APPLYING DDPG ALGORITHM TO SWING-UP AND BALANCE CONTROL FOR A DOUBLE INVERTED PENDULUM ON A CART

Trong-Nguyen Ho[1,*], Thanh-Sang Tat[2], Hoang-Anh Ngo[2], Truong-Son Nguyen[2], Duc-Anh Bui[2], Thanh-Trung Le[2], Vu-Loc Le[2], Lac-Thien Huynh[2]

[1] Onsemi Viet Nam
17A, No.10, Bien Hoa City, Dong Nai Province, 810000, Viet Nam

[2] Ho Chi Minh City University of Technology and Education
Vo Van Ngan St., No. 01, Thu Duc City, Ho Chi Minh City, 700000, Vietnam

* Corresponding author. E-mail: nguyen.ho@email.com

Abstract: In this study, we apply the Deep Deterministic Policy Gradient (DDPG) algorithm in reinforcement learning to control a double inverted pendulum on a cart (DIPC)- a high order single input-multi output (SIMO) system . The simulation results demonstrate DDPG's stability and effectiveness in achieving swing-up and balance, showing its potential for tackling challenging control tasks in robotics.
Keywords: Reinforcement learning, DDPG, double inverted pendulum on a cart, swing-up, balance.

1. Introduction

In the field of control engineering, applying optimal control methods yields high efficiency but demands knowledge of the system and the ability to construct the system's mathematical equations. This poses challenges, particularly for highly complex systems, those operating in noisy environments, subject to environmental influences, or with high manufacturing inaccuracies in mechanical systems. An effective alternative approach in such cases is the application of machine learning methods, allowing the system to autonomously learn how to control itself without requiring in-depth knowledge of the system, while also enhancing the system's adaptability to its environment.

Reinforcement Learning (RL) is a machine learning subfield that trains systems to make decisions for achieving optimal outcomes through trial-and-error interactions with the environment, without human intervention. Recent advancements in RL have expanded its application to various control problems, such as AlphaGo and AlphaZero in games, autonomous driving, and game management, offering new opportunities for self-learning and adaptation in real-world scenarios.

The primary goal of RL is to train an agent to perform tasks in an unfamiliar environment. The agent receives observations and rewards from the environment and responds by taking actions, with the reward serving as a measure of the action's success in achieving the task's objective.

This paper specifically focuses on the application of the Deep Deterministic Policy Gradients (DDPG) algorithm, DDPG is designed for solving continuous action space problems in which an agent learns to interact with an environment by taking actions to maximize cumulative rewards. It combines elements of both actor-critic methods and deep neural networks to find optimal policies for continuous control tasks. It has shown success in various applications, including autonomous driving and robot control. In this study, we employ the DDPG algorithm to control the balance of a double inverted pendulum system mounted on a cart, renowned for its high instability, strong nonlinearity, and chaotic behavior. Our evaluation of the control method is conducted within a simulated environment using MATLAB.

2. Deep Deterministic Policy Gradients

2.1. Background

Reinforcement learning is a computer-based approach in which a machine learns to perform tasks by interacting with an unknown environment, aiming to maximize the total amount of rewards or cumulative reward through decision-making without human intervention and without being explicitly programmed to achieve the task.

The objective of reinforcement learning is to train an agent to acquire an optimal or nearly optimal policy that maximizes the reward function or another reinforcement signal provided by the user, which is constructed from immediate rewards. The agent interacts with the environment, receiving feedback in the form of observations and rewards, and sends actions to the environment. Rewards indicate the level of success in completing the task.

Deep Deterministic Policy Gradient (DDPG) is a reinforcement learning algorithm designed to tackle challenges in environments with continuous action spaces. It combines concepts from policy-based methods and value-based methods to efficiently learn policies for tasks with continuous control. The algorithm employs a policy network (actor) for selecting actions and a value function (critic) for evaluating those actions.

DDPG incorporates a replay buffer to store past experiences and target networks to stabilize training. This algorithm efficiently learns in environments with continuous action spaces, making it applicable to tasks like robotic control and autonomous systems.

2.2. Mathematical Formula

We denote a set of states as S, a set of actions as A, and a set of rewards as R. At each time step t, the agent receives the environmental state representation, denoted as $S_t \in S$. Based on this state, the agent selects an action $A_t \in A$, resulting in the state-action pair (S_t, A_t). In the next time step, $t+1$, the environment transitions, leading to the new state $S_{t+1} \in S$. At this time step $t+1$, the agent receives a reward $R_{t+1} \in R$ for the action A_t taken from the state S_t. The following figure (Fig. 1) shows a general representation of a reinforcement learning scenario.

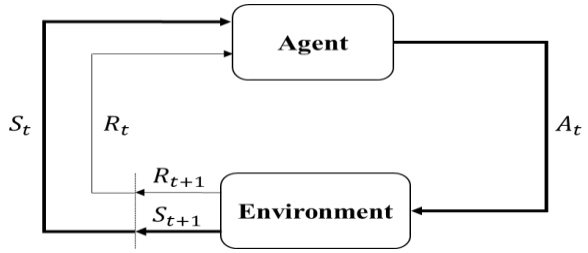

Fig. 1. A reinforcement learning scenario.

The expected return can be represented as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (1)$$

With the discount factor γ is introduced here to guide the agent's attention towards immediate rewards rather than distant future rewards. The value of γ typically ranges between 0 and 1.

If at time t, an agent is following policy π, then $\pi(a|s)$ represents the probability that the action taken at time step t is $A_t = a$ and the state is $S_t = s$. The state-value function for policy π denoted v_π as determines the goodness of any given state for an agent who is following policy π:

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s] \quad (2)$$

The action-value function (3) below determines the goodness of the action taken by the agent from a given state for policy π.

$$q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\,|S_t = s, A_t = a] \quad (3)$$

We consider π to be an optimal policy if it is better than or at least the same as all other policies π' is represented by equation.

$$\pi \geq \pi' \text{ if only if } v_\pi(s) \geq v'_\pi(s) \text{ for all } s \in S \quad (4)$$

The optimal state-value function v_* which gives the largest expected return achievable by any policy π for each state $s \in S$ defined as:

$$v_*(s) = \max_\pi v_\pi(s) \quad (5)$$

Optimal action-value function, or optimal Q-function, which gives the largest expected return achievable by any policy π for $s \in S$ and $a \in A$ defined as:

$$q_*(s,a) = \max_\pi q_\pi(s,a) \quad (6)$$

We denoted as Q_*, and using the Bellman equation. It is given by:

$$Q_*(s,a) = \underset{s' \sim P}{E}\left[r(s,a) + \gamma \max_{a'} Q_*(s',a')\right] \quad (7)$$

where $s' \sim P$ is shorthand for the next state s', is sampled by the environment from a distribution $P(.|s,a)$, and a' is next action.

Assuming the approximator is a neural network $Q_\phi(s,a)$, with parameters ϕ, and given a set $\mathcal{D}$ of transitions of transitions (s,a,r,s',d) (where d indicates whether state s' is terminal). We can set up a mean-squared Bellman error (MSBE) function (8):

$$L(\phi,\mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}}\left[(Q_\phi(s,a) - (r + \gamma(1 - d)\max_{a'} Q_\phi(s',a')))^2\right] \quad (8)$$

The term $r + \gamma(1-d)\max_{a'} Q_\phi(s',a')$ is called target, when we minimize the MSBE loss to make the Q-function be more like this target. The parameters of the target network are denoted ϕ_{targ}. In DDPG algorithms, the target network is updated once per main network update by "*polyak averaging*" (where ρ is a hyperparameter between 0 and 1):

$$\phi_{targ} \leftarrow \rho\phi_{targ} + (1-\rho)\phi \quad (9)$$

Putting it all together, Q-learning in DDPG is performed by minimizing the following MSBE loss with stochastic gradient descent (where $\mu_{\theta targ}$ is the target policy):

$$L(\phi,\mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}}\left[(Q_\phi(s,a) - (r + \gamma(1 - d)Q_{\phi targ}(s',\mu_{\theta targ}(s'))))^2\right] \quad (10)$$

Finally, the policy learning is performed by:

$$\max_\theta \underset{s\sim\mathcal{D}}{\mathrm{E}}\left[Q_\phi(s,\mu_\theta(s))\right] \quad (11)$$

DDPG learns two parameterized functions: a Q-function $Q_\phi(s,a)$ and a policy $\mu_\theta(s)$. In terms of terminology, the Q-function is referred to as the critic, while the policy is known as the actor.

DDPG agents use the critic to estimate the policy values. The critic takes the current observation and action as inputs, producing a scalar output representing the estimated discounted cumulative long-term reward when executing the specified action from the current state and following the policy thereafter. Additionally, DDPG agents employ an actor designed for continuous action spaces. The continuous deterministic actor takes the current observation as input and deterministically outputs an action based on the observation. DDPG operates as an off-policy learning algorithm, indicating that the improvement of the learned policy depends on a separate policy for action selection.

To enhance stability in learning, target networks are implemented for both the critic and actor in DDPG. These target networks are updated based on the soft-update rule, gradually incorporating information from the corresponding main networks. [1].

DDPG uses a replay buffer to avoid focusing too much on recent experiences. By storing and randomly picking past experiences, it helps the algorithm learn more effectively and stay stable during training. This also encourages better exploration of the environment. Its structure is illustrated in (Fig. 2).

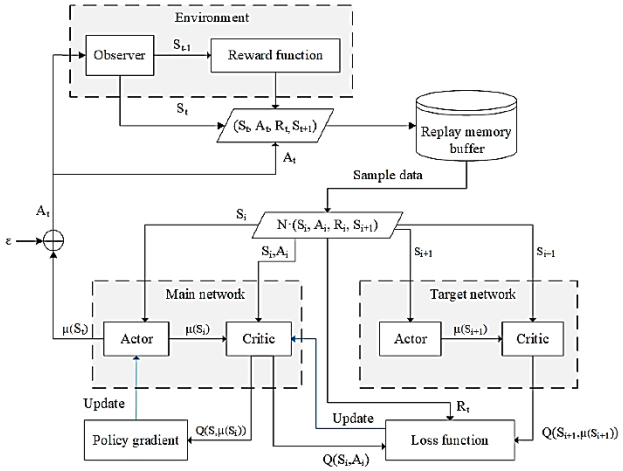

Fig. 2. DDPG diagram [9]

Pseudo-code for the algorithm is provided in (Fig. 3). This is the algorithm we have followed for computing a policy for controlling our DIPC.

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_\phi(s, a) - y(r, s', d))^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

    end for
  end if
until convergence

Fig. 3. DDPG algorithm pseudo-code [7].

3. Double Inverted Pendulum on a Cart

3.1. Background

DIPC comprises two linked pendulums attached to a cart moving along a track (Fig. 4). Designing a controller for these coupled pendulums introduces an extra layer of complexity compared to a single inverted pendulum system. The inclusion of a second pendulum not only presents an additional challenge but also offers an opportunity to demonstrate advanced control concepts or serve as a basis for research.

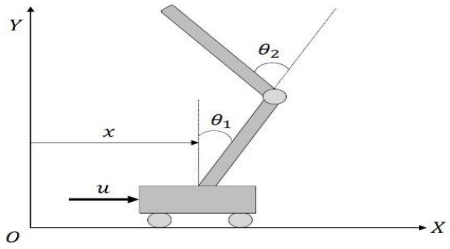

Fig. 4. Mathematical model of DIPC

In this article, our focus is not on solving the mathematical equations that model the system. As previously mentioned, reinforcement learning methods diverge from traditional approaches; RL does not require explicit knowledge and the solution of the system's equations for control computation. The training goal is to make both of pendulums stand upright without falling over from downward position using minimal control effort (swing-up and balance tasks). The system parameters are presented in (Tab. 1)

Tab. 1. Parameters of DIPC

Nr.	Parameter	Description	Value	Unit
1	m_0	The cart mass	0.350	kg
2	m_1	The pendulum 1 mass	0.133	kg
3	m_2	The pendulum 2 mass	0.025	kg
4	A_1	The pendulum 1 length	0.5	m
5	A_2	The pendulum 2 length	0.5	m
6	b_0	Damping coefficient of the cart	0.05	Ns/m
7	b_1	Damping coefficient of the pendulum 1	0.001	Ns/m
8	b_2	Damping coefficient of the pendulum 2	0.001	Ns/m

3.2. State Space

The state space s is composed of 6 continuous states variables $s \in \mathbb{R}^6$, that is represented as:

$$s = \begin{bmatrix} x & \dot{x} & \theta_1 & \dot{\theta}_1 & \theta_2 & \dot{\theta}_2 \end{bmatrix}^T \quad (12)$$

To represent the observation of state s in DDPG, sin/cos functions are used to describe deflection angle changes. It simplifies the representation of fluctuations and rotations, thanks to the inherent simplicity and cyclical nature of these trigonometric functions. The periodic graphs produced by sin/cos functions make it easy to observe and analyze the system's vibrational characteristics. Using $o(s) \in \mathbb{R}^8$, it becomes:

$$o(s) = \begin{bmatrix} x & \dot{x} & \sin\theta_1 & \cos\theta_1 & \dot{\theta_1} & \sin\theta_2 & \cos\theta_2 & \dot{\theta_2} \end{bmatrix}^T \quad (13)$$

where:

$x\,[m]$: correspond to the cart position in the x axis, the domain of this variable is $x \in [-2.4, 2.4]$.

$\theta_1[rad]$ and $\theta_2\,[rad]$: represent respectively the angle of the first and second pendulums as we can see in (Fig. 2). The upward balanced pendulum position is $(0; 0)$ rad, and the downward hanging position is $(\pi; 0)$ rad.

$\dot{x}[m/s]$, $\dot{\theta}_1[rad/s]$, and $\dot{\theta}_2[rad/s]$: represent the velocities of the cart, pendulum 1, and pendulum 2, respectively.

3.3. Action Space

The action space $u(N)$ is continuous where $= [-15, 15]$, $a(u) \in \mathbb{R}^1$. It correspond to the force applied to the cart in one or the other direction.

3.4. Reward Function

The reward r_t, provided at every time step t, is:

$$r_t = -w_0\left(w_1{\theta_{1t}}^2 + w_2{\theta_{2t}}^2 + w_3{x_t}^2 + w_4{u_{t-1}}^2\right) - V_pF \quad (14)$$

where ω_0, ω_1, ω_2, ω_3, ω_4 are the weight variables that indicate the importance or priority of each state in influencing the system's performance or learning behavior. Adjusting these weights allows for a targeted focus on specific states, influencing the optimization process in reinforcement learning or autonomous control systems. V_p is the penalty value with F is a flag (1 or 0) that indicates whether the cart is out of bounds. Their values are shown in (Tab. 2).

Tab. 2. Values of weight matrix

Nr.	Sysbol	Value
1	ω_0	0.1
2	ω_1	5
3	ω_2	5
4	ω_3	1
5	ω_4	0.05
6	V_p	100

3.5. Discount Factor

We have chosen a discount factor γ is 0.99 [5]. This relatively high discount factor encourages the agent to minimize the likelihood of entering a terminal state whenever possible.

3.6. Critic Network

To model the parameterized Q-value function within the critic, utilize a neural network with two input layers, one for the observation channel and the other for the action channel, and one output layer that returns the scalar value.

Critic network Q uses a 2-layer MLP with 400 and 300 neurons. Action is included in the first layer, and "*ReLU*" is the activation function for all hidden layers. Training utilizes the "*Adam optimizer*" with a learning rate of $1e^{-3}$ [2]. Utilize functions provided by Reinforcement Learning Library of MATLAB, it is represented as (Fig. 4)

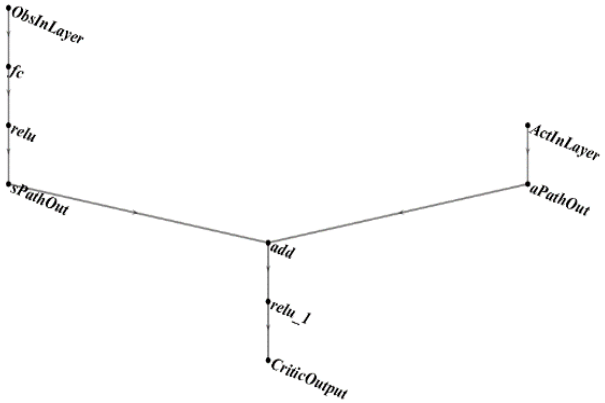

Fig. 4. Critic network structure.

3.7. Actor Network

To define the parameterized policy in the actor, employ a neural network with a single input layer, receiving information from the environment observation channel, and a sole output layer, generating actions for the environment action channel.

Actor network μ is a MLP that has 2 hidden layers of 400 and 300 neurons respectively. The last layer of the network is "*tanhLayer*" constrains output within $[-15\ 15]$ (N), adjust the network output to align with the action range through "*scalingLayer*". "*Adam optimizer*" has been used with a learning rate of $1e^{-4}$ [2]. Utilize functions provided by Reinforcement Learning Library of MATLAB, it is represented as (Fig. 5).

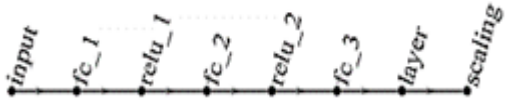

Fig. 5. Actor network structure.

3.8. Experience Reply Buffer

The replay buffer is of finite size $\mathbb{R}$ and the set of transitions $\mathcal{D}$ is randomly sampled from it. Each transition (s, a, r, s', d) sampled from the environment by using the exploration policy is stored in the replay buffer. In this article, we utilize a replay buffer with a size of $1e^6$ and a mini-batch size of 256.

3.9. Training Setup

To train the agent, we set training options in MATLAB to run for a maximum of 10000 episodes, each lasting up to 1000 time-steps with sample time 0.02s. Monitor the training progress through Episode Manager Dialog box of MATLAB. Stop training if the agent achieves an average cumulative reward exceeding -150

across five consecutive episodes. At this point agent's proficiency in efficiently balancing the pendulum in an upright position with minimal control effort.

4. Simulation

The training simulation build as (Fig. 6).

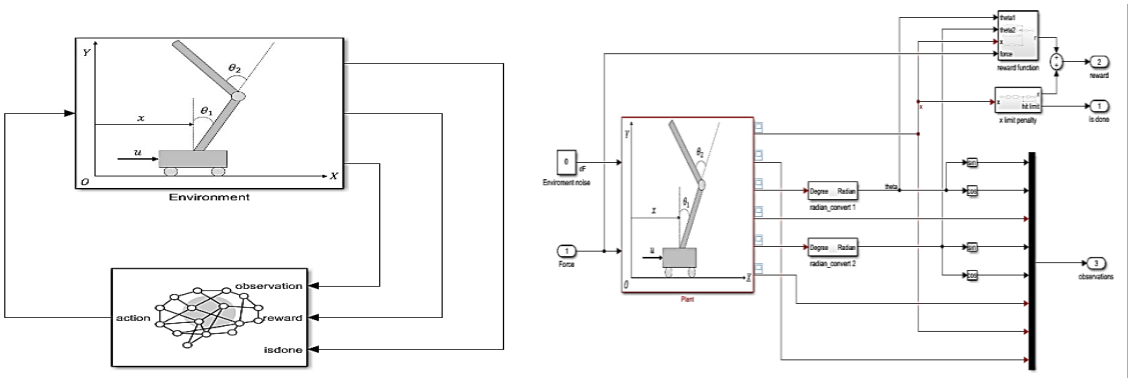

Fig. 6. DIPC System and DDPG Agent in MATLAB Simulink.

We employ Simscape in MATLAB to simulate the physical model of the DIPC system as (Fig. 7). The reliability of Simscape has been validated [8]. The Simscape Multibody model is constructed using physical connections, allowing bidirectional energy flow between components. Additional pendulum stages can be easily added using copy and paste, leveraging the convenience of physical connections. Visualizing the pendulum's behavior during training is facilitated through Simscape monitoring, enabling easier monitoring and adjustment.

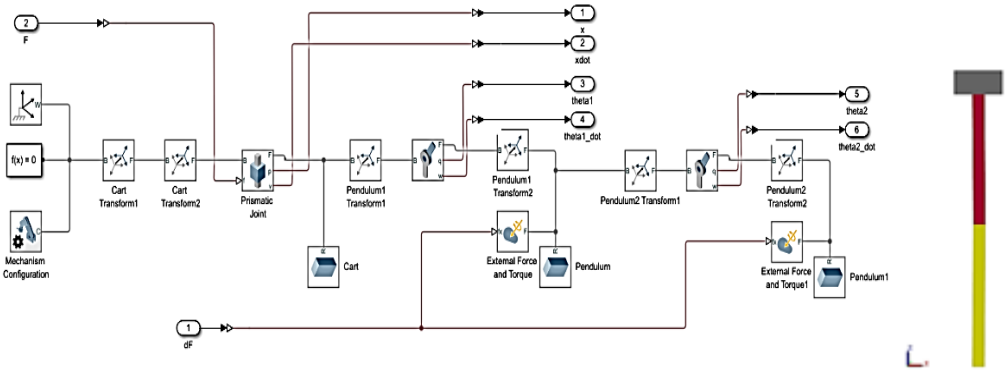

Fig. 7. Building a DIPC in Simscape.

After performing training process, training results are presented in (Fig. 8). The DDPG algorithm demonstrated convergence after exceeding 1000 episodes, successfully achieving the desired outcomes by the 1216th episode. At this point, the agent showcased its ability to swing up the system from initial states and maintain the pendulum in the upright position with minimal control effort. The average reward at this stage was registered at -134.849.

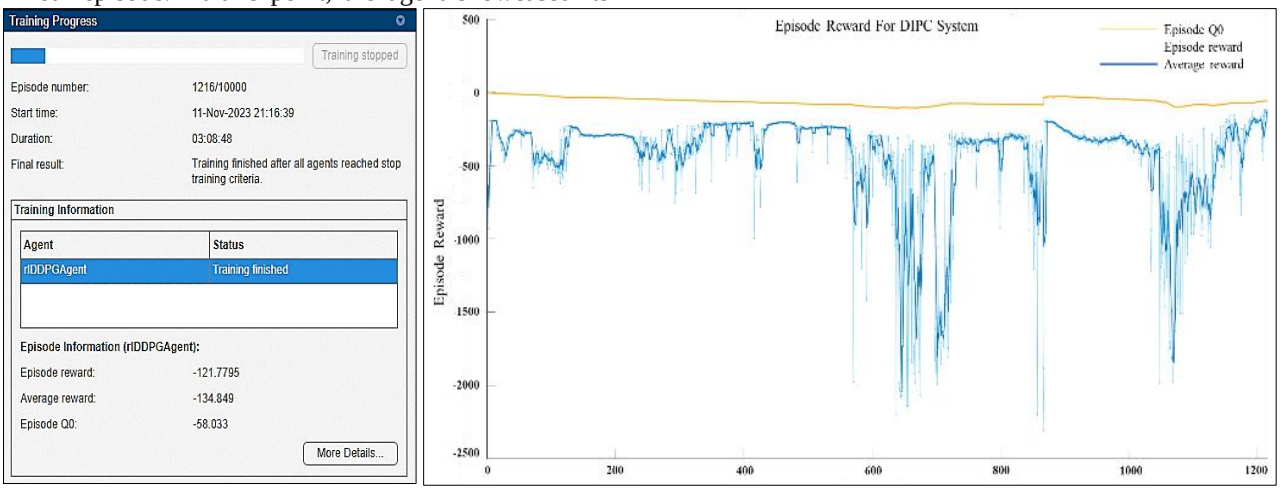

Fig. 8. Training progress.

The output responses for DIPC system are depicted in (Fig. 9), (Fig. 10), (Fig. 11), (Fig. 12), (Fig. 13) respectively. A video showing the DIPC system's natural behavior when swing-up and balance can be found at: https://www.youtube.com/watch?v=e2OT6WeYf9I.

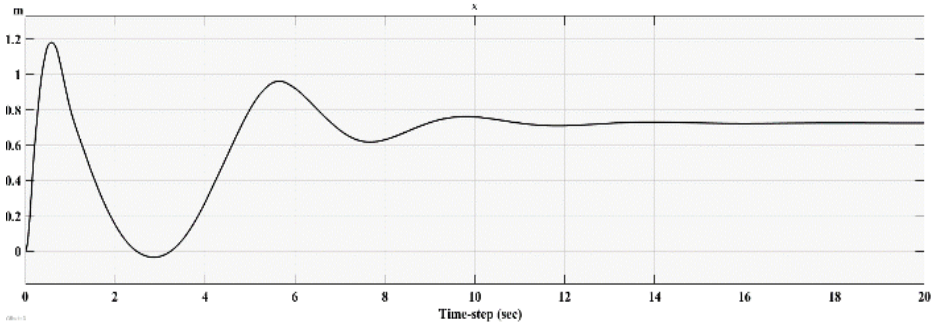

Fig. 9. Cart position (m).

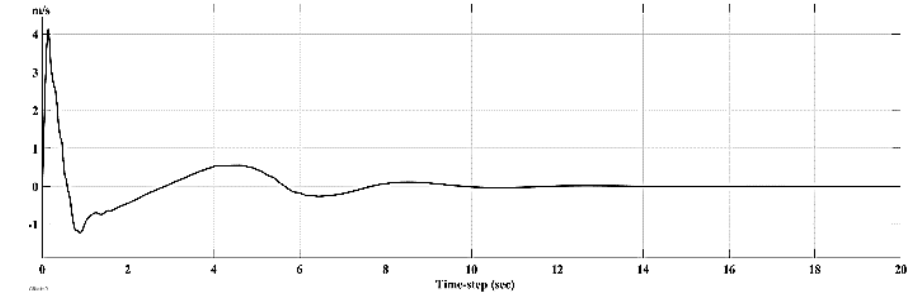

Fig. 10. Cart velocity (m/s).

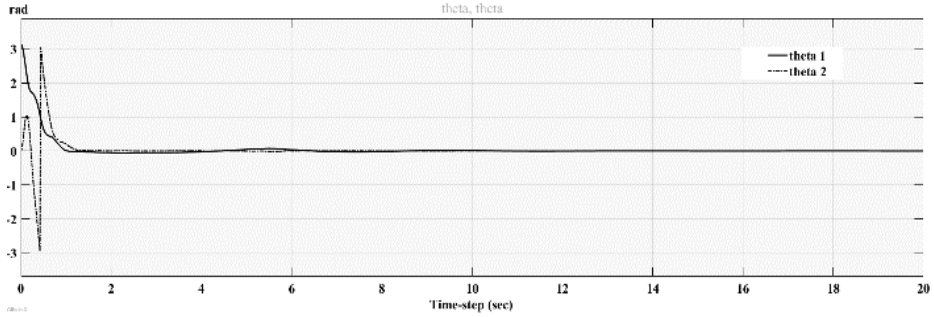

Fig. 11. Angle of Pendulum 1 and 2 (rad).

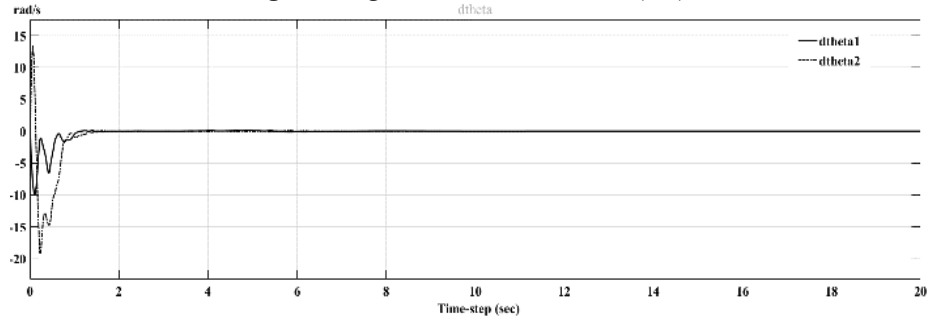

Fig. 12. Angular velocity of Pendulum 1 and 2 (rad/s).

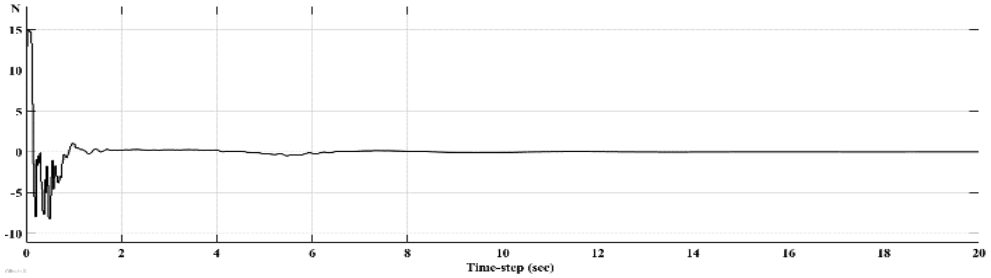

Fig. 13. Force acting on the cart (N)**.**

5. Conclusions

Application of DDPG for controlling DIPC has yielded promising results. The controller successfully maneuvers pendulum to balanced position from initial state in approximately 1 sec. It demonstrates efficient and rapid stabilization. Furthermore, system maintains stability with two inverted pendulums, showcasing the robustness of the DDPG-based control approach. These findings highlight the efficacy of DDPG in addressing the challenging dynamics of DIPC, laying the foundation for further exploration and application in the realm of dynamic control systems. As a future endeavor, we aim to apply the DDPG algorithm to a physical model for real-world testing. This will provide insights into the algorithm's effectiveness, robustness, and adaptability in practical environments. The objective is to bridge the gap between simulation and reality, refining DDPG controller for diverse and dynamic scenarios.

Acknowledgement

We want to give thanks to PhD. Van-Dong-Hai Nguyen (faculty of Electronics and Electrical Engineering, HCMUTE) due to his supervision for this research.

6. References

[1] Sutton R.S., and Andrew G. Barto.: "Reinforcement Learning: An Introduction, Second edition", Adaptive Computation and Machine Learning, Cambridge, Mass: The MIT Press, 2018.

[2] Gustin J., Houyon J.: "Inverted Double Pendulum: Searching High-Quality Policies to Control an Unstable Physical System", INFO8003: Optimal decision making for complex problems, 2022.

[3] Tran V.D, Ho T.N, Nguyen M.T, Nguyen V.D.H.: "Balancing Control For Double-Linked Inverted Pendulum On Cart: Simulation And Experiment", Journal of Technical Education Science, No. 44A, 2017.

[4] Bogdanov A.: "Optimal Control of a Double Inverted Pendulum on a Cart", Technical Report CSE-04-006, 2004.

[5] Lillicrap T.P., Hunt J.J., Pritzel A., Heess N., Erez T., Tassa Y., Silver D, Wierstra D.: "Continuous Control with Deep Reinforcement Learning", ICLR, 2016.

[6] Gustafsson F.: "Control of Inverted Double Pendulum using Reinforcement Learning", 2016.

[7] OpenAI.: "Deep Deterministic Policy Gradient", 2018.

[8] Matlab.: "Single Pendulum in Simulink and Simscape Multibody", 2023.

[9] Panjapornpon C., Chinchalongporn P., Bardeeniz S., Makkayatorn R., Wongpunnawat W.: "Reinforcement Learning Control with Deep Deterministic Policy Gradient Algorithm for Multivariable pH Process", MDPI, 2022.