Proceedings of The 2018 IEEE International
Conference on Real-time Computing and Robotics
August 1-5, 2018, Kandima, Maldives

Towards High Level Skill Learning: Learn to Return Table Tennis Ball Using Monte-Carlo Based Policy Gradient Method

Yifeng Zhu[1], Yongsheng Zhao[2], Lisen Jin[1], Jun Wu[1], and Rong Xiong[1]

***Abstract*— Deep learning has achieved a great success in both visual and acoustic recognition and classification tasks. The accuracy of many state-of-the-art methods have surpassed that of human beings. However, in the field of robotics, it remains to be a big challenge for a real robot to master a high-level skill using deep learning methods, even though human can easily learn the task from demonstration, imitation and practice. Compared to Go and Atari games, this kind of tasks is usually continuous in both state space and action space, which makes value based reinforcement learning methods unavailable. Making a robot learn to return a ball to a desired point in table tennis is such a typical task. It would be a promising step if a robot can learn to play table tennis without the exact knowledge of the models in this sport just as human players do. In this paper, we consider such a kind of motion decision skill learning, a one-step decision making process, and give a Monte-Carlo based reinforcement learning method in the framework of Deep Deterministic Policy Gradient. Then we apply this method in robotic table tennis and test it on two tasks. The first one is to return balls to a desired point first, and the second one is to return balls to randomly selected landing points. The experimental results demonstrate that the trained policy can successfully return balls of random motion state to both a designated point and randomly selected landing points with high accuracy.**

I. INTRODUCTION

Deep learning methods, especially deep convolutional neural networks and deep recurrent neural networks, have brought great breakthrough in speech recognition, visual object detection, recognition and localization, as well as many other domains such as natural language translation, image segmentation, image understanding, and etc. The explosive increase of the amount of well-labelled data and the parallel computation ability has facilitated the great success of deep learning methods. Many of those state-of-the-art methods achieve higher accuracy than or comparable to that of human in some visual recognition tasks. Besides, reinforcement learning as a promising method in decision making and optimization has achieved remarkable breakthroughs, e.g., the amazing success of AlphaGo [1] and Deep Q-Network playing human-level performance in Atari games [2]. While tasks like mapping from input data to output target, the hierarchical feature detection and representation, and the parameter training involved in the learning process can all be handled by using logical operation on a computer without interacting with the real environment, it remains to be a big challenge for a robot to master a motion decision skill that needs to be learned by interacting with the real world.

In the field of robotics, making a motion decision when interacting with the real environment in real-time is a quiet common but challenging task for a robot. Playing table tennis is such a typical task involving the dynamic motion decision, which is considered as a high-level skill. For such a game it is very important to decide when, where and how to strike a ball with an arbitray initial motion state back to the desired landing point. It is an essential prerequisite for robots to play competitive table tennis.

For such a motion decision problem in playing table tennis, Muelling et al proposed a four-stage striking process, hitting the ball in the virtual hitting plane [3]. They also provided strategies for robots to win in a table tennis game [4] [5]. Nevertheless, it remains a great challenge in the problem of how a robot chooses the velocity and the pose of the racket to strike balls of various incoming motion states and returns them to a target point, due to the complexity of physical models. The complexity of the task lies in its continuous state space and continuous action space, as well as its interaction with high dimensional and nonlinear environments.

Matsushima et al conducted a research on returning the ball to a target point using a slide-way structured robot [6]. However, its learned policy lacks flexibility and is unfit for a standard table tennis game. Therefore, we propose a novel learning framework in this paper for a humanoid robot to strike the ball to a given target point with high accuracy, meeting the requirement of a standard and competitive table tennis game.

The nature of this motion decision is actually a problem of optimization. However, some complex issues involved have made traditional methods of optimization inapplicable. First of all, the flying model of a spinning ball is high dimensional, nonlinear, and coupled. Given a perceived initial start-point and a designated target end-point, there are infinite solutions that satisfy the condition, since the flying velocity and spin velocity of the ball are all in continuous domain. In other words, the robot theoretically has infinite choices of action to strike the ball back to the target point if not considering the constraints of the robot. Therefore, it is impossible to deduce the analytic solution to back-propagate hitting action of the end-effector, given the target landing point. Considering the real-time requirement of the task, it is also difficult to use a traditional optimization

*This work was supported by the National Nature Science Foundation of China (Grant No.61473258)

[1]Yifeng Zhu, Jun Wu, Hongxiang Yu, and Rong Xiong are with the State Key Laboratory of Industrial Control and Technology, Zhejiang University, Hangzhou, 310027, P. R. China. Rong Xiong is the corresponding author. rxiong@zju.edu.cn

[2]Yongsheng Zhao is with the Binhai Industrial Technology Research Institute of Zhejiang University, Tian, 300457, P. R. China.

978-1-5386-6868-9/18/$31.00 © 2018 IEEE

method to search the optimal numerical solutions in a high-dimensional continuous domain. However, human players can learn to play table tennis very well by demonstration, imitation, and practice without knowing the specific motion model. Inspired by this, we seek to a methodology with a good generalization ability that could help the robot learn the high level skill of striking the ball back to a desired landing point within tolerable errors. This leads us to the method of deep reinforcement learning, which is a good alternative for both optimization problem and decision making problem. But we face the following difficulties. First of all, the state space of the environment is often continuous. Secondly, the choice of the motion is often continuous in the domain of definition. Thirdly, physical models of the environment are often complicated and non-linear, which makes this category of problem more complicated.

Recently, reinforcement learning brings great success. D. Silver et al proposed Deep Deterministic Policy Gradient (DDPG) to effectively solve problems in continuous states and actions [7]. It has been successfully tested in playing Atari games, displaying human-parity performance. This success has an instructive meaning in our research. In this work, we are dealing with a one-step process. That is, given the input of a state, we generate an action and then get a result from actuating this action. The difficulty from the aspect of time series vanishes, yet the complication of high-order, nonlinear physical models and vast range of state space and action space rises.

In order to overcome the foregoing difficulties, we seek to a good simulation of models for reinforcement learning. Therefore, we establish a simulation environment of table tennis game using several well-established models to make our simulated result convincing. Based on this simulation environment, we then apply a Monte Carlo based policy gradient method and deliberately define the reward in this task to deal with this one-step Markov Decision Process in the framework of our humanoid robotic system, successfully bringing about our desired result.

The rest of the paper is organized as follows. Section II is the related work about existing reinforcement learning theories and methods, as well as reinforcement learning tasks in robotic table tennis. Section III presents details of our solution including the mathematical formulation and algorithms. Section IV briefly introduces the simulation environment we have established. At Last, Section V gives results of our experiment and section VI summarizes our work and gives a prospect on future works.

II. RELATED WORK

Reinforcement learning has long been a promising research field in decision making for robots and artificial intelligence. In the past, discretized methods such as Q-learning, SARSA have performed well in many problems such as maze or cart-and-pole [8]. Since original versions of these approaches deal with discrete state and actions, the performance of these methods is limited, as well as their applications. The method of actor-critic is also proposed, dealing with the limitation of actor-only (learning policy only) or critic-only (learning value functions only) methods. Reinforcement Learning in continuous state space was once proposed by Kenji [9], utilizing the continuous method of dynamic programming– Hamilton-Jacobi-Bellman Equations (HJB) which gives an example of solving this complicated task. Several other successful actor-critic methods have been brought forth like natural actor-critic [10], A3C method [11].

In recent years, many new approaches of reinforcement learning and its successful applications have been demonstrated, such as game of Go, Atari games, and classical control problems in OpenAI environment [1] [2]. A leap in the problem of continuous state and action is the introduction of deterministic policy gradient [12], which is the expected policy gradient of the action value functions. This method generally outperforms the stochastic policy gradient in high-dimensional continuous state and action space. With this fundamental theory, the deep deterministic policy gradient (DDPG), which is the deterministic policy gradient using deep neural networks as function approximators. This method also includes experience replay, which can make the convergence faster, and what is more, enable the agent to learn from experience, effectively utilizing data in the past.

Reinforcement learning with neural networks have many successful real-world applications. Levine et al used the method of end-to-end learning to make the robot learn to screw a cap on a bottle [13]. Levine et al also used an asynchronous reinforcement learning method on robot arms to learn to perform a door opening skill from scratch [14].

Reward setting is also a crucial element in the problem of reinforcement learning. The setting of reward is also a key to the efficiency of algorithms. Andrew Ng et al proved that modifying the reward with potential function preserves the property of convergence of the original problem while speeding up the convergence [15].

Learning methods in the field of robotic table tennis has been explored before. Muelling et al used a biomimetic approach to create a human-alike stroke movement, proposing discrete movement stages hypothesis and virtual hitting point hypothesis [3]. It is a general trajectory generator in robotic table tennis. The movement learning has been explored by J.Peters et al, based on DMP (Dynamic Motor Primitives). J.Kober et al represented elementary movement with meta-parameters, and they made the robot to learn mappings from circumstances to meta-parameters using reinforcement learning [16]. Muelling et al also proposed MoMP (Mixture of Motor Primitives) [17] to learn motor actions. It is a method which can choose appropriate motor primitives while generalized motor among them. These works focus more on the motion planning of a robot arm in Robotic Table Tennis.

III. FRAMEWORK AND LEARNING METHOD DETAILS

The main framework of learning to play table tennis is shown in figure 1. The environment consists of several models involved in this task. Pitching Server emits a ball of random states. Motion Model is composed of a trajectory prediction model and a rebound model between ball and

table. Given an initial motion state, this model gives out the state of the ball at the hitting plane or at the landing point. Taking the motion state of ball at the hitting plane as input, the actor generates an action. With the motion state of ball and the action, the Rebound Model of the racket can be simulated, and the ball flies from the robot to the other side. At last, the landing point of the ball can be obtained using the trajectory model again. The reward is given after one episode, and the information of one trial can be stored in a replay buffer from which the critic samples and trains itself as well as the actor.

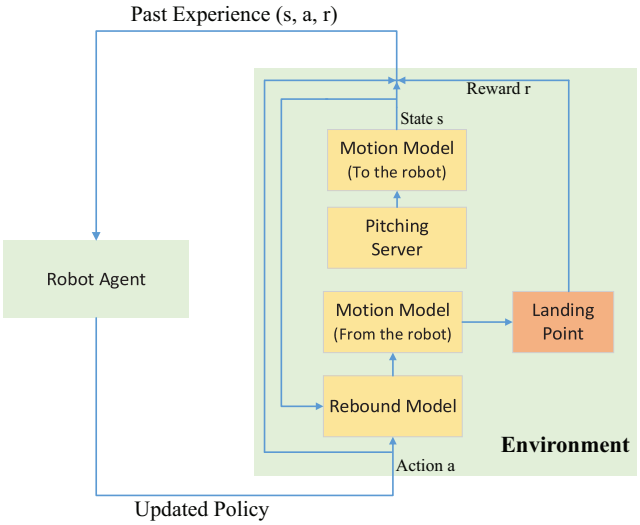


Fig. 1. Main framework of learning to play table tennis

A. Mathematical formulation

In order to solve the problem of one-step decision making that interests us, the method of reinforcement learning we use is modified from the work of DDPG [7]. In DDPG, the target value for critic to learn is defined as: $y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1}|\theta^Q))$

We denote the critic and actor as $Q = Q(s, a|\theta^Q)$ and $\mu = \mu(s|\theta^\mu)$ respectively. θ^Q are weight parameters in critic network and θ^μ are weight parameters in actor network. s_t is the state a robot needs to interact with, and a_t is the action it chooses to respond to the state. rs_t, a_t is the reward an agent or a robot receives after it takes action.

Since our case is one-step MDP problem, target value y decays into:

$$y_t = r(s_t, a_t) \tag{1}$$

The reinforcement learning problem is to optimize two objective functions for critic and actor respectively:

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

$$\begin{aligned}\max_{\theta^\mu} J &= \int_S \rho^\mu(s) r(s, \mu_\theta(s)) \mathrm{d}s \\ &= E_{S\sim\rho^\mu}[r(s, \mu_\theta(s))]\end{aligned} \tag{3}$$

Given large data, we use Adam Optimizer for optimization. Gradients in (2) can be easily calculated and the parameters can henceforth be changed, yet (3) is a little bit complicated. According to the theorem proposed in [12], the gradient can be obtained as follows:

$$\bigtriangledown_{\theta^\mu} J = \frac{1}{N} \bigtriangledown_a Q(s, a, |\theta^Q) \bigtriangledown_{\theta^\mu} \mu(s, a|\theta^\mu) \tag{4}$$

Using this sampled gradient, we can show the actor of agent the direction it should follows for better update locally.

B. Representation of action-value function and policy

Both the state-action-value function (critic) and policy of action (action) are represented by a nonlinear neural network. The structures of neural networks and initial parameters mostly use the ones mentioned in [7].

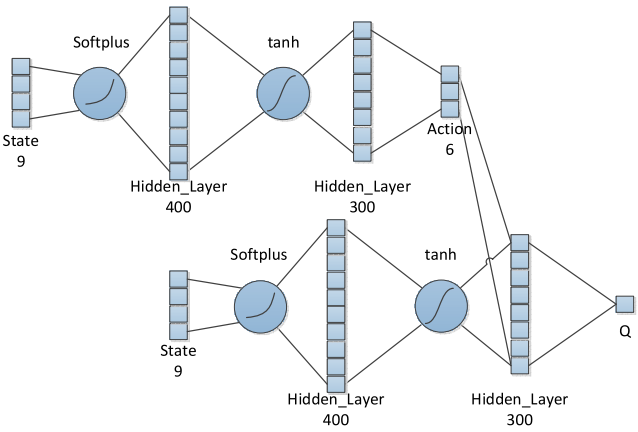


Fig. 2. Actor (top) and Critic (bottom) neural network structures

In figure 2, we vividly present the structures of our actor and critic and relation between them. The numbers indicated below blocks are the numbers of each components respectively.

C. Algorithm

Here we shall give the algorithm for solving this kind of reinforcement learning problem. Note that this is a variant of Deep Deterministic Policy Gradient. Since our method is Monte Carlo based rather than Temporal Difference based, our target value for critic is the reward itself and we do not have to use another networks of critic and actor to store the settings of previous version. See the Algorithm 1 for more details of this Monte Carlo based problem.

So far, we have briefly introduced the way of solving this one-step, high-level skill problem. Now we focus on a specific problem, which is robotic table tennis.

D. Application in robotic table tennis

In this section, we are going to present the application of our proposed Monte-Carlo based method into robotics table tennis. This part is organized as follows. First of all, we define the convention of coordinates so that we can discuss the problem by a uniform standard. Then, we define state and

Algorithm 1 Monte-Carlo-Based DDPG

Randomly initialize critic network Q and actor network μ
Initialize replay buffer R
Set the number of episodes for training M
for $k = 1 \rightarrow M$ **do**
    Initialize noise N for action exploration
    Receive observation states s_k
    Generate action $a_k = \mu(s_k|\theta^\mu)$ from actor network
    Add exploration noise to the generated action
    Execute action a_k, observe reward r_k and landing point p_{target}
    Store the trial (s_k, a_k, r_k)
    Sample a random minibatch of N trials (s_i, a_i, r_i)
    Set $y_i = r_i$
    Update critic network by minimizing the loss:

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

    Update actor network using sampled policy gradient:

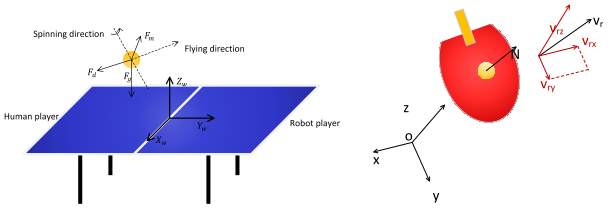
$$\bigtriangledown_{\theta^\mu} J = \frac{1}{N} \bigtriangledown_a Q(s, a, |\theta^Q) \bigtriangledown_{\theta^\mu} \mu(s, a|\theta^\mu)$$

end for

actions for one-designated-landing-point task and randomly-chosen-landing-points task respectively. At last, we will talk about the reward setting in these specific table tennis cases.

1) Convention of Coordinates: The origin of our world coordinate is set at the center of the whole table. The direction perpendicular to the plane of the table upwards is defined as positive direction of z-axis. The direction from the center to the robot side is defined as positive direction of y-axis. And x-axis is perpendicular to the y-z plane and its positive direction is defined according to Right-Hand Rule.

The origin of our racket coordinate is set at the center of the racket. The direction of the normal vector of the racket plane towards the side of opponent is defined as positive direction of z-axis. The direction from the bar of the racket to the center of racket is defined as positive direction of y-axis. Positive direction of x-axis can be obtained by Right-Hand Rule, which is the same way in world coordinate.



(a) World Coordinate (b) Racket Coordinate

Fig. 3. Convention of Coordinates

The racket coordinate is established for calculation of racket rebound model. Except this, variables and values are all described in world coordinate. Positions on the half-side of robot's has positive y value, while negative on the half-side of opponent's. For example, the hitting plane of the robot is set at $y = 1.199m$, while the desired landing point in the first task is set $x = 0.0m, y = –0.87m$.

2) One designated landing point: First of all, we apply this framework to a primitive task. That is, return a ball to a single target landing point. We have defined the designated landing point, which is defined at $x = 0.0m, y = –0.87m$.

a) State: The state of a ball has 9 components. Three for ball's position at the hitting plane, three for flying velocity, and three for spin velocity, which are denoted as $\vec{p}_{in}$, $\vec{v}_{in}$ and $\vec{\omega}_{in}$ respectively. We use s to denote the state of a incoming ball at the hitting plane and it is denoted as $s = \begin{bmatrix} \vec{p}_{in} & \vec{v}_{in} & \vec{\omega}_{in} \end{bmatrix}^\intercal$.

As previously described, we use the physical status of a ball as input to our task. Many reinforcement learning works used images as the input for end-to-end learning, yet our mission requires the accurate estimation of the ball's spin state, which is difficult to get from the image sequence directly without any prior knowledge. Therefore, compared to the physical description of a ball at the hitting plane, it is inefficient to take images as input.

The components of three physical variables of a ball is described as follows: $\vec{p}_{in} = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^\intercal$, $\vec{v}_{in} = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^\intercal$, $\vec{\omega}_{in} = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^\intercal$. Given the feasible region for the robot and the limitations of ball's motion, the boundary of states are given in Table I. Note that balls from the pitching machine have a low ω_y and this component of spin velocity has little impact in models. Therefore, we set ω_y as a constant to simulate the situation that ω_y is not vanished yet has little influence on the simulation.

TABLE I
STATE OF INCOMING BALL AT THE HITTING PLANE

Component	Range of value
p_x	[-0.5 m , 0.5 m]
p_y	1.199m
p_z	[-0.5m, 0.5m]
v_x	[-0.5 m/s , 0.5 m/s]
v_y	[-3.0 m/s, 0.0 m/s]
v_z	[-0.5m/s, 0.5m/s]
ω_x	[-100.0rad/s, 100.0rad/s]
ω_y	10.0rad/s
ω_z	[-100.0rad/s,100.0rad/s]

b) Action: The action generated from the policy consists of five components, three for the translational velocity of the racket and two for the pose of the racket. The position of the racket is not considered, since the system is fully capable of predicting the position of the ball at the hitting plane with high accuracy, and it is assured that the racket can reach the position of the ball during the striking stage. The translational velocity of the racket is described by its components in x, y and z direction in the world coordinate, which is $\vec{v}_r = \begin{bmatrix} v_{rx} & v_{ry} & v_{rz} \end{bmatrix}^\intercal$.

The pose of a racket is usually described by a normal vector $\vec{n}_{3\times1} = \begin{bmatrix} n_x & n_y & n_z \end{bmatrix}^\intercal$. The normal vector is or-

thogonal to the surface of the racket in the world coordinate. However, it is redundant to describe the pose of a racket using a vector of three components, since in rebound model, it is not our concern whether the bar of the racket is at top or at bottom. In other words, we do not care about the rotation along the direction of y-axis in the world coordinate. Thus, the pose of the racket when hitting the ball is described by the other two components n_x and n_z, while we set $n_y = -1$.

And the action is denoted as $\vec{a} = \begin{bmatrix} \vec{v}_r & \vec{n}_{3\times1} \end{bmatrix}^\intercal$. Given the fact that the robot has its own limits in actuation, the available domain of action must be constrained. We set the boundary of the action, according to the ability of our robotic system Wu&Kong, as described in Table II.

TABLE II
ACTION GENERATED BY POLICY

Component	Range of value
v_{rx}	[-0.5 m/s , 0.5 m/s]
v_{ry}	[-3.0 m/s, 0.0 m/s]
v_{rz}	[-0.5m/s, 0.5m/s]
n_x	[-0.5, 0.5]
n_y	-1
n_z	[-0.5,0.5]

3) Randomly selected landing points: Based on the method we proposed, we also explore on the problem of learning to return balls to multiple landing points. In order to do this, we take the landing points into account.

a) State: In spite of the nine states of the position, rotation velocities and linear velocities, we add two-dimensional state p_{target} (the height is always the same when it hits the table) into the input to our framework. Now the input of states becomes $s = \begin{bmatrix} \vec{p}_{in} & \vec{v}_{in} & \vec{\omega}_{in} & \vec{p}_{target} \end{bmatrix}^\intercal$. Now, we will train the model using randomly distributed target point value, but we have constrained the target point in a reasonable range which would meet the requirement of competitive table tennis. The constraint is listed in the table.

TABLE III
RANGE OF TARGET POINTS AS INPUT STATES

Component	Range of value
$p_{targetx}$	[-0.8m, 0.8m]
$v_{targety}$	[-1.1m, -0.3m]

b) Action: In this problem, the action is the same as the one landing point task.

4) Reward Setting: The setting of reward is a crucial part in the problem of reinforcement learning. How reward is formulated decides how well the task is learned. In the two tasks mentioned above, our reward is given as follows:

$$R = -k(\|p - p_{target}\| + \|z_{set} - z_{net}\|) \tag{5}$$

In the equation above, k is a scalar coefficient which can be adjusted due to different situations and training of networks. Here we set $k = 5$. p is the actual landing point, p_{target} is the target landing point. The first term in reward is the Euclidean distance between these two points. z_{net} is the actual height when the ball flies across above the net, and z_{set} is the set height, which is 0.27m, higher than the net height of 0.1525m. The second term is the Euclidean distance of these two heights.

The second term is a crucial one. It not only reduces the number of solutions, but also constrains the height of the ball when flying across the table without deliberately checking if the ball is netted every time (Since if the ball is netted, the second term is relatively large, away from the set height and result in low reward. Therefore, the algorithm will seek for other solutions with smaller absolute value of the second term. If the ball is too high, it might hit the roof of the room, which also needs to be circumvented). It is a similar idea from converting constrained problem to the unconstrained problem in traditional optimization, yet suitable for this reinforcement learning task.

IV. SIMULATION ENVIRONMENT FOR LEARNING

In this problem, it is essential to build an environment that is consistent with the real-world situation. There are three essential parts involved, which are the trajectory model of a flying-spinning ball, the rebound model of a spinning ball with table and the rebound model of spinning ball with racket. The following is the brief introduction of our simulation environment on which our results rely.

A. Trajectory model of a flying spinning ball

The trajectory model, or in other words, trajectory prediction model of a flying-spinning ball is based on a continuous motion model (CMM) derived by Zhao et al [18]. With this model, we can effectively take spinning velocity into account without undermining the accuracy of this prediction with real observation.

B. Rebound Model of a spinning ball on table

The rebound model of a spinning ball is established according the work of Zhao et al [19]. The model uses the mean value theorem, momentum theorem and angular momentum theorem. The effectiveness and accuracy of this model was verified in the mentioned work.

C. Rebound model of spinning ball on racket

This model is our recent progress. The model deals with the spin velocity of the ball which makes the problem very complicated. First of all, we convert variables in world coordinate into racket coordinate for decoupling. And then we predict the states of the ball flying back following formulations as follows:

$$\begin{cases} v_z^{out} = -\alpha_z v_z^{in} + \beta_z v_z^{racket} \\ v_x^{out} = v_x^{in} + f_{\mu x}(\vec{v}_{in}, \vec{\omega}_{in}, \vec{v}_{racket})(v_z^{out} - v_z^{in}) \\ v_y^{out} = v_y^{in} + f_{\mu y}(\vec{v}_{in}, \vec{\omega}_{in}, \vec{v}_{racket})(v_z^{out} - v_z^{in}) \\ \omega_x^{out} = \omega_x^{in} + \dfrac{m}{I} f_{\mu lx}(\vec{v}_{in}, \vec{\omega}_{in}, \vec{v}_{racket})(v_z^{out} - v_z^{in}) \\ \omega_y^{out} = \omega_y^{in} + \dfrac{m}{I} f_{\mu ly}(\vec{v}_{in}, \vec{\omega}_{in}, \vec{v}_{racket})(v_z^{out} - v_z^{in}) \\ \omega_z^{out} = e_z \omega_z^{in} \end{cases} \tag{6}$$

Where $f_{\mu x}, f_{\mu y}, f_{\mu lx}, f_{\mu ly}, \alpha_z, \beta_z, e_z$ are coefficients of collision, m is the mass of a ball and I is the inertia of momentum of a ball.

We have briefly presented models that we use in our simulation environment. The simulation environment is visualized as in figure 4. In next section, we would present our results of our solution.

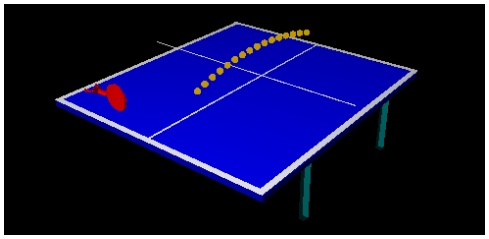


Fig. 4. Simulation environment

V. Experimental Results

A. One designated landing point

Given the fact that the diameter of a table tennis ball is $0.04m$, we consider $\pm 0.05m$ as acceptable errors from the target landing point. The intuitive interpretation is that the actual landing point within the area the same as the surface of a racket is acceptable. Given the actuator error of the robot and the uncertainty within each model, it is reasonable to accept this level of error. We evaluate the performance of a learned policy by simulating 5000 times of various incoming ball and examine the result of landing points.

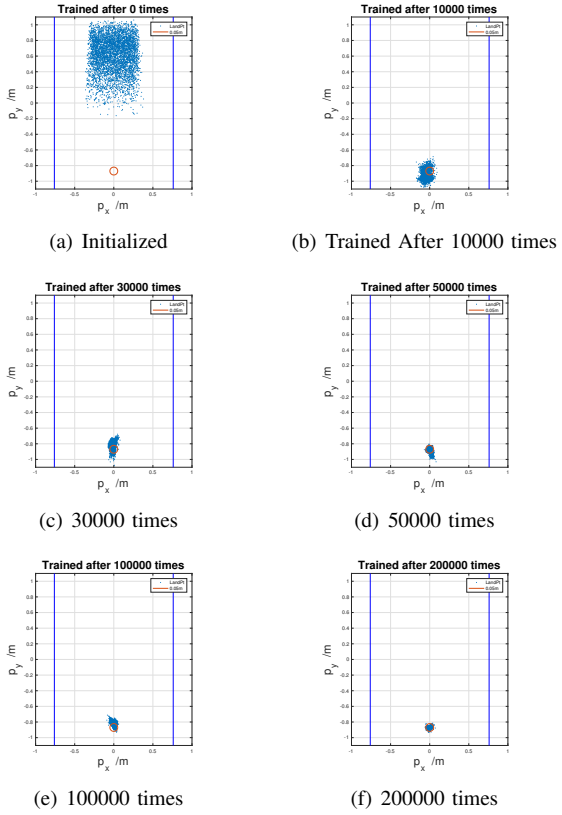
We trained the problem for 20000 episodes (times) and results are shown in figure5(a) to 5(f) and figure 6. Figure 5(a) is the result of randomly initialized policy , and figure 5(b)to 5(f) shows the performance after training 10000 times, 30000 times, 50000 times, 100000 times and 200000 times respectively. At the end of actor-critic training, we obtain a good performance of a policy, as presented in figure 6. To show this result more clearly, we have presented the percentage of landing points within a range of $0.05m$ of the target landing point in table IV, which consists from the intuitive observations of figures. Note that during the training, the performance might be undermined since it is changing from one optimum to another one, which result in the lower performance after 100000 times than after 50000 times.

TABLE IV

Performance of One Designated Landing Point

Numbers of Training	Within $\pm 0.05m$ of target landing point
0	0.00%
10000	14.06%
30000	50.98%
50000	78.92%
100000	67.58%
200000	99.22%

To make sure that our agent learns a converged value function and policy, we set the learning rates of critic and



(a) Initialized (b) Trained After 10000 times

(c) 30000 times (d) 50000 times

(e) 100000 times (f) 200000 times

Fig. 5. Distribution of landing points after numbers of learning

actor 0.001 and 0.0005 respectively, in accordance to the theory of convergence in actor-critic algorithm [20]. What is more In order to learn the policy robustly and converge to local optima, we reduce the mean and variance of exploration noise as the number of episodes increases.

B. Randomly selected landing points

In this subsection, we evaluate the performance of the model on 10 randomly selected landing points.

We have also trained the model for 200000 times in total, and we show the process of learning from 7(a) to 7(f). Among all the figures, 7(a) is the result from randomly initialized weights, 7(b) to 7(f) are results from 10000, 30000, 100000, 200000 times respectively. After the training finished, we get a pretty satisfactory result just as the last figure indicates. For better demonstration of the performance, we place the results in the table V. For each target landing point, we show the percentage of the balls landing within the range of 5cm of the landing point out of 5000 balls.

We also present the percentage of landing points within a range of $0.05m$ of the target landing point in table while set the learning rate the same as in the problem of one designated landing point.

C. Real-time Requirement

The time the actor takes to generate an action takes about $0.3ms$. The traditional nonlinear optimization method takes $1.7s$ in calculating one solution of action using Matlab. We can see that our method has better real-time performance than

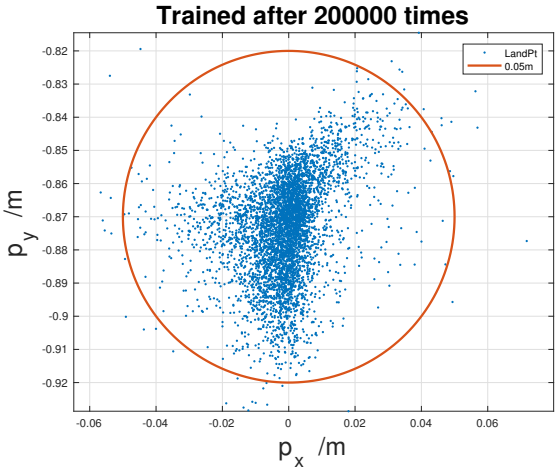


Fig. 6. Result after 200000 times

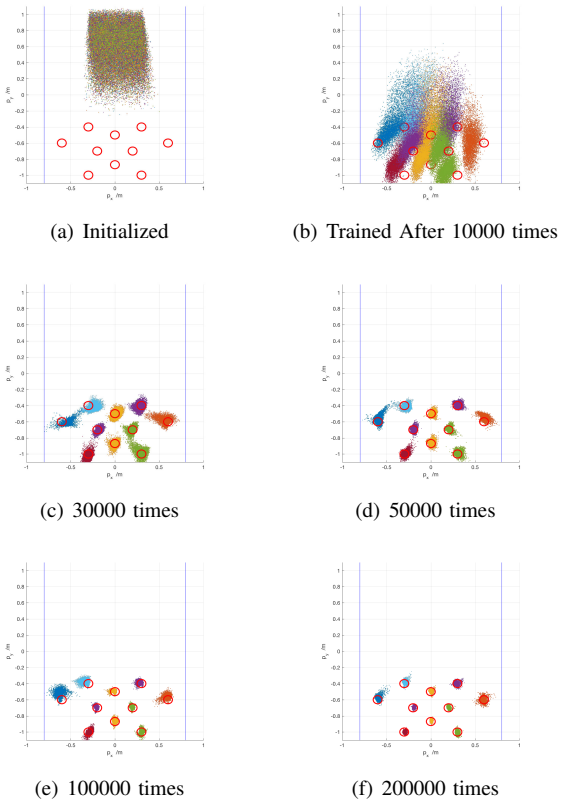
TABLE V
PERFORMANCE OF MULTIPLE LANDING POINTS

	Landing Points (within $\pm 0.05m$)			
Training	**(0.0,-0.87)**	**(0.3,-0.4)**	**(0.3,-1)**	**(-0.3,-0.4)**
0	0.00%	0.00%	0.00%	0.00%
10000	0.33%	3.78%	0.37%	6.87%
30000	78.37%	49.22%	53.00%	17.72%
50000	88.10%	86.67%	72.12%	52.57%
100000	97.97%	91.12%	95.13%	66.43%
300000	99.92%	98.12%	98.70%	96.30%
	(-0.3,-1.0)	**(-0.6,-0.6)**	**(0.6,-0.6)**	**(0.0,-0.5)**
0	0.00%	0.00%	0.00%	0.00%
10000	5.00%	4.90%	0.38%	9.60%
30000	68.77%	17.73%	55.85%	56.23%
50000	77.72%	66.28%	39.93%	86.30%
100000	86.84%	89.53%	62.47%	96.85%
300000	99.93%	93.15%	91.95%	99.48%
	(-0.2,-0.7)	**(0.2,-0.7)**	**Average**	
0	0.00%	0.00%	0.00%	
10000	5.97%	1.90%	3.42%	
30000	70.48%	77.60%	54.49%	
50000	92.65%	87.85%	75.02%	
100000	98.58%	98.37%	82.33%	
300000	100%	99.80%	97.74%	

the traditional optimization method. The program runs in a computer using 24-core Intel X5670 CPU and the system uses Ethernet communication between the robot and the computer. Therefore, our methods can meet the real-time requirement in our existent system.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we consider a category of problem, which concerns high-level, one-step skill. Then we deal with the problem of returning a flying ball with rotation to a target landing point. We model this problem as a one-step Markov Decision Process, and then treat it as a Monte Carlo based reinforcement learning task. Our work shows that our solution, which is a Monte Carlo version of Deep Deterministic Policy Gradient, deals with the task of returning spinning balls to a desired landing point with good performance. Yet how to make the function approximators (critic and actor) converge to better local optimum needs more exploration,



(a) Initialized (b) Trained After 10000 times

(c) 30000 times (d) 50000 times

(e) 100000 times (f) 200000 times

Fig. 7. Distribution of landing points after numbers of learning

given the complexity and non-convexity of non-linear neural networks.

Our result shows the possibility for more accurate control in robotic table tennis and gives the opportunity for a human-like robot to play in a competitive table tennis game. More work will be devoted into the transfer learning of policy, which would have one learned policy applied to any landing point within feasible region for the robot. Moreover, our idea and our reward setting have an instructive meaning in the learning of other similar sports such as badminton when the incoming object is in various dynamic states instead of static states.

REFERENCES

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] K. Mülling, J. Kober, and J. Peters, "A biomimetic approach to robot table tennis," *Adaptive Behavior*, vol. 19, no. 5, pp. 359–376, 2011.

[4] K. Mülling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *The International Journal of Robotics Research*, vol. 32, no. 3, pp. 263–279, 2013.

[5] Z. Wang, A. Boularias, K. Mülling, and J. Peters, "Modeling opponent actions for table-tennis playing robot." in *AAAI*, 2011.

[6] M. Matsushima, T. Hashimoto, M. Takeuchi, and F. Miyazaki, "A learning approach to robotic table tennis," *IEEE Transactions on robotics*, vol. 21, no. 4, pp. 767–771, 2005.

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

[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[9] K. Doya, "Reinforcement learning in continuous time and space," *Neural computation*, vol. 12, no. 1, pp. 219–245, 2000.

[10] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7, pp. 1180–1190, 2008.

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

[12] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 387–395.

[13] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[14] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," *arXiv preprint arXiv:1610.00633*, 2016.

[15] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *ICML*, vol. 99, 1999, pp. 278–287.

[16] J. Kober, E. Öztop, and J. Peters, "Reinforcement learning to adjust robot movements to new situations," in *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, vol. 22, no. 3, 2011, p. 2650.

[17] K. Muelling, J. Kober, and J. Peters, "Learning table tennis with a mixture of motor primitives," in *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*. IEEE, 2010, pp. 411–416.

[18] Y. Zhao, Y. Zhang, R. Xiong, and J. Wang, "Optimal state estimation of spinning ping-pong ball using continuous motion model," *IEEE Transactions on Instrumentation and Measurement*, vol. 64, no. 8, pp. 2208–2216, 2015.

[19] Y. Zhao, R. Xiong, and Y. Zhang, "Rebound modeling of spinning ping-pong ball based on multiple visual measurements," *IEEE Transactions on Instrumentation and Measurement*, vol. 65, no. 8, pp. 1836–1846, 2016.

[20] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, 2000, pp. 1008–1014.