Ballbot Simulation System: Modeling, Verification, and Gym Environment Development

Abdelrahman Nashat, Abdelrahman Morsi, and Mohamed M. M. Hassan
Electrical Engineering Department
Assuit University
Assiut, Egypt
{abdelrahman.nashat, abdelrahman.morsi}@aun.edu.eg, mohamed.hassan4@eng.au.edu.eg

Mustafa Abdelrahman
School of Engineering
University of Hull
Hull, United Kingdom
murabeei@aun.edu.eg

***Abstract*—This paper introduces the development of a sophisticated simulation model for the ball-balancing robot (ballbot), implemented as a Gymnasium (Gym) environment in Python. The main purpose of this environment is to facilitate the application of Reinforcement Learning (RL) techniques to effectively control the ballbot system. Initially, a standard ballbot model is created using Solidworks, and a Unified Robot Description Format (URDF) file is generated to precisely capture the dynamics of the ballbot. To validate the model, simulations are performed within MATLAB/Simulink, comparing the ballbot URDF model against its nonlinear mathematical model with a simple PID controller. The simulation results demonstrate that the URDF model accurately represents the ballbot dynamics, exhibiting a comparable response to the mathematical model. Subsequently, the high-fidelity URDF model is integrated into the Gym environment using the Pybullet simulator. The ongoing objective of this research is to utilize the developed ballbot environment for RL-based control design.**

Keywords*—*ballbot system, modeling and simulation, control applications, Gym environment, reinforcement learning

I. INTRODUCTION

Robots typically have more than three contacts with the ground, including at least two independent driving wheels, to be statically stable with a large base for support [1]. These robots, while they can rotate at any point, cannot move freely in every direction and rotate instantly. In crowded or narrow areas, they will have trouble in avoiding obstacles.

A ball-balancing robot (ballbot) is a vertical robotic platform mounted on top of a ball that rolls on the ground thus moving the robot. Actuation is realized by a set of omni-directional wheels (omniwheels) that sit on the ball and are connected to the rest of the robot body. Compared to the wheeled ground robots, previously mentioned, which are inherently stable, the ballbot is inherently unstable. Therefore, the ballbot system must operate under an active control to be stabilized and be saved from falling. On the other hand, the same instability confers much more agility (e.g., it can move fast by tilting, and it can track any trajectory on the horizontal surface) [2].

In literature, there were different designs of the ballbot. The first design in [1] was based on inverse mouse-ball mechanism. It used drive rollers to move the ball. In [3], the design was based on inverted pendulum mechanism and used omniwheels for the first time to move the ball. This new feature made the yaw motion possible.

The ballbot dynamics are often modeled either as a simple planar model, which will be referred to as 2D model [1], [4], [5], [6] or as a coupled 3D model [7], [8], [4], [6]. This part will be covered thoroughly in Section II.

Several control methods were applied to the ballbot to control both balancing and tracking:

- Double loop controller which consists of a Linear Quadratic Regulator (LQR) controller in the outer loop and a Proportional-Integral (PI) controller in the inner loop [1], [5].
- Proportional-Derivative (PD) controller with inverse dynamic [8].
- Fuzzy-based controller [9], [10].
- Sliding mode controller [4], [11].
- Reinforcement Learning (RL) controller [12].

The controller design and its performance, especially if it depends on Artificial Intelligence (AI) as in RL, highly depends on the accuracy of the model. RL [13] is a branch of machine learning that focuses on enabling agents to learn from their interactions with an environment to achieve specific goals. RL algorithms can be used directly on the actual physical ballbot. However, this could lead to many problems: the ballbot's motors could burn out if the torque is too high or the mechanical parts could be damaged due to the constant falls of the ballbot during training. These hazards can be avoided by training using simulated ballbot rather than the physical one.

The main goal of this work is twofold. First, is to design an accurate simplified simulation system of the ballbot. Second is to build a reliable Gymnaisum (Gym) environment for the designed ballbot simulation system, which can ease the application of RL-based control technique. To achieve that, a mechanical model of the ballbot system is built using Solidworks. Then, a Unified Robot Description Format (URDF) file, which can capture the dynamics of the ballbot system, is generated. To verify the performance of the ballbot simulation system, a simple Proportional-Integral-Derivative (PID) controller is considered using MATLAB/Simulink.

The rest of the paper is organized as follows: Section II discribes the mathematical and Solidworks modeling of the

ballbot. Verification of the models is discussed in Section III. In Section IV, a reliable ballbot Gym environment is built. Finally, the conclusions are given in Section V.

II. MODELING

In this section, a brief mathematical derivation of the ballbot model will be presented as well as the description of the Solidworks implementation.

A. Mathematical Model

Two different model types are presented to describe the ballbot dynamics: the 2D model and the 3D model. Regarding the 2D model, the robot dynamics are decoupled into three 2D models: *xz*-plane, *yz*-plane, and *xy*-plane, where each one is described by two Degrees Of Freedom (DOF), one for the translation of the ball and the other one for the rotation of the bot. The three motors and omniwheels are modeled as three virtual actuating wheels, one in each plane for driving the ball. The coupling between the three 2D models is neglected, i.e., they are decoupled and hence the balancing controller consists of three independent controllers, one for each plane, see Fig. 1a.

On the other hand, for the 3D model, the coupling between the planes is considered. This makes the 3D model be described by 5 DOFs, two of them are for the position of the ball, and the other three are for the rotation of the bot. Therefore, the model can capture all aspects of the ballbot dynamics accurately and forms a more reliable basis for the controller design which can handle fast and complex maneuvers, see Fig. 1b.

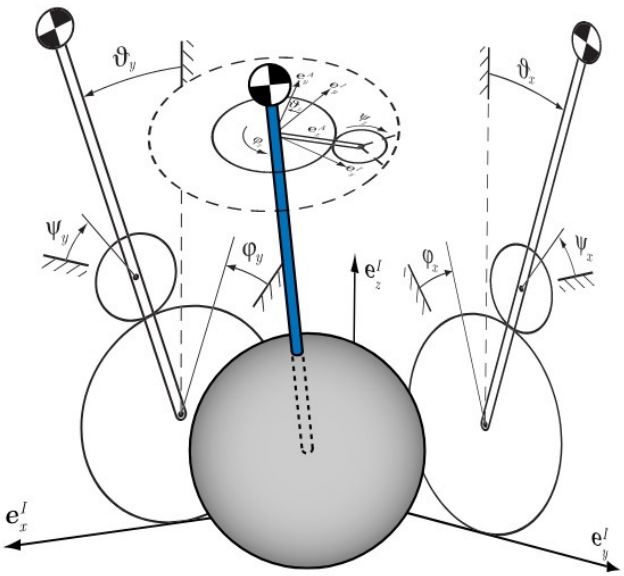
In spite of the high accuracy and reliability of the 3D model of the ballbot, it is complex and needs more computations. Therefore, in this paper, the 2D model is considered as it is much simpler and can provide a sufficient approximation to qualitatively understand the dynamics of the ballbot system and to design balancing controllers.

The dynamic modeling of the ballbot is valid under the following assumptions:

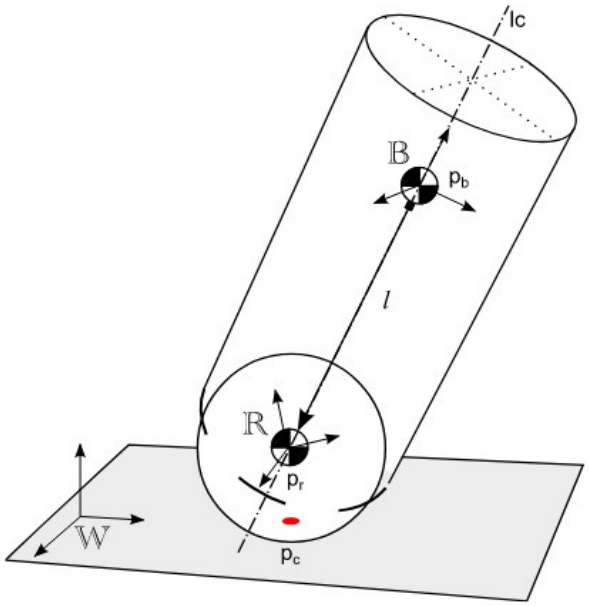
1) There is no slipping between the ball and the wheels or between the ball and the ground.
2) The motion in the *yz*-plane and *xz*-plane are decoupled and their equations of motion are symmetric.
3) The contacts between the wheels and the ball and between the ball and the ground are point contacts.
4) Rigid body dynamics are considered with no deformation.
5) The floor is assumed to be flat.
6) Only viscous friction is considered.

The dynamic equations of the 2D model can be derived using the Euler Lagrange approach by calculating the kinetic and potential energies of each plane, the whole derivation can be found in [4]. The equation of motion in the *yz*-plane is written in matrix form as follows:

$$M(q_x)\ddot{q}_x + C(q_x, \dot{q}_x)\dot{q}_x + D(\dot{q}_x) + G(q_x) = Q_x \tau_x \quad (1)$$



(a) 2D model [6].



(b) 3D model [8].

Fig. 1: 2D and 3D models of ballbot.

where $q_x = \begin{bmatrix} y_k & \theta_x \end{bmatrix}^T$ with θ_x is the body tilt angle around the *x*-axis, y_k is the ball position on the floor in the *y*-axis, and τ_x is the virtual torque delivered by the virtual wheel acting on the ball around the *x*-axis. $M(q_x)$, $C(q_x, \dot{q}_x)$, $G(q_x)$, $D(\dot{q}_x)$, and Q_x are the mass matrix, the Coriolis, the gravity, the friction, and the input vectors, respectively, which are given as follows:

$$M(q_x) = \begin{bmatrix} \frac{J_k}{r_k^2} + \frac{J_w}{r_w^2} + M_k + M_b & M_b l \cos\theta_x - \frac{J_w r_k}{r_w^2} \\ M_b l \cos\theta_x - \frac{J_w r_k}{r_w^2} & J_b + M_b l^2 + \frac{J_w r_k^2}{r_w^2} \end{bmatrix} \quad (2)$$

$$C(q_x, \dot{q}_x) = \begin{bmatrix} 0 & -M_b l \dot{\theta}_x \sin\theta_x \\ 0 & 0 \end{bmatrix} \tag{3}$$

$$G(q_x) = \begin{bmatrix} 0 \\ -M_b g l \sin\theta_x \end{bmatrix} \tag{4}$$

$$D(\dot{q_x}) = \begin{bmatrix} B_{vk}\dot{y}_k \\ 0 \end{bmatrix} \tag{5}$$

$$Q_x = \begin{bmatrix} \frac{1}{r_w} \\ -\frac{r_k}{r_w} \end{bmatrix} \tag{6}$$

where the definitions of J_k, J_w, r_k, r_w, M_k, M_b, l, J_b, g, and B_{vk} are given in Table I.

Following the assumptions given in Section II-A, the xz-plane has identical dynamics to the yz-plane. Therefore, to obtain the equation of motion in the xz-plane, θ_x, y_k, and τ_x in (1) are replaced by θ_y, x_k, and τ_y, respectively, where θ_y is the body tilt angle around the y-axis, x_k is the ball position on the floor in the x-axis, and τ_y is the virtual torque around the y-axis.

On the other hand, the equation of motion in the xy-plane is given as:

$$\ddot{\theta}_z = \frac{J_k r_k r_w}{J_k J_z r_w^2 + 3(J_k + J_z) J_w r_k^2 \sin^2\alpha} \tau_z \tag{7}$$

where θ_z is the body tilt angle around the z-axis and τ_z is the virtual torque around the z-axis, see Table I for the definitions of J_z and α.

Regarding the step of designing a balancing controller, see Section III-A, the motors' torques are considered as the control inputs. Therefore, the virtual torques, i.e., τ_x, τ_y, and τ_z, are needed to be converted to the torques of the actual wheels as follows:

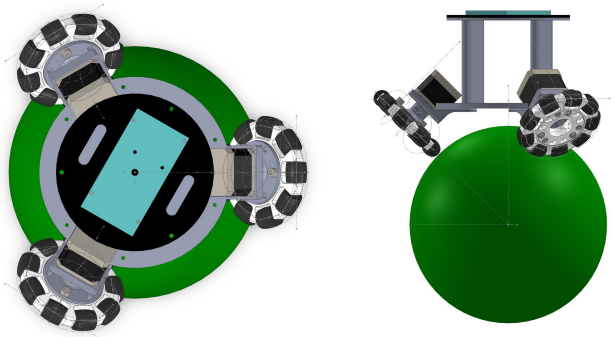
$$\begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} = \begin{bmatrix} \frac{2r_w}{3r_k\cos\alpha} & \frac{-r_w}{3r_k\cos\alpha} & \frac{-r_w}{3r_k\cos\alpha} \\ 0 & \frac{r_w}{\sqrt{3}r_k\cos\alpha} & \frac{-r_w}{\sqrt{3}r_k\cos\alpha} \\ \frac{r_w}{3r_k\sin\alpha} & \frac{r_w}{3r_k\sin\alpha} & \frac{r_w}{3r_k\sin\alpha} \end{bmatrix}^{-1} \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \tag{8}$$

B. Solidworks Model

In this subsection, a simplified mechanical representation of the ballbot, i.e., a Computer-Aided Design (CAD) file, is designed using Solidworks [14]. The design consists of three parts:

1) The body of the ballbot including the three motors which will be referred to as bot.
2) The omniwheels that apply the torque required to move the ball.
3) The ball.

For simplification, the bot, in the designed CAD file, does not include controllers nor batteries, as shown in Fig. 2. The physical properties of the CAD ballbot are given in Table II.



(a) Top view. (b) Side view.

Fig. 2: Solidworks model of the ballbot.

The most important feature of this CAD representation is to create a suitable URDF file for later use in building a full ballbot simulation environment. This file is an XML (Extensible Markup Language) specification used to model multibody systems such as robotic manipulator arms for manufacturing assembly lines and animatronic robots. It simply divides the whole robot into a series of links and joints that are responsible for the motion of these links.

This exact file has been chosen for two reasons:

1) Ease of integration between Solidworks and URDF exporter tool.
2) All available robotic simulators such as Pybullet, Gazebo, Simscape, etc. are easily integrated to import and simulate URDF files.

To fully capture the dynamics of the ballbot, two URDF files, which are considered the basis for the full ballbot simulation system, have been generated:

1) The bot file which contains four links: the bot and three omniwheels. Each omniwheel is connected to the bot via a rotating joint with three joints in total.
2) The ball file which consists of only one link, which is the ball, with no joints.

TABLE I: Description of planar model parameters.

Symbol	Unit	Description
J_k	$kg.m^2$	Inertia of the ball
J_w	$kg.m^2$	Inertia of the omniwheel
J_b	$kg.m^2$	Inertia of the bot
J_z	$kg.m^2$	Inertia of the bot around z-axis
r_k	m	Radius of the ball
r_w	m	Radius of the omniwheel
M_k	kg	Mass of the ball
M_b	kg	Mass of the bot
l	m	Distance from Center of Mass (CoM) of ball to CoM of bot
g	m/s^2	Gravity
B_{vk}	$N/(m/s)$	Viscous friction of ball to ground
α	$^\circ$	Wheel zenith angle

TABLE II: CAD ballbot physical properties.

Parameter	Value	Parameter	Value
J_k	$0.0112\ kg.m^2$	r_w	$0.05\ m$
J_w	$0.000149\ kg.m^2$	M_k	$1.437\ kg$
J_b	$0.0583\ kg.m^2$	M_b	$2.228\ kg$
J_z	$0.0137\ kg.m^2$	l	$0.142\ m$
r_k	$0.112\ m$	α	45°

The reason for this separation, between the bot and ball files, is that the ball cannot be considered as one of the links in the bot file. This is due to the tree structure of the URDF file, which means that if the bot is considered as the base link, it has three child links (omniwheels) attached to it. However, each omniwheel is attached to the ball, therefore, the ball should be a child to all three omniwheels at the same time. This is not applicable because it defies the tree structure of the URDF file. Moreover, considering the ball as the base link is also not applicable for the same reason.

III. CAD VERIFICATION

This section discusses the verification step of the CAD ballbot representation created in Section II-B.

The ballbot simulation environment is meant to be used as a tool for RL development. Therefore, it is better to perform the verification in an RL-based simulator such as Pybullet [15] or Gazebo [16]. To achieve that, a simple balancing controller, e.g., PID controller, can be applied to stabilize the bot on the ball without falling off it. However, using PID controllers in the Pybullet and Gazebo simulators can be tricky to add or develop. Moreover, failing to understand the dynamics of the physics engine can be an extra problem.

To solve this issue, we use another reliable simulator, i.e., MATLAB [17], which has built-in controller designs with an option to tune. Moreover, it has a library inside Simulink for multibody robotics which is Simscape. This is where we import the URDF files and build the environment.

In this work, a simple model based on the mathematical equations is built, see Section II-A. Then, this model is used to design a stable balancing controller for the CAD ballbot. Next, we import the CAD ballbot to the Simulink environment and apply the same controller to verify the model. In the sequel, these steps will be discussed.

A. Nonlinear Mathematical-Based Simulation

Based on (1), a simple mathematical nonlinear simulation is developed in Simulink. The choice of B_{vk}, see Table I, is arbitrary, which depends on the material of both the ball and the plane it contacts. From [2], we consider this value $B_{vk} = 49.849$.

The obtained model is used to fine-tune the PID controller using the built-in Matlab PID Tuner toolbox. The resultant parameters of the considered PID controller are $k_p = 20.39$, $k_i = 56.57$, $k_d = 1.34$, and filter coefficient $N = 327$.

Two different simulation scenarios have been considered to assess the performance of the PID controller. The first scenario is applying a step of 57.3° on the tilt angle θ_x. Fig. 3 shows the corresponding trajectories of θ_x, $\dot{\theta}_x$, y, and $\dot{y}$. It is worthy to mention that applying this step value (57.3°) is for simulation only and cannot be implemented practically. This is due to the dynamics of the ballbot, as if the tilt angle is increased over 30°, the ballbot falls immediately. The step analysis obtained from the trajectory of θ_x in Fig. 3 are as follows:

- Settling time = $0.397s$.
- Overshoot = 18%.

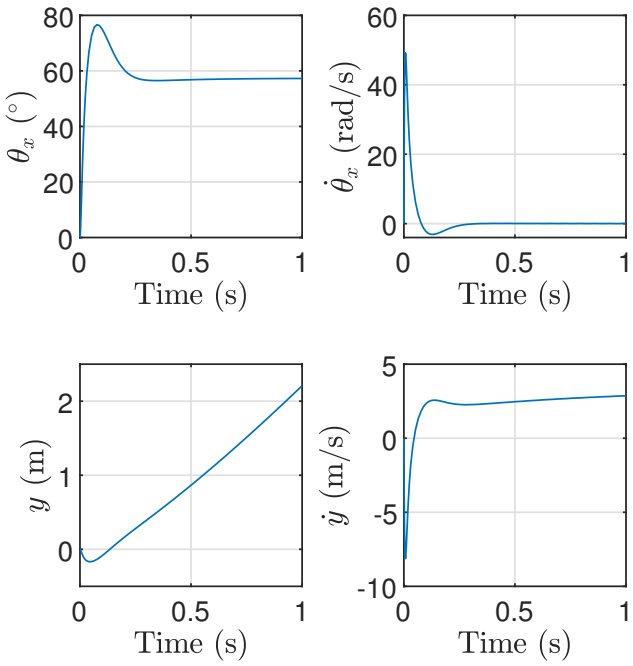


Fig. 3: Step-response of mathematical-based model.

In the second scenario, we test the balancing of the ballbot, which is achieved by applying a reference of 0° tilt angle. In this case, the PID controller achieves efficient balancing results, as shown in Fig. 4. Based on that, this PID controller can be used to validate the simulated CAD model of the ballbot.

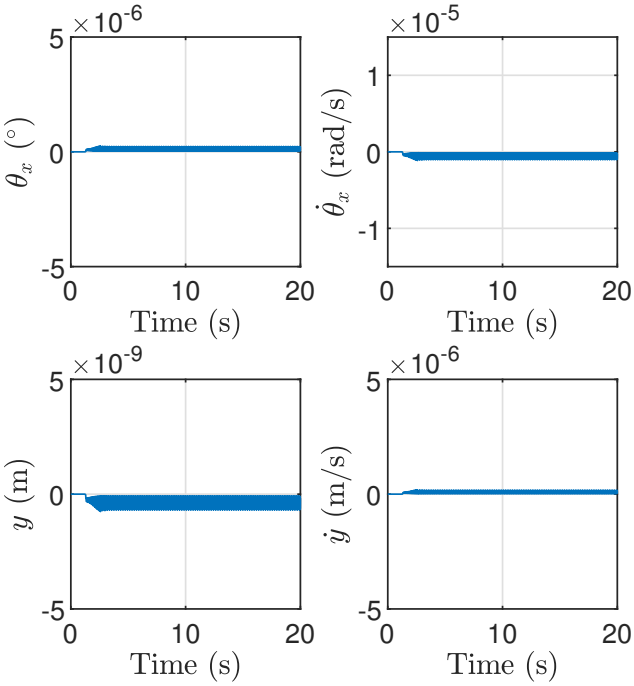


Fig. 4: Balancing trajectories of mathematical-based model.

B. CAD-Based Simulation

To construct a complete ballbot simulation system using the Simscape library within Simulink, the process involves the following steps, building upon the CAD ballbot model created in Section II-B:

1) Preparing the Simscape environment inside Simulink by specifying the world frame, configuration, solver settings, and ground plane.
2) Importing the URDF file generated by Solidworks into Simulink using the command `smimport`.
3) Testing each link and joint in the simulation to make sure everything works correctly.
4) Adding a 6-DOF block to the ball and to the bot to make them fall on the plane.
5) There is no physics engine inside Simscape library. Therefore, contacts between any two objects inside the model must be added manually. To achieve that, we add spatial contact force blocks between ball and plane, bot and plane, each omniwheel and ball, each omniwheel and bot, each omniwheel and plane, and between bot and ball. The parameters of these blocks are arbitrary and dependent on the surface materials of each object as well as the dynamics of the simulator itself. The parameters chosen for this model are given in Table III.
6) Adding sensors to measure the tilt angles of the bot, the angular velocities of the bot, and position of the ball.

After applying the previous steps, the ballbot simulation system has been successfully integrated into Simulink, see Fig. 5.

To validate the ballbot system created in Simulink, we test the balancing scenario. To achieve that, the same PID controller, used with the nonlinear mathematical model, has been applied onto both θ_x and θ_y.

Fig. 6a shows the trajectories of θ_x and θ_y during the simulation. It demonstrates that the designed ballbot system can efficiently balance within $3.16\ s$, which is an acceptable duration. It is worthy to mention that, this duration $3.16\ s$ has been measured when the trajectories of θ_x and θ_y have reached a small range of $[\pm 2^\circ]$.

The corresponding trajectories of x_k and y_k are shown in Fig. 6b. It shows that the ballbot keeps moving in order to achieve the balancing as there is no position control applied on it.

TABLE III: Spatial contact forces parameters.

Parameter	Unit	Between Ball & Omniwheel	Other pairs
Stiffness	N/m	10^6	$40,000$
Damping	$N/(m/s)$	10^8	$4,000$
Transition Region Width	m	10^{-4}	10^{-4}
Static Friction Coefficient	–	1.1	1.1
Dynamic Friction Coefficient	–	1.0	1.05
Critical Velocity	m/s	0.1	0.1

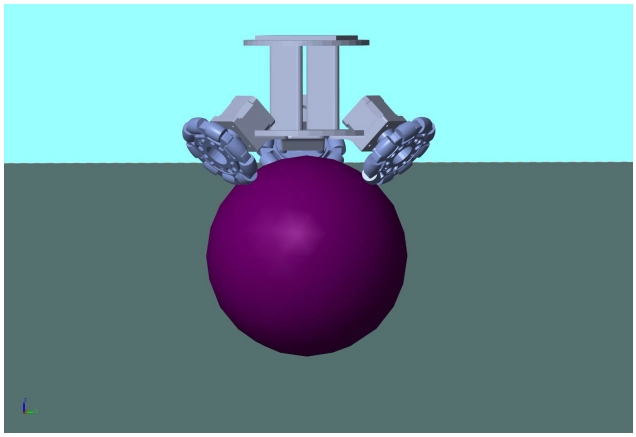


Fig. 5: Ballbot simulation system in Simulink.

Fig. 6c shows the trajectories τ_1, τ_2, and τ_3, which have very realistic and reasonable performance. Note that these trajectories continuously decrease over time to nearly $\pm 5\ N.m$ as the ballbot reaches a stable state.

To further assess the performance of the ballbot simulation system using the considered PID controller, different slight inclines of 1°, 2°, and 3° in θ_x have been applied on the ballbot initially, as shown in Fig. 7. The controller has been able to stabilize the ballbot. Therefore, the designed Solidworks model of the ballbot system can be considered as a high-fidelity model which can be used to make a robust environment for RL training.

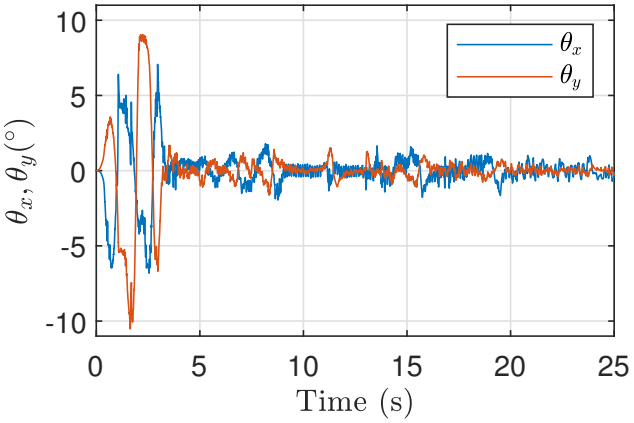
IV. BALLBOT GYM ENVIRONMENT

In this section, a reliable environment for the designed ballbot simulation system has been built. This environment can be efficiently used for training an RL agent. The agent is trained using Stable-Baselines3 (SB3) [18] Python library which is a set of reliable implementations of RL algorithms in PyTorch. Therefore, the ballbot must be wrapped in the Gym [19] environment class to be integrated with SB3. Note that, Gym is an application programming interface (API) standard for RL.

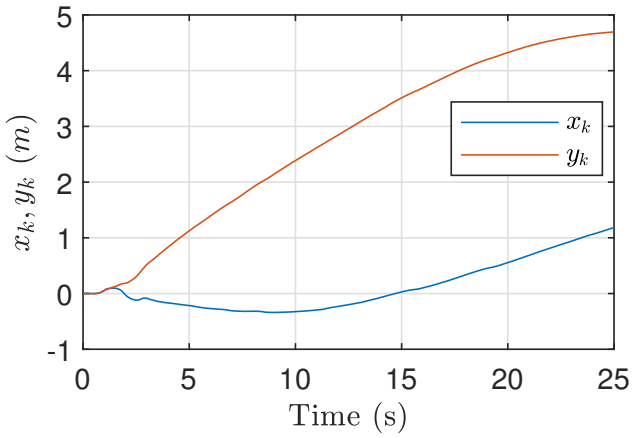
The first step is to find a suitable simulator which can visualize the ballbot and its dynamics. Note that, in Simulink, as previously mentioned in Section III-B, there is no physics engine. This lengthens the simulation time and makes Simulink inefficient to train an RL agent.

Based on [20], the search has been narrowed down into these simulators: Raisim, Nvidia Isaac, Webots, Pybullet, Gazebo, and MujoCo. Some additional properties have been added to choose the most suitable simulator among them. The desired simulator properties are as follows:

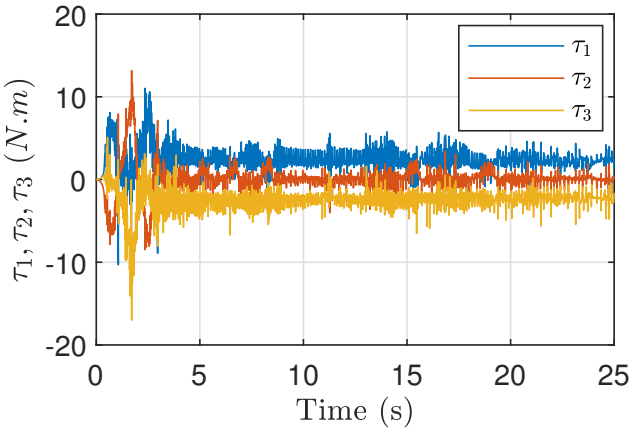
- Accurate physics engine.
- Integration with Python language and RL.
- Stable in terms of errors and code bugs.
- Lightweight in terms of processing speed.
- User-friendly.



(a) Trajectories of θ_x and θ_y.

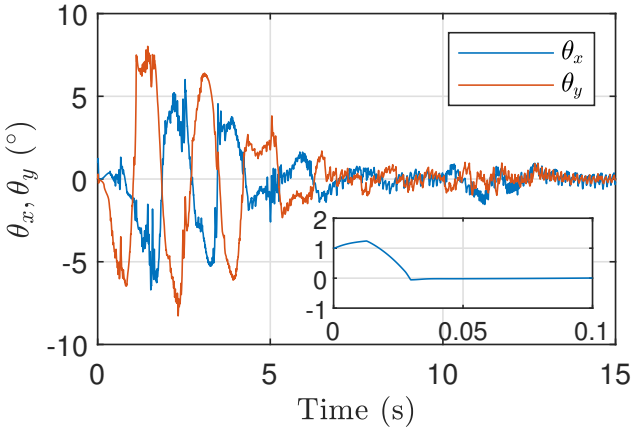


(b) Trajectories of x_k and y_k.

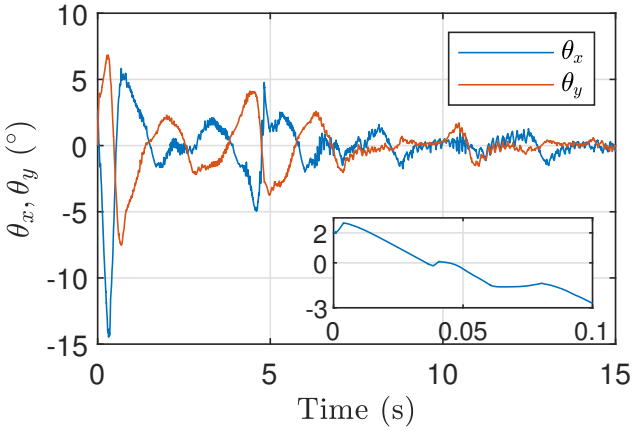


(c) Trajectories of τ_1, τ_2, and τ_3.

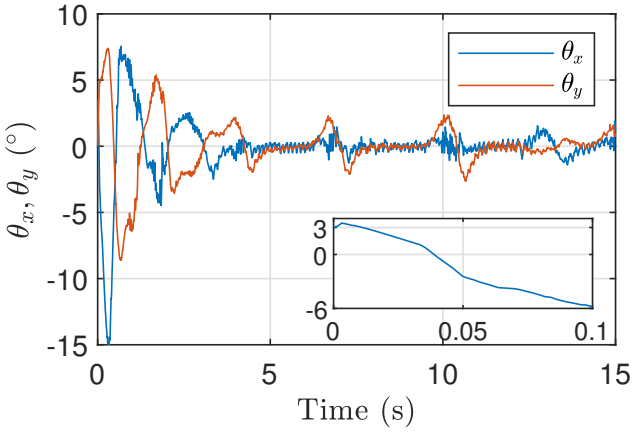
Fig. 6: Balancing trajectories of ballbot simulation system.



(a) Initial 1° incline.



(b) Initial 2° incline.



(c) Initial 3° incline.

Fig. 7: Balancing trajectories of ballbot simulation system with different initial inclines in θ_x.

- Moderate graphics capabilities.
- Available documentation.

Following this criterion, the simulator Pybullet [15] has been considered in this work.

The next step is to build a Python class to wrap the Solidworks implementation. This class includes the required functions which can be used for interfacing with the ballbot system, e.g., sensing different parameters and sending actuation commands, setting joint dynamics, and other basic functions.

Finally, the Python class is built to wrap the previous class into the Gym environment. This class provides the necessary following Gym functions: `reset()`, `step()`, `render()`,

and `close()`, as well as the initialization of the environment parameters and dynamics. The ballbot Gym environment is well integrated with SB3 but no RL agent is trained yet. Fig. 8 shows a frame of the ballbot system using the Pybullet simulator.

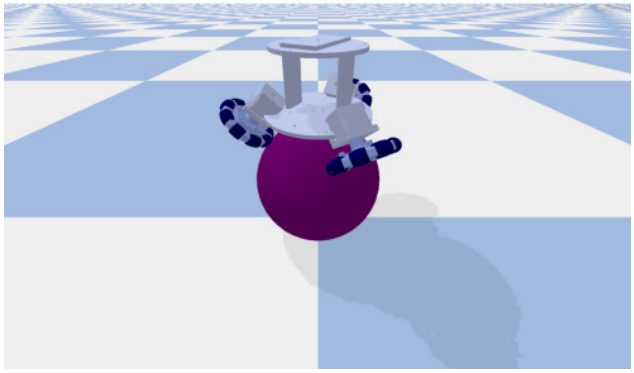


Fig. 8: Ballbot simulation system in Pybullet.

V. CONCLUSIONS

In this paper, a high-fidelity CAD model of the ballbot system has been designed. The performance of this model has been successfully validated in MATLAB/Simulink by applying a standard PID controller which has also been used with the nonlinear mathematical model of the ballbot system. The simulation results have demonstrated that the CAD model can efficiently represent the nonlinearity of the ballbot system and fully capture its dynamics. Based on the designed CAD model of the ballbot system, a reliable Gym environment has been constructed to be used with the Pybullet simulator. As a next step of this work, an RL-based control will be applied onto the designed ballbot environment. Moreover, new different scenarios will be considered in simulation and experimentally, such as: reference tracking and disturbance rejection.

REFERENCES

[1] T. B. Lauwers, G. A. Kantor, and R. L. Hollis, "A dynamically stable single-wheeled mobile robot with inverse mouse-ball drive," in *2006 IEEE International Conference on Robotics and Automation (ICRA)*, 2006, pp. 2884–2889.

[2] M. A. Alyousify, H. S. Abbas, M. M. M. Hassan, and M. H. Amin, "Parameter identification and control of a ball balancing robot," in *2022 8th International Conference on Mechatronics and Robotics Engineering (ICMRE)*, 2022, pp. 91–97.

[3] M. Kumagai and T. Ochiai, "Development of a robot balancing on a ball," in *2008 International Conference on Control, Automation and Systems*, 2008, pp. 433–438.

[4] T. K. Jespersen, "Kugle-modelling and control of a ball-balancing robot," *Master Thesis, Aalborg University, Aalborg, Denmark*, 2019.

[5] D. B. Pham, H. Kim, J. Kim, and S.-G. Lee, "Balancing and transferring control of a ball segway using a double-loop approach [applications of control]," *IEEE Control Systems Magazine*, vol. 38, no. 2, pp. 15–37, 2018.

[6] P. Fankhauser and C. Gwerder, "Modeling and control of a ballbot," B.S. thesis, Eidgenössische Technische Hochschule Zürich, 2010.

[7] A. Bonci, "New dynamic model for a ballbot system," in *2016 12th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA)*, 2016, pp. 1–6.

[8] A. N. Inal, Ö. Morgül, and U. Saranlı, "A 3d dynamic model of a spherical wheeled self-balancing robot," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5381–5386.

[9] C.-H. Chiu and W.-R. Tsai, "Design and implementation of an omni-directional spherical mobile platform," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 3, pp. 1619–1628, 2015.

[10] Y. Yunong, H.-M. Ha, Y.-K. Kim, and J. myung Lee, "Balancing and driving control of a ball robot using fuzzy control," in *2015 12th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, 2015, pp. 492–494.

[11] D. B. Pham and S.-G. Lee, "Hierarchical sliding mode control for a two-dimensional ball segway that is a class of a second-order underactuated system," *Journal of Vibration and Control*, vol. 25, no. 1, pp. 72–83, 2019. [Online]. Available: https://doi.org/10.1177/1077546318770089

[12] Y. Zhou, J. Lin, S. Wang, and C. Zhang, "Learning ball-balancing robot through deep reinforcement learning," in *2021 International Conference on Computer, Control and Robotics (ICCCR)*, 2021, pp. 1–8.

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[14] Dassault-Systèmes, "Solidworks premium 2021," 1995. [Online]. Available: https://www.solidworks.com/

[15] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016. [Online]. Available: http://pybullet.org

[16] B. Gerkey, R. T. Vaughan, A. Howard *et al.*, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th international conference on advanced robotics*, vol. 1, 2003, pp. 317–323.

[17] MathWorks, "Statistics and machine learning toolbox," Natick, Massachusetts, United States, 2023. [Online]. Available: https://www.mathworks.com/help/stats/index.html

[18] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[19] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, "Gymnasium," Mar. 2023. [Online]. Available: https://zenodo.org/record/8127025

[20] J. Collins, S. Chand, A. Vanderkop, and D. Howard, "A review of physics simulators for robotic applications," *IEEE Access*, vol. 9, pp. 51 416–51 431, 2021.