# CS885 Project Report
# Reinforcement Learning-based Control for Cyclic Motions

**Sahand Ajami** SAHAND.AJAMI@UWATERLOO.CA
*Department of Mechanical and Mechatronics Engineering*
*Faculty of Engineering, University of Waterloo*

## Abstract

Many of the robotics interfaces perform cyclic tasks and there exists a number of approaches to generate cyclic motion. The first approach is to use time-independent motion generator such as a Partial Derivative (PD) controller. However, those time-based trajectory controllers are not robust to external disturbances. The other approach is to use time-independent ones such as Virtual Energy Regulator (VER). According to VER's time-independent nature, it is able to optimally coordinate with an external disturbance to utilize its benefits or minimize its impacts. However, designing VER requires deep knowledge about the dynamics of the system as it is sensitive to internal factors such as friction. This project aims to design Reinforcement Learning (RL) based controllers for cyclic behaviour and compare it with time-independent and time-dependent motion generators. The actor critic method is used in designing the controller and Deep Deterministic Policy Gradient (DDPG) and Twin Delayed DDPG (TD3) algorithms were used to update the agent. The performance of the RL-based controllers were compared with PD and VER on Franka-Emika robot and RL-based controller did not exhibit satisfactory performance.

## 1. Introduction

Many robotic systems, such as rehabilitation robots (Shi et al., 2019), industrial manipulators (Carabin et al., 2017), and mobile robots (Silva and Machado, 2012) perform cyclic tasks. Accordingly, having a cyclic pattern generator is the first requirement of such robotic systems.

It has been more than a decade since robotic researchers are designing different toolboxes to generate cyclic motions. Adaptive Frequency Oscillator (AFO) (Righetti et al., 2006), Central PatternGenerator (CPG) (Ijspeert, 2008), Dynamic Movement Primitive (DMP) (Ijspeert et al., 2013), and Adaptive Natural Oscillator (ANO) (Khoramshahi et al., 2017) are some of the toolboxes developed for cyclic tasks. AFO is a classic version of adaptive oscillators, which adapts the frequency and provides a phase lock to a reference trajectory. CPG is a general-purpose and popular toolbox that generates cyclic trajectories using a set of dynamical equations. DMP uses a primitive trajectory and improves it by minimizing the desired cost function. ANO is another types of adaptive oscillators that optimize the frequency and shape of the reference trajectory to minimize energy consumption.

Regardless of the pattern generator type, in the current control scheme of robotic systems, the pattern generator provides the reference trajectory for a tracking controller (e.g., a PD controller). In this scheme, most of the pattern generators are blind to the robot states and the controller enforces the reference trajectory. Such systems are not successful in applications where the robot should effectively collaborate with external factors; such as a human in human robot interaction (HRI) (Goodrich et al., 2008), another robot in robot-

robot interaction (R2I) (Ahmadabadi and Nakano, 2001), external disturbances in manipulators (Chiaverini et al., 1999), and unknown environments in legged robots (Bermudez et al., 2012). This defect in different conditions leads to miss-coordination, performance reduction, and even instability.

A solution for the miss-coordination problem is cyclic motion generation using the natural dynamics concept. There are two main approaches to attain this goal: (1) Natural Dynamics Exploitation (NDE) and (2) Natural Dynamics Modification (NDM). In NDE, the nonlinear modes of the robot are excited and it performs the cyclic patterns (Della Santina and Albu-Schaeffer, 2020). The exploited cyclic patterns are natural behaviour of the robotic system, hence, they are energy efficient. But, the possible nonlinear modes are highly limited and mostly not the preferred ones. In NDM, the dynamics of the robot are modified to perform a stable limit cycle. To do so, pure nonlinear static feedback (robot positions and velocities) replaces with controller and pattern generator dynamical equations; no additional dynamics are added.

The proposed static feedback modifies the natural dynamics to a nonlinear limit cycle performing the desired task. Since, the control objective is defined in the state space domain (and it is a norm of position and velocity), instead of the time domain which is widely used for tracking controllers, the controller is called Virtual Energy Regulator (VER). This is initially introduced in (Nasiri et al., 2021) and utilized in (Nasiri et al., 2022) to resolve the human-exoskeleton coordination problem as an HRI task.

The main advantages of VER are (1) combining the controller and pattern generator blocks, (2) removing systematic delay between the generated limit cycle and the robot's state, (3) full-time independence, (4) full awareness of the states, and (5) high robustness against external factors. Due to limitations imposed by solely using static feedback, this method has two main disadvantages. (1) Since VER works based on natural dynamics modification, it is sensitive to the robot dynamics cancellation performance (especially friction). (2) Due to its reliance on pure static feedback, it can only generate a certain class of limit cycles that we call feasible limit cycles.

This project aims to benefit from reinforcement learning techniques to design controllers to address the weakness of VER. For this purpose, the actor-critic method was used and Deep Deterministic Policy Gradient (DDPG) and Twin Delayed DDPG (TD3) algorithms were used to train the agent. The training results of the agents are then compared with each other and their control outcome is compared with VER.

## 2. Mathematics

### 2.1 Virtual Energy Regulator

Consider the dynamics of an n-DOF manipulator as

$$M(\ddot{\vec{x}})\ddot{\vec{x}} + H(\vec{x}, \dot{\vec{x}}) = \vec{\tau} \tag{1}$$

where $\vec{x}$, $\dot{\vec{x}}$, $\ddot{\vec{x}} \in \mathbb{R}^n$ are the vectors of joint positions, velocities and accelerations, $M \in \mathbb{R}^{n\times n}$ is mass matrix, $\vec{\tau} in \mathbb{R}^n$ is the vector of torques applied by motors, and $H \in \mathbb{R}^n$ contains all additional dynamical terms such as Coriolis, Centrifugal, motor dynamics, and gravity forces. Having a dynamic compensation as $\vec{\tau} = \tilde{M}(\ddot{\vec{x}})\ddot{\vec{v}} + \tilde{H}(\vec{x})$, the robot dynamics can be rewritten as

$$\ddot{\vec{x}} = \delta(\vec{x}, \dot{\vec{x}}) + M^{-1}\tilde{M}\vec{v}(\vec{x}, \dot{\vec{x}}), \delta(\vec{x}, \dot{\vec{x}}) = M^{-1}(\tilde{H} - H) \tag{2}$$

where $\tilde{M}$, $\tilde{H}$ are estimations of $M$ and $H$, respectively. Assuming a proper dynamic compensation ($\delta \approx 0$, $\tilde{M} \approx M$, and $\ddot{\vec{x}} = \vec{v}(\vec{x}, \dot{\vec{x}})$), without loss of generality, for an arbitrary joint we have $\ddot{x} = v(x, \dot{x})$. The goal is to design the static feedback $v(x, \dot{x})$ such that $\langle x(t), \dot{x}(t)\rangle$ (position and velocity of the joint) performs a feasible desired limit cycle $\langle x_d, \dot{x}_d\rangle$ (desired position and velocity of the joint over the desired limit cycle).

### 2.1.1 Definitions

1. *Centre:* The desired limit cycle centre is defined as $c = \langle x_c, v_c\rangle$ where $x_c = 1/T \int_T x_d(t)dt$ and $v_c = 1/T \int_T \dot{x}_d(t)dt = x(t+T) - x(t)$; since the limit cycle is periodic, $x(t) = x(t \pm T), v_c = 0$ therefore $c = \langle x_c, 0\rangle$.

2. *Radius* and *phase:* In this formulation, any point in the phase plane ($\langle x, \dot{x}\rangle$) is transformed to $\langle r, \theta\rangle$ where $r = \sqrt{(x - x_c)^2 + \dot{x}^2}$ and $\theta = atan2(\dot{x}, x - x_c)$ are radius and phase defined about the desired limit cycle centre.

3. *Feasible Limit Cycle:* A feasible limit cycle for $\ddot{x} = v(x, \dot{x})$ is a clock-wise, sufficiently smooth, and not self-crossing closed curve in the phase plane. Thus: **(1)** its radius is a function of phase, **(2)** its position contains one minimum and one maximum in each period, **(3)** if the velocity ($\dot{x}_d$) is positive(negative), the position ($x_d$) is monotonically increasing(decreasing), and **(4)** its acceleration ($\ddot{x}_d$) is a function of position and velocity.

4. *Desired radius:* The limit cycle radius defined about the limit cycle centre is the desired radius ($r_d$) which is by definition a function of phase; i.e. $r_d = \sqrt{(x_d - x_c)^2 + \dot{x}_d^2} = f(\theta), \theta \in [-\pi, \pi]$. Hence, the desired position and velocity are also function of the phase as $x_d = x_c + r_d(\theta)\cos(\theta)$ and $\dot{x}_d = r_d(\theta)\sin(\theta)$.

### 2.1.2 Control Law

The VER control law to generate desired synchronous limit cycles is defined as $v(x, \dot{x}) = \alpha(x, \dot{x}) + \beta(x, \dot{x} + \gamma(x, \dot{x}))$ where $\alpha$ is *attractor*, $\beta$ is *accelerator*, and $\gamma$ is *synchronizer*. Attractor is responsible to generate a convergent dynamics towards the desired limit cycle and satisfy $r = r_d$. Accelerator generates the desired acceleration to move over the desired limit cycle. Synchronizer provides a dynamical coupling between independent limit cycles at different joints and synchronizes them.

1. *Attractor:* To generate a convergent dynamics towards the desired limit cycle $r = r_d$ should be satisfied. To have a well-posed mathematics, the control objective is redefined as $r^2 = r_d^2$ as where $V(x, \dot{x}) = r^2 = (x - x_c)^2 + \dot{x}^2$ and $V_d(\theta) = r_d^2 = (x_d - x_c)^2 + \dot{x}_d^2$ are the virtual energy and the desired energy of the joint, respectively. Accordingly, the attractor term to generate a convergent dynamics towards the desired limit cycle is $\alpha = PS(\dot{x})(V_d(\theta) - V(x, \dot{x}))$ where $P > 0$ is attractor gain and $S$ is a function of velocity such that $S(\dot{x})\dot{x} > 0$.

2. *Accelerator:* Although attractor term grants convergent dynamics towards the equilibrium points on the desired limit cycle, it is not sufficient to generate a stable limit cycle. Hence, the accelerator term is added to make the stable equilibrium points locally unstable and govern the desired limit cycle. Based on **Definition. 1**, the desired acceleration is a function of position and velocity and therefore a function of phase. Hence, by having a desired position trajectory and its second derivative ($\ddot{x}_d$), the accelerator term is $\beta = \ddot{x}_d(\theta)$.

3. *Synchronizer:* The stable limit cycles can be generated by accelerator and attractor terms, but the generated limit cycles in different joints are blind to each other; they are independend. Hence, synchronizer term is needed to enforce them for synchronization. To synchronize the limit cycles, we define the limit cycle pitch ($\rho$) which is the first harmonic phase of the limit cycle. To synchronize the limit cycles at $i$th and $j$th joints with $\phi = \rho_i(\theta_i) - \rho_j(\theta_j)$ phase difference, we have $\gamma_i = K(\sin(\rho_j - \phi) - \sin(\rho_i))$, respectively.

## 2.2 Reinforcement Learning Controller

For this control problem, actor-critic method was used. The block diagram of the controller is represented in Fig. 1. According to the figure, actor receives the observations from the environment and outputs the actions. Critic observes both the states and the actions and produces a Q-value. Afterwards, the error is calculated by comparing the Q-value (from critic) and the reward value (from environment). The error is then used to update both actor and critic. Since the environment is time-continuous the actor is deterministic and the critic produces a Q-value, Deep Deterministic Policy Gradient Descent (DDPG) and Twin Delayed DDPG (TD3) were chosen for the agent. These algorithms are shown in Algorithm 1 and Algorithm 2, respectively (Lillicrap et al., 2015) (Fujimoto et al., 2018).

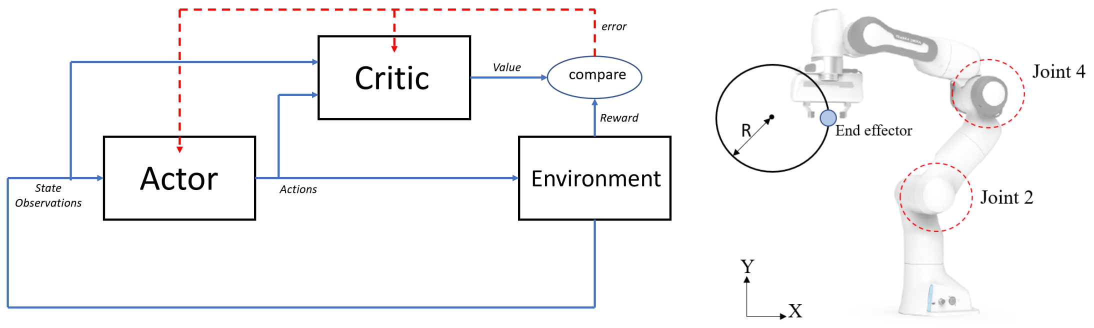

Figure 1: The block diagram of the system (left) and the environment (right). The experimental setup is Franka Emika robot (Panda arm) with 7-DOF where only joint 2 (first joint) and joint 4 (second joint) are used for the experiment and the rest are controlled to hold a fixed position. The goal is to control the end effector (which is shown by a blue point) over the desired circle in the sagittal plane with $R = 10cm$.

### 2.2.1 Definitions

The objective of the controller is to guide the end-effector of the robot (see Fig. **??**) to follow the desired trajectory. The desired trajectory was predefined in the sagittal plane of the robot and joints 2 and 4 were used for this purpose. Therefore, position and velocity of those joints ($x_i, \dot{x}_i, i \in \{2, 4\}$) with their difference from the desired values ($e_i = x_{d,i} - x_i, i \in \{2, 4\}$) were defined as states, and their torques ($\tau_i, i \in \{2, 4\}$) were defined as actions.

The reward function was defined as:

$$R = +10(|e| \leq 0.1) - 1(|e| > 0.1) - 100F \tag{3}$$

where $e$ is error (the difference between the desired and the actual position of the end effector) it determines that if the end effector of the robot follows the desired trajectory with the accuracy of 0.1, the agent receives +10 reward. Otherwise, it receives -1. $F$ is a boolean denoting to the boundaries of the workspace. For safety reasons, if the end effector crosses the predefined boundaries, the agent receives -100, and the episode restarts.

The networks of the actor and critic are shown in Fig. 2. The actor consists of one fully-connected layer with `Tanh` as its activation function. The critic contains two paths: one from the state observations and the other from the actions. The state observation path consists of a fully-connected layer with 120 neurons, `ReLU` activation and another fully-connected layer with 80 neurons. Its action path contains a fully-connected layer with 80 neurons. Both paths are then added together and activated by `ReLU` function.

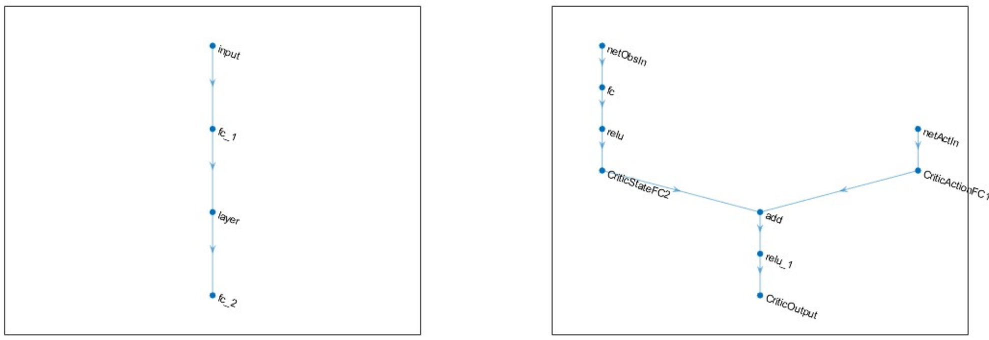

Figure 2: The architecture of the neural networks for actor (left) and critic (right).

## 3. Experimental Results and Discussion

The setup is shown in Fig. 1 and the task was to move the end effector of the robot in a circular trajectory by using joints 2 and 4. For this purpose, the desired joint trajectories were obtained by the inverse kinematics and the control law is to produce torques for joints 2 and 4 which correspond to their desired trajectory. For training the RL agent, to reduce the stochasticity, the training was performed with 5 seeds and to minimize the risk of memorization, different circles with different radii ($8cm < R < 13cm$) and different periods of motions ($1sec < T < 2sec$) were fed to the agent in each episode. The training curves for the agent are shown in Fig. 3. In can be observed that TD3 agent performed better than DDPG, in both episode $Q_0$ and the learning curve. This can be because TD3 solves the overestimation bias of DDPG. TD3 is based on DDPG with three major improvements: (1) additive clipped noise on actions, (2) double critics and actors, (3) delayed actors update, which can address variance and the quality of the value function estimation.

Since TD3 performed better than DDPG in training, it was used for evaluation. Fig. 4 shows the position of the end effector in moving over a circular trajectory. It can be seen from the figure that both VER and PD performed very accurately. TD3, however, was not able to follow the desired trajectory and the robot was stopped since it passed its safety boundaries.

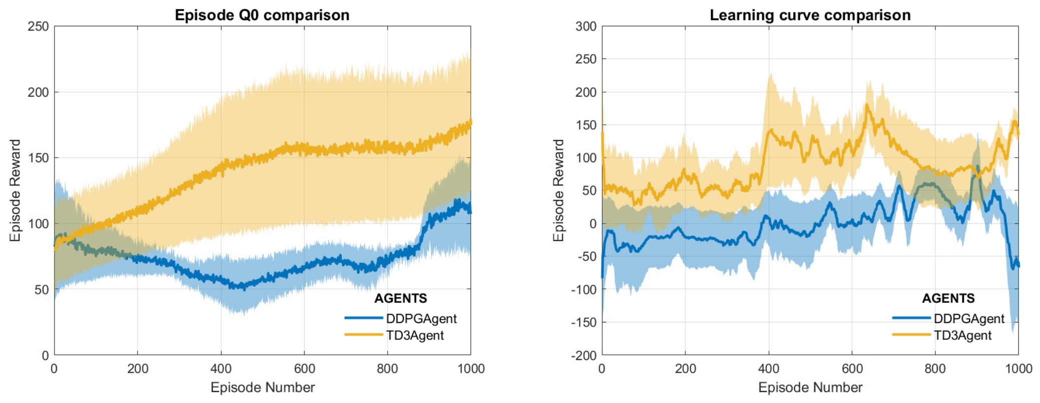

Figure 3: Comparison between the training with TD3 and DDPG: The left plot shows the episode $Q_0$ comparison and the right plot represents the learning curves for both algorithms.

## 4. Conclusion and Future works

The objective of this project was to design an RL-based controller and compare its performance with existing two types of cyclic motion generators (time-dependent and time independent). To the best of the author's knowledge, this is the first project to compare the results with both types of motion generators. It was hypothesized that the RL-based controller can address problems of both types i.e. being robust to external disturbances and less sensitive to the internal dynamics. However, it failed to follow the desired trajectory. Future works aim to improve the design of the controller. The potential improvements are described as follows:

- *Reward function:* Even though the reward function seems reasonable and exhibited promising results in other problems [1], it did not help the agent learn this control law. To make the reward function less sparse, $e^2$ was used instead of $|e|$. However, it did

1. I tested this reward function for a servo control (goal-conditioned) problem. 1 DOF mass was simulated and the objective was to follow a step function i.e. reach a stationary target.

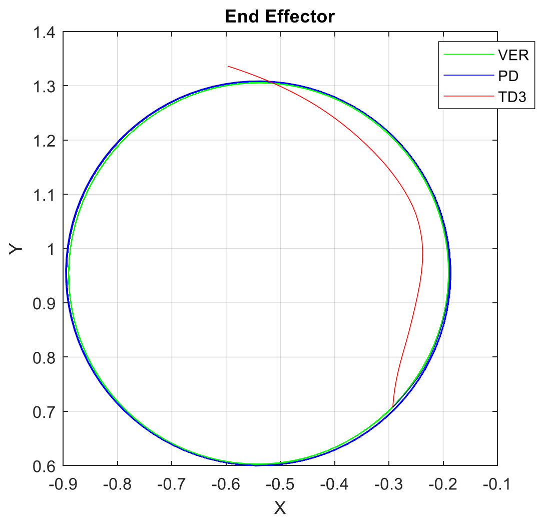

Figure 4: Comparison between VER, PD and TD3 in evaluation. The control objective was that the end effector of the robot should perform a circular motion.

not show any improvements either. Future work should aim to change the reward function in a way that the agent can benefit the most.

- *Actor-critic structure:* The architecture of the networks used for the both actor and critic were discussed previously. Different number of layers with different number of neurons were tested to reach an optimum level of learning. However, they did not help agent fully learn the control law. Future work can aim in fine-tuning the architecture with the hyperparameters to reach the most optimum learning.

- *Different RL techniques:* In this project, actor critic method was used which is common in control problem. However, this technique was not followed by an acceptable performance. Future work aims to evaluate the other methods such as multi-agent RL, inverse RL and imitation learning. In imitation learning, for instance, the agent will be designed to imitate some of the features from time-dependent some from time-independent motion generator.

## References

Majid Nili Ahmadabadi and Eiji Nakano. A" constrain and move" approach to distributed object manipulation. *IEEE Transactions on Robotics and Automation*, 17(2):157–172, 2001.

Fernando L Garcia Bermudez, Ryan C Julian, Duncan W Haldane, Pieter Abbeel, and Ronald S Fearing. Performance analysis and terrain classification for a legged robot over rough terrain. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 513–519. IEEE, 2012.

Giovanni Carabin, Erich Wehrle, and Renato Vidoni. A review on energy-saving optimization methods for robotic and automatic systems. *Robotics*, 6(4):39, 2017.

Stefano Chiaverini, Bruno Siciliano, and Luigi Villani. A survey of robot interaction control schemes with experimental comparison. *IEEE/ASME Transactions on mechatronics*, 4 (3):273–285, 1999.

Cosimo Della Santina and Alin Albu-Schaeffer. Exciting efficient oscillations in nonlinear mechanical systems through eigenmanifold stabilization. *IEEE Control Systems Letters*, 5(6):1916–1921, 2020.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

Michael A Goodrich, Alan C Schultz, et al. Human–robot interaction: a survey. *Foundations and Trends® in Human–Computer Interaction*, 1(3):203–275, 2008.

Auke Jan Ijspeert. Central pattern generators for locomotion control in animals and robots: a review. *Neural networks*, 21(4):642–653, 2008.

Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.

Mahdi Khoramshahi, Rezvan Nasiri, Mohammad Shushtari, Auke Jan Ijspeert, and Majid Nili Ahmadabadi. Adaptive natural oscillator to exploit natural dynamics for energy efficiency. *Robotics and Autonomous Systems*, 97:51–60, 2017.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Rezvan Nasiri, Mohammad Shushtari, Hossein Rouhani, and Arash Arami. Virtual energy regulator: A time-independent solution for control of lower limb exoskeletons. *IEEE Robotics and Automation Letters*, 6(4):7699–7705, 2021.

Rezvan Nasiri, Hannah Dinovitzer, and Arash Arami. A unified gait phase estimation and control of exoskeleton using virtual energy regulator (ver). In *2022 International Conference on Rehabilitation Robotics (ICORR)*, pages 1–6. IEEE, 2022.

Ludovic Righetti, Jonas Buchli, and Auke Jan Ijspeert. Dynamic hebbian learning in adaptive frequency oscillators. *Physica D: Nonlinear Phenomena*, 216(2):269–281, 2006.

Di Shi, Wuxiang Zhang, Wei Zhang, and Xilun Ding. A review on lower limb rehabilitation exoskeleton robots. *Chinese Journal of Mechanical Engineering*, 32(1):1–11, 2019.

Manuel Fernando Silva and JA Tenreiro Machado. A literature review on the optimization of legged robots. *Journal of Vibration and Control*, 18(12):1753–1767, 2012.

## Appendix

**Algorithm 1** Deep Deterministic Policy Gradient

```
 1: Input: initial policy parameters $\theta$, Q-function parameters $\phi$, empty replay buffer $\mathcal{D}$
 2: Set target parameters equal to main parameters $\theta_{targ} \leftarrow \theta$, $\phi_{targ} \leftarrow \phi$
 3: **repeat**
 4:     Observe state $s$ and select action $a = clip(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
 5:     Execute $a$ in the environment
 6:     Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
 7:     Store $(s, a, r, s', d)$ in replay buffer $\mathcal{D}$
 8:     If $s'$ is terminal, reset environment state.
 9:     **if** it is time to update **then**
10:         **for** however many updates **do**
11:             Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12:             Compute targets:
            $y(r, s', d) = r + \gamma(1-d)Q_{\phi_{targ}}(s', \mu_{\theta_{targ}}(s'))$
13:             Update Q-function by one step of gradient descent using
            $\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d)\in\mathcal{B}} (Q_\phi(s,a) - y(r,s',d))^2$
14:             Update policy by one step of gradient ascent using
            $\nabla_\theta \frac{1}{|B|} \sum_{s\in B} Q_\phi(s, \mu_\theta(s))$
15:             Update target networks with
            $\phi_{targ} \leftarrow \rho\phi_{targ} + (1-\rho)\phi$
            $\theta_{targ} \leftarrow \rho\theta_{targ} + (1-\rho)\theta$
16:         **end for**
17:     **end if**
18: **until** convergence
```

**Algorithm 2** Twin Delayed DDPG (TD3)

```
1: Input: initial policy parameters $\theta$, Q-function parameters $\phi$, empty replay buffer $\mathcal{D}$
2: Set target parameters equal to main parameters $\theta_{targ} \leftarrow \theta$, $\phi_{targ} \leftarrow \phi$
3: repeat
4:     Observe state $s$ and select action $a = clip(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
5:     Execute $a$ in the environment
6:     Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
7:     Store $(s, a, r, s', d)$ in replay buffer $\mathcal{D}$
8:     If $s'$ is terminal, reset environment state.
9:     if it is time to update then
10:        for however many updates do
11:            Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12:            Compute target actions:
               $a'(s') = clip(\mu_{\theta_{targ}}(s') + clip(\epsilon, -c, c), a_{Low}, a_{High}), \epsilon \sim \mathcal{N}(0, \sigma)$
13:            Compute targets:
               $y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{targ,i}}(s', a'(s'))$
14:            Update Q-function by one step of gradient descent using
               $\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in \mathcal{B}} (Q_\phi(s, a) - y(r, s', d))^2$ for $i = 1, 2$
15:            if $j$ mod policy_delay $= 0$ then
16:                Update policy by one step of gradient ascent using
                   $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_\theta(s))$
17:                Update target networks with
                   $\phi_{targ,i} \leftarrow \rho \phi_{targ,i} + (1 - \rho)\phi_i$
                   $\theta_{targ} \leftarrow \rho \theta_{targ} + (1 - \rho)\theta$
18:            end if
19:        end for
20:    end if
21: until convergence
```