Reinforcement Learning for Ballbot Balancing

Giulia Buzzetti[1], Davide Zappetti[2] and Giovanni Iacca[1]

***Abstract*— The present work investigates the influence of different simulation modelling choices on the performance and stability of RL-based ballbot balancing. In this study, three modelling paradigms are compared: a commonly used linear planar model, an omniwheel model with explicit spherical rollers, and a capsule-based model with anisotropic friction. For each configuration, a PPO-based controller is trained and evaluated on the same balancing task. The findings indicate that simulation fidelity exerts a substantial influence on learning behavior and control stability. In contrast to the planar and roller-based models, which demonstrate oscillatory or unstable behavior and lack reliable balancing, the anisotropic-friction model facilitates asymptotically stable balancing, ensuring smooth control actions and precise velocity tracking. These findings demonstrate that accurate modelling of the interaction between the wheel and the ground is critical for successful control of the ballbot using RL, and that anisotropic friction provides an effective and computationally tractable solution for stable learning in simulation.**

I. Introduction

Ballbots are robots that balance on a ball using wheels: over the past years, different ballbot models have been implemented, with configurations ranging from the number and placement of omniwheels used to balance on the ball [6], to different sizes [13] and application scopes [7].

Balancing on a single sphere allows ballbots to occupy little space and to perform swift, agile motions, while also being easy to move by users. Due to this configuration, however, ballbots operate around an unstable equilibrium. This means that they must continuously move in order to maintain balance, i.e., to avoid tipping over or falling. For this reason, reliable and stable balance control is essential.

Most of the literature in this area focuses on classical control approaches, such as PID [1], LQR [14], or combinations of the two [13]. On the other hand, while Reinforcement Learning (RL) approaches have been extensively used in various robotic systems, only a limited number of RL-based approaches for ballbot balancing have been proposed, and they all rely on the planar model described in Section II-A.1. In [19], a deep neural network is trained to recover stability after perturbations, while a PID controller is used to maintain balance. In contrast, [11] proposes a general framework for RL-based ballbot balancing; however, no agent is actually trained, and no balancing results are reported. More recently, Salehi et al. [15] introduced an RL framework for ballbot navigation on uneven terrain using omniwheels modeled as capsules with anisotropic friction. Nevertheless, to the best of our knowledge, the impact of different simulation modeling choices on RL-based ballbot balancing has not yet been systematically investigated, and no comparative evaluation under controlled conditions has been reported.

To address this gap, this work presents a systematic comparison of different ballbot simulation models for RL-based balancing. The main contributions of the paper are:

- a comparative study of different ballbot modeling choices for RL-based balancing;
- an empirical analysis of how commonly used modeling abstractions affect learning stability and performance;
- the identification of a simple and simulation-stable modeling setup that enables smooth, drift-free control.

The rest of the paper is structured as follows: Section II describes the different models that are compared, and how the RL framework is applied to each model; Section III reports the results; and Section IV concludes the paper.

II. Methodology

A. Modeling paradigms for ballbot control

To address the problem of balancing ballbots, we conduct a comparative analysis of three different modeling paradigms and evaluate their applicability within an RL framework. For simplicity, the ballbot's body is modelled as a cylinder, neglecting the arms and other specific components such as the battery and the detailed mechanical structure of the wheel–body interface.

Modeling and simulating the interaction between the ballbot body and the sphere is particularly challenging due to the complexity of accurately representing the omniwheels. To address this challenge, three models with different levels of abstraction are investigated to determine which most effectively captures the real behavior of the ballbot.

The following subsections provide a comprehensive overview of the three models.

1) Modeling the ballbot with a linear model: The full 3D robot is first approximated as three independent planar subsystems, consistent with the literature [4], [14], [9], [10], [19], [11]. Two planes, denoted *xz* (sagittal) and *yz* (coronal), are used to describe balancing while moving forward/backward and left/right, respectively, and are assumed to be identical. The third plane, denoted *xy*, captures yaw motion (rotation about the vertical axis) and is generally treated separately or given less emphasis for balance control.

Each planar subsystem is described using two degrees of freedom (DoF): one DoF for the ball motion along the ground (equivalently, the ball's rolling angle in that plane), and one DoF for the body rotation (tilt) in that plane.

[1]Giulia Buzzetti and Giovanni Iacca are with the Department of Information Engineering and Computer Science, University of Trento, 38123 Trento {giulia.buzzetti,giovanni.iacca}@unitn.it.

[2]Davide Zappetti is with Enchanted Tools, Paris, France davide@enchanted.tools.

The real actuation consists of three omniwheels; however, the planar model replaces them with a single "virtual" actuating wheel per plane, producing an equivalent driving torque on the ball. This is a modeling convenience and does not represent the physical wheel geometry directly. Consequently, when implementing control on hardware, the virtual-plane torques must be mapped back to the three real motor torques via a suitable conversion.

The following assumptions are made in order to capture the dominant balancing dynamics:

- The decoupled planes are assumed to be independent, i.e., no coupling between planes is considered.
- No slippage occurs between the ball and the ballbot body.
- The distance between the ball and the body is fixed.
- No friction is present at the ball–body contact.
- The contact between the ball and the body is non-deformable, and all components are treated as rigid bodies.
- Motor torque is treated as a direct input.
- The floor is assumed to be flat.

Under these assumptions, the ballbot in a plane can be modelled as a rigid body (often idealized as a cylinder) mounted on a rolling sphere, resembling an inverted pendulum on a rolling base.

The limitations of this model stem from the idealized nature of these assumptions. In particular, the model does not account for wheel–ball slippage, motor friction, or possible loss of contact between the wheels and the sphere.

2) Modeling the omniwheels with spheres: In this model, the omniwheels are based on the single-contact-line design originally proposed by Asama et al. [2]. This configuration, characterized by a single row of passive rollers, has been widely adopted due to its mechanical simplicity and its ability to accommodate surface irregularities. Alternative omnidirectional wheel designs incorporate rollers with cylindrical geometry mounted at a constant inclination with respect to the wheel axis, most commonly at an angle of 45°, as in conventional Mecanum-type wheels [3].

Owing to the passive rotation of the rollers, omniwheels are able to transmit traction forces along the wheel's rolling direction while allowing unconstrained motion in the perpendicular direction [5]. Consequently, the wheel–ground interaction can be decomposed into a tangential driving force generated by wheel rotation and a lateral component associated with the free rolling of the rollers, analogous to the sliding behavior observed in spherical contacts.

Several approaches have been proposed to model this behavior in simulation. A high-fidelity method consists of explicitly modeling each roller and its contact with the ground, potentially using rollers of varying effective radii to ensure smooth transitions between successive contacts [18]. While such approaches improve numerical continuity, they significantly increase computational cost and can severely degrade real-time simulation performance. Similar drawbacks have been reported in dynamic simulations of omnidirectional and Mecanum wheels, where discretization of rollers leads to contact-induced vibrations and reduced simulation efficiency.

Here, following the simplified strategy adopted in [3], the omniwheel is modelled as a cylindrical hub equipped with ten evenly spaced, fixed rollers distributed along its circumference, as illustrated in Fig. 1. Although this representation captures the essential kinematic properties of the omniwheel, it introduces non-ideal contact transitions between adjacent rollers, which manifest as small oscillations during motion. The simulation itself is unstable: because of the gaps between adjacent spheres, the ballbot intermittently loses continuous contact with the ground, causing it to bounce on the globe.

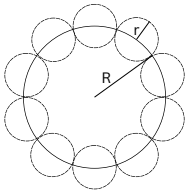

Fig. 1. The omniwheel model uses a cylindrical hub surrounded by ten small, free-rotating rollers that allow passive lateral motion.

To mitigate this effect, ball arresters are introduced. These are fixed spheres placed on the opposite side of each omniwheel. However, this modification alone is insufficient to ensure stability. When the arresters are positioned too close to the globe, they constrain the omniwheels and prevent proper rotation. Conversely, when they are placed too far from the globe (i.e., too close to the ball), the ballbot continues to exhibit the bouncing behavior.

3) Modeling the omniwheels as capsules with anisotropic friction: This model is inspired by the fundamental kinematic behavior of omniwheels equipped with a single row of passive rollers. When the omniwheel rotates around its main axis, it generates a tangential velocity (denoted $v_{\parallel}$) at the contact point with the globe. This velocity lies in the plane of the wheel and is perpendicular to the wheel's axis of rotation. It is the only velocity component that the wheel actively imposes on the contacting surface.

Due to the presence of the passive rollers, the omniwheel exhibits a fundamentally different behavior in the transverse direction. The rollers are free to rotate about their own axes, which are orthogonal to the main wheel axis. As a result, the contact point can move in the direction perpendicular to $v_{\parallel}$ with negligible resistance. If the globe has a velocity component in this transverse direction, it is not opposed by significant friction at the contact, because this motion is accommodated by the rotation of the rollers.

Therefore, the velocity of the contact point between the omniwheel and the globe can be decomposed into two orthogonal components: one parallel to the wheel's driven direction and one perpendicular to it. While such a decomposition is always possible, the defining characteristic of an omniwheel is that only the parallel component is constrained by friction and actuation, whereas the perpendicular component is effectively unconstrained.

This property is essential for ballbot operation. A typical ballbot employs three omniwheels arranged at 120 deg inter-

vals around the globe. Each wheel drives the globe along its own local parallel direction, which differs from that of the other wheels. If transverse motion were not permitted at each contact, the wheels would mechanically interfere with one another, generating conflicting constraints that would prevent smooth rolling. The passive rollers eliminate these conflicts by allowing relative motion in the transverse directions, enabling the combined wheel actions to produce arbitrary planar motion of the globe.

One solution to replicate this mechanism in simulation was proposed in [12]. The authors simplify the simulation of Mecanum wheels by replacing each roller's rotating joint with a fixed connection, which greatly reduces the number of constraints and speeds up computation. To preserve the physical behavior of the rollers, they assign direction-dependent (anisotropic) friction to the roller links. High friction is applied along the roller's free-rolling axis to transmit driving force, while low friction is used across it, allowing sideways slip similar to that of real rollers.

Another simple yet effective solution was proposed in [15]. In their work, the authors simulate the omniwheels as capsules with anisotropic friction. The friction in the direction of wheel motion is set high to ensure force transmission, while in the perpendicular direction, it is low to simulate the effect of the passive rollers. This solution was made possible by a custom feature implemented in the MuJoCo simulator [17]. To the best of our knowledge, MuJoCo is the only simulator to allow anisotropic friction to be specified on the same body.

B. Reinforcement Learning based balancing

We now describe how the RL framework for balancing was implemented for the different models under study. We first outline the elements that are common across all experiments and then discuss the model-specific design choices.

1) Common experimental setup: To address the balancing problem using RL, we employ a deep neural network and adopt a symmetric actor–critic architecture trained with the Proximal Policy Optimization (PPO) algorithm [16]. In all experiments, the network consists of three fully connected layers with sizes 512, 256, and 128, as this architecture provided stable convergence while keeping the model compact. The actor and critic networks share the same architecture and use the same observation vector, consistent with the symmetric actor–critic formulation.

The control is velocity-based. The physics simulation runs at 200 Hz, while the policy outputs actions at 50 Hz. The different models use different action spaces because they have different actuators; the specific action definitions for each model are detailed in the following subsections.

The observation vector is identical across all experiments. It includes the projected gravity expressed in the trunk frame, the velocity of the actuated DoF, the linear velocities of the trunk, the angular velocity of the trunk, the desired velocities, and the previous action. Since the task is to balance at a fixed point, the velocity commands are set to zero. The projected gravity in the trunk frame represents the direction of gravity in the local coordinate system of the trunk and therefore encodes its orientation with respect to gravity. The horizontal components (g_0, g_1) correspond to the trunk tilt in the sagittal and lateral directions, while the vertical component g_z measures the alignment of the trunk's vertical axis with gravity.

The reward function is composed of five different terms. One term provides positive feedback when the ballbot's base is close to horizontal:

$$r_o = -\omega_o \sum_{j=0}^{1} g_j^2 + c, \tag{1}$$

where $\omega_o > 0$ is a weighting factor, c is a constant chosen to keep the reward positive, and g_0 and g_1 are the x and y components of the projected gravity of the ballbot's base, respectively. A second term penalizes large control actions:

$$r_a(t) = \omega_a \sum_{j=0}^{2} a_j^2(t), \tag{2}$$

where $\omega_a < 0$ is a weighting factor and $a(t) = [a_0(t), a_1(t), a_2(t)]$ denotes the action vector at time t. A third term penalizes large variations in the actions:

$$r_l(t) = \omega_l \sum_{j=0}^{2} \big(a_j(t-1) - a_j(t)\big)^2, \tag{3}$$

where $\omega_l < 0$ is a weighting factor. Another term tracks the desired linear velocity along the x and y axes:

$$r_{vl}(t) = e^{\left(\frac{\Sigma\left(c_l(t) - v_l(t)\right)^2}{\sigma_l}\right)}, \tag{4}$$

where $c_l(t)$ is the desired trunk velocity in the x–y plane, $v_l(t)$ is the measured trunk velocity, and σ_l is a scaling constant. Finally, the angular velocity around the z axis is tracked by

$$r_{va}(t) = e^{\left(\frac{\Sigma\left(c_a(t) - v_a(t)\right)^2}{\sigma_a}\right)}, \tag{5}$$

where $c_a(t)$ and $v_a(t)$ are the desired and measured angular velocity around the z axis, respectively, and σ_a is a constant.

Each episode terminates when the robot reaches a lean angle of 20°, which is set as a threshold beyond which the ballbot can no longer regain stability. In addition, an episode is terminated if the network outputs an action at the maximum allowed magnitude. This choice was made to avoid bang-bang control behavior that would compromise the stability of the robot. If none of these conditions are met, the maximum episode length is set to 40 s, based on the assumption that if the robot is able to balance for more than 40 s, it has reached asymptotic stability.

2) Reinforcement Learning for the ballbot linear model: The RL framework implemented using the linear model was trained in the Isaac Gym environment [8]. Isaac Gym provides a GPU-based platform in which both the simulation and the training of the algorithm are executed, enabling fast and highly parallelized learning. Leveraging this architecture, the agent was trained using 2048 parallel vectorized environments for 1000 episodes. The velocity-based controller runs

at 50 Hz and outputs the desired velocities for each of the actuated linear planes of the ballbot, in which the dynamics are decomposed, namely the roll, pitch, and yaw of the ball.

3) Reinforcement Learning for the omniwheels modelled with spheres: This model is computationally heavier than the planar one. Therefore, it was not possible to simulate the same number of parallel vectorized instances. Consequently, the agent was trained in Isaac Gym using 1024 parallel instances for 2000 episodes. The network outputs the desired velocities of the three omniwheels, which are simulated as cylinders with rollers on their surfaces.

4) Reinforcement Learning for the omniwheels modelled with capsules with anisotropic friction: This framework could not be implemented in Isaac Gym because anisotropic friction on the same body is not supported. The RL environment was therefore built on top of MuJoCo, and both the simulation and the training pipeline were executed on the CPU. As a result, large-scale parallelization was not possible, and the network was trained using 10 parallel environments for 4,000,000 episodes. The network outputs the desired velocities of the wheels, which are simulated as capsules with anisotropic friction.

III. Results

We first present the results of the linear model, followed by those obtained with the omniwheels modelled as spheres, and finally those obtained with the model where omniwheels are represented as capsules with anisotropic friction.

A. Results with the linear model

Figs. 2 and 3 show the orientation and the velocity of the trunk during a test episode. While the system can maintain balance for about a minute, it does not reach a stable equilibrium. We can notice a visible oscillation in the trunk orientation and an even more marked one in the yaw velocity.

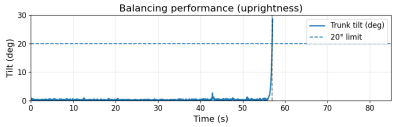

Fig. 2. Trend of the ballbot trunk's deviation from upright, measured from the projection of the gravity vector onto the trunk z-axis. The dashed line denotes the 10° allowable tilt.

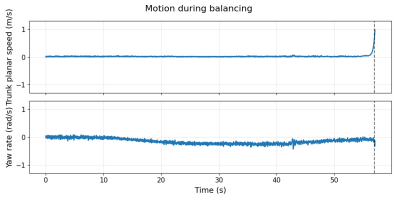

Fig. 3. Motion of the trunk during balancing. The top panel shows the magnitude of the linear velocity in the horizontal (xy) plane, while the bottom panel shows the angular velocity about the vertical (z) axis.

An inspection of the control output, presented in Fig. 4, reveals that the actions are not smooth, but they present numerous spikes, which contribute to this instability. However, looking at Fig. 5, we can observe that the model reasonably tracks the reference velocities until the onset of instability. Once the ballbot begins to fall, recovery is impossible, and trajectory tracking deteriorates rapidly.

The total training time was circa 20 minutes. The reward curve shown in Fig. 6 indicates convergence to a steady value, suggesting that learning has stabilized and further training is unlikely to improve performance significantly.

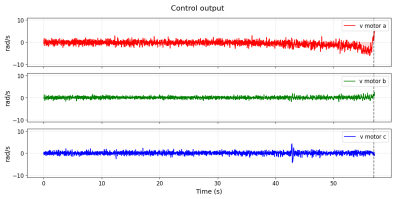

Fig. 4. Control outputs of the neural network: desired angular velocities of the globe about the roll (top), pitch (middle), and yaw (bottom) axes.

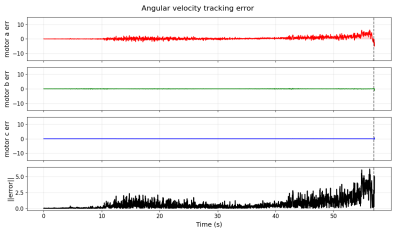

Fig. 5. Tracking error between desired and measured globe angular velocities. The first three panels show roll, pitch, and yaw errors, and the bottom panel shows the Euclidean norm of the error vector.

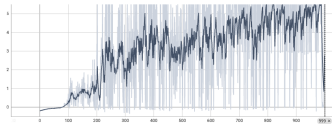

Fig. 6. Episode reward over training iterations with the planar mode. The smoothed curve highlights the underlying performance trend.

B. Results with the omniwheels modelled with spheres

In the model in which the omniwheels are represented as cylinders with spherical rollers, the agent is unable to maintain balance for more than half a second. Fig. 7 presents the trend of the trunk tilt over time. It can be observed that the robot reaches the limit angle of 20° after only 0.8 s; once this limit is reached, the system is unable to recover stability. Fig. 8 shows the norm of the trunk's linear velocity over time and the angular velocity of the trunk about the z-axis. In comparison with Fig. 3, it can be noted that both the linear and angular velocity magnitudes are larger than those observed in the linear model.

If we consider the tracking error between the desired output velocities of the motors and the actual velocities, as presented in Fig. 9, we can observe that, in this model, it is more difficult to accurately track the velocities with respect to the planar model. However, the real reason for the strong

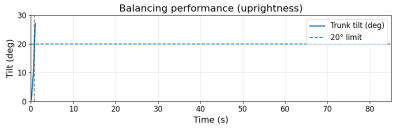

Fig. 7. Trunk tilt over time, measured via the projection of the gravity vector onto the trunk z-axis. The dashed line marks the 10° allowable tilt.

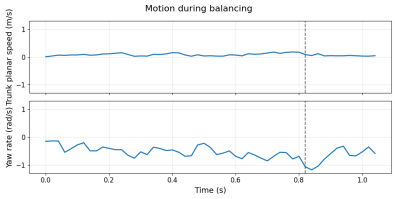

Fig. 8. Trunk motion during balancing. The upper panel depicts the magnitude of the linear velocity in the horizontal (xy) plane, and the lower panel depicts the angular velocity about the vertical (z) axis.

instability can be found by analyzing the control output of the network, presented in Fig. 10. It can be seen that the network outputs the maximum allowed velocity on one motor, thereby terminating the experiment immediately.

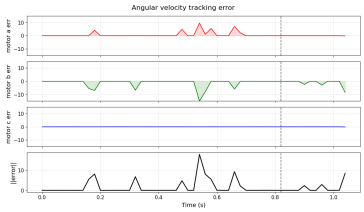

Fig. 9. Tracking error between desired and measured angular velocities of the wheels. The first three panels show the error for each motor, and the bottom panel shows the Euclidean norm of the error vector.

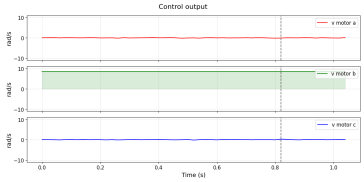

Fig. 10. Control outputs of the neural network: desired angular velocities of the three omniwheels.

By examining the trend of the reward function depicted in Fig. 11, it can be observed that this strategy of outputting the maximum velocity on one control command yields higher rewards than those obtained by actually balancing the robot.

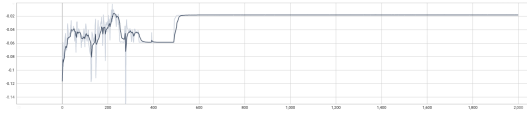

Fig. 11. Learning progress of the roller-based omniwheel ballbot model, measured by episode reward over training iterations. The smoothed curve illustrates the trend of learning performance.

It should be noted that the overall training process was heavier in this case, with a training time of circa 3 hours.

C. Results with the omniwheels modelled with capsules with anisotropic friction

Lastly, we analyze the model in which the omniwheels are represented as capsules with anisotropic friction.

As we can see from Fig. 12, the balance is asymptotically stable: the initial oscillation of the tilt angle of the trunk reduces over time, and the tilt angle never reaches the threshold of 20°. Additionally, as depicted in Fig. 13, the trunk linear and angular velocities also follow the same pattern. Moreover, differently from the previous models, the curves are smoother, indicating a smaller change in velocities, which indeed improves the overall stability.

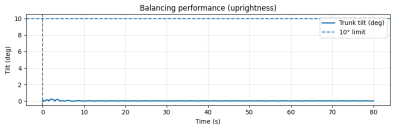

Fig. 12. Trunk tilt angle over time, obtained from the projection of the gravity vector onto the trunk z-axis.

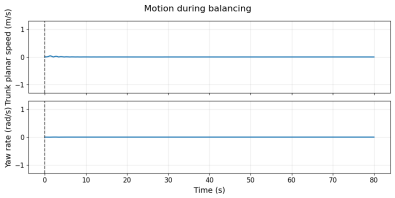

Fig. 13. Balancing dynamics of the trunk: linear velocity magnitude in the horizontal (xy) plane (top) and angular velocity about the vertical (z) axis (bottom).

An analysis of the network outputs depicted in Fig. 14 further shows that, in this framework, the control actions are smoother. After an initial transient peak, the control actions gradually converge to lower magnitudes. This behavior is consistent with that of an asymptotically stable system: once balance is achieved, only small corrective actions are required to maintain the upright configuration. As depicted in Fig. 15, the capsules track the reference velocities closely, with the residual error arising primarily from the one-step delay introduced by the discrete simulation timestep.

As shown in Fig. 16, the reward function reaches a plateau toward the end of training, indicating convergence of the learning process. In this case, the total training time was 94 minutes, which lies between those of the other two configurations considered.

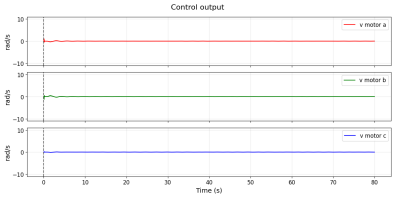

Fig. 14. Control signals generated by the neural network, showing the desired angular velocities of the three omniwheels.

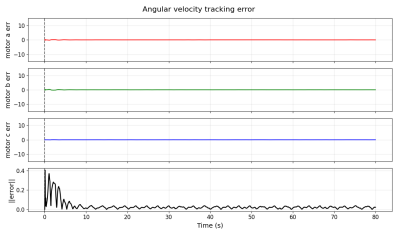

Fig. 15. Angular velocity tracking performance of the wheels. The upper three plots depict the error for each motor, and the lower plot depicts the Euclidean norm of the error vector.

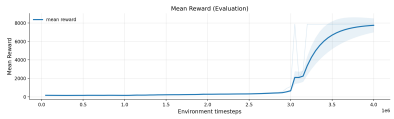

Fig. 16. Learning progress of the roller-based omniwheel ballbot model, measured by episode reward over training iterations. The smoothed curve illustrates the trend of learning performance.

Table I reports the maximum balancing time $t_{\max}$, the peak tracking error $e_{\max}$, and the total training time for the three models. The tracking error is defined as the maximum Euclidean norm of the difference between the desired and measured actuator velocities over the evaluation episode.

TABLE I

COMPARISON OF THE THREE MODELS.

Model	$t_{\max}$ **[s]**	$e_{\max}$ **[rad/s]**	**Training time**
Linear	57.3	7.0	20 min
Spheres	0.8	14.0	180 min
Capsules	80.0	0.4	94 min

IV. CONCLUSIONS

In this paper, we presented a comparative analysis of how different ballbot modeling choices affect the training of a deep RL agent. Contrary to initial expectations, the model most commonly adopted in the ballbot literature, namely the planar model, exhibits intrinsic difficulties when applied to RL. This is likely related to the fact that the roll, pitch, and yaw reference frames change with the rotation of the ball, introducing additional nonlinearities and increasing the complexity of the control problem faced by the agent. By contrast, explicitly modeling the omniwheels as cylinders with passive rollers did not provide sufficient mechanical stability for reliable policy learning in our experiments. The capsule-based model with anisotropic friction produced the best results among the considered alternatives, enabling the successful training of an agent capable of maintaining balance in an asymptotically stable manner.

This study was intentionally conducted without sensor noise, actuator noise, or domain randomization, to isolate the effects of the modeling assumptions themselves and preserve a controlled comparison between simulation models. A natural direction for future work is to bridge the sim-to-real gap by deploying the best-performing model on real hardware, using domain randomization to improve robustness.

REFERENCES

[1] JW Blonk. Modeling and control of a ball-balancing robot. Master's thesis, University of Twente, 2014.

[2] D. Chugo, K. Kawabata, H. Kaetsu, H. Asama, and T. Mishima. Development of omnidirectional vehicle with step-climbing ability. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 3, page 3849–3854 vol.3, sept 2003.

[3] Cătălin Dosoftei, Vasile Horga, Ioan Doroftei, Tudor Popovici, and Ştefan Custura. Simplified mecanum wheel modelling using a reduced omni wheel model for dynamic simulation of an omnidirectional mobile robot. In *2020 International Conference and Exposition on Electrical And Power Engineering (EPE)*, pages 721–726, 2020.

[4] Peter Fankhauser and Corsin Gwerder. Modeling and control of a ballbot. B.S. thesis, Eidgenössische Technische Hochschule Zürich, 2010.

[5] Masaaki Kumagai and Takaya Ochiai. Development of a robot balancing on a ball. In *2008 International Conference on Control, Automation and Systems*, pages 433–438, 2008.

[6] Tom B Lauwers, George A Kantor, and Ralph L Hollis. A dynamically stable single-wheeled mobile robot with inverse mouse-ball drive. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2884–2889. IEEE, 2006.

[7] Zhongyu Li and Ralph Hollis. Toward a ballbot for physically leading people: A human-centered approach. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4827–4833. IEEE, 2019.

[8] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning, 2021.

[9] Umashankar Nagarajan, George Kantor, and Ralph L. Hollis. Trajectory planning and control of an underactuated dynamically stable single spherical wheeled mobile robot. In *2009 IEEE International Conference on Robotics and Automation*, page 3743–3748, May 2009.

[10] Umashankar Nagarajan, Anish Mampetta, George A. Kantor, and Ralph L. Hollis. State transition, balancing, station keeping, and yaw control for a dynamically stable single spherical wheel mobile robot. In *2009 IEEE International Conference on Robotics and Automation*, page 998–1003, May 2009.

[11] Abdelrahman Nashat, Abdelrahman Morsi, Mohamed M. M. Hassan, and Mustafa Abdelrahman. Ballbot simulation system: Modeling, verification, and gym environment development. In *2023 Eleventh International Conference on Intelligent Computing and Information Systems (ICICIS)*, pages 198–204, 2023.

[12] Yoshito Okada, Kazuya Oguma, Kenta Gunji, Yoshiki Yokota, Hanif Aryadi, Shotaro Kojima, Ranulfo Bezerra, Masashi Konyo, Kazunori Ohno, and Satoshi Tadokoro. Fast and accurate simulation of mecanum wheels with passive rollers emulated by fixed joints and anisotropic friction. In *2023 21st International Conference on Advanced Robotics (ICAR)*, pages 592–598. IEEE, 2023.

[13] Dinh Ba Pham, Hyung Kim, Jaejun Kim, and Soon-Geul Lee. Balancing and transferring control of a ball segway using a double-loop approach [applications of control]. *IEEE Control Systems Magazine*, 38(2):15–37, 2018.

[14] Joel Roos and Marco Sütterlin. Rezero with arm: A three degrees of freedom manipulator balancing on a ball. Master's thesis, ETH Zurich, 2017.

[15] Achkan Salehi. Reinforcement learning for ballbot navigation in uneven terrain. *arXiv preprint arXiv:2505.18417*, 2025.

[16] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[17] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[18] Marvin Wiedemann. Simulation of highly dynamic omnidirectional robots in isaac sim. Slides presented at ROSCon 2023, 2023. Accessed 2025-12-14.

[19] Yifan Zhou, Jianghao Lin, Shuai Wang, and Chong Zhang. Learning ball-balancing robot through deep reinforcement learning. In *2021 International Conference on Computer, Control and Robotics (ICCCR)*, page 1–8, January 2021.