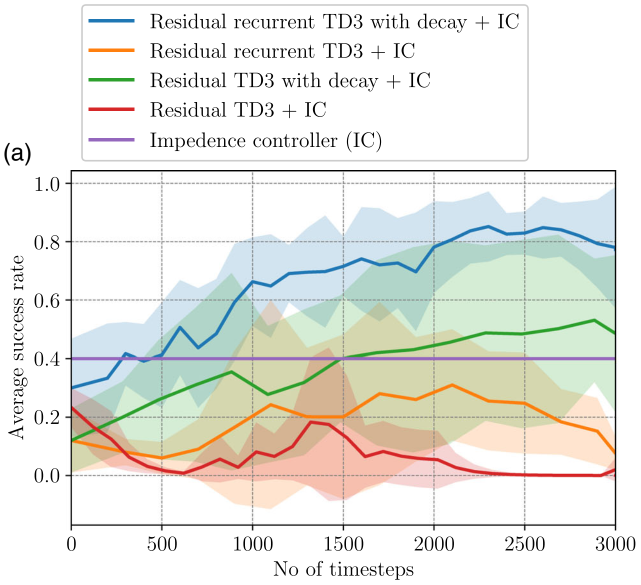
our approach by applying it to the task of inserting the disks on a stacking tower and the real-world application of hanging wine glasses into racks.

4.2.1. *Uncertain Goal Position*

In this scenario, we introduce a Gaussian noise of 0 mean and 15 mm standard deviation along the X, Y, and Z-axes of the frame. The setup for this experiment is shown in Figure 3. We conduct five experiments using different random seeds to initialize the proposed RL algorithm, residual recurrent TD3 with a decaying nominal controller policy. Furthermore, in an ablation study, we compare our method with recurrent TD3, residual TD3, using a decay factor with residual TD3, and using the IC without any RL. Note that the output of these RL algorithms is passed to the IC, which in turn controls the robot.

The success rate and average episode length plots of these five runs are shown in **Figure 4**. It can be seen that our method learns to insert the object with over 80% success rate in 3000 time steps. **Figure 5** shows the average success rate of five runs against training time in minutes. It can be seen that our method learns the task of insertion in less than 4 min of real-world training with a success rate of more than 80%. Furthermore, we perform 100 insertions using the best-learnt policy to evaluate the performance of the learnt policy with our algorithm, residual TD3, both with and without the decay factor and the using the impedance control without any RL. **Table 1** shows the number of successful insertions. Video stills of the robot inserting the object using our method are shown in **Figure 6**.

(a)



(b)

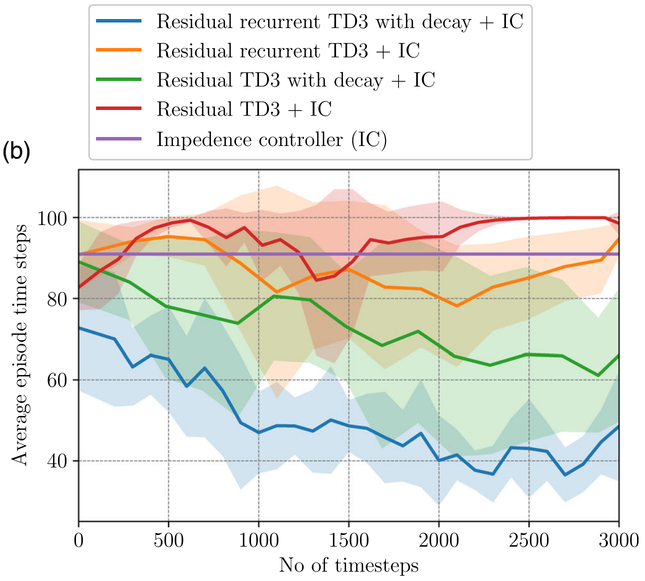


Figure 4. a) Average success rate and b) average episode length plots for five real-world robotic experiments for inserting an object in the frame. It can be seen that our method works better than other variations of RL algorithms. This can be attributed to the fact that adding recurrence to the TD3 formulation can help in faster learning. In addition, using a decay factor adds the implicit constraint on the algorithm that correcting trajectories using RL is more important as the robot is closer to the goal.

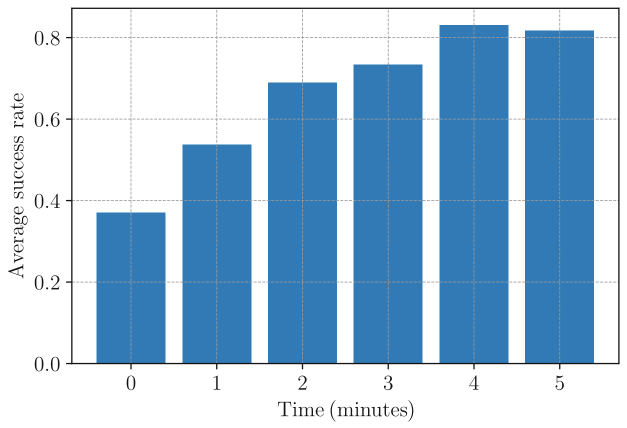


Figure 5. Average success rate and training time plot for five real-world robotic experiments for inserting an object in the frame. It can be seen that our method achieves an average success rate of over 80% with 4 min of real-world interactions.

4.2.2. Object Property Change and Uncertain Goal Position

Here, we wrap the object in foam to make the object deformable and to change object size, as shown in the second image of Figure 3c, and the insertion setup same as shown in Figure 3a. We perform 100 insertions of the object using our algorithm's policy with the highest success rate in the insertion setup.

Our policy was able to insert the object 85/100 times successfully. However, we observed that the average episode length when the object is wrapped in the foam was observed to be 69 time steps instead of 45 time steps with the original object. This can be attributed to the fact that the enveloping object in the foam affects robot–object–hanger system dynamics. Moreover, due to the increased diameter of the object, when the hanger obstructs the object, the contact area increases. Due to this, the hanger–object contact time before the object can be inserted can increase. These reasons contribute to the robot needing more time to complete the task.

Despite these changed dynamics, our approach can successfully insert the object in the frame without additional training. Video stills of the robot inserting the object using our method are shown in **Figure 7**.

Table 1. The result of evaluating different residual learning methods combined with Cartesian impedance control and using only the Cartesian impedance control without any RL for 100 attempts of inserting the object in the frame.

Method	% of successful grasps
Residual recurrent TD3 with decay + IC	94%
Residual recurrent TD3 + IC	24%
Residual TD3 with decay + IC	69%
Residual TD3 + IC	10%
Impedance control without RL	40%

4.2.3. Effect of Change in Frame Position and Orientation

Here, we rotate the frame and reposition it to create a hanging scenario, as shown in Figure 3b. We conduct 100 hanging experiments using the policy learnt in the insertion setup. Our policy was able to hang the object successfully for 86/100 attempts. Video stills of the robot hanging the object are shown in **Figure 8**.

We do not conduct any additional training in this scenario. For our RL approach, observed states consist of the forces on the end-effector and the difference between the assumed goal position and the end-effector's current position, both expressed in the hanger frame. None of these variables rely on the absolute position and orientation of the end-effector. Thus, as long as we can define the insertion axis along the negative Z-axis of the hanger, the states and actions of our RL's approach remain consistent

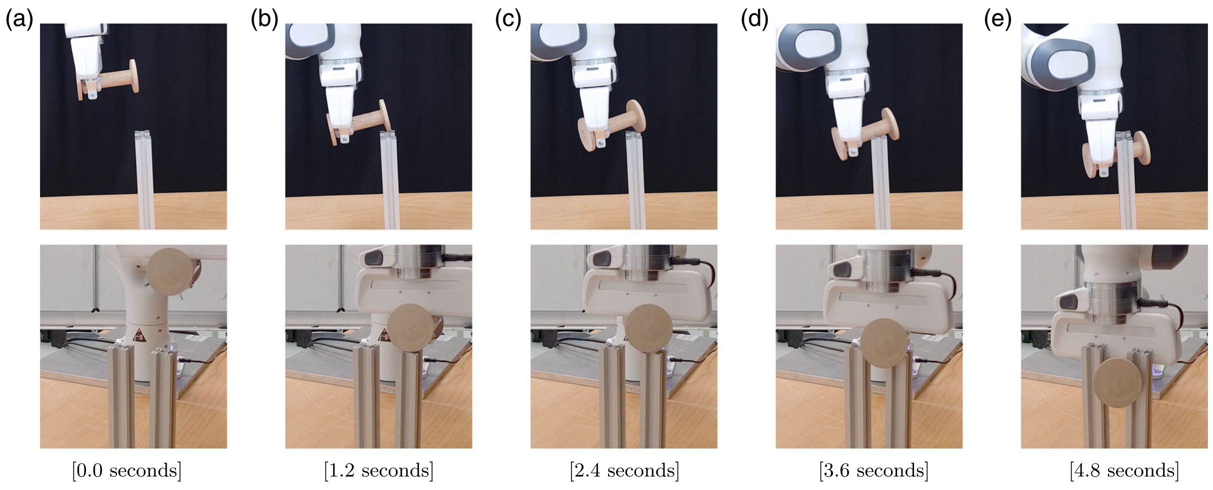


Figure 6. Video stills of the robot inserting the object. The first row is the side view, and the second row is the front view of the corresponding image. The captions are the timestamps of image stills. In images (b) and (c), it can be seen that the frame obstructs the object. The subsequent stills show that the policy can insert the object despite the erroneous goal position. The image captions show real-world timestamps.

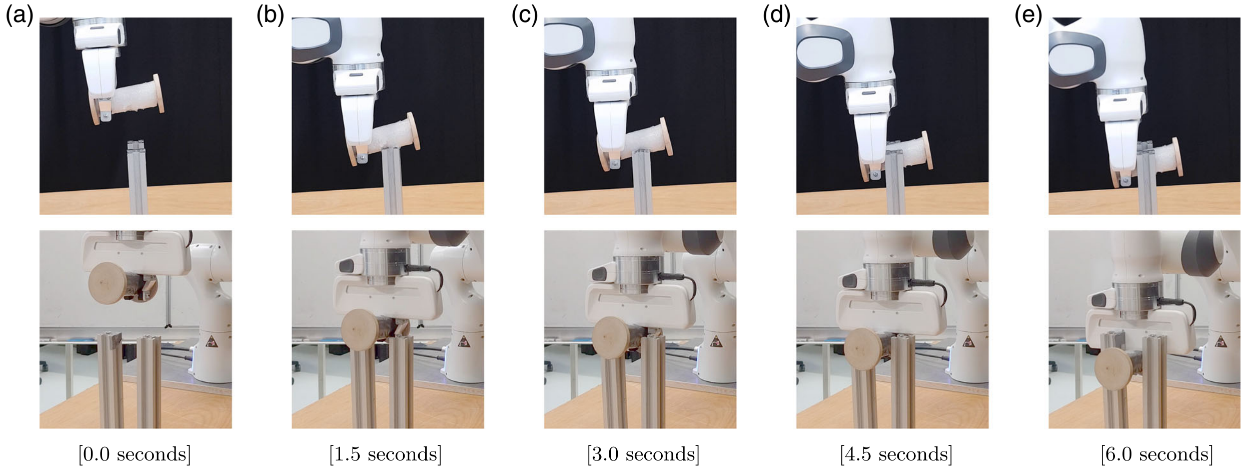


Figure 7. Video stills of the robot inserting the foam-enveloped object. The first row is the side view, and the second row is the front view of the corresponding image. In images (b) and (c), it can be seen that the frame obstructs the object. The subsequent stills show that the policy can insert the object despite the erroneous goal position. The image captions show real-world timestamps.

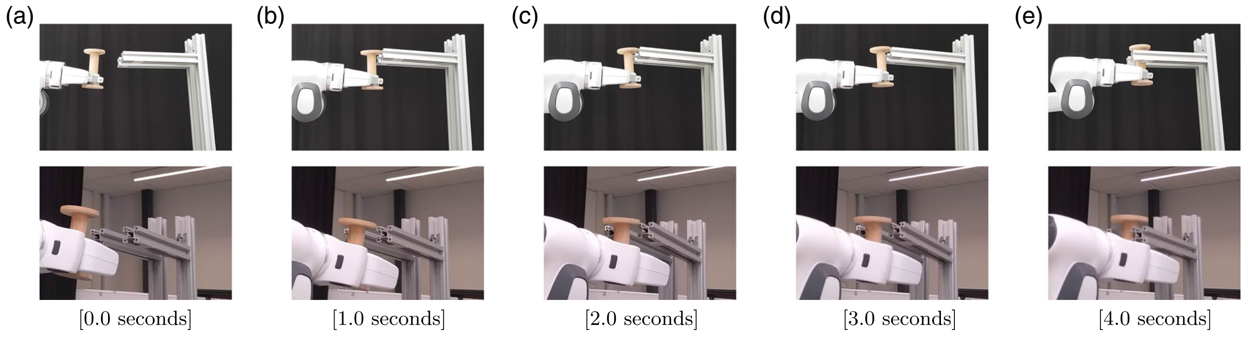


Figure 8. Video stills of the robot hanging the object. The first row is the side view, and the second row is the closer front view of the corresponding image. In images (b) and (c), it can be seen that the frame obstructs the object. The subsequent stills show that the policy is able to hang the object despite the erroneous goal position. The image captions show real-world timestamps.

with the learned policy. Thus our policy is valid for every orientation and position change of the hanger.

In this hanging scenario, we needed to reset the experiment after every ten attempts, as the robot's joints were close to joint limits. Thus, learning the task in this configuration was not suitable for the robot. By choosing the end-effector pose-agnostic state and action representation, we learnt the policy in the insertion setup and applied it directly to the hanging setup successfully.

Note that in Section 3.2.4, we assume that the RL method does not need to learn orientation uncertainties for most assembly tasks in our framework. Our experiments validate this assumption. For the presented tasks of inserting the object in the frame, the object moves and rotates in the robotic hand due to environmental collisions, as shown in Figure 6 and 8. In this scenario, the object is at an angle with respect to the frame. The IC can handle these rotations, whereas RL adapts to the position uncertainties. Thus our approach can complete these tasks, even when position and orientation uncertainties are present, without having RL learn additional orientation corrections.

4.3. Validation Experiments

We validate our approach using two experimental setups, stacking discs onto a tower and hanging wine glasses in the rack, and, prove that our method can be used in real-world scenarios.

4.3.1. Stacking Discs onto a Tower

In this setup, the task is to insert wooden disks onto a wooden tower in the presence of position uncertainties. The discs have varying diameters of 8, 7.5, and 7 cm. The wooden stick of the tower and the disc hole have a clearance of 2 mm. We add a Gaussian noise of 0 mean and 15 mm standard deviation in the goal position of the discs. While training, only one disc of diameter 7.5 cm is used. Like the previous experiments, we train the policy for 3000 time steps. We conduct 100 disc insertions using the trained policy. Our policy was able to insert the disc successfully for 88/100 attempts. The number of successful insertions without using RL was 56/100. Thus, our approach was able to achieve high success rate for a novel task within

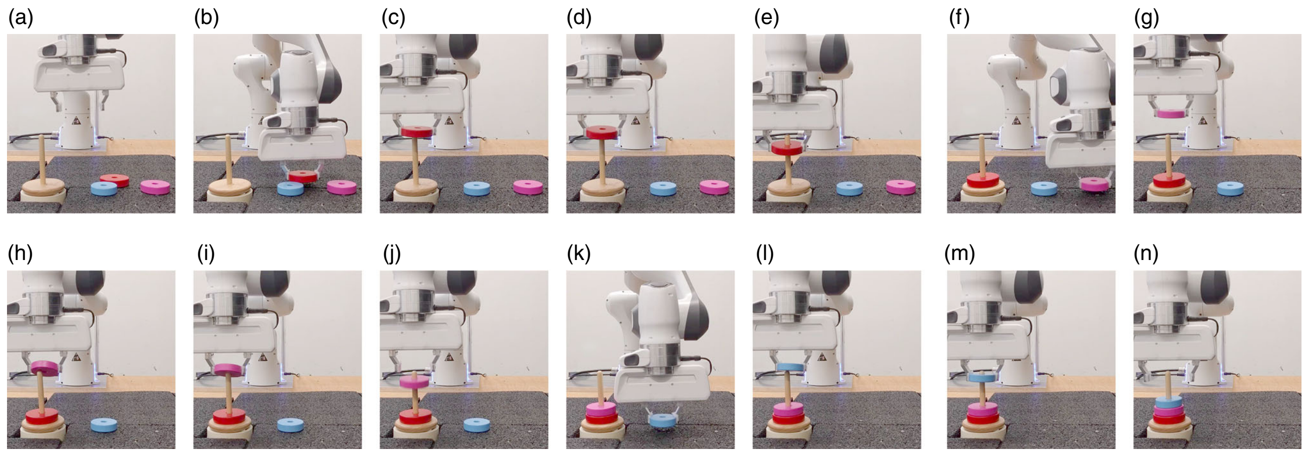


Figure 9. Video stills of the robot stacking the discs onto the wooden tower. The sequence shows that the learnt policy can be successfully integrated into the whole pick-and-place pipeline. c,d,h,i,l) The disc obstructed by the wooden stick. The subsequent images for each show that despite the initial erroneous goal positions, the robot is able to place all three discs onto the tower.

5 min of real world training. Furthermore, we use this policy in a complete pick-and-place pipeline. While validating our approach in this setup, the robot picks up a disc and moves it on top of the assumed goal. At this point, the robot starts using the learnt policy to insert the object. While using only impedance control fails in the particular scenario, as shown in **Figure 9**, our method can successfully insert the discs onto the tower.

4.3.2. Hanging Wine Glasses

In this setup, we use our method to hang wine glasses in a rack in the presence of position uncertainties. This setup is particularly challenging because 1) grasp forces on the glass cannot increase, as greater forces might break the glass, and 2) grasps on the wine glasses are not form-closure grasps due to the unavailability of a stable surface. These two reasons cause the glass to rotate inside the robot hand and fall with the application of a small external force. Here, we needed to identify a low-stiffness setting of the IC that does not exceed this force while interacting with the environment. We use the same policy that was used for the hanging task in Section 4.2.3, as both the objects share similar characteristics in terms of the "neck" part that goes into the rack or frame.[45] We hang the wine glass 100 times in the rack, while adding Gaussian noise of 0 mean and 15 mm standard deviation in the goal position. Without using any RL, the glass could be hung 38/100 times. Our policy was able to hang the glass successfully for 81/100 attempts. Our approach could generalize the trained policy for novel objects that shared similar features with the objects used in the training, with a high success rate. Furthermore, we use this policy in a complete pick-and-place pipeline. While validating our approach in this setup, the robot picks up a glass and moves it in front of the assumed goal. At this point, the robot starts using the learnt policy to insert the glass. Our approach can successfully hang the glass, as shown in **Figure 10**.

5. Conclusion and Future Work

This article proposed an approach combining conventional methods and RL to conduct assembly tasks robustly in the presence of uncertainties. We extended the algorithm TD3 using LSTM

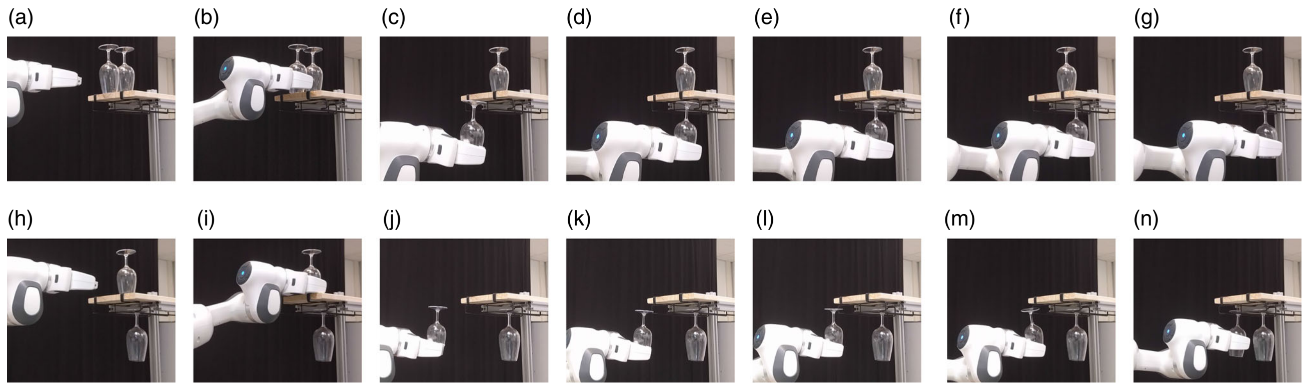


Figure 10. Video stills of the robot hanging the wine glasses in the racks. The sequence shows that the learnt policy can be successfully integrated into the whole pick-and-place pipeline in a real-world scenario. d,e,k,l,m) The glasses obstructed by the rack. The subsequent images for each of them show that despite the initial erroneous goal positions, the robot could hang the two glasses into the rack.

networks and formulated the recurrent TD3 algorithm. We developed a nominal control strategy and augmented it using the recurrent TD3 algorithm and IC to adapt to uncertainties in the environment. The proposed approach could learn to hang and insert objects in a few minutes of real-world training.

With a Gaussian goal position uncertainty of 0 mean and 15 mm variance, our method was able to insert the object with a 94% success rate, while impedance control alone could achieve only 40%. Learning in task space allowed us to generalize the learnt policy for different frame positions and orientations. After changing the object's deformability and size by enclosing the object in a foam sheet and after changing hanger configurations, our method was able to insert and hang the object with at least an 85% success rate. Thus, we show that the learnt policy is robust to changes in the object's surface properties and agnostic to the frame's position and orientation. Furthermore, our approach outperforms pure impedance control and the variations of residual control methods with and without the decay factor and recurrence. Moreover, our policy can be applied to different setups, such as inserting discs on a tower or hanging wine glasses in a rack with a few minutes of training.

In the future, we would like to extend this method to use vision to learn RL policies that can adapt to objects with very different shapes and characteristics. Another direction for future research is to automatically compute the weight for the RL method and nominal controller instead of using a fixed decay function.

Acknowledgements

This research was funded by the Netherlands Organization for Scientific Research project Cognitive Robots for Flexible Agro-Food Technology, grant P17-01. *The Interactive Supporting Information of this article can be found at*: https://www.authorea.com/doi/full/10.22541/au.162671181.19429485/.

Conflict of Interest

The authors declare no conflict of interest.

Data Availability Statement

Research data are not shared.

Keywords

industrial robots, reinforcement learning, robot manipulation

Received: May 15, 2021
Revised: July 7, 2021
Published online: September 2, 2021

[1] P. Zou, Q. Zhu, J. Wu, J. Jin, in *Proc. – 2019 Chinese Automation Congress, CAC 2019*, IEEE, Piscataway, NJ **2019**, pp. 3269–3273.
[2] A. Bicchi, V. Kumar, in *Proc. 2000 ICRA. Millennium Conf. IEEE Int. Conf. on Robotics and Automation. Symp. Proc.* (Cat. No.00CH37065), IEEE, Piscataway, NJ **2020**, p. 1.
[3] A. Billard, D. Kragic, *Science* **2019**, *364*, 6446.
[4] L. Manuelli, W. Gao, P. Florence, R. Tedrake, *kPAM: KeyPoint Affordances for Category-Level Robotic Manipulation*, arXiv, **2019**.
[5] H. Bruyninckyx, S. Dutre, J. De Schutter, in *Proc. – IEEE Int. Conf. on Robotics and Automation*, vol. *2*, IEEE, Piscataway, NJ **1995**, pp. 1919–1924.
[6] F. Dietrich, D. Buchholz, F. Wobbe, F. Sowinski, A. Raatz, W. Schumacher, F. M. Wahl, in *IEEE/RSJ 2010 Int. Conf. on Intelligent Robots and Systems, IROS 2010 – Conf. Proc.*, IEEE, Piscataway, NJ **2010**, pp. 2313–2318.
[7] J. Kober, J. A. Bagnell, J. Peters, *Int. J. Robotics Res.* **2013**, *32*, 1238.
[8] S. Hofer, K. Bekris, A. Handa, J. C. Gamboa, M. Mozifian, F. Golemo, C. Atkeson, D. Fox, K. Goldberg, J. Leonard, C. Karen Liu, J. Peters, S. Song, P. Welinder, M. White, *IEEE Trans. Autom. Sci. Eng.* **2021**, *18*, 398.
[9] G. Schoettler, A. Nair, J. Luo, S. Bahl, J. A. Ojea, E. Solowjow, S. Levine, in *IEEE Int. Conf. on Intelligent Robots and Systems*, IEEE, Piscataway, NJ **2019**, pp. 5548–5555.
[10] Y. P. Pane, S. P. Nageshrao, J. Kober, R. Babuška, *Eng. Appl. Artif. Intell.* **2019**, *78*, 236.
[11] S. Dong, D. K. Jha, D. Romeres, S. Kim, D. Nikovski, A. Rodriguez, *Tactile-RL for Insertion: Generalization to Objects of Unknown Geometry*, arxiv, **2021**.
[12] O. M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, W. Zaremba, *Int. J. Robotics Res.* **2020**, *39*, 3.
[13] S. Gu, E. Holly, T. Lillicrap, S. Levine, in *Proc. – IEEE Int. Conf. on Robotics and Automation*, IEEE, Piscataway, NJ **2016**, pp. 3389–3396.
[14] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, W. Zaremba, *Adv. Neural Inf. Process. Syst.* **2017**, *2017*, 5049.
[15] O. Kroemer, S. Niekum, G. Konidaris, *J. Mach. Learn. Res.* **2021**, *22*, 1, http://jmlr.org/papers/v22/19-804.html.
[16] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, S. Levine, *Int. J. Robot. Res.* **2021**, *40*, 698.
[17] C. C. Beltran-Hernandez, D. Petit, I. G. Ramirez-Alpizar, K. Harada, *Appl. Sci. (Switzerland)* **2020**, *10*, 1.
[18] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, R. Tachibana, in *IEEE Int. Conf. on Intelligent Robots and Systems*, Vol. *2017*, IEEE, Piscataway, NJ **2017**, pp. 819–825.
[19] Y. Fan, J. Luo, M. Tomizuka, in *Proc. – IEEE Int. Conf. on Robotics and Automation*, IEEE, Piscataway, NJ **2018**, p. 811.
[20] J. Luo, E. Solowjow, C. Wen, J. A. Ojea, A. M. Agogino, in *IEEE Int. Conf. on Intelligent Robots and Systems*, IEEE, Piscataway, NJ **2018**, pp. 2062–2069.
[21] G. Thomas, M. Chien, A. Tamar, J. A. Ojea, P. Abbeel, in *Proc. – IEEE Int. Conf. on Robotics and Automation*, IEEE, Piscataway, NJ **2018**, pp. 3524–3531.
[22] Y. Wu, M. Mozifian, F. Shkurti, *Shaping Rewards for Reinforcement Learning with Imperfect Demonstrations using Generative Models*, arXiv, **2020**.
[23] Z. Zhu, H. Hu, *Robotics* **2018**, *7*, 17.
[24] M. Kyrarini, M. A. Haseeb, D. Ristić-Durrant, A. Gräser, *Autonomous Robots* **2019**, *43*, 239.
[25] G. Ding, Y. Liu, X. Zang, X. Zhang, G. Liu, J. Zhao, *Sensors (Switzerland)* **2020**, *20*, 1.
[26] K. Kronander, E. Burdet, A. Billard, in *Workshop on Human-Robot Interaction for Industrial Manufacturing Robotics Science and Systems* **2014**.
[27] L. Rozo, M. Guo, A. G. Kupcsik, M. Todescato, P. Schillinger, M. Giftthaler, M. Ochs, M. Spies, N. Waniek, P. Kesper, M. Büerger, in *IEEE Int. Conf. on Intelligent Robots and Systems*, IEEE, Piscataway, NJ **2020**, pp. 9072–9079.
[28] J. F. Broenink, M. L. J. Tiernego, in *Proc. 8th European Simulation Symp., Simulation in Industry*, Society for Computer Simulation International, Genoa, Italy **1996**, pp. 504–508.

[29] N. Hogan, in *Proc. of the American Control Conf.*, Vol. *1*, IEEE, Piscataway, NJ **1984**, pp. 304–313.
[30] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, S. Levine, in *Proc. – IEEE Int. Conf. on Robotics and Automation*, IEEE, Piscataway, NJ **2018**, p. 6023.
[31] S. Fujimoto, H. van Hoof, D. Meger, in *Proceedings of the 35th International Conference on Machine Learning*, (Eds: J. Dy, A. Krause), Proceedings of Machine Learning Research, Vol. 80, PMLR, July **2018**, pp. 1587– 1596, http://proceedings.mlr.press/v80/fujimoto18a.html.
[32] D. Wierstra, A. Foerster, J. Peters, J. Schmidhuber, in *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. *4668 LNCS*, Springer Verlag, Berlin **2007**, pp. 697–706.
[33] N. Heess, J. J. Hunt, T. P. Lillicrap, D. Silver, *Memory-based control with recurrent neural networks*, arXiv **2015**.
[34] H. Dang, P. K. Allen, *in Proc. – IEEE Int. Conf. on Robotics and Automation*, IEEE, Piscataway, NJ **2012**, pp. 2392–2397.
[35] J. Xu, Z. Hou, Z. Liu, H. Qiao, *Compare Contact Model-based Control and Contact Model-free Learning: A Survey of Robotic Peg-in-hole Assembly Strategies*, arXiv **2019**.
[36] T. Lefebvre, J. Xiao, H. Bruyninckx, G. De Gersem, *Adv. Robotics* **2005**, *19*, 479.
[37] C. Ott, *Cartesian Impedance Control of Redundant and Flexible-Joint Robots*, Springer, Berlin **2008**.
[38] *Libfranka*, https://github.com/frankaemika/libfranka/ (accessed: May 2020).
[39] J. Pajarinen, V. Kyrki, *Artif. Intell.* **2017**, *247*, 213.
[40] S. P. Singh, T. Jaakkola, M. I. Jordan, *In Machine Learning Proc. 1994*, Elsevier, Amsterdam **1994**, pp. 284–292.
[41] L. T. Dung, T. Komeda, M. Takagi, *Appl. Artif. Intell.* **2008**, *22*, 761.
[42] M. Hausknecht, P. Stone, *AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents (AAAI-SDMIA15)*, Arlington, VA, USA, November **2015**.
[43] *Franka Imeka Robot*, https://www.franka.de/ (accessed: May 2020).
[44] S. Guadarrama, A. Korattikara, O. Ramirez, P. Castro, E. Holly, S. Fishman, K. Wang, E. Gonina, N. Wu, E. Kokiopoulou, L. Sbaiz, J. Smith, G. Bartók, J. Berent, C. Harris, V. Vanhoucke, E. Brevdo, *TF-Agents: A library for Reinforcement Learning in TensorFlow*, **2018**, https://github.com/tensorflow/agents (accessed: April 2021).
[45] A. Varava, D. Kragic, F. T. Pokorny, *IEEE Trans. Robotics* **2016**, *32*, 1479.