symmetry

MDPI

Article

Toward Self-Driving Bicycles Using State-of-the-Art Deep Reinforcement Learning Algorithms

SeungYoon Choi [1,†], Tuyen P. Le [1,†], Quang D. Nguyen [1], Md Abu Layek [1], SeungGwan Lee [2,*] and TaeChoong Chung [1]

[1] Artificial Intelligence Lab, Computer Science and Engineering, Kyung Hee University, Yongin-si, Gyonggi-do, Gyeonggi 446-701, Korea; sychoi84@khu.ac.kr (S.C.); tuyenple@khu.ac.kr (T.P.L.); quangnd@khu.ac.kr (Q.D.N.); layek@khu.ac.kr (M.A.L.); tcchung@khu.ac.kr (T.C.)

[2] Humanitas College, Kyung Hee University, Yongin, Gyeonggi 446-701, Korea

* Correspondence: leesg@khu.ac.kr

† These authors contributed equally to this work.

Received: 22 January 2019; Accepted: 19 February 2019; Published: 23 February 2019

Abstract: In this paper, we propose a controller for a bicycle using the DDPG (Deep Deterministic Policy Gradient) algorithm, which is a state-of-the-art deep reinforcement learning algorithm. We use a reward function and a deep neural network to build the controller. By using the proposed controller, a bicycle can not only be stably balanced but also travel to any specified location. We confirm that the controller with DDPG shows better performance than the other baselines such as Normalized Advantage Function (NAF) and Proximal Policy Optimization (PPO). For the performance evaluation, we implemented the proposed algorithm in various settings such as fixed and random speed, start location, and destination location.

Keywords: deep reinforcement learning; deep deterministic policy gradient (DDPG); machine learning; self-driving bicycle

1. Introduction

Bicycles are efficient vehicles in terms of their environment-friendly, affordable, and user-friendly characteristics. However, due to the unstable dynamics of bicycles, riders spend significant effort practicing and being observant while riding. There are already self-driving cars and autonomous air vehicles, but self-driving bicycles are still being developed [1]. Many studies have proposed methods to improve such bicycles both in terms of their mechanisms and controllers [2–5]. Particularly, studies [2,3] focus on the physical enhancement of bicycles, and studies [4,5] develop bicycle controllers based on control theory and bicycle dynamics. However, their proposed controllers only work properly in simulation environments and fails to apply to the real world due to disturbances present in a real environment. A reinforcement learning-based controller is able to interact with the surroundings and is adaptable to various environments [6–8]. Despite these possibilities, very few studies have used reinforcement learning to develop bicycle controllers [5,9,10]. Randlov [9] built a controller based on the SARSA algorithm [11], which could not handle the highly variable state and action space. Jie Tan [5] used shallow neural network controllers that apply policy gradient methods [12] to train parameters. However, shallow neural network controllers have limitations in expressing highly nonlinear environments such as bicycles. Tuyen [10] used a deep neural network to represent the bicycle controller. In his implementation, the controller is quickly trained by using an algorithm called the deep deterministic policy gradient (DDPG) [13]. The controller allows the bicycle to perfectly balance itself but fails to lead the bicycle to any location. In this study, we propose an improved controller that can lead the bicycle to any location. Note that a bicycle with fixed speed will be difficult

to turn while moving. Therefore, in this paper, we extend the existing bicycle models so that the bicycle velocity can be controlled by a neural network controller.

In fact, the paper is an extended version of work published in [10,14]. We extend our previous work by modifying the bicycle dynamics and learning the controller to adapt to the new dynamics. The contributions of this paper are highlighted as follows. First, we redefine the bicycle dynamics to adaptively control the velocity of the bicycle. Second, we propose the learning process in which we use a reward function for not only balancing the bicycle but also leading the bicycle to a given destination. The learning process uses the DDPG algorithm.

The rest of the paper is arranged as follows. Section 2 shortly reviews background knowledge. Section 3 introduces the dynamics of the bicycle. Section 4 describes the overall learning process. Section 5 shows the results of our proposed controller. Finally, Section 6 summarizes our work and provides some future directions on this topic.

2. Background

2.1. Reinforcement Learning and Policy Gradient Based Method

Reinforcement Learning (RL) [11] is a subfield of machine learning that uses reward values directly received from the environment to learn an agent. Basically, Markov decision process (MDP) is formally used to describe RL problems. It is modelled with a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$ which consists of a state space $\mathcal{S}$; an action space $\mathcal{A}$; a transition function $\mathcal{P}(s_{t+1}|s_t, a_t)$ that predicts the next state s_{t+1} given a current state-action pair (s_t, a_t); $r\,(s_t, a_t)$ that defines the immediate reward achieved at each state-action pair, and $\gamma \in (0, 1)$ denotes a discount factor. Agent collects a sequence of state-action pairs (s_t, a_t) called a trajectory ξ_t (e.g., episode, rollout) with discounted cumulative reward given by

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

The policy of a RL problem is a function π which maps the state space to the action space. An RL algorithm tries to find an optimal policy π^* 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 are one approach to find the optimal policy. In the policy gradient-based methods, the policy is parameterized by a parameters vector $\boldsymbol{\theta}$ 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 the learning rate and k is the current update number.

2.2. Deep Deterministic Policy Gradient Algorithm

DDPG [13] is an off-policy algorithm that uses deep neural networks to represent the policy. The features of this algorithm are introduced as follows. First, the algorithm inherits an actor-critic framework [11]. This means that there are two components in the algorithm, the actor and the critic. The actor takes responsibility for a policy, which receives a state as the input and generates an action. The critic estimates the action value function, which is used to assess the goodness of the actor. Second, the algorithm uses two deep neural networks, one each for the actor and the critic. Such a framework powerful enough to represent a highly non-linear task such as controlling a bicycle. Third, the algorithm uses a deterministic policy gradient [15] to train the actor network as follows:

$$\nabla_\theta J(\pi(\theta^\mu)) \approx \mathbb{E}\left[\nabla_a Q\left(s, a|\theta^Q\right) \times \nabla_{\theta^\mu}\mu\left(s|\theta^\mu\right)\right]$$

where $\nabla_\theta J(\pi(\theta^\mu))$ is the policy gradient, $\nabla_a Q\left(s, a|\theta^Q\right)$ is the gradient of the action value function w.r.t action a, and $\nabla_{\theta^\mu}\mu\left(s|\theta^\mu\right)$ is the gradient of the actor w.r.t parameter θ^μ. Finally, the DDPG algorithm inherits two features from Deep Q-Learning [16]. The first feature is maintaining a copy for each network, e.g., copies of the actor network and critic network. The copy ones improve the stability of the learning process. The second feature is maintaining a replay memory that stores all of the sample data during interacting with the environment. At each time step, we randomly sample a batch of data from the replay memory and use them to train the networks. The replay memory removes the correlation in the sequence of the data sample. Using a deterministic policy is more stable than a stochastic policy where the actions are drawn from a distribution.

3. Extended Bicycle Dynamics

Studies on bicycle dynamics are often studied by researchers [2,5,9,17]. The first bicycle studied by Randlov [9] is illustrated in Figure 1.

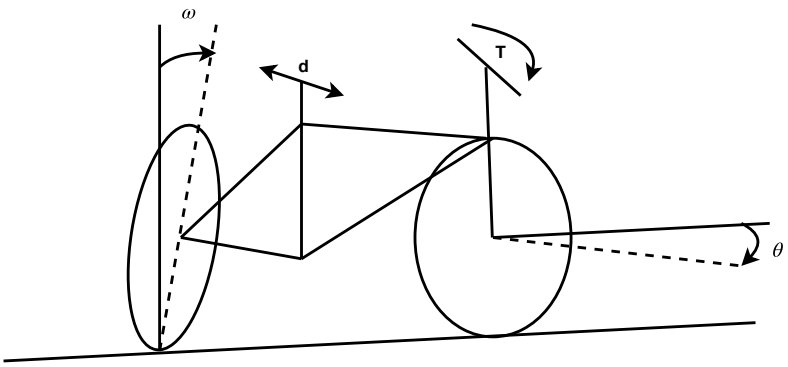
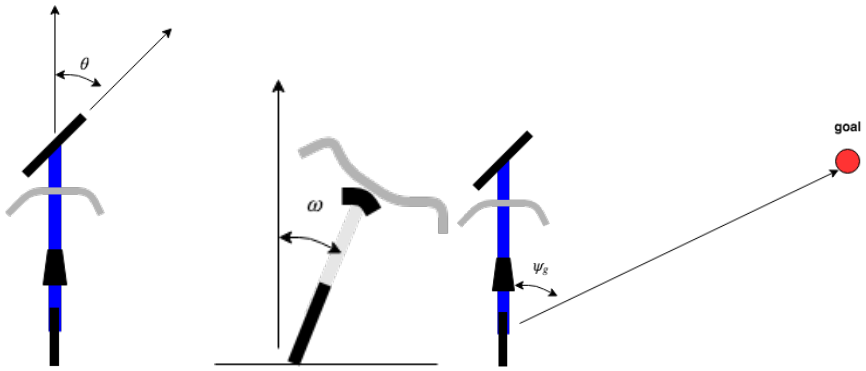


Figure 1. Illustration of a bicycle.

This paper not only utilizes the work of Randlov [9] study, but also extends it to where the velocity of the bicycle is dynamic. Particularly, the bicycle has six dimensional states: $(\omega, \dot{\omega}, \ddot{\omega}, \theta, \dot{\theta}, \psi_g)$ where ω, $\dot{\omega}$, $\ddot{\omega}$ are the angle, angular velocity, and angular acceleration of the bicycle relative to the vertical plane. $\theta, \dot{\theta}$ are the angle and angular velocity of the handlebars and ψ_g is the angle formed by the bicycle and a specified goal g. The bicycle states are demonstrated in Figure 2. To control the bicycle, the agent chooses three actions. The first action is the torque applied to the handlebar (T). The second action is the displacement (d) between the center of mass and the bicycle plan (Figure 3). The third action is the force (F) applied to the pedal of the bicycle.



(**a**) $\theta, \dot{\theta}$ (**b**) $\omega, \dot{\omega}, \ddot{\omega}$ (**c**) ψ_g

Figure 2. Demonstration of 6-dimensional states.

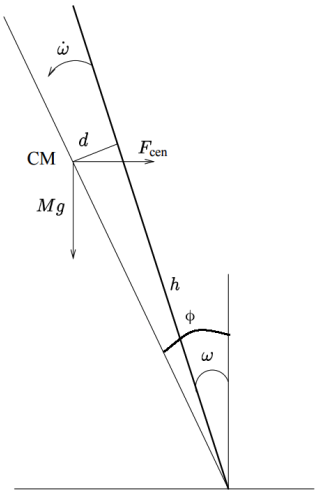


Figure 3. The bicycle seen from behind. The thick line represents the bicycle [9].

The equations of the position of the tires for the front tire:

$$\begin{bmatrix} x_f \\ y_f \end{bmatrix}_{(t+1)} = \begin{bmatrix} x_f \\ y_f \end{bmatrix}_{(t)} + vdt \begin{bmatrix} -\sin\left(\psi + \theta + sign(\psi + \theta)\arcsin(\frac{vdt}{2r_f})\right) \\ \cos\left(\psi + \theta + sign(\psi + \theta)\arcsin(\frac{vdt}{2r_f})\right) \end{bmatrix}, \tag{1}$$

and for the back tire:

$$\begin{bmatrix} x_b \\ y_b \end{bmatrix}_{(t+1)} = \begin{bmatrix} x_b \\ y_b \end{bmatrix}_{(t)} + vdt \begin{bmatrix} -\sin\left(\psi + sign(\psi)\arcsin(\frac{vdt}{2r_b})\right) \\ \cos\left(\psi + sign(\psi)\arcsin(\frac{vdt}{2r_b})\right) \end{bmatrix}, \tag{2}$$

where ψ is angle make by bicycle and horizontal line, and r_b and r_f (Figure 4) are radii of the front tire and back tire, respectively. The radii are given by:

$$r_f = \frac{l}{|\cos(\frac{\pi}{2} - \theta)|} = \frac{l}{|\sin\theta|.} \tag{3}$$

and

$$r_b = l\left|\tan\left(\frac{\pi}{2} - \theta\right)\right| = \frac{l}{|\tan\theta|} \tag{4}$$

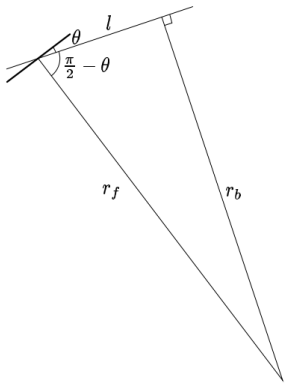


Figure 4. The bicycle seen from above. The thick line represents the front tire [9].

The angular acceleration $\ddot{\omega}$ can be calculated as:

$$\ddot{\omega} = \frac{1}{I_{bicycle}}\left(Mhg\sin\phi - \cos\phi\left(I_{dc}\dot{\sigma}\dot{\theta} + sign(\theta)v^2\left(\frac{M_d r}{r_f} + \frac{M_d r}{r_b} + \frac{Mh}{r_{CM}}\right)\right)\right), \tag{5}$$

where angle ϕ is the total angle of tilt of the center of mass (CM) (Figure 3), and is defined as:

$$\phi = \omega + arctan(\frac{d}{h}). \tag{6}$$

The angular acceleration $\ddot{\theta}$ of the front tire and the handle bar is

$$\ddot{\theta} = \frac{T - I_{dv}\dot{\sigma}\dot{\omega}}{I_{dl}}. \tag{7}$$

The moment of inertia has the formula:

$$I_{bicycle} = \frac{13}{3}M_c h^2 + M_p(h + d_{CM})^2, \tag{8}$$

where various moments of inertia for a tire (Figure 5) are estimated to: $I_{dc} = M_d r^2$, $I_{dv} = \frac{3}{2}M_d r^2$, $I_{dl} = \frac{1}{2}M_d r^2$.

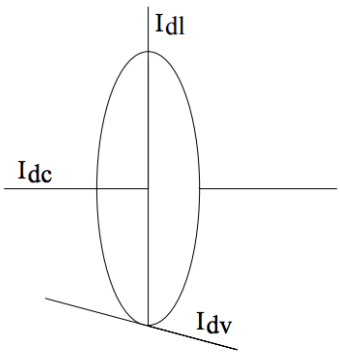


Figure 5. Axis for the moments of inertia for a tire [9].

The velocity of the bicycle can be adjusted by learning the force applied to the pedal. Figure 6 shows how the force applied to the pedal can be transmitted to the bicycle. Using the static equilibrium assumption, we can write the following torque equations:

$$F_1 R_1 = F_2 R_2 \tag{9}$$

and

$$F_3 R_3 = F_4 R_4. \tag{10}$$

Since $F_2 = F_3$, we can combine the above two equations to give an expression for F_4:

$$F_4 = F_1 \frac{R_1 R_3}{R_2 R_4}. \tag{11}$$

The force F_4 determines the acceleration of the bicycle. Particularly, the acceleration of the bicycle is as follows:

$$a = \frac{F_4}{M_c + M_d + M_p}. \tag{12}$$

From the acceleration, we can calculate the velocity of the bicycle:

$$v = v + \Delta t \times a. \tag{13}$$

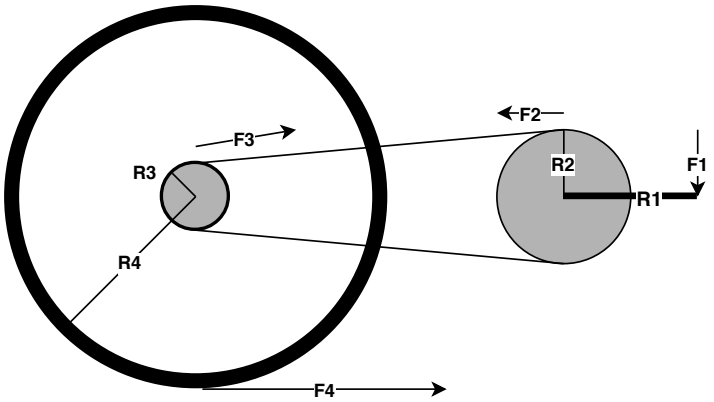


Figure 6. Force transmission from the pedal.

Various parameters for the bicycle dynamics are shown in Table 1.

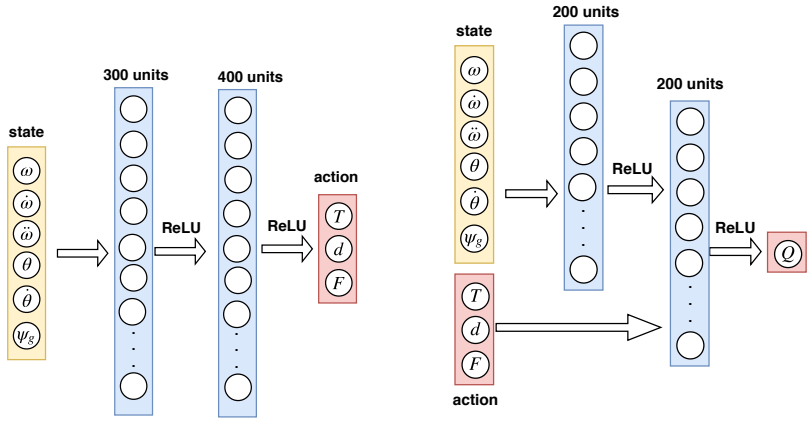
Table 1. Parameters of bicycle dynamics [9].

Notation	Description	Value
c	Horizontal distance between the point, where the front wheel touches the ground and the CM	66 cm
CM	The Center of Mass of the bicycle and cyclist as a whole	
d	The agent's choice of the displacement of the CM perpendicular to the plane of the bicycle	
d_{CM}	The vertical distance between the CM for the bicycle and for the cyclist	30 cm
h	Height of the CM over the ground	94 cm
l	Distance between the front tire and the back tyre at the point where they touch the ground	111 cm
M_c	Mass of the bicycle	15 kg
M_d	Mass of a tire	1.7 kg
M_p	Mass of the cyclist	60 kg
r	Radius of the tire	34 cm
$\dot{\sigma}$	The angular velocity of a tire	$\dot{\sigma} = \frac{v}{r}$
T	The torque the agent applies to the handlebars	
dt	Time step	0.025 s

4. Method to Control Bicycle

4.1. Network Structure

There are two networks used for training, namely a critic network and an actor network, which are illustrated in Figure 7. The input of the actor network is a 6-dimensional state vector and the output of the actor network is a 3-dimensional action vector. Meanwhile, the input of the critic network is both a state vector and an action vector, and the output of the critic network is a Q action value. The configurations of the actor network and critic network are shown in Table 2. Two hidden layers of the actor network have 300 units and 400 units, respectively, while both hidden layers of the critic network have 200 units. The action vector only joins the network at the second hidden layers. The parameters of the networks are initialized randomly and are optimized using the ADAM algorithm [18]. The target networks are updated using a soft-updating technique with learning rate $\tau = 0.001$. We did not try different numbers of layers of units. From our experience, however, neural networks are quite flexible and can cope with a variety of settings.



(**a**) Actor network (**b**) Critic network

Figure 7. The two neural networks used in this paper.

Table 2. Parameters of the actor network and the critic network.

Name	Actor	Critic
Input layer	A state vector (s_t)	State vector and action vector (s_t, a_t)
1st fully-connected layer	400 units	200 units
2nd fully-connected layer	300 units	200 units
Output layer	An action vector (a_t)	Q-value
Initial parameters	Uniformly random between $[-3e^{-3}, 3e^{-3}]$	Uniformly random between $[-3e^{-3}, 3e^{-3}]$
Learning rate	0.001	0.001
Optimizer	ADAM [18]	ADAM [18]

4.2. Network Training

Critic network training. Critic networks include a main critic network (Q) parameterized by θ^Q and a target critic network (Q') parameterized by $\theta^{Q'}$. At every discrete time step, the main network is updated using a batch of samples, which obtains data from the replay memory. Particularly, θ^Q is optimized to minimize the loss function as follows:

$$L = \frac{1}{N}\sum_i \left(y_i - Q\left(s_i, a_i|\theta^Q\right)\right)^2, \tag{14}$$

where i indicates the ith sample in the batch and

$$y_i = r_i + \gamma Q'\left(s_{t+1}, \mu'\left(s_{t+1}|\theta^{\mu'}\right)|\theta^{Q'}\right). \tag{15}$$

$\theta^{Q'}$ is coupled with θ^Q using a soft-updating technique with learning rate τ as follows:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}. \tag{16}$$

Actor network training. Similarly, actor networks include a main actor network (μ) parameterized by θ^μ and a target actor network (μ') parameterized by $\theta^{\mu'}$. The main actor network is updated using the deterministic policy gradient theorem [15] as follows:

$$\nabla_{\theta^\mu}\mu|_{s_i} \approx \frac{1}{N}\sum_i \nabla_a Q\left(s, a|\theta^Q\right)|_{s=s_i, a=\mu(s_i)} \times \nabla_{\theta^\mu}\mu\left(s|\theta^\mu\right)|_{s_i}, \tag{17}$$

where $\nabla_a Q$ is the gradient of the critic w.r.t. action a and $\nabla_{\theta^\mu}\mu$ is the gradient of actor w.r.t. parameter θ^μ. $\theta^{\mu'}$ is updated using a soft-updating technique with learning rate τ as follows:

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}. \tag{18}$$

4.3. Learning Process

Applying reinforcement learning to the bicycle problem is described as follows. The state of a bicycle includes $(\omega, \dot{\omega}, \ddot{\omega}, \theta, \dot{\theta}, \psi_g)$, where ω, $\dot{\omega}$, $\ddot{\omega}$ are the angle, angular velocity, and angular acceleration of the bicycle relative to the vertical plane; $\theta, \dot{\theta}$ are the angle and angular velocity of the handlebars and ψ_g is the angle formed by the bicycle and a specified goal g. These states are sent to the controller at each time step and the controller returns the values of d, the torque T, and the force F applied to the pedal.

How the DDPG algorithm trains the controller is summarized in Figure 8. Let $Q(s,a|\theta^Q)$ and $Q'(s,a|\theta^{Q'})$ be the main network and target network of the critic, respectively, $\mu(s,a|\theta^\mu)$ and $\mu'(s,a|\theta^{\mu'})$ be the main network and the target network of the actor, respectively, and R be the experience replay. The learning process is described as follows:

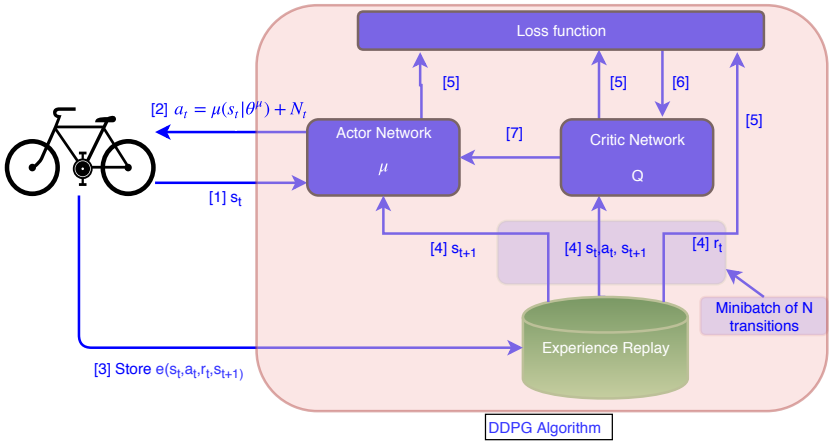


Figure 8. Workflow of the deep deterministic policy gradient (DDPG) algorithm applied to the bicycle.

(1) The agent observes state s_t from the bicycle and feeds it to the actor network (step [1] in Figure 8) for estimating the next action a_t (step [2] in Figure 8) as follows:

$$a_t = \mu\left(s_t|\theta_\mu\right) + N_t, \tag{19}$$

where N_t is small random noise for exploring the action space.

(2) The bicycle transits to next state s_{t+1} and returns reward r_t to the agent.

(3) The sampled data (s_t, a_t, r_t, s_{t+1}) is then stored the experience replay for later use (step [3] in Figure 8).

(4) From the experience replay memory, we randomly select a batch of N samples and use them to train the networks (step [4] in Figure 8).

(5) Train the critic network (step [5] and step [6] in Figure 8) by minimizing the loss function as Equation (14).

(6) Train the actor network (step [7] in Figure 8) using the deterministic policy gradient as Equation (17).

(7) Thereafter, the parameters of the target networks ($\theta^{\mu'}$ and $\theta^{Q'}$) are updated using soft-update techniques as Equations (16) and (18).

4.4. Reward Function

The reward function is defined as follows:

$$r(s,a) = \begin{cases} \text{the last reward before falling down} & |\omega| > \frac{\pi}{6} \\ -(\omega^2 + 0.1\dot{\omega}^2 + 0.01\ddot{\omega}^2) - 2.0\psi_g^2 & |\omega| < \frac{\pi}{6}, \end{cases} \tag{20}$$

where the term $-(\omega^2 + 0.1\dot{\omega}^2 + 0.01\ddot{\omega}^2)$ takes responsibility for balancing the bicycle and the term $-2\psi_g^2$ is for leading the bicycle to the goal. In this reward function, the bicycle is considered as falling down if the angle between the bicycle and the vertical plane is greater than $\frac{\pi}{6}$ rad (or 30 degree). When the bicycle falls down, the reward at this time is used until the end of the episode. The coefficients for each term are selected based on their contributions to the reward. Particularly, we use a coefficient of 1.0 for ω^2, which is the most important in the balancing term. A coefficient of 2.0 is used for ψ_g to highlight the importance of the go-to-goal term. Figure 9 shows the effects of the components on the reward value. Initially, the term $-2\psi_g^2$ has a small value compared to $-\omega^2$, $-0.1\dot{\omega}^2$ and $-0.01\ddot{\omega}^2$. This indicates that this term is the most important to the reward function. However, during 5000 training episodes, the gap between the $-2\psi_g^2$ term and the other terms is decreases and all terms are tend to zero.

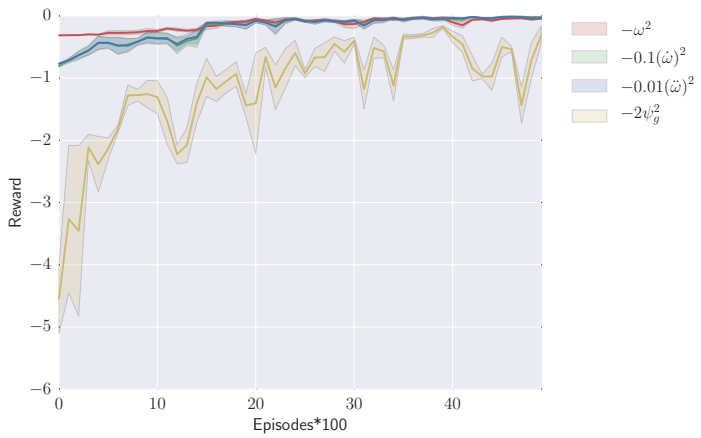


Figure 9. Contributions of the components to the reward function.

5. Experiments

5.1. Settings

The simmlation environment is as follows. The operating system is Linux (Ubuntu 16.04 LTS) with 64 GB DDR3 memory. We use PyCharm as an integrated development environment and experiment with Python as a language. Parameters of the algorithm are shown in Table 3, where we use the Ornstein-Uhlenbeck process [19] to explore the action space. The experience replay can contain up to 500,000 data samples. At each training step, we randomly obtain a batch of 64 samples from the experience replay and use them to train the controller.

Table 3. Parameters of algorithm.

Name	Value
Input dimension	6 (states)
Output dimension	3 (actions)
Discounted factor	0.99
Random noise	Ornstein-Uhlenbeck process [19] with $\theta = 0.15$ and $\sigma = 0.2$
Experience memory capacity	500,000
Batch size	64 samples

5.2. Baselines

NAF (Normalized Advantage Function) [20] uses a Q neural network for the entire problem. To adapt to the continuous control tasks, the Q network is decomposed into a state value term V and an advantage term A:

$$Q(s,a|\theta^Q) = A(s,a|\theta^A) + V(s|\theta^V)$$

The advantage A is parameterized as a quadratic function of nonlinear features of the state:

$$A(s,a|\theta^A) = -\frac{1}{2}(a-\mu(s|\theta^\mu))^T P(s|\theta^P)(a-\mu(s|\theta^\mu)).$$

$P(s|\theta^P)$ is a square matrix with formula:

$$P(s|\theta^P) = L(s|\theta^P)L(s|\theta^P)^T$$

where $L(s|\theta^P)$ is a lower-triangular matrix with entries come from a linear output layer of a neural network. This representation of the Q-network can deal with continuous action tasks.

PPO (Proximal Policy Optimization) [21] is an on-policy algorithm. This means that the policy is learned from the trajectories that are generated from current policy instead of the trajectories from the replay memory. PPO gets rid of the computation created by constrained optimization. PPO implements the idea of TRPO's constraint [22], which does not allow the policy to change too much but instead uses a simpler form of equation. The features of this algorithm can be summarized as follows. First, denote the probability ratio between the old and new policies as

$$r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}.$$

PPO imposes the constraint by forcing $r(\theta)$ to stay within a small interval around 1, precisely $[1-\epsilon, 1+\epsilon]$, where ϵ is a hyperparameter.

$$J^{CLIP}(\theta) = \mathbb{E}\Big[min(r(\theta)\hat{A}_{\theta_{old}}(s,a), clip(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\theta_{old}}(s,a))\Big]$$

The function clip $(r(\theta), 1-\epsilon, 1+\epsilon)$ clips the ratio within $[1-\epsilon, 1+\epsilon]$. The objective function of PPO uses the minimum value between the original value and the clipped version, and therefore we lose the motivation for increasing the policy update to extremes to obtain better rewards.

When applying PPO to the network architecture with shared parameters for both the policy (actor) and value (critic) functions, in addition to the clipped reward, the objective function is augmented with an error term in the estimation (the second term in the formula) and an entropy term (the third term in the formula) to encourage sufficient exploration.

$$J^{CLIP'}(\theta) = \mathbb{E}[J^{CLIP}(\theta) - c_1(V_\theta(s) - V_{target})^2 + c_2 H(s, \pi_\theta(.))]$$

Here, both c_1 and c_2 are hyperparameter constants.

5.3. Results and Discussion

In this section, we compare the performance of a controller-based DDPG with other baselines. In addition, we show the performance of the bicycle before and after considering the velocity. The results are shown below.

5.3.1. Simulation without Controlling the Velocity

Comparison with baselines. In the first evaluation, we compare the performance of a bicycle trained by DDPG algorithms with the performance of the bicycle trained by other algorithms.

Particularly, we compare DDPG with the NAF algorithm (Normalized Advantage Function algorithm) and PPO algorithm (Proximal Policy Optimization algorithm). Both algorithms are state-of-the-art deep reinforcement learning algorithms that can deal with a highly continuous action space. Both algorithms produce deterministic policies that are expected to have lower variance and predictable performance compared to a stochastic policy. Figure 10 shows the performance of a bicycle that randomly starts at $(50, 50)$ m and wants to reach a goal at $(60, 65)$ m. The speed of the bicycle is fixed at 10 km/h and the displacement d is in the range from -20 cm to 20 cm. We report the performance throughout three runs of 5000 episodes. Each episode has 400 time steps.

Figure 10. Performance comparison of DDPG with baselines algorithms.

From the figure, we can see that the controller trained by DDPG outperforms the controller trained by other algorithms in term of variance and reward values. At the beginning of the learning process, DDPG seems to has a bigger variance than other algorithms. However, at the end of the learning process, the variance of DDPG algorithm is decreased while the variance of PPO algorithm is increased and bigger than other algorithms. The algorithm PPO (under development) is expected to gradually train a stable controller but not better than the DDPG algorithm on bicycle domain. The difficulty in tuning PPO's hyperparameters might be the reason for the reported results. In addition, PPO takes a long time to obtain the same performance as DDPG. Even though NAF can learn something on bicycle controller, it cannot obtain a good controller for stabilizing the bicycle. Figure 11 shows values of 6-dimensional states of a successful trajectory. The figure shows that all of states are stable and gradually converges to zero.

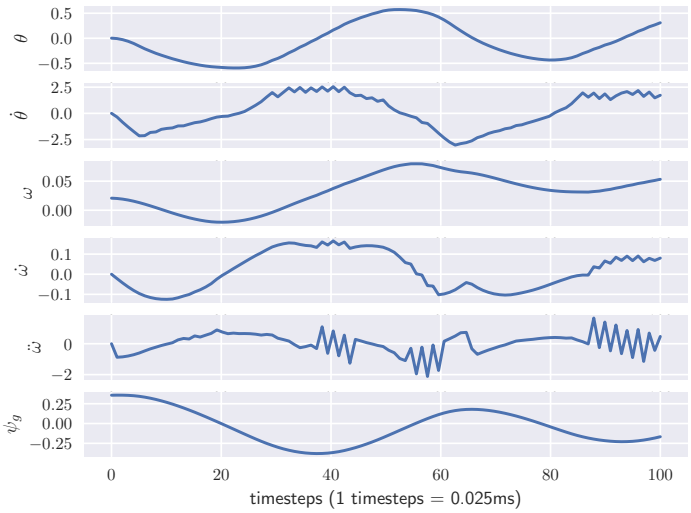
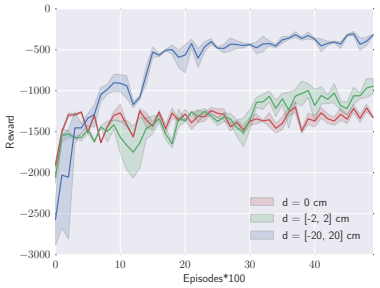
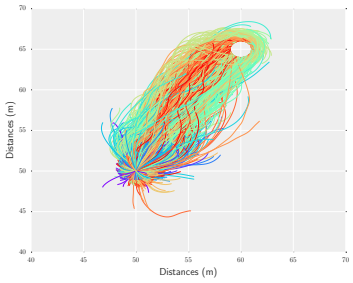


Figure 11. Observed states of a successful trajectory.

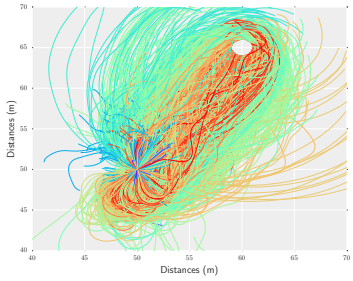
Fixed start-fixed goal. The second evaluation shows the performance of a bicycle with different values of d. The bicycle randomly starts at $(50, 50)$ m with a random direction and wants to reach a goal at $(60, 65)$ m. The speed is fixed at 10 km/h. We train for 5000 episodes of 400 time steps. We learn for 5000 episodes of 400 time steps. The result reported in Figure 12a shows that an agent using a big displacement to adjust the center of mass will outperform an agent using a small displacement and an agent without displacement (only steering the handlebar). Intuitively, without considering d, the bicycle will easily fall down when it tries to turn the bicycle at a high speed. Figure 12 shows the trajectories of the back wheel of the bicycle during the learning process. During this time, the bicycle gradually reaches the goal (blue lines are early trajectories and red lines are late trajectories). However, the bicycle that starts at the opposite position of the goal location often falls down.



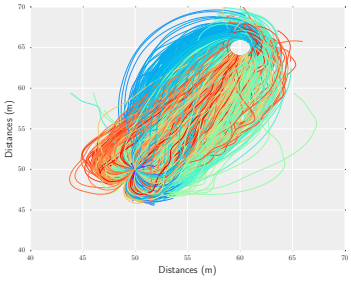
(**a**) Performance for different values of d.



(**b**) d = 0.0 cm.



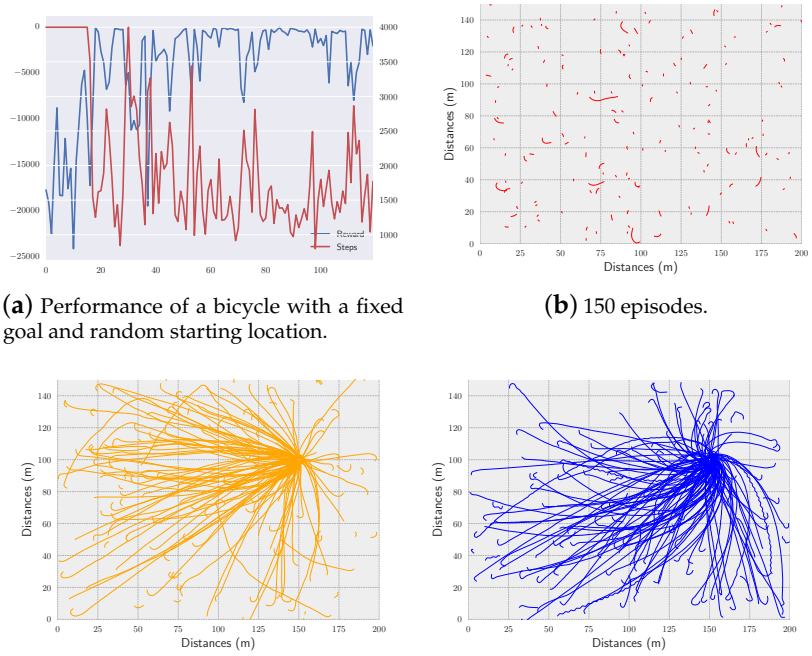
(**c**) d is in range -2 cm and 2 cm.



(**d**) d is in range -20 cm and 20 cm.

Figure 12. Trajectory of a bicycle starting at $(50, 50)$ m and reaching the goal at $(60, 65)$ m.

Random start-fixed goal. The next evaluation is performed for a bicycle that starts at a random location and learns to reach a goal position at $(150, 100)$ m. After around 700 episodes, the bicycle almost reaches the goal from any staring location. The reward converges to zero and the average number of steps to reach decreases from the initial steps (4000) to around 1000 (Figure 13a). Figure 13 shows the trajectories of the bicycle during the learning process.



(**a**) Performance of a bicycle with a fixed goal and random starting location.

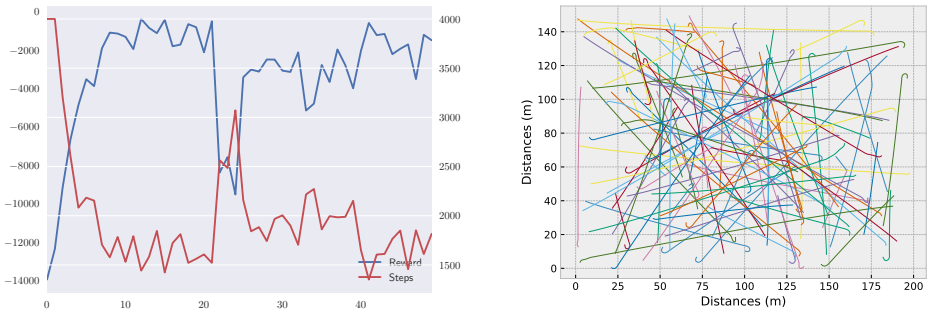
(**b**) 150 episodes.

(**c**) 300 episodes.

(**d**) 600 episodes.

Figure 13. Bicycle trajectories during learning.

Random start-random goal. In the last evaluation of this section, we report the performance of a bicycle controller that is trained to start at random locations and reach a random goal. Figure 14a shows the same behaviors as the previous evaluations. This means the average cumulative reward converges to zero and the average number of steps decreases. Figure 14b shows 100 trajectories for a trained bicycle that starts at random locations and reaches pre-defined random goals. In all of the cases, the bicycle reaches the goal.



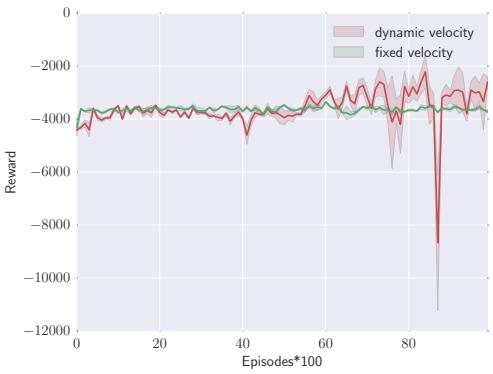
(**a**) Performance of a bicycle with random goals and random starting locations

(**b**) Trajectories of a trained bicycle

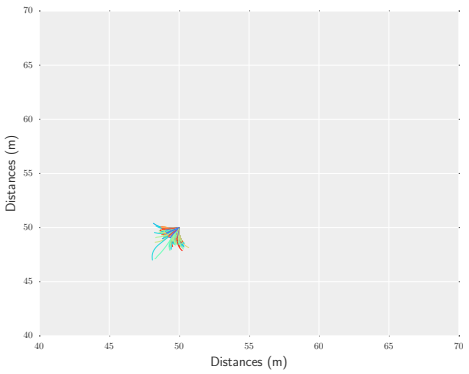
Figure 14. Bicycle with random goal locations and random starting locations.

5.3.2. Simulation with Controlling the Velocity

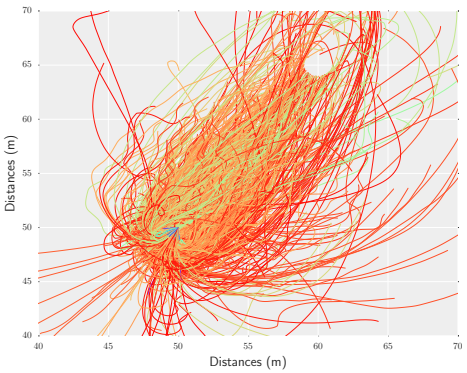
Learning to turn. Understanding the force applied to the pedal is necessary to adjust the velocity of the bicycle. This evaluation shows the efficiency of the controller that learns the force applied to pedal simultaneously. The bicycle in this evaluation only uses a small displacement (from -2 cm to 2 cm) and randomly starts at a position that is in the opposite direction of the goal at $(60, 65)$ m. We compare the performances between a controller that learns the velocity and a controller that does not. The performance is reported through 10,000 episodes of 500 steps each. The initial velocity is 2.5 m/s. Figure 15a shows the average reward, while Figure 15b,c show the trajectories of the bicycle during the learning process. From the figure, we can see that learning a pedal's force helps the bicycle turn the wheel when the bicycle is facing the opposite direction from the goal. The way a bicycle turns the wheel by adjusting the pedal's force is shown in Figure 15d. If the bicycle is facing the opposite direction from the goal's location, the bicycle will adjust the pedal's force to reduce the speed. Under this low speed, the bicycle can easily turn. After the bicycle turns to face direction of the goal's location, the bicycle increases the speed to get to the goal as soon as possible. In the case the bicycle already faces the direction as the goal's location, the bicycle simply gradually increases the speed and heads toward to the goal.



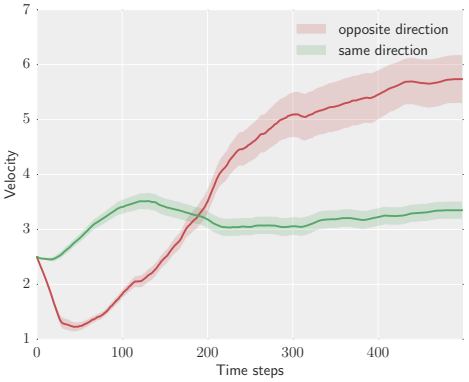
(a) Comparison between a bicycle with and without learning the pedal's force



(b) Trajetories of a bicycle without learning the pedal's force



(c) Trajetories of a bicycle after learning the pedal's force

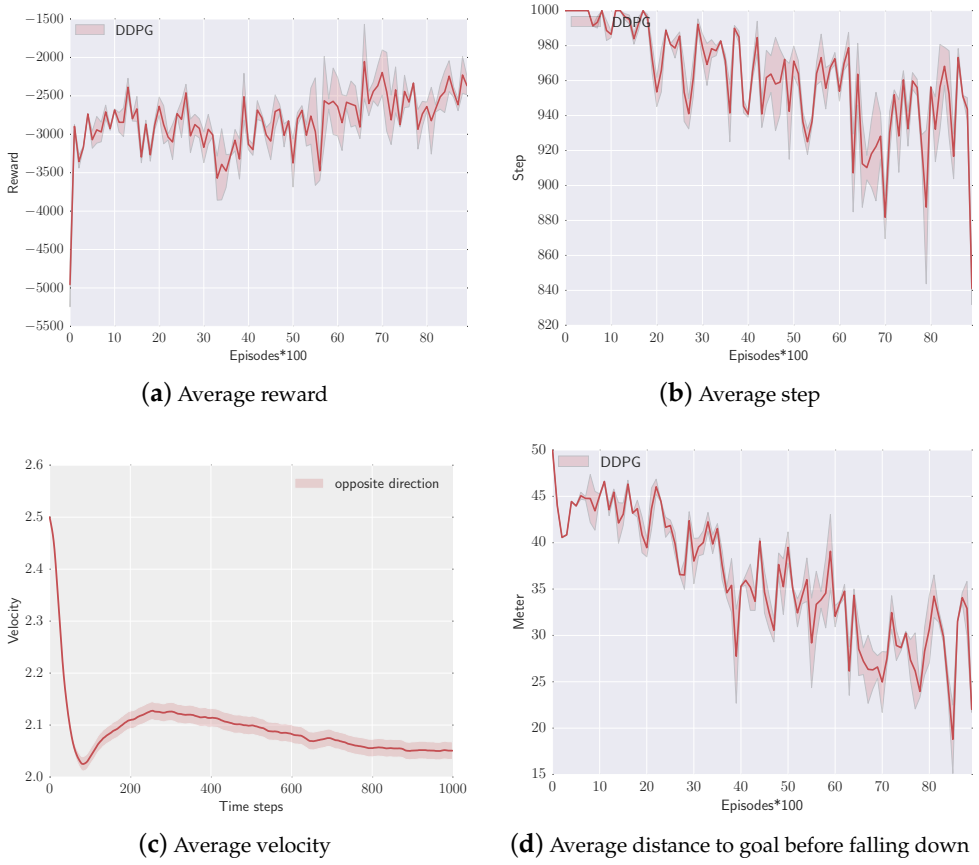


(d) The speed of a bicycle during episodes.

Figure 15. A bicycle learning to turn.

Random start-random goal-learned velocity. In the final evaluation, we increase the complexity of the domain's state space by allowing the bicycle to start at any location with any direction and requiring it reach to a random location. The speed of the bicycle ranges from 1 m/s to 5 m/s and is adjusted by the force applied to the pedal, which ranges from -100 N to 100 N. The initial speed of the bicycle is 2 m/s. We average the performance of the controller after three runs of 10,000 episodes.

The report shown in Figure 16 indicates that the controller is gradually improved by increasing the average reward and decreasing average step and distance to the goal. The speed of the bicycle is decreased to 2 m/s in the first 100 steps and then increases a little before being stabilizing at around 2 m/s. Intuitively, the trained controller decreases the velocity to help the bicycle become stable when turning time. However, due to the curve of dimensionality, the controller needs to explore more of the state space and action space, requiring training via a millions of episodes to obtain an optimal controller.



(**a**) Average reward (**b**) Average step

(**c**) Average velocity (**d**) Average distance to goal before falling down

Figure 16. The performance of a bicycle starting at any place to reach a random location.

6. Conclusions and Future Works

In this paper, we propose a method to control a bicycle using the DDPG algorithm and show that it can be successfully controlled. A controller with a deep neural network can rotate the bicycle handlebars, move the center of gravity displacement, and adjust the speed so that the bicycle can change directions without collapsing. The agent using the proposed neural network controller in this paper was able to reach a specified position and generate a gentle and smooth trajectory. For future work, first, we can enhance the controller by forcing the bicycle to follow a pre-defined trajectory such as a bicycle running on the road. In addition, the DDPG algorithm requires picking a step size that falls into the right range. If it is too small, the training progress will be extremely slow. If it is too large, training tends to be overwhelmed by noise, leading to poor performance. The DDPG algorithm does not assure monotonically improved performance of the controller. Therefore, the second future work is using PPO with some modifications to obtain a more stable controller. Finally, making a real autonomous bicycle needs to consider many aspects such as the effect of a cyclist on the handlebar (T), the effect of the cyclist's foot on the force to pedal, or the hardware needed to build the bicycle. However, most of them have been ignored in this study for simplifying the learning problem. A study on these aspects will be valuable to make a full understanding of an autonomous bicycle.

Author Contributions: Conceptualization, S.L. and T.C.; methodology, S.C. and T.P.L.; software, S.C. and T.P.L.; validation, Q.D.N. and M.A.L.; formal analysis, S.L. and T.C.; investigation, S.L. and T.C.; resources, T.P.L.

Funding: The authors are grateful to the Basic Science Research Program through the National Research Foundation of Korea (NRF-2017R1D1A1B04036354).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Nederland, G. Introducing the self-driving bicycle in The Netherlands. Available online: https://www.youtube.com/watch?v=LSZPNwZex9s (accessed on 10 December 2018).
2. Keo, L.; Yamakita, M. Controlling balancer and steering for bicycle stabilization. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, St. Louis, MO, USA, 10–15 October 2009; pp. 4541–4546.
3. Meijaard, J.P.; Papadopoulos, J.M.; Ruina, A.; Schwab, A.L. Linearized dynamics equations for the balance and steer of a bicycle: a benchmark and review. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*; The Royal Society: London, UK, 2007; Volume 463, pp. 1955–1982.
4. Schwab, A.; Meijaard, J.; Kooijman, J. Some recent developments in bicycle dynamics. In *Proceedings of the 12th World Congress in Mechanism and Machine Science*; Russian Academy of Sciences: Moscow, Russia, 2007.
5. Tan, J.; Gu, Y.; Liu, C.K.; Turk, G. Learning bicycle stunts. *ACM Trans. Gr. (TOG)* **2014**, *33*, 50. [CrossRef]
6. Lu, M.; Li, X. Deep reinforcement learning policy in Hex game system. In Proceedings of the IEEE Chinese Control And Decision Conference (CCDC), Shenyang, China, 9–11 June 2018; pp. 6623–6626.
7. Bejar, E.; Moran, A. Deep reinforcement learning based neuro-control for a two-dimensional magnetic positioning system. In Proceedings of the IEEE 4th International Conference on Control, Automation and Robotics (ICCAR), Auckland, New Zealand, 20–23 April 2018; pp. 268–273.
8. Yasuda, T.; Ohkura, K. Collective behavior acquisition of real robotic swarms using deep reinforcement learning. In Proceedings of the Second IEEE International Conference on Robotic Computing (IRC), Laguna Hills, CA, USA, 31 January–2 February 2018; pp. 179–180.
9. Randløv, J.; Alstrøm, P. Learning to drive a bicycle using reinforcement learning and shaping. *ICML* **1998**, *98*, 463–471.
10. Le, T.P.; Chung, T.C. Controlling bicycle using deep deterministic policy gradient algorithm. In Proceedings of the 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), Jeju, Korea, 28 June–1 July 2017; pp. 413–417.
11. Sutton, R.; Barto, A. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 1998; Volume 1.
12. Peters, J.; Schaal, S. Reinforcement learning of motor skills with policy gradients. *Neural Netw.* **2008**, *21*, 682–697. [CrossRef] [PubMed]
13. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:1509.02971.
14. Le, T.P.; Quang, N.D.; Choi, S.; Chung, T. Learning a self-driving bicycle using deep deterministic policy Gradient. In Proceedings of the 18th International Conference on Control, Automation and Systems (ICCAS), Pyeongchang, Korea, 17–20 October 2018; pp. 231–236.
15. Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; Riedmiller, M. Deterministic policy gradient algorithms. In Proceedings of the 31st International Conference on Machine Learning, Beijing, China, 21–26 June 2014; pp. 387–395.
16. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529. [CrossRef] [PubMed]
17. Hwang, C.L.; Wu, H.M.; Shih, C.L. Fuzzy sliding-mode underactuated control for autonomous dynamic balance of an electrical bicycle. *IEEE Trans. Control Syst. Technol.* **2009**, *17*, 658–670. [CrossRef]
18. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
19. Uhlenbeck, G.E.; Ornstein, L.S. On the theory of the Brownian motion. *Phys. Rev.* **1930**, *36*, 823. [CrossRef]

20. Gu, S.; Lillicrap, T.; Sutskever, I.; Levine, S. Continuous deep q-learning with model-based acceleration. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 2829–2838.
21. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
22. Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; Moritz, P. Trust region policy optimization. In Proceedings of the International Conference on Machine Learning, Lille, France, 6–11 July 2015; pp. 1889–1897.