2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)
June 28 - July 1, 2017 at Maison Glad Jeju, Jeju, Korea

Controlling Bicycle Using Deep Deterministic Policy Gradient Algorithm

Le Pham Tuyen[1], TaeChoong Chung[1]
[1]Computer Science and Engineering Department, Kyung Hee University, Yongin, Gyeonggi, 446-701, South Korea
E-mail: {tuyenple,tcchung}@khu.ac.kr

***Abstract*—Controlling a bicycle without human interaction is still a challenge for researchers. Most of the studies on this topic focus on the physical area of bicycle or designing controllers based on automatic control knowledge such as feedback controller, LQR controller. This study focuses on applying a state-of-the-art deep reinforcement learning algorithm called Deep Deterministic Policy Gradient to control the bicycle. The bicycle can use the learned controller (agent) to keep balancing or reach a specified goal**

***Keywords*—Deep Reinforcement Learning, Deep Deterministic Policy Gradient, Deep-Q Network, Balancing Bicycle, Go-to Bicycle**

1. Introduction

Bicycle was invented in 19th-century [1], which has two wheels attached to a frame. To move and balance the bicycle, it needs a rider (human). During the next two centuries, a lot of designs of the bicycle were studied. Most of them focus on physical and mechanical area [2][3]. The goal of these studies is to make the bicycle more stable and balance. Inspired by bicycle topic, some studies have focused on design bicycle robots, which make the bicycle balance without the human control. [4] and [5] design bicycle robots, which includes a rider balancing on a bicycle frame. [6] and [7] design LEGO-based robots, which can balance without human control. Most of the studies based on knowledge of automatic control and mechanical dynamics such as feedback controller and LQR controller. However, designing controllers for bicycle robot is a difficult task, which requires experiments from experts in mechanical and control. Reinforcement learning (RL) is an approach to learning a controller without knowledge of mechanical and control. The controller of a bicycle is updated during a learning process and can adapt to different dynamics of a robot system. The first study of controlling bicycle using RL was proposed in [8]. In this study, the authors use SARSA(λ) [9] to learn the controller of a bicycle. However, the algorithm only deals with discrete state and action space. Thus, it is hard to apply in practical applications, where state and action space are continuous domains. Study [10] uses bicycle domain to evaluate their RL algorithm, called Least-Squares Policy Iteration. The algorithm work with continuous state space and discrete action space. Even though the algorithm shows the improvement of the controller after learning process, it contains a limitation by only dealing with discrete action space. Because of dealing with continuous state and action space in robotics control, we need a RL algorithm, which can work with high-dimensional continuous state and action space. Recently, due to the innovation in computer systems, researchers focus on a class of RL algorithms called Deep Reinforcement Learning (Deep RL), which employs Deep Neural Networks (DNNs) policy into it. One of the Deep RL algorithms is Deep Deterministic Policy Gradient (DDPG) [12], which can deal with high dimensional continuous action space and uses DNNs to represent the policy. Besides, the algorithm inherits traditional approaches of RL such as policy gradient [15], actor-critic [9].

This study focuses on applying DDPG Algorithm to control the bicycle; simulating a learning process with a learned agent; and built a bicycle robot based on LEGO [14] and using it for applying the algorithm in practical application. The paper is organized as following: in the section 2, we introduce the backgrounds of RL; we describe the way to control bicycle using DDPG in section 3. The section 4 shows simulation results based on some bicycle tasks. Finally, the conclusion section gives us the overall of the paper and suggest the things to do in the future.

2. Background

In this section, we review the background of Reinforcement Learning, Markov Decision Process, Deep Deterministic Policy Gradient Algorithm and Controlling Bicycle using Reinforcement Learning

2.1. Reinforcement Learning and Markov Decision Process

Reinforcement Learning (RL) is a part of machine learning focused on interact with an environment, receive a reward from the environment and learn from the received reward. Usually, RL problems are modeled as a discrete-time Markov Decision Process (MDP) with a tuple of $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$. It includes a state space $\mathcal{S}$; an action space $\mathcal{A}$; a transition function $\mathcal{P}(s_{t+1}|s_t, a_t)$, that measures the probability of obtaining the next state s_{t+1} given a current state-action pair (s_t, a_t); $r(s_t, a_t)$ defines the immediate reward achieved at each state-action pair, and $\gamma \in (0, 1)$ denotes a discount factor. A sequence of state-action pairs (s_t, a_t) creates a trajectory ξ_t (also called an episode) with discounted cumulative reward given by

$$R(\xi) = \sum_{t=0}^{T} \gamma^t r(s_t, a_t).$$

978-1-5090-3056-9/17/$31.00 ©2017 IEEE

A RL algorithm tries to find an optimal policy π^* in order to maximize the expected total discounted reward as follows:

$$J(\pi) = \mathbb{E}[R(\xi)] = \int p(\xi|\pi)R(\xi)d\xi.$$

Policy gradient based methods is one of approach to find the optimal policy. In the policy gradient based methods, the policy is parameterized by parameters vector θ and is updated along the gradient direction of the expected total discounted rewards as

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \nabla_{\boldsymbol{\theta}} J(\pi(\boldsymbol{\theta}_k)),$$

where α denotes a learning rate and k is current update number. Policy gradient based method guarantee to converge to an optimum (local optimum). The main computation of a policy gradient method is how to estimate the gradient from given trajectories such that its variance is small. Many policy gradient methods are summaried in [15] such as finite difference, REINFORCE, actor-critic policy gradient and natural policy gradient.

2.2. Actor-Critic Architecture

Actor-Critic architecture is introduced in [9]. It becomes a model to develop algorithms. This architecture is divided into two parts: actor part and critic part. The actor part receives a state from an environment and uses a policy to predict an action. The critic part receives the state and the reward from the environment to evaluate how good at that state. The policy of actor part is updated during learning process based on the evaluation of critic part. The actor-critic architecture is demonstrated in Fig. 1

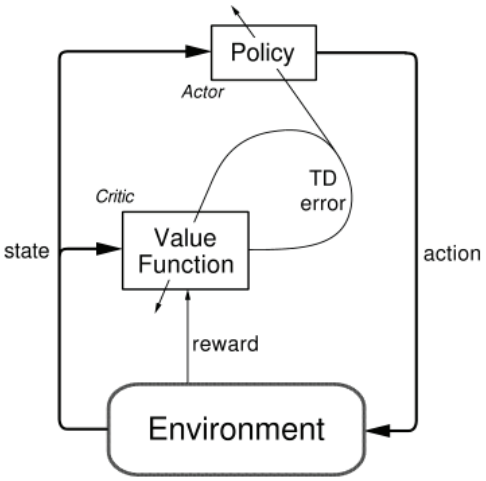


Fig. 1. Actor critic architecture [9]

2.3. Deep Deterministic Policy Gradient Algorithm

Deep Deterministic Policy Gradient (DDPG) are a state-of-the-art Deep RL algorithm [12]. It inherits characteristics of a policy gradient method [15] and an actor-critic method [9]. It uses two Deep-Q Networks [13] to represent the policies: one for actor part and one for critic part. Thus, the policies can represent problems with high complexity. One of the big advantages of DDPG is dealing with high dimensional continuous state and action space, which is necessary to apply to control problem such as robotics. To have this feature, the policy of actor part (a DNN) receive a state from an environment and generate a continuous action. The network parameters in actor's policy are updated based on Deterministic Policy Gradient [11], which is a recent method in policy gradient.

2.4. Controlling Bicycle using Reinforcement Learning

The bicycle system is used in this paper is a derived version of the bicycle system proposed in [8] and [10]. At each time step (0.01 s), the agent will receive information about states of the bicycle: the angle and angular velocity of the handle bars $(\theta, \dot{\theta})$; the angle, angular velocity and acceleration of bicycle $(\omega, \dot{\omega}, \ddot{\omega})$. The agent can choose a continuous action $T \in [-2N, 2N]$, which is the torque applied to the handle bars. The velocity of back wheel is a fixed value of 10 km/h, which is enough to balance the bicycle with only steering the front wheel. The bicycle starts around coordinate $(0, 0)$ m. For more details about the bicycle kinematics and how the control inputs and states constitute in the controller, we can reference to [8]. Fig. 2 shows overall process of controlling a bicycle using a RL algorithm. In this figure, the bicycle is in the role of an environment, which provides states to a RL algorithm and react to an action generated by the algorithm.

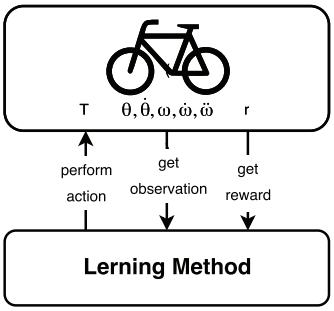


Fig. 2. Controlling bicycle using Reinforcement Learning

3. Controlling Bicycle using DDPG

Overall DDPG procedure is demonstrated in Fig. 3. In this figure, actor network is a DNN with parameters θ^{μ} and $\theta^{\mu'}$. Critic network is another DNN with parameters θ^{Q} and $\theta^{Q'}$. The procedure of DDPG algorithm is explained as follows (The detail parameters are shown in APPENDIX section):

[1] The bicycle observes state s_t and transfers to Actor Network.

[2] The actor network receives s_t as an input and generates an action a_t as an output. After that, the action is add a small noise and is sent back to the bicycle.

$$a_t = \mu\left(s_t|\theta_{\mu} + N_t\right)$$

[3] The bicycle observers a reward r_t, and next state s_{t+1}. The tuple of $\langle s_t, a_t, r_t, s_{t+1}\rangle$ is stored to a experience pool

[4] In the experience pool, we randomly select a batch of N tuples and use them to learn the policies

[5] Compute loss function (e.g. TD error) as follow:

$$L = \frac{1}{N}\sum_{i}\left(y_i - Q\left(s_i, a_i|\theta^Q\right)\right)^2$$

with

$$y_i = r_i + \gamma Q'\left(s_{t+1}, \mu'\left(s_{t+1}|\theta^{\mu'}\right)|\theta^{Q'}\right)$$

[6] Update critic network by minimizing the loss L

[7] Update actor network using deterministic policy gradient theorem [11]

$$\nabla_{\theta^\mu}\mu|_{s_i} \approx \frac{1}{N}\sum_{i}\nabla_a Q\left(s, a|\theta^Q\right)|_{s=s_i, a=\mu(s_i)} \times \nabla_{\theta^\mu}\mu\left(s|\theta^\mu\right)|_{s_i}$$

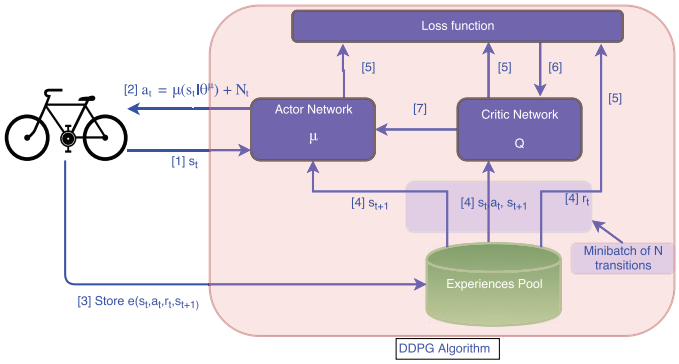


Fig. 3. Procedure of DDPG algorithm

4. Experiments

We control the bicycle on two tasks: balancing task and go-to task. For balancing task, the agent controls the bicycle so that the bicycle can balance as long as possible. The task finished when the bicycle can balance for 60 seconds. I evaluate the learning process with different reward functions. For each reward function we have different convergence speed and different trajectories of bicycle. First, we evaluate the learning algorithm with a simple reward function (Eq. 3), which gives 1 if the bicycle is still balance and 0 if the bicycle falls down. Intuitively, when the bicycle can balance in long time, it can receive more reward.

$$r(s,a) = \begin{cases} 0 & |\omega| > \frac{\pi}{6} \\ 1 & |\omega| < \frac{\pi}{6} \end{cases} \quad (1)$$

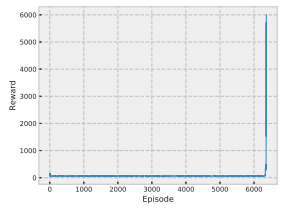
The second evaluated reward function is shown in Eq. 4. The reward will be high if the value of $(\omega, \dot{\omega}, \ddot{\omega})$ is small and otherwise. In this case, the bicycle will receive a high reward if it can keep the straight pose in the long time.

$$r(s,a) = \begin{cases} 0 & |\omega| > \frac{\pi}{6} \\ -\left(\omega^2 + 0.1\dot{\omega}^2 + 0.01\ddot{\omega}^2\right) & |\omega| < \frac{\pi}{6} \end{cases} \quad (2)$$

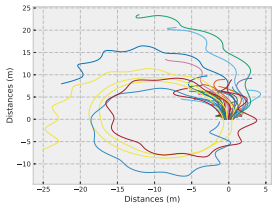
The third evaluated reward function (Eq. 5). is similar to the second one. However, we reverse the coefficients of $(\omega, \dot{\omega}, \ddot{\omega})$. Therefore, in the second function, ω is the most impacted parameter to the magnitude of reward function. In the case of the third reward function is parameter $\ddot{\omega}$. It means that the bicycle will receive a high reward if it does not suddenly change the acceleration.

$$r(s,a) = \begin{cases} 0 & |\omega| > \frac{\pi}{6} \\ -\left(0.01\omega^2 + 0.1\dot{\omega}^2 + \ddot{\omega}^2\right) & |\omega| < \frac{\pi}{6} \end{cases} \quad (3)$$

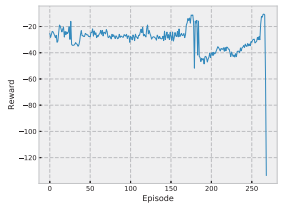
Fig. 4. reports performance and trajectory of bicycle task for different reward functions. With the Eq. 3., the balancing task needs to be trained over 6000 episodes. The training process for balancing task using Eq. 4. and Eq. 5. is faster than using Eq. 3. In the initial stage of learning process, the bicycle easily falls down Using Eq. 5. gives the learning process converge gradually. Even though each learning process has different convergence speed, both of them can obtained a good policy, which can keep bicycle balancing in long time.



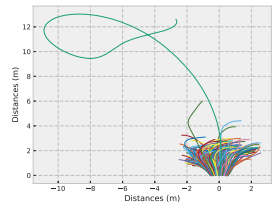
(a) Performance using Eq. 3.



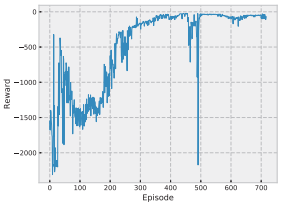
(b) Trajectory using Eq. 3.



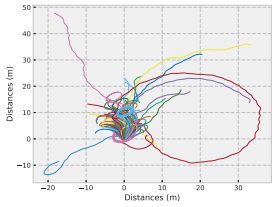
(c) Performance using Eq. 4.



(d) Trajectory using Eq. 4.



(e) Performance using Eq. 5.



(f) Trajectory using Eq. 5.

Fig. 4. Bicycle trajectory and performance of balancing task for each reward function

A bicycle using the learned policy for balancing will generate circular trajectories (Fig. 5). In this figure, we show five trajectories (in five different colors) of a bicycle started at $(0, 0)$ m with a slightly changing in direction

The go-to task is more difficult than balancing task. It requires the agent control the bicycle to reach a specified goal

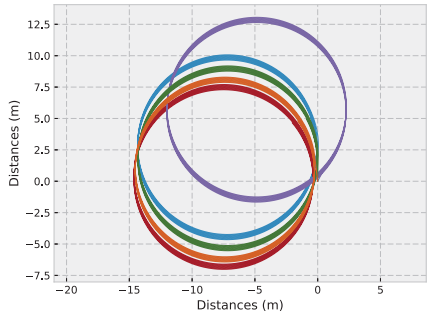


Fig. 5. Trajectories of a bicycle using learned policy

without falling down. Fig. 6(a) demonstrates our settings for go-to task. In our settings, the bicycle starts around $(0, 0)$ m and perpendiculars to the goal located at $(250, 0)$ m (the red circle). The reward function for go-to task as following,

$$r(s,a) = \begin{cases} 0 & |\omega| > \frac{\pi}{6} \\ -(0.01\omega^2 + 0.1\dot{\omega}^2 + \ddot{\omega}^2) - 10\Delta d_g & |\omega| < \frac{\pi}{6} \end{cases} \quad (4)$$

where

$$\Delta d_g = d_g - d_g^{'}.$$

d_g and $d_g^{'}$ are distance between bicycle and the goal in the current step and previous step correspondingly. The first part in Eq. 6. similar to Eq. 5, which leads the bicycle in balancing situation and the second part helps the bicycle reach a specified goal. Trajectories of bicycle is reported in Fig. 6(b). During the learning process (around 25000 episodes), the bicycle is attracted by the goal and moves to the goal rather than moving randomly. However, the obtained policy only let the bicycle head to the goal but not exact goal position.

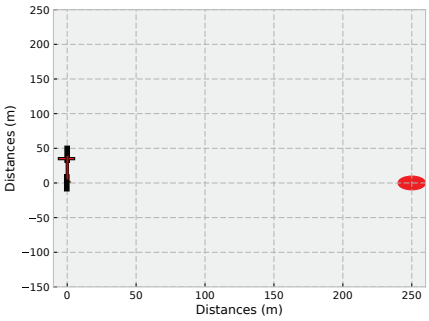
A computer simulation (using Panda3D [18]) is made to evaluate the learned controller (Fig. 7.)

5. Conclusion

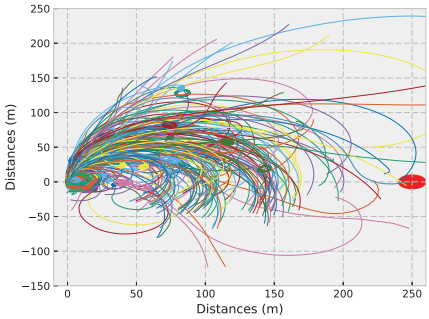
DDPG algorithm is a state-of-the-art RL algorithm for high-dimensional continuous tasks. To apply the algorithm in a practical application, we develop a bicycle robot using LEGO EV3 platform (Fig. 8.) inspired from [14]. However, directly applied the DDPG algorithm on LEGO EV3 is difficult due to the limitation of the EV3 platform (CPU and RAM). In the future, we can consider other approaches to run DDPG on LEGO platform such as remote control or built a stronger bicycle robot system.

Acknowledgement

This research was supported by the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science, and Technology (2014R1A1A2057735); and the Kyung Hee University in 2016 (KHU-20160601).



(a) Initial state of bicycle



(b) Training Trajectory trajectories

Fig. 6. Bicycle trajectory of go-to task

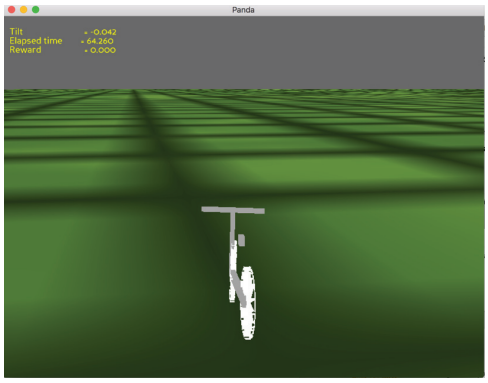


Fig. 7. Bicycle simulation

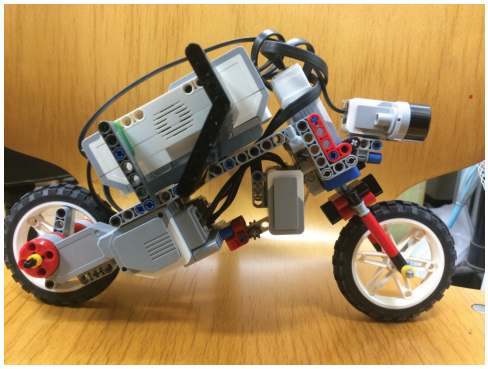


Fig. 8. Bike robot using LEGO

REFERENCES

[1] Herlihy, David V. Bicycle: the history. Yale University Press, 2004.
[2] Schwab, Meijaard, and Kooijman. "Some recent developments in bicycle dynamics." Proceedings of 12th World Congress in Mechanism and Machine Science. 2007.
[3] Meijaard, Jaap P., et al. "Linearized dynamics equations for the balance and steer of a bicycle: a benchmark and review." Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences. Vol. 463. No. 2084. The Royal Society, 2007.
[4] Murata Boy. Retrieved from: http://www.murata.com/en-us/about/mboymgirl/mboy
[5] Bicycle Robot. Retrieved from: http://ai2001.ifdef.jp/primer_V2/primer_V2.html
[6] Basso, Michele, and Giacomo Innocenti. "Legobike: A challenging robotic lab project to illustrate rapid prototyping in the mindstorms/simulink integrated platform." Computer Applications in Engineering Education 23.6 (2015): 947-958.
[7] Basso, Michele, Giacomo Innocenti, and Alberto Rosa. "Simulink meets lego: Rapid controller prototyping of a stabilized bicycle model." 52nd IEEE Conference on Decision and Control. IEEE, 2013.
[8] Randlv, Jette, and Preben Alstrm. "Learning to Drive a Bicycle Using Reinforcement Learning and Shaping." ICML. Vol. 98. 1998.
[9] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. Vol. 1. No. 1. Cambridge: MIT press, 1998.
[10] Lagoudakis and Ronald. "Model-free least-squares policy iteration." NIPS. Vol. 14. 2001.
[11] Lever, Guy. "Deterministic policy gradient algorithms." (2014).
[12] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).
[13] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533.
[14] Retrieved from: https://www.lego.com/en-us/mindstorms/products/mindstorms-ev3-31313
[15] Peters, Jan, and Stefan Schaal. "Reinforcement learning of motor skills with policy gradients." Neural networks 21.4 (2008): 682-697.
[16] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).
[17] Uhlenbeck, George E and Ornstein, Leonard S. "On the theory of the brownian motion." Physical review, 36(5):823, 1930
[18] Goslin, Mike, and Mark R. Mine. "The Panda3D graphics engine." Computer 37.10 (2004): 112-114.

APPENDIX

Actor Network Parameters

TABLE 1
PARAMETERS OF ACTOR NETWORK.

Name	Value
Input layer	A state (s_t)
1st fully-connected layer	400 units
2nd fully-connected layer	300 units
Output layer	An action (a_t)
Initial parameters of hidden layer	Uniformly random between $\left[-\frac{1}{\sqrt{f}}, \frac{1}{\sqrt{f}}\right]$ where f is the fan-in of the layer
Initial parameters of output layers	Uniformly random between $\left[-3e^{-3}, 3e^{-3}\right]$
Learning rate	0.0001
Optimizer	Adam [16]

Critic Network Parameters

TABLE 2
PARAMETERS OF CRITIC NETWORK.

Name	Value
Input layer	State-action pair (s_t, a_t)
1st fully-connected layer	400 units
2nd fully-connected layer	300 units
Output layer	Q-value
Initial parameters of hidden layer	Uniformly random between $\left[-\frac{1}{\sqrt{f}}, \frac{1}{\sqrt{f}}\right]$ where f is the fan-in of the layer
Initial parameters of output layer	Uniformly random between $\left[-3e^{-3}, 3e^{-3}\right]$
Learning rate	0.001
Optimizer	Adam [16]

Experiment Parameters

TABLE 3
PARAMETERS OF ALGORITHM

Name	Value
Input dimension	5
Output dimension	1
Discounted factor	0.99
Random noise	Ornstein-Uhlenbeck process [17] with $\theta = 0.15$ and $\sigma = 0.2$
Experience pool capacity	500000
Batch size	256 samples