Learning and Generalization of Dynamic Movement Primitives by Hierarchical Deep Reinforcement Learning from Demonstration

Wonchul Kim, Chungkeun Lee, and H. Jin Kim

Abstract— **This paper presents an approach to learn and generalize robotic skills from a demonstration based on deep reinforcement learning (Deep RL). Dynamic movement primitives (DMPs) formulate a nonlinear differential equation and produce the observed movement from demonstration. We build a network to represent this differential equation, and learn and generalize the movements by optimizing the shape of DMPs with respect to the rewards up to the end of each sequence of movement primitives. In order to do this, we consider an actor-critic algorithm for Deep RL and implement a hierarchical strategy. This drastically reduces the search space for a robot by decomposing the task, which allows to solve the sparse reward problem from complex tasks. In order to integrate DMPs with hierarchical deep RL, the differential equation is considered as temporal abstraction of option. The overall structure is mainly composed of two controllers: meta-controller and sub-controller. The meta-controller learns a policy over intrinsic goals, and a sub-controller learns a policy over actions to accomplish the given goals. We demonstrate our approach on a 6 degree-of-freedom (DOF) arm with a 1-DOF gripper and evaluate it in a pick-and-place task.**

I. INTRODUCTION

Deep reinforcement learning (Deep RL) has shown potential for robot control in the aspect of increasing autonomy and flexibility of robots [1]. However, in actual robotic experiments, even a simple robotic task such as an arm approaching a target is still challenging due to the high-dimensional exploration required. In order to reduce the search space and maintain a model-free approach, we consider learning dynamic movement primitives (DMPs), which can represent complex motion. Since DMPs are highly dependent on nonlinear differential equations for the shape of movement primitives, we specifically aim to learn those equations from a demonstration and improve them to acquire new motion skills for everyday tasks such as pick-and-place.

However, this is not simple for reinforcement learning (RL) because of the requirement to represent knowledge at multiple levels of temporal abstractions and to explore the environment efficiently. In particular, the primary difficulty arises due to the insufficient exploration that prevents from obtaining robust value functions and results in a sparse feedback problem. In order to tackle this problem, we apply hierarchical reinforcement learning (HRL) which treats a complex task as a sequence of multiple tasks, rather than a single task. For example, pick-and-place can be decomposed to approaching an object, grasping it, and transporting it to another location. Thus, it is necessary to define temporal abstraction which can be used to represent the hierarchy of each decomposed task as detailed in [2].

Within HRL, [3] proposes temporal abstraction of option to define a multi-step action policy and formalizes the learning problem as a discrete semi-Markov decision process. The key challenge is learning and operating policies over different levels of temporal abstractions in tasks. In this paper, we consider DMPs [4] as options and we propose a deep learning approach to learn the differential equations of DMPs as the output of the policy for a given task. Some desirable properties of this representation are described in [5].

In order to learn and generalize DMPs, we implement an actor-critic, model-free algorithm based on the deterministic policy gradient which can operate over continuous action spaces [6]. Then, two actor-critic networks are used for meta-controller and sub-controller, respectively. The overall structure is composed of two actor-critic networks, namely, a meta-controller and a sub-controller. The method for using two controllers is presented in [7] and we extend it to an actor-critic algorithm with continuous control for a manipulator. Specifically, the meta-controller produces goals over given states and the sub-controller generates actions over given states and goals to achieve the goal, by maximizing the cumulative reward function regarding a given task. Therefore, the robot can perform sequences of movement primitives by learning corresponding nonlinear differential functions of DMPs from a demonstration, which also does not require a model of the robot or the environment.

The main contributions of our work in applying HRL for learning the sequences of movement primitives are:

- Deep RL can generate DMPs for different trajectories or goals from demonstration. It does not require a pack of labeled DMP to learn various DMPs.
- RL improves the performance of DMPs. When the demonstration is not perfect or not from an expert, our method can produce a better trajectory.
- HRL enables a robot to learn the sequences of movement primitives at once, i.e. it does not require to learn, label each of DMPs, and subsequently execute them with some symbolic planners. Thus, the connection of each DMP will be more smoother.
- Considering options as DMPS, it generates task hierarchy comprising sequences of movement primitives and solves the sparse feedback problem.

The rest of this paper is constructed as follows. After discussing related works, we briefly describe DMPs and deep

*This work was not supported by any organization

Wonchul kim, Chungkeun Lee, and H. Jin Kim are with Faculty of Aerospace and Mechanical Engineering, Seoul National University, 08826, South Korea onedang2@snu.ac.kr, elkein@snu.ac.kr, Taewan Kim, and hjinkim@snu.ac.kr

RL algorithm with pre-training networks that form the basis of our work in section III. Based on this, HRL is presented with a pseudo-algorithm and some analysis in Section IV. Then, our algorithm in a pick-and-place task is demonstrated in Section V. Conclusion is made in Section VI.

II. RELATED WORKS

DMPs are useful in generating complex motions based on demonstration for control and planning of robots. When an RL algorithm is combined with DMPs, a robot can learn new motion skills, and DMPs can efficiently reduce the search space during learning. A huge motion library has been built in [8] to learn robotic motor skills from human demonstration and generalize a pick-and-place task to new situations. [9] uses a stochastic policy improvement to learn motor skills from demonstration on the humanoid robot. Similarly, the work in [10] modulates dynamical systems initialized from the demonstration via DMPs and implements an expectation-maximization-based RL algorithm to allow a robot to acquire new motor skills.

However, since those approaches learn each movement of a complex motion respectively and also need to execute each movement one by one, they are not practical and fully autonomous for performing tasks requiring several movements. In order to solve this, we consider hierarchical learning approach, which gives hierarchy in a sequence of tasks. The works [11] and [12] applied RL at the highest level of abstraction, while [13] learns the policy parameters in sequences of policies simultaneously at multiple levels of temporal abstraction. Formally, most of HRL approaches are described in the SMDP [3], where a collection of policies comprised of multi-step actions, sequentially call lower-level policies. At each step, the higher-level of the hierarchy chooses representation of a multi-step action and then, the lower-level makes decisions for a one-step action. Therefore, the agent is able to represent knowledge at multiple levels of temporal abstractions and exploiting temporal abstraction makes HRL much more efficient than normal RL, when the agent perform a certain task involving long-range planning.

In this paper, we apply HRL to learn and generalize DMPs for a pick-and-place task. The HRL approach proposed in [5] applies PI^2 algorithm [14] to DMPs and evaluates the approach in a pick-and-place task with an 11 degree-of-freedom (DOF) arm. They represent options as DMPs and describe some benefits of this representation in terms of scalability to higher dimensional tasks and applicability to continuous action and state spaces. Because they use HRL approach only in learning of goal parameter, not in the entire level of learning, they do not generate task hierarchy and they learn each of DMPs and execute each of them separately in an order. However, in this paper, we consider generating task hierarchies by building two controllers. The controllers optimize the task with respect to the cost of the entire sequence in the higher-level controller, whereas the lower-level controller is optimized with regrading to the cost for achieving its particular subgoals. This method is presented in [7] with hierarchical value functions operating at different temporal scales. We propose to extend this scheme to an actor-critic learning algorithm [6] to represent temporal abstraction that involves simultaneously learning options and a control policy to compose options with continuous control. Overall, we generate task hierarchy comprising sequences of movement primitives where the robot can learn the sequences of movement primitives at once, without a need to learn and perform each of DMPs one by one with some symbolic planners. Thus, the connection of each DMP will be smoother.

Although HRL can make better performance than normal RL due to efficient exploration, robotic tasks involving high-dimensional, time and efforts associated with continuous action spaces and high costs for exploration trials on physical systems still remain a challenge [2] and [15]. Here, we propose to use a deep learning approach to build HRL structures for learning DMPs and use a demonstration for the initialization of the DNNs.

III. MODEL-FREE REINFORCEMENT LEARNING WITH DYNAMIC MOVEMENT PRIMITIVES

In this section, we briefly describe dynamic movement primitives (DMPs) framework [4] and discuss how we integrate deep RL with DMPs.

A. Pre-training DMPs for networks of Deep RL

DMPs are presented as a formulation of movement primitives with nonlinear differential equations, which can create complex motions based on time evolution with discrete or rhythmic movements. Here, we focus on discrete movements. A one-dimensional movement is generated by integrating a linear spring system under an external forcing term, and it is referred to as a transformation system such as:

$$\tau \dot{v} = K(g-x) - Dv + (g-x_0)f \tag{1}$$

$$\tau \dot{x} = v, \tag{2}$$

where x and v are position and velocity of the system; x_0 and g are the start and goal positions; τ and $(g-x_0)$ are temporal and spatial scaling terms; D determines how the system is critically damped and K is a spring constant; f is a nonlinear differential function allowing the generation of the complex movements learned from the demonstration. It does not directly depend on time. Instead, it depends on a phase variable (s) that monotonically changes during movements, which is obtained by the equation

$$\tau \dot{s} = -\alpha s \tag{3}$$

where α is a pre-defined constant.

In this paper, we propose a deep learning approach to represent the nonlinear function generated from demonstration by using deep neural networks (DNNs). According to the equation (1), $f_{target}(s)$ is formulated as

$$f_{target}(s) = \frac{-K(g-x) + Dv + \tau \dot{v}}{g - x_0} \tag{4}$$

where x_0 and g are set to $x(0)$ and $x(T)$, respectively. In order to obtain f_{target}, a set of movements $x(t)$ is recored

from a demonstration and the derivatives $v(t)$ and $\dot{v}(t)$ are computed for each time step $t = 0, ..., T$. Then, the canonical system is exploited with an appropriately adjusted temporal scaling τ.

As the function f_{target} is dependent on a phase variable, the input of the DNNs is aslo a phase variable and the output is a forcing factor. For a linear regression, we minimize the loss criterion $L = \sum_s (f_{target}(s) - f(s))^2$ using the Adam [16] optimizer with the following parameters: learning rate (0.0001), stability constant (0.001), and Tensorflows default exponential decay rates β. The network architecture consists of three hidden layers, which is same as the actor network structure [17]. We will cover the details of the network in Section IV.

B. Reinforcement Learning to learn DMPs

As mentioned earlier, DMPs can represent various motion primitives due to the inclusion of the nonlinear differential function f to the discrete dynamic equations and f determines the shape of motion primitives toward a goal position. Thus, for the purpose of this paper, we implement the RL algorithm to learn and improve function f.

Using the pre-trained models, RL algorithm can imitate the observed trajectory from a demonstration. However, replicating the imitated trajectory is not the purpose, but rather it serves as an initialization for further improvement through learning. In this paper, the policy is improved to maximize the following cumulative reward by optimizing the weights and parameters of DNNs based on the initialization.

$$R = -r_{t_N} - \int_{t_0}^{t_N} r_t + \frac{1}{2} \dot{\mathbf{x}}_t R \dot{\mathbf{x}}_t^T \, dt, \tag{5}$$

where r_{t_N} and r_t the reward at terminal time and the immediate reward, and $\frac{1}{2}\dot{x}_t R \dot{x}_t^T$ is an immediate control reward. The reward function depends on a given task. Then, the policy is improved through an iterative process of exploration and parameter update, as presented in Fig. 1.

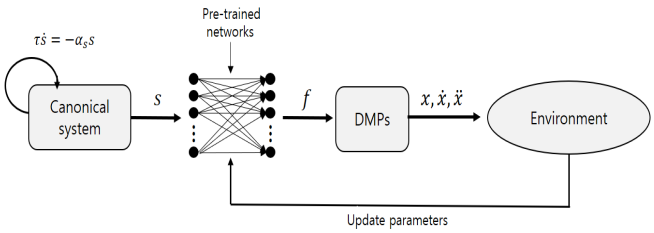

Fig. 1. Policy improvement loop

According to the initialization of the pre-trained network, it initially produces the observed movements and then, through several iterations with parameter update, it is expected to generate movements that maximizes the reward function value in the future. Then, the process continues with the new parameters as the basis for the exploration. The exploration is performed by the results of DMPs formulation given in equation (1).

IV. Hierarchical Deep Reinforcement Learning

Hierarchical reinforcement learning (HRL) defines temporal abstractions as a Semi-Markov Decision Process (SMDP). In this paper, we implement it to solve the sparse feedback problem for control and planning through task decomposition. The detailed description is presented in [3]. Thus, we apply hierarchical approach [7] to the actor-critic algorithm based on the deterministic policy gradient [6] for a manipulation task.

In order to use a framework of two controllers, we build two actor-critic networks and assign each of them as meta-controller and sub-controller as Fig. 2 shows. The meta-controller receives state s_t and chooses a goal g_t and then, the sub-controller selects an action a_t over s_t and g_t. The objective for the sub-controller is to maximize cumulative intrinsic reward, r^{int}. Similarly, the objective of the meta-controller is to optimize the cumulative extrinsic reward, r^{ext}.

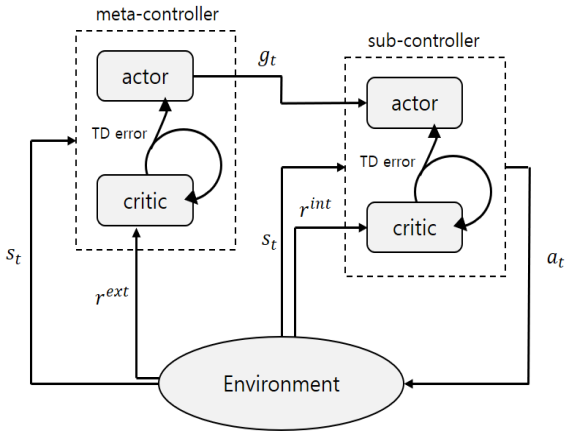

Fig. 2. Overview of hierarchical actor-critic learning structure

For each of controllers, the actor networks are made up of three fully connected layers of 32 units for hidden layers and the critic networks are composed of the same number of hidden layers of 128 units. Each of layers exploits a rectified linear unit, except for the output layer. At each learning process, the highest predicted Q-value is chosen from the actor network and we can obtain its corresponding action for the given state. Since the actor networks determine the policy, pre-training is also conducted only for each actor network of two controllers, as described in Section III-B. Then, using this pre-trained model, we initialize the models and follow the algorithm illustrated in Algorithm 1.

Within Algorithm 1, before executing the loops of algorithm, actor networks of meta-controller and sub-controller are initialized with the pre-trained models. An episode represents sequences of DMPs to complete the task and it is made up of steps for executing atomic actions. Considering the purpose of this paper, which is to learn DMPs, the step is dependent on same time step as the system of DMPs. Each of goals categorizes DMPs involving actions to achieve one sequence of DMPs. When the agent accomplishes the goal, a new goal is given from the meta-actor network.

Algorithm 1 Hierarchical actor-critic algorithm

Initialize meta-actor and sub-actor, and their target networks with pre-trained models
Initialize meta-critic, sub-critic, and their target networks
Initialize the experience replay buffer
for Repeat (for each episode): **do**
  Initialize DMPs and get initial state (s_0)
  $g \leftarrow$ actor(s_0)
  $R^{ext} \leftarrow 0$
  $sub-terminal, terminal \leftarrow$ False
  for Repeat (for each step of episode): **do**
    Get action(a) from the sub-actor(s, g)
    Take action a, observe r^{int}, r^{ext}, s', $sub-terminal$, $terminal$
    Store experience $(s, g, a, r^{int}, s', g)$ for Sub. Contr.
    UPDATE PARAMETERS for Sub. Contr.
    UPDATE PARAMETERS for Contr.
    $R^{ext} \leftarrow R^{ext} + r^{ext}$
    $s \leftarrow s'$
    if $sub-terminal$ is True **then**
      $g \leftarrow$ actor(s)
      $sub-terminal \leftarrow$ False
      Store experience (s_0, g, R^{ext}, s') for Contr.
      $s_0 \leftarrow s$
      $R^{ext} \leftarrow 0$
    end if
  end for
end for

$sub-terminal$ represents 'True' when the agent finish the goal and 'False' when the agent does not. $terminal$ means if the agent successfully accomplishes the whole task (all sequence of DMPs) or not.

Via-point task. We now introduce a via-point task to illustrate the better performance of HRL than normal RL. In this task, each of algorithm is pre-trained with a black line shown in Fig. 3 and they both are required to pass through two via-points before arriving at the destination. The reward function is expressed as

$$R(t) = \int_{t_0}^{t_N} 10(C_1(t) + C_2(t)) + 10^{-2}\dot{\mathbf{x}}\dot{\mathbf{x}}^T dt \qquad (6)$$

$$C_1(t) = \delta(t-20)|\mathbf{x} - [-0.075, -0.1, 0.1]^T|,$$

$$C_2(t) = \delta(t-55)|\mathbf{x} - [-0.175, 0, 0.05]^T|,$$

where x is a desired position, $C_1(t)$ and $C_2(t)$ are the distance between the current position and the desired location (i.e. point 1 and point 2, respectively) at a given time, and $\dot{\mathbf{x}}\dot{\mathbf{x}}^T$ is to avoid high speed.

Within this task, meta-actor network chooses the goals over the state. The goals are made up of via-points and the last position, and the state is the output of canonical system. Similarly, sub-actor networks provide the forcing term, f, according to the state which consists of the output of meta-actor network, goals, and the output of the canonical system.

As the Fig. 3 describes, HRL successfully generates the trajectory to pass through two via-points while normal RL fails and passes only the via-point 1.

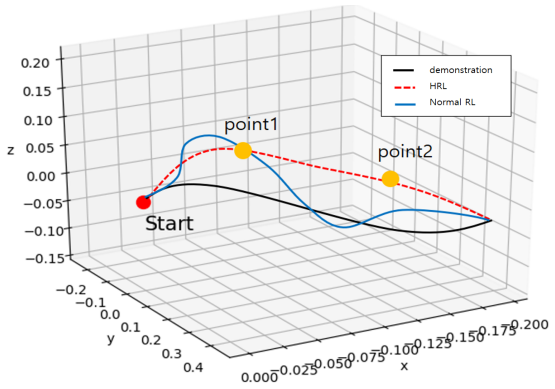

Fig. 3. Comparison between normal reinforcement learning and HRL

V. EXPERIMENTS

The following section describes how we applied HRL to the framework of DMPs on the UR3 arm to perform a pick-and-place task. The arm platform we used is a 6-DOF robot arm with 1-DOF gripper as in Fig. 4. and the controller available from Universal Robots A/S was used for low-level control of the arm.

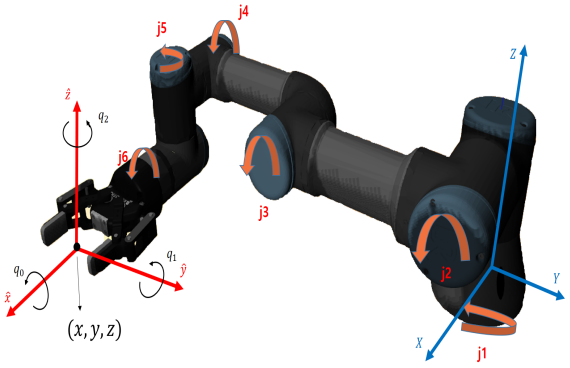

Fig. 4. UR3 arm [mm]

For experiments, the demonstration includes seven variables, representing the end-effector's position (x, y, z) in the Cartesian space, the end-effector's orientation (q_0, q_1, q_2) in the quaternion space, and the gripper's position (e), which represents 'open' when $e = 0$ and 'close' when $e = 1$. Then, we calculate the forcing term through the transformation system in equation (1). We assume that the position of the target is known since perceptual component such as visual processing is currently not the focus of our research. All the information is recorded at 125Hz. The corresponding velocities and accelerations for position and attitude were computed numerically by differentiating the position.

A. Learning and improvement

In order to validate that RL can learn and improve DMPs from a demonstration, we use a poor demonstration on purpose. The demonstrated trajectory which detours from the start to the end position. Given this poor trajectory, we pre-train the DNNs and then, carry out Deep RL algorithm with

executing DMPs at each time step. Fig. 5 describes how RL can improve the given demonstration.

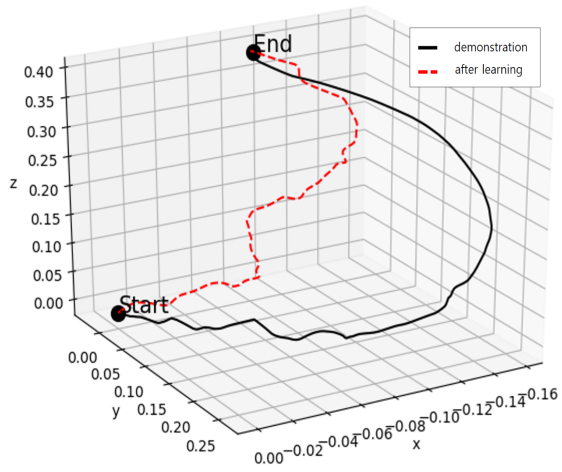

Fig. 5. Learning and improvement of DMPs from a demonstration [m]

Although RL improves the trajectory by making shorter to an end position from a start position, the generated trajectory is not straight unlike common expectation. This could be a local optima, because RL optimizes values of the nonlinear differential equations which generate movement primitives based on the demonstration.

B. learning and Generalization

DMPs alone cannot reproduce a trajectory for a new goal which is not included in the demonstration. The goal parameter in DMPs formulation should be changed and the new trajectory should be learned from the demonstration. In this section, we describe how RL can be used for generalization of DMPs based on a demonstration. We first pre-train the DNNs with an initial trajectory and then, implement the RL algorithm with the newly assigned goals around the initial goal position.

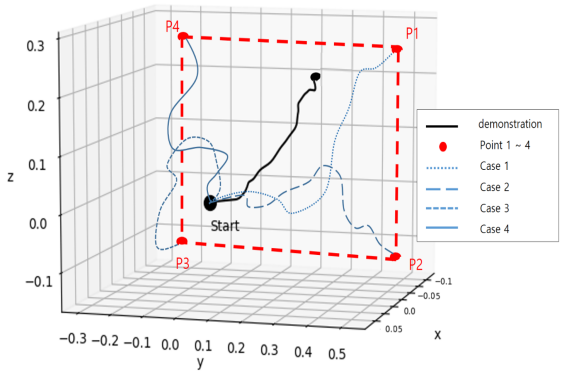

Fig. 6. Learning and generalization of DMPs from a demonstration [m]

As Fig. 6 presents, we consider four different target points (P1 $\sim$ P4) around the initial target, which are not specified by the demonstration. First, we train the arm to learn a demonstration and then, based on the demonstration, we use RL to train the arm to approach the four new targets. For all the cases (P1 $\sim$ P4), the arm reaches the goals after the about 80 $\sim$ 100 iterations. This confirms that RL can generate new trajectories according to the corresponding goals, which are not involved in the demonstration.

C. Pick and Place Task

In this section, we apply hierarchical Deep RL to a pick-and-place task, where the robot reaches for grasping an object on the ground, and place it on a shelf as shown in Fig. 7. The shelf is in the front left of the robot and the object is placed on the ground in the front right of the robot. Grasp planning for complex objects is not the focus of this paper and a cylindrical object is chosen for simplicity.

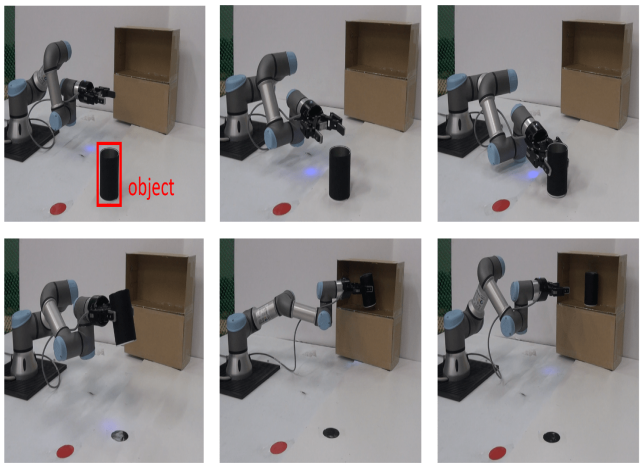

Fig. 7. Pick-and-place task setting

For a demonstration, we only provide horizontal and vertical movements, which are not as efficient for reaching a target as diagonal movements. The black solid line in Fig. 8 presents the demonstrated trajectory. Based on this, after pre-training procedure described in II-A, the meta-controller produces the goal positions, which are positions for grasping, releasing, and the destination, from the output of the canonical system at each time step. Similarly, the sub-controller provides the forcing term for DMPs over the same state to the meta-controller and the goals.

The reward function for this task is

$$R(t) = 10(C_1(t) + C_2(t)) + \int_{t_0}^{t_N} D(t) + 50G(t) + 10^{-2}\dot{\mathbf{x}}\dot{\mathbf{x}}^T dt, \tag{7}$$

where the immediate cost consists of three components. The term $G(t)$ is 0 if the object is in the gripper during the transport, and 1 otherwise. Because this particular arrangement of the object and shelf requires the arm to turn as well as move up and down, the cost for the end-effector orientation is specified in $D(t)$. It penalizes if the orientation is different from the demonstration. Also, to avoid hitting the top or bottom of the shelf, we want the end-effector to approach the shelf's opening horizontally in order to release the object on the shelf. So the term $D(t)$ specifies that it is desirable for the

end-effector to get to the same height as the releasing point before arriving. The terminal cost terms $C_1(t)$ and $C_2(t)$ are used to reduce the distance between the current end-effector position and the desired location for grasping and releasing, respectively.

We consider two experiments (Case 1 and Case 2). For case 1, we demonstrate inefficient movements to perform the task. For case 2, we change the object position for grasping. Therefore, we can confirm that our approach can improve pre-trained movements more efficiently to complete the task, and given a changed object position for grasping, it is also able to generate new motions to achieve the task. The results of this task are presented in Figs. 8 - 11.

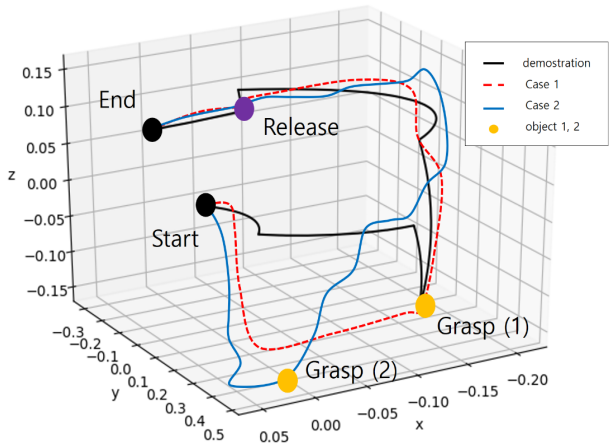

Fig. 8. Trajectory of the end-effector [m]

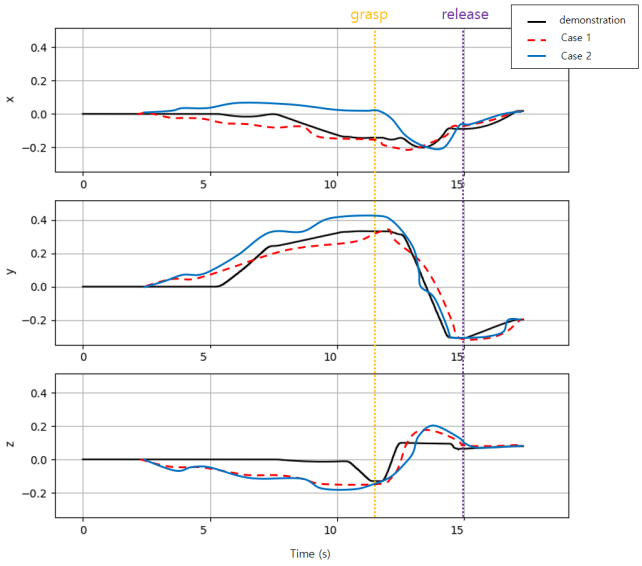

Fig. 9. Position of the end-effector [m]

As Figs. 8 - 11 present, the end-effector successfully performs a pick-and-place task from a demonstration and improves the performance by generating diagonal movements with changing orientations. Additionally, for case 2, it

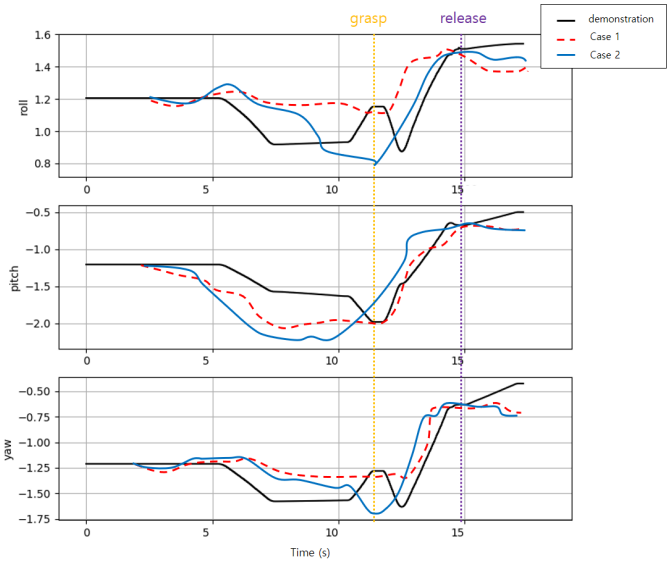

Fig. 10. Orientation of the end-effector [m]

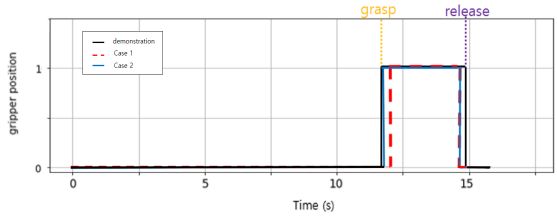

Fig. 11. Position of the gripper

can complete the task given the changed object position for grasping.

VI. CONCLUSIONS

This paper addresses a deep learning approach to learn DMPs (dynamic movement primitives) and implementation with hierarchical Deep RL for learning the sequences of movement primitives from a demonstration. First, we pre-train the DNNs (deep neural networks) for Deep RL (reinforcement learning) from a demonstration using DMPs. Then, Deep RL algorithm improves the performance of DMPs. For example, when the demonstration is not perfect or not from an expert, our method can produce a better trajectory. Additionally, Deep RL can generalize DMPs for different trajectories or goals. Even though some targets are not specified by the demonstration, out approach can generate new trajectories to those targets based on the initial demonstration. Thus, we do not require a pack of labeled DMPs to learn various DMPs. Finally, we apply the hierarchical strategy considering DMPs as options for temporal abstractions. This allows a robot to build hierarchy with sequences of movement primitives so that the robot learns the sequences of movement primitives at once, not each of movement one by one. We evaluate the proposed approach for a pick-and-place task on a 6-DOF robot arm with a 1-DOF gripper.

REFERENCES

[1] 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.

[2] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, no. 4, pp. 341–379, 2003.

[3] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[4] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert, "Learning movement primitives," in *Robotics research. the eleventh international symposium*. Springer, 2005, pp. 561–572.

[5] F. Stulp and S. Schaal, "Hierarchical reinforcement learning with movement primitives," in *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*. IEEE, 2011, pp. 231–238.

[6] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[7] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Advances in neural information processing systems*, 2016, pp. 3675–3683.

[8] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, "Learning and generalization of motor skills by learning from demonstration," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 763–768.

[9] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert, "Control, planning, learning, and imitation with dynamic movement primitives," in *Workshop on bilateral paradigms on humans and humanoids, IEEE International Conference on Intelligent Robots and Systems*, 2003, pp. 1–21.

[10] P. Kormushev, S. Calinon, and D. G. Caldwell, "Robot motor skill coordination with em-based reinforcement learning," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, 2010, pp. 3232–3237.

[11] F. Stulp and M. Beetz, "Refining the execution of abstract actions with learned action models," *Journal of Artificial Intelligence Research*, vol. 32, pp. 487–523, 2008.

[12] B. Nemec, M. Tamošiūnaitė, F. Wörgötter, and A. Ude, "Task adaptation through exploration and action sequencing," in *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*. IEEE, 2009, pp. 610–616.

[13] S. Hart and R. Grupen, "Learning generalizable control programs," *IEEE Transactions on Autonomous Mental Development*, vol. 3, no. 3, pp. 216–231, 2011.

[14] E. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *Journal of Machine Learning Research*, vol. 11, no. Nov, pp. 3137–3181, 2010.

[15] J. Morimoto and K. Doya, "Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning," *Robotics and Autonomous Systems*, vol. 36, no. 1, pp. 37–51, 2001.

[16] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[17] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.