Learning Ball-balancing Robot Through Deep Reinforcement Learning

Yifan Zhou[1], Jianghao Lin[2], Shuai Wang[3], Chong Zhang[3]

Abstract— **The ball-balancing robot (ballbot) is a good platform to test the effectiveness of a balancing controller. Considering balancing control, conventional model-based feedback control methods have been widely used. However, contacts and collisions are difficult to model, and often lead to failure in balancing control, especially when the ballbot tilts a large angle. To explore the maximum initial tilting angle of the ballbot, the balancing control is interpreted as a recovery task using Reinforcement Learning (RL). RL is a powerful technique for systems that are difficult to model, because it allows an agent to learn policy by interacting with the environment. In this paper, by combining the conventional feedback controller with the RL method, a compound controller is proposed. We show the effectiveness of the compound controller by training an agent to successfully perform a recovery task involving contacts and collisions. Simulation results demonstrate that using the compound controller, the ballbot can keep balance under a larger set of initial tilting angles, compared to the conventional model-based controller.**

Keywords*—*Ball-balancing robot; Balance control; Reinforcement Learning.

I. INTRODUCTION

Ballbot is a dynamically-stable mobile robot balancing on a single spherical wheel. This unique way of moving makes ballbot move in omnidirectional and thus exceptionally agile and maneuverable compared to other ground vehicles. Also, continuously position adjusting allows robot to be tall enough for human-robot interaction and having large accelerations without tipping over. Balancing on a ball, apart from the impressive look and behavior, also makes the robot more applicable to human environments. The underlying dynamics of robotic system is highly nonlinear and unstable, which requires full system dynamics planning and control. While conventional feedback control method can solve task such as balancing or velocity control in omnidirectional direction, applications that involve contacts between the wheel and the ball, and between the ball and the ground are difficult to approach with conventional control method. We usually use pure viscous damping friction model, neglecting the static friction and nonlinear dynamic friction. Also, the large tilt angle will lead to the no-contact, collision and additional force, which cannot be modeled. Identifying and modeling contacts and friction under these circumstances is difficult, even if a reasonable contact behavior is provided by a carefully identified physical model. Hence, it is often difficult to achieve adaptable yet robust behavior as soon as the unmodelable elements are introduced.

RL algorithms have been widely used to learn policies for unstructured environments, because they allow agent to interact with their environments continuously and to iterate the policy spontaneously. Moreover, RL can handle control problems that are difficult to approach with conventional controllers because the control goal can be specified indirectly as a term in a reward function and not explicitly as the result of a control action [1]. However, standard RL methods cannot be solely used in the ballbot, since it requires the robot learn through interaction, which can lead to instability. And collecting the amount of interaction that is needed to learn a complex skill can be time-consuming.

In this paper, we attempt to combine the benefits of both conventional feedback control and RL. We study control problem that the ballbot have large initial tilt angle, which cannot be described by the model. The problems possess the modelable structure that can be partially handled with conventional feedback control. And the unmodelable part of the control task, which must consider contacts and friction, is solved with RL.

The rest of the paper is organized as follows. In Section II, related work on ballbot is reviewed and followed by the RL in control problems. Section III provide an overview of ballbot system and introduce the fundamental of RL. In Section IV, we present the control system. We give the dynamic modeling of the ballbot and design a feedback controller, followed by the introduction of the DDPG method. Section V shows the simulation results implemented on ballbot, while Section VI concludes the paper.

II. RELATED WORK

A. Ballbot

Many ball-balancing robots have been built in the past years. The main ballbot research platform is shown in Fig. 1. The CMU Ballbot [2] used an inverse mouse-ball drive mechanism, consisting of four rollers to drive the ball. BallP [3], Rezero [4], Kugle [5] use three omni-wheels to drive the ball. A tall robot is built up for the convenince to interact with people. With a small leaning angle, the projection of the center of mass on the horizontal plane moves out of its supporting triangle. To increase the contact force or avoid the fact that the ball gets pushed out during large inclinations, the Kugle [5] robot is equipped with a skirt and the Rezero [4] robot is equipped with ball arrester, which hold the ball in place. These equipment reduce the unmodelable elements' influence on the ballbot.

[1]Y. Zhou is with Zhejiang University, Hangzhou, China (e-mail: yifanzhou7@gmail.com)

[2]J. Lin is with Shanghai Jiao Tong University, Shanghai, China (e-mail:chiangel@sjtu.edu.cn)

[3]S. Wang and C. Zhang are with Tencent Robotics X, Shenzhen, China (e-mail: {shawnshwang, chongzzhang}@tencent.com)

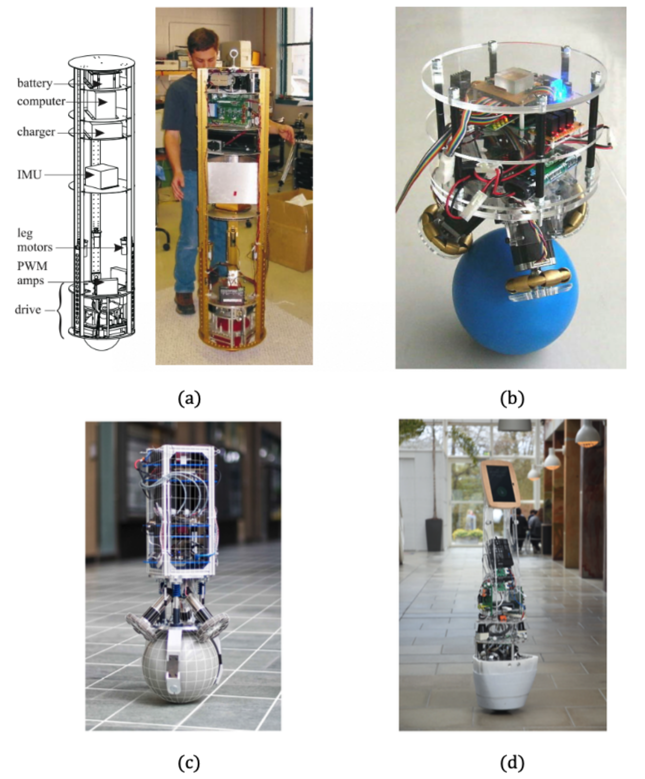

Fig. 1. Main ballbot contributions: (a) Ballbot from CMU; (b) “BallIP” from TGU; (c) “Rezero” from ETH (d) “Kugle” from AAU.

In [2] [3] and [4], linear feedback control approaches are used to maintain balance and achieve motion. The CMU ballbot uses an inner loop that maintains the body at desired inclination angles and an outer loop linear quadratic regulator (LQR) controller that achieves desired ball motions by commanding body angles to the inner controller. Later in [3] [6], CMU ballbot is introduced with arms and uses its planning procedure to plan in the space of both body lean angles and arm angles to achieve desired ball motions. The Kugle robot uses both linear feedback controllers [5] and nonlinear sliding mode is designed to control the body angles to follow a given path.

B. RL in Control Problems

With RL getting increasingly popular, a large number of new complex robot control applications have been achieved (e.g., solving Rubik’s Cube [7], learning dexterous in-hand manipulation [8], learning agile walk for legged robots [9], and trajectory tracking of a rotorcraft aerial robot [10]).

However, the dynamically stable robot is much less investigated due to the inherent instability which makes it impossible to collect enough data. And the random initial will easily cause robot tipping over or being destroyed. Hence, approaches that combine the convectional control method and the RL was proposed. A residual RL is used in robot arm control. The dynamical system consists of a fully actuated robot and underactuated objects. By decomposing them into a part that is solved efficiently by conventional feedback control methods, and the residual which is solved with RL, an agent was trained successfully to perform a real-world block assembly task involving contacts and unstable objects [11]. For the ballbot, a two-step algorithm is introduced which combines the proven performance of a model-based controller with a model-free method for compensating for model inaccuracy [12]. The authors build a linearized model of ballbot and design an iLQG controller, which can already stabilize the unstable system, enabling the learning algorithm to sample useful data. For the subsequent learning, the RL algorithm uses the designed iLQG controller as an optimized initial guess for the parameters for they share the same cost function. In this method, the RL is used to adapt the model-based controller to the real environment, rather than cope with the unmodelable problems.

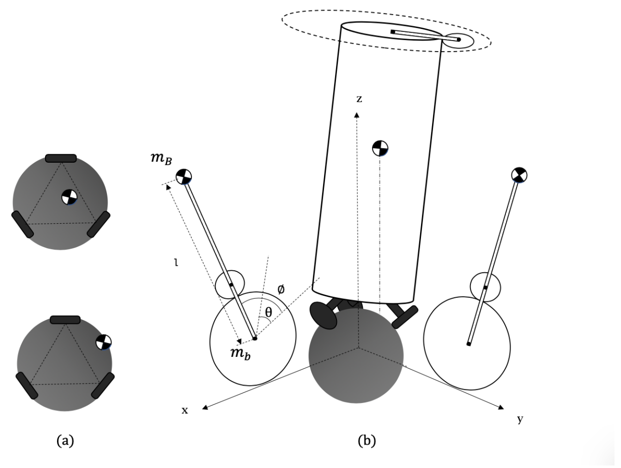

Fig. 2. The model of the ballbot: (a) Top view of the ballbot; (b) Ballbot is approximated into two decoupled plant.

III. PRELIMINARIES

In this section, we provide an overview of ballbot system. Also, the RL that we build on in our approach is introduced.

A. System Overview

The ballbot system consists of three rigid bodies: the ball, the actuating wheels and the body. The ball transmits and bears all arising forces, and the actuating wheels are three omniwheels mounted with $120°$ spacing. Additionally, the following assumptions are made [2]:

- *Rigid Body Assumption*: In this model, all the objects are considered as rigid bodies, which means all the deformation should be neglected.
- *Friction Assumption*: Friction between the wheel and the floor and between the wheel and the body is modeled as pure viscous damping. Forces due to static friction and nonlinear dynamic friction are neglected.
- *No Slip Assumption*: There is no slip between the ball and floor and between the ball and the wheel.

The three contacting points on the ball form a supporting triangle, which is essential to the system. Consider two cases of the ballbot shown in Fig. 2 (a). As the first case, the ballbot

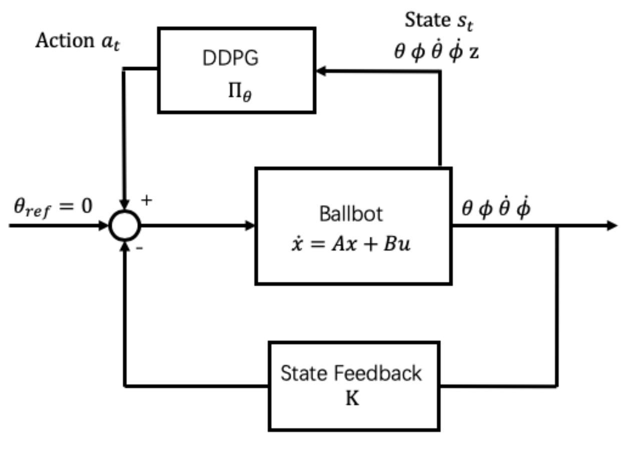

Fig. 3. Structure of the system.

tilt a small angle and its center of gravity is directly above the supporting triangle, which guarantees that all wheels are contacting with the ball. However, for some reason, the inclination angles of a ballbot increase to a large extent, leading to the center of gravity out of the supporting triangle. With the gravity torque, the contact condition and the friction change a lot, thus the assumptions we made when modeling the robot are no longer satisfied. More specifically, there is a practical limit on the magnitude of the tilt angle. Exceeding this limit induces lots of unpredictable factors not present in the ballbot model. The unpredictable factors such as contact and friction in this model makes it impossible to identify the model accurately. Based on the discussion above, we consider the ballbot a system that consists of modelable part and unmodelable one.

B. Reinforcement Learning

In RL, we consider an agent interacting with an environment E over discrete timesteps, which can be modeled as a Markov Decision Process. At every timestep t, the agent is in a state s_t, takes an action a_t, and the environment provides agent with a scalar reward r_t and the state s_{t+1}. An agent's behavior is defined by a policy π, which maps states to a probability distribution over actions $\pi : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$. Policy learning can be subdivided into two categories: 1) Stochastic 2) Deterministic. Stochastic policy learns the conditional probability of taking an action a, given that it is in a state s, $\pi(a|s) = P[A_t = a|S_t = s]$. A deterministic policy on the other hand learns the action as a function of state, $a = \pi(s)$, which means that the agent will give a deterministic action in each state s.

We denote the return by $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r_i$, where T is the horizon that the agent optimizes over and $\gamma \in [0, 1]$ is a discount factor for future rewards. In RL, the agent aims to learn a policy $u_t = \pi(s_t)$ to maximize the expected return.

IV. BALANCE CONTROL

Based on the analysis in Section III, we introduce a control system that consists of two parts. The first part is a conventional model-based linearized feedback controller that is able to keep the ballbot balance around the balancing point. It might have decreased performance on unmodelable conditions. And RL method is superposed and maximize the reward function we define. The control structure is illustrated in Fig. 3. In this way, the performance near the equilibrium will be guaranteed, and the iteration time may be decreased.

A. Task Definition

In this paper, we aim to design a controller that performs better than the conventional controller in the unmodelable section. To illustrate the ability of two controllers, we choose a balance recovery task, aiming to stabilize the unstable system from a tilt angle (with zero angular velocity) and keep balance for a period of time.

Now we will give the parameters to describe the initial state of the robot. For the ballbot standing on the ball with all three omniwheels contacting with ball, the body can be regarded revolving around the center of the ball. As shown in Fig. 4, the orientation of the ball can be described: 1) rotates α around the z-axis of the ball 2) rotates β around the y-axis of the ball. Generally speaking, the body has 2 DOFs: the inclination angle β and the direction α to which the robot tilts.

B. Conventional Feedback Controller

1) Simplified Mathematical Model: A nonlinear 3D model of the system dynamics of ballbot has been analytically derived in [2], where the wheel dynamics are neglected. In this model, the three-dimensional system is approximated in three decoupled planes. As shown in Fig. 2 (b), the median sagital plane (xz-plane) and the median coronal plane (yz-plane) related to the system moving forward or backward on the flat floor are identical and share same equations of motion. Wheels are simplified as one actuating wheel that fits in a plane. The third plane (xy-plane) describes the rotation around the z-axis in the body fixed reference frame, to which we pay less attention in balance control. Under these circumstances, the controller for the three-dimensional system can be designed by analysing and designing independent controllers for the two separate and identical systems.

We choose the median sagital plane to derive the model. In detail, the body angle θ represents the change of ball position calculated by x, which is read from sensors.

$$\theta = \frac{x}{r_b}, \tag{1}$$

where r_b is the radius of the ball. Another configuration is ϕ, aiming to describe the position of the robot above the ball. ϕ is accessed by IMU. The mass of the ballbot and the ball are given by m_B and m_b respectively. The distance from the center of mass of the ballbot to the center of mass of body is given by l. The Lagrangian formulation is used to derive the nonlinear equations of motion for the simplified model. According to Euler-Lagrange equations, the energies of the system obeys:

$$\frac{d}{dt}\Big(\frac{\partial \mathcal{L}}{\partial \dot q}\Big) - \frac{\partial \mathcal{L}}{\partial q} = Q, \tag{2}$$

where $\mathcal{L}$ is the difference between the kinetic energy and the potential energy, q is generalized coordinate, $\dot{q}$ is the derivative of generalized coordinate q, and Q is the external force, including the friction and the torque between the ball and the wheel.

The kinetic energy K_b of the ball is computed with rotation energy and transnational kinetic energy:

$$K_b = \frac{I_b\dot{\theta}^2}{2} + \frac{m_b(r_b\dot{\theta})^2}{2}, \tag{3}$$

where I_b, m_b, and r_b are the moment of inertia, the mass and the radius of the ball respectively. The potential energy of the ball is $V_b = 0$. The kinetic energy K_B and potential energy V_B of the body are

$$\begin{aligned} K_B =& \frac{m_B}{2}(r_b^2\dot{\theta}^2 + 2r_bl(\dot{\theta}^2 + \dot{\theta}\dot{\phi})\cos(\theta+\phi) + l^2(\dot{\theta}+\dot{\phi})^2) \\ &+ \frac{I_B}{2}(\dot{\theta}+\dot{\phi})^2, \\ V_B =& m_Bgl\cos(\phi+\theta), \end{aligned} \tag{4}$$

where I_B is the moment of inertia of the body with respect to the center of the ball, m_B is the mass of the body, and g is the gravity acceleration. The total kinetic energy is $K = K_b + K_B$ and the total potential energy is $V = V_b + V_B$. Define the system configuration vector $q = \begin{bmatrix}\theta & \phi\end{bmatrix}^T$. The Lagrangian $\mathcal{L}$ is a function of q and $\dot{q}$:

$$\mathcal{L}(q,\dot{q}) = K - V = K_B + K_b - V_B - V_b. \tag{5}$$

Let τ be the component of the torque applied between the ball and the body in the direction normal to the plane. To model the viscous friction terms, define the vector

$$D(\dot{q}) = \begin{bmatrix}\mu_\theta\dot{\theta} \\ \mu_\phi\dot{\phi}\end{bmatrix}, \tag{6}$$

where μ_θ and μ_ϕ are the viscous damping coefficients that model ball-ground and ball-body friction, respectively. Using this notation, the Euler-Lagrange equations of motion for the simplified ballbot model are

$$\frac{d}{dt}\Big(\frac{\partial\mathcal{L}}{\partial\dot{q}}\Big) - \frac{\partial\mathcal{L}}{\partial q} = \begin{bmatrix}0\\ \tau\end{bmatrix} - D(\dot{q}). \tag{7}$$

After computing the derivatives in the Euler-Lagrange equations and rearranging terms, the equations of motion can be expressed as

$$M(q)\ddot{q} + C(q,\dot{q}) + G(q) + D(\dot{q}) = \begin{bmatrix}0\\ \tau\end{bmatrix}. \tag{8}$$

The mass matrix $M(q)$ is

$$M(q) = \begin{bmatrix}\Gamma_1 + 2m_Br_bl\cos(\theta+\phi) & \Gamma_2 + m_Br_bl\cos(\theta+\phi) \\ \Gamma_2 + m_Br_bl\cos(\theta+\phi) & \Gamma_2\end{bmatrix}, \tag{9}$$

where

$$\Gamma_1 = I_b + I_B + m_br_b^2 + m_Br_b^2 + m_Bl^2, \tag{10}$$

$$\Gamma_2 = I_B + m_Bl^2. \tag{11}$$

The vector of Coriolis and centrifugal forces is

$$C(q,\dot{q}) = \begin{bmatrix}-m_Br_bl\sin(\theta+\phi)(\dot{\theta}+\dot{\phi})^2 \\ 0\end{bmatrix}, \tag{12}$$

and the vector of gravitational forces is

$$G(q) = \begin{bmatrix}-m_Bgl\sin(\theta+\phi) \\ -m_Bgl\sin(\theta+\phi)\end{bmatrix}. \tag{13}$$

Premultiply both sides of (8) by the inverse of matrix M:

$$\ddot{q} = M(q)^{-1}\Big(\begin{bmatrix}0\\ \tau\end{bmatrix} - C(q,\dot{q}) - G(q) - D(\dot{q})\Big). \tag{14}$$

Rewrite the expression:

$$\ddot{q} = \begin{bmatrix}\ddot{\theta} \\ \ddot{\phi}\end{bmatrix} = \begin{bmatrix}\frac{b(\Gamma_2 + lm_Br_b\cos(\phi+\theta)) - a\Gamma_2}{c} \\ \frac{a(\Gamma_2 + lm_Br_b\cos(\phi+\theta)) - b(\Gamma_1 + 2lm_Br_b\cos(\phi+\theta))}{c}\end{bmatrix}, \tag{15}$$

where

$$\begin{aligned} a =& m_Bgl\sin(\phi+\theta) - \mu_\theta\dot{\theta} + lm_Br_b\sin(\phi+\theta)(\dot{\phi}+\dot{\theta})^2, \\ b =& \tau - \mu_\phi\dot{\phi} + glm_B\sin(\phi+\theta), \\ c =& \Gamma_2^2 - \Gamma_1\Gamma_2 + l^2m_B^2r_b^2\cos^2(\phi+\theta). \end{aligned} \tag{16}$$

To put these equations into standard nonlinear state space form, define the state vector to be $x = \begin{bmatrix}q^T & \dot{q}^T\end{bmatrix}^T$ and define the input $u = \tau$. The relationship $\sin(\theta+\phi) = \theta+\phi$ and $\dot{\phi} = \dot{\theta} = 0$ are satisfied when the ballbot system is at the equilibrium point. Then the linearized state space model can be described as:

$$\begin{cases}\dot{x} = Ax + Bu \\ y = Cx + Du,\end{cases} \tag{17}$$

where the inner state is

$$x = \begin{bmatrix}\theta & \phi & \dot{\theta} & \dot{\phi}\end{bmatrix}^T, \tag{18}$$

and the system input is

$$u = \tau \tag{19}$$

with

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{m_B^2l^2gr_b}{H} & \frac{m_B^2l^2gr_b}{H} & \frac{\mu_\theta\Gamma_2}{H} & \frac{-\mu_\phi F}{H} \\ \frac{m_Bgl(F-G)}{H} & \frac{m_Bgl(F-G)}{H} & \frac{-\mu_\theta(\Gamma_2+m_br_bl)}{H} & \frac{\mu_\phi G}{H}\end{bmatrix}, \tag{20}$$

$$B = \begin{bmatrix}0 & 0 & \frac{F}{H} & -\frac{G}{H}\end{bmatrix}^T, \tag{21}$$

where $F = \Gamma_2 + m_Br_bl$, $G = \Gamma + 2m_Br_bl$ and $H = \Gamma_2^2 - \Gamma_1\Gamma_2 + m_B^2l^2r_b^2$. C and D are the identity matrix and zero matrix with appropriate dimensions respectively.

2) Design of Controller: Now a linear state feedback controller can be used that stabilizes the system about $\phi = 0$. We apply the state feedback control law

$$u = -Kx \tag{22}$$

to system, and we choose the structure of K to be

$$K = \begin{bmatrix}k_1 & k_2 & k_3 & k_4\end{bmatrix}. \tag{23}$$

For the linearized system, a control law can be designed to ensure stability by traditional linear method.

C. Balance Control Based on Deep Reinforcement Learning

In this paper, we use DDPG algorithm [13], a model-free, off-policy actor-critic algorithm using deep function approximators that provides us with a good framework to train the neural network for learning highly non-linear functions.

1) Deep Deterministic Policy Gradient: DDPG is an actor-critic policy learning method [14] with additional target networks to stabilize the learning process. The action-value function describing the expected return after taking an action a_t in the state s_t:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{r_{i\geq t}, s_{i>t}\sim E, a_i > t\sim\phi}[R_t|s_t, a_t], \qquad (24)$$

and the recursive version of (24), known as the Bellman equation, is as follows:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1}\sim E}[r(s_t, a_t) + \gamma\mathbb{E}_{a_{t+1}\sim\pi}[Q^\pi(s_{t+1}, a_{t+1})]] \qquad (25)$$

DDPG is based on DQN [15], a commonly used off-policy algorithm. It uses the greedy policy $\pi(s) = \arg\max_a Q(s, a)$. We consider function approximators parameterized by θ^Q, which we optimize by minimizing the loss:

$$L(\theta^Q) = \mathbb{E}_{s_t\sim\rho^\beta, a_t\sim\beta, r_t\sim E}[(Q(s_t, a_t|\theta^Q) - y_t])^2, \qquad (26)$$

where

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q). \qquad (27)$$

In DDPG algorithm the target policy is deterministic We can describe it as a function $\mu : \mathcal{S} \to \mathcal{A}$ and avoid the inner expectation:

$$Q^\mu(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1}\sim E}\left[r(s_t, a_t) + \gamma Q^\mu(s_{t+1}, \mu(s_{t+1}))\right] \qquad (28)$$

The actor is updated by following the applying the chain rule to the expected return from the starting distribution J with respect to the actor parameters:

$$\begin{aligned}\nabla_{\theta^\mu} J \approx&\mathbb{E}_{s_t\sim\rho^\beta}[\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)}]\\ =&\mathbb{E}_{s_t\sim\rho^\beta}[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)}\nabla_{\theta_\mu}\mu(s|\theta^\mu)|_{s=s_t}]\end{aligned} \qquad (29)$$

To guarantee the samples' independence and identical distribution, a replay buffer R is used, which also makes efficient use of hardware optimizations. Transitions are sampled from the environment according to the exploration policy and the tuple (s_t, a_t, r_t, s_{t+1}) is stored in the replay buffer. When this buffer becomes full, oldest samples are discarded. The actor and critic are updated by sampling a minibatch uniformly from the buffer.

Additionally, DDPG treat the problem of exploration independently from the learning algorithm to deal with the in exploration in continuous action space. So the exploration policy μ' is constructed by adding noise sampled from a noise process $\mathcal{N}$ to the actor policy μ

$$\mu'(s_t) = \mu(s_t|\theta_t^\mu) + \mathcal{N}. \qquad (30)$$

When updating the target values, DDPG uses "soft" target update, rather than directly copying the weights: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ with $\tau \ll 1$, ensuring the consistently train without divergence. For more details, refer to Algorithm 1.

Algorithm 1: DDPG algorithm

```
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor policy
$\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$,
$\theta^{\mu'} \leftarrow \theta^\mu$
Initialize relay buffer $R$
for episode =1 $\to M$ do
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $s_1$
  for $t = 1 \to T$ do
    Select and execute action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$
    Observe reward $r_t$ and new state $s_{t+1}$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
    Sample a minibatch of transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu(s_{i+1}|Q^{\mu'})|\theta^{Q'})$
    Update critic network by minimizing the loss function:
        $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^\mu))^2$
    Update actor policy using the sampled policy gradient:
        $\nabla_{\theta_\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, s=\mu(s_i)}\nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
    Update the target networks:
        $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
        $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
  end for
end for
```

2) State Vector And Network Architecture: Consistent with the conventional controller, the chosen state vector consists of the body angle ϕ and the angular velocity $\dot{\phi}$ in both the median sagital plane and the median coronal plane. As the large inclination angle will cause the problem that the ball might be pushed out, only this state cannot identify the full state of the ballbot and the ball. We introduce the the attitude of the robot z to identify the full state of the ballbot.

And the action vector is the target velocity of the virtual wheel in each plane, which can be converted into the velocity for the real three motors. Considering the ability to improve the performance of the conventional controller, we decide the amplitude of the action approximately equals the max speed that the motor will reach.

The main object of our task is to provide a set of motor input that are needed to keep the ballbot in balance. The only element included in the reward function is β, the included angle between the z-axis in the body fixed reference frame and the vertical direction. The final reward function is as follows

$$r_t = -||\beta|| + c, \qquad (31)$$

where c is a constant to make sure the reward function is always positive.

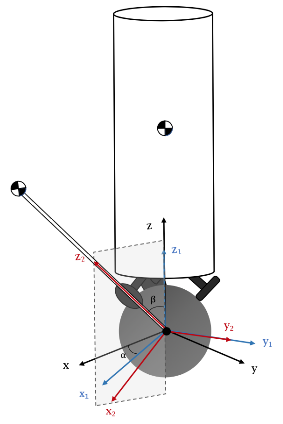

Fig. 4. Parameters used to describe the initial state.

V. EXPERIMENTS

We evaluate our method in simulation. In this section, we start by reporting how the experimental system is set up. Then the application of conventional feedback control method is detailed, followed by the improvement using DDPG algorithm. Several experiments are implemented, with results being analyzed.

A. *Experimental Setup*

We use CoppeliaSim 4.1.0 shown in Fig. 5, a full-featured simulator for model-based optimization considering body contacts, to evaluate our method in simulation. This environment consists of a ballbot with three omni wheels, and without equipment such as skirt or ball arrester. The omniwheel is driven by a motor controlled by a speed incremental controller. Also, we assume the full robot state information is available thanks to the virtual sensor on the body. To speed up the training process, we use the RL library, stable-baselines, which is an open-source library for reinforcement leaning that offers a simple interface to train, evaluate agents and do hyperparameter tuning for a variety of applications.

The neural architectures for actor and critic network are both two fully connected Multi-Layer Perceptron with layer normalization. The other hyperparameters we used are listed in Table I.

TABLE I
HYPERPARAMETERS USED FOR DDPG

Hyperparameter	**Value**
Hardware configuration	1 CPU + 4 CPU cores
Optimizer	Adam
Actor learning rate	0.0001
Critic learning rate	0.0001
Weight decay regularization	0.00001
Minibatch size	128
Replay buffer size	50000
Constant c in reward function	0.7

B. *Experimental Results*

In the test run shown in Fig. 6 and Fig. 8, a ballbot was commanded to recover from different initial states, with conventional controller and the compound controller respectively. The radial coordinate represents the angle β, and the angular coordinate represents the angle α. All angles are expressed in degrees. Yellow points indicate that the ballbot can recover at those points, and purple points otherwise. The three lines mounted with $120°$ spacing represent the wheel direction. Boundaries between yellow and purple points in both figures are roughly triangular. Define the yellow part as recovery area. Compared with the conventional controller, the compound controller has larger recoverable area. However, the ballbot has the least tilt angle β with either conventional controller or the compound controller in the direction of the three wheels ($\alpha = 90°$, $210°$ or $330°$). The compound controller has very limited expending capacity at these directions. When the ballbot tilts a large angle in the direction $\alpha = 90°$, only one omniwheel contacts with the ball. The force between the ball and the wheel is in the tangential direction. The motor rotation contributes less to the balancing. When $\alpha = 30°$, $150°$ or $270°$, the ballbot has two wheels contacting with the ball when $\alpha = 30°$, which allows the ball moving under two DOFs. Under these α angles, β can be up to $12° \sim 13°$. Although the RL-based controller learns efficiently through contacting with the environment, it cannot break physical limitations of the system.

Detailed states and inputs of the ballbot with respect to time are shown in Fig. 7 and Fig. 9. When $\alpha = 210°$, $\beta = 7.5°$, the ballbot with conventional controller falls over but the one with compound controller recovers. Body angles ϕ_x, ϕ_y and the angular velocities of three motors are depicted. Oscillation exists during the adjusting time. Under this situation, the three wheels change their rotating directions rapidly and the ballbot “kicks” the ball to keep balance. Collision, bounce and slipping exist in this system, which are unpredictable and unmodelable.

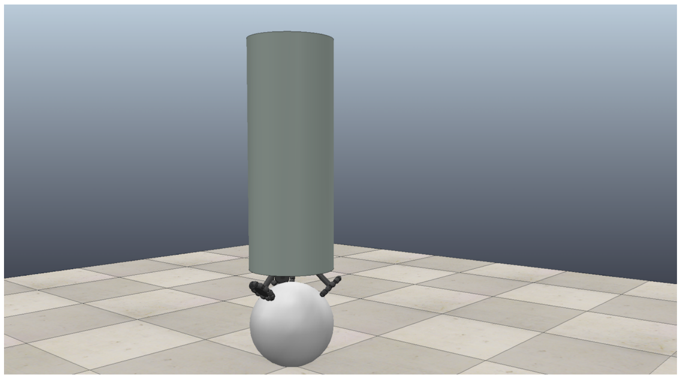

Fig. 5. Ballbot in CoppeliaSim.

VI. CONCLUSION

In this paper, RL algorithm is superposed to the conventional controller, making up a compound controller. Both the conventional feedback controller and the compound controller can stabilize the ballbot system, while the compound controller has larger recovery area. When the compound controller tries to balance the ballbot from an initial state that

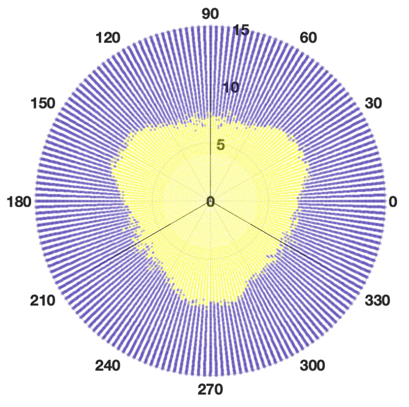

Fig. 6. The recovery area of the conventional controller.

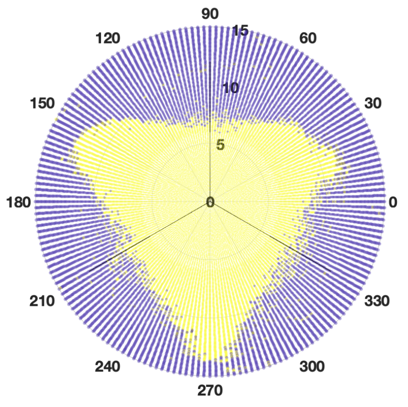

Fig. 8. The recovery area of the compound controller.

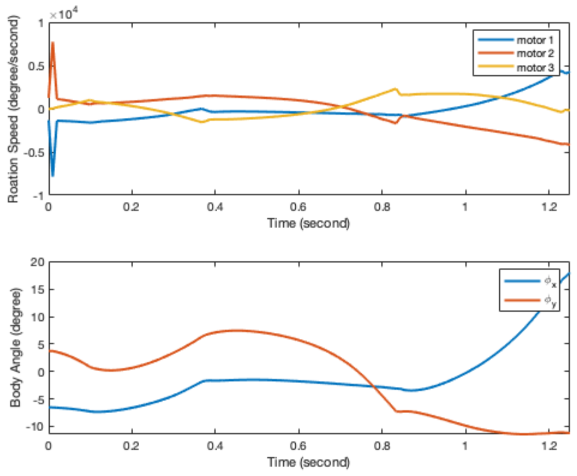

Fig. 7. The body angle ϕ_x ,ϕ_y and the rotation velocity of three motors with conventional controller in 1.25s.

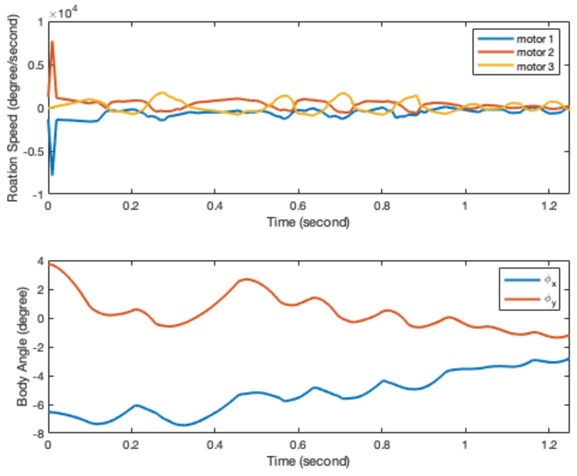

Fig. 9. The body angle ϕ_x, ϕ_y and the rotation velocity of three motors with compound controller in 1.25s.

the conventional controller fails, the ballbot "kicks" the ball, which is an unmodelable but interesting phenomenon. This control policy can only be designed by RL. It is demonstrated that under certain circumstances, RL-based approach could give better or even unpredictable policies that can rarely be derived by conventional control framework.

References

[1] Wang, Cong, et al. "Learning Mobile Manipulation through Deep Reinforcement Learning." *Sensors* 20.3 (2020): 939.

[2] Lauwers, Tom B., George A. Kantor, and Ralph L. Hollis. "A dynamically stable single-wheeled mobile robot with inverse mouse-ball drive." *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006*. ICRA 2006.. IEEE, 2006.

[3] Kumagai, Masaaki, and Takaya Ochiai. "Development of a robot balancing on a ball." *2008 International Conference on Control, Automation and Systems*. IEEE, 2008.

[4] Fankhauser, Péter; Gwerder, Corsin (2010). *Modeling and control of a ballbot*. Bachelor Thesis, ETH.

[5] Thomas Kølbæk Jespersen (April 2019). *Kugle - Modelling and Control of a Ball-balancing*. Master Thesis, AAU.

[6] Sonnleitner, Fabian, Roberto Shu, and Ralph L. Hollis. "The mechanics and control of leaning to lift heavy objects with a dynamically stable mobile robot." *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019.

[7] Akkaya, Ilge, et al. "Solving rubik's cube with a robot hand." *arXiv preprint arXiv:1910.07113* (2019).

[8] Andrychowicz, OpenAI: Marcin, et al. "Learning dexterous in-hand manipulation." *The International Journal of Robotics Research* 39.1 (2020): 3-20.

[9] Hwangbo, Jemin, et al. "Learning agile and dynamic motor skills for legged robots." *Science Robotics* 4.26 (2019).

[10] Palunko, Ivana, et al. "A reinforcement learning approach towards autonomous suspended load manipulation using aerial robots." *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013.

[11] Johannink, Tobias, et al. "Residual reinforcement learning for robot control." *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019.

[12] Farshidian, Farbod, Michael Neunert, and Jonas Buchli. "Learning of closed-loop motion control." *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014.

[13] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." *arXiv preprint arXiv:1509.02971* (2015).

[14] Rosenstein, Michael T., et al. "Supervised actor-critic reinforcement learning." *Learning and Approximate Dynamic Programming: Scaling Up to the Real World* (2004): 359-380.

[15] Mnih, Volodymyr, et al."Human-level control through deep reinforcement learning." *nature* 518.7540 (2015): 529-533.