2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

DexDribbler: Learning Dexterous Soccer Manipulation via Dynamic Supervision

Yutong Hu*, Kehan Wen and Fisher Yu

***Abstract*— Learning dexterous locomotion policy for legged robots is becoming increasingly popular due to its ability to handle diverse terrains and resemble intelligent behaviors. However, joint manipulation of moving objects and locomotion with legs, such as playing soccer, receive scant attention in the learning community, although it is natural for humans and smart animals. A key challenge to solve this multitask problem is to infer the objectives of locomotion from the states and targets of the manipulated objects. The implicit relation between the object states and robot locomotion can be hard to capture directly from the training experience. We propose adding a feedback control block to compute the necessary body-level movement accurately and using the outputs as dynamic joint-level locomotion supervision explicitly. We further utilize an improved ball dynamic model, an extended context-aided estimator, and a comprehensive ball observer to facilitate transferring policy learned in simulation to the real world. We observe that our learning scheme can not only make the policy network converge faster but also enable soccer robots to perform sophisticated maneuvers like sharp cuts and turns on flat surfaces, a capability that was lacking in previous methods. Video and code are available at github.com/SysCV/soccer-player.**

I. INTRODUCTION

Artificial Intelligence is getting embedded into robotic bodies to become more accessible and useful. Among those body designs, legged locomotion stands out for its flexibility and anthropomorphic characteristics. Thanks to the advances in control theories and engineering, legged robots [1]–[3] nowadays can perform complex maneuvers like walking, running, dancing, crawling, jumping, etc., in pre-determined environments, much richer than other forms of locomotion such as wheels. Those technologies further enable the creation of humanoid robots [4], poised to revolutionize social productivity.

However, the legs can do much more than locomotion. They can also manipulate objects as well. Humans constantly move objects with their legs for work and entertainment, such as playing soccer. Similar to manipulation with arms and hands, the legs will also need to cope with the uncertainties of the interacted objects and the environment. More challenging than the arms, the legs must perform those manipulations while supporting the body. This difficulty is compounded by challenging terrains and high-degree freedom of articulated limbs, prohibiting effective solutions based on traditional control.

Recently, reinforcement learning has enjoyed great success in learning generalizable legged locomotion policy on various terrains [3]–[8]. However, despite the prevalence of legged manipulation in the natural world, few works focus on the challenging legged manipulation problem. Several learning-based methods based on model-free reinforcement learning, such as DribbleBot [1] and OP3 [9] were proposed for this task. Those policy learning try to infer limb actions from the states and objectives of manipulated objects. Unfortunately, this indirect supervision usually leads to poor performance and generalizability in the manipulation task. For example, in the soccer case, the learned policy cannot dribble the ball on smooth surfaces as the ball speed can be high, and the overrun of the robot is necessary to stop the ball.

This paper aims to design an efficient and effective learning scheme for the legged manipulation task. We mainly consider the case of ball manipulation because it is common in human world and the ball motion is complex. We observe that although it is challenging to infer the limb articulation on unknown terrains from the ball status, it is much easier to estimate the necessary body motion based on traditional control. Therefore, we propose integrating a feedback block based on PID control to estimate the body motion and supervise the policy training. This supervision is dynamic because it depends on the target states. Besides, a neural-aided Kalman Filter is used to estimate the ball states taking the fusion of information from both neural network and camera

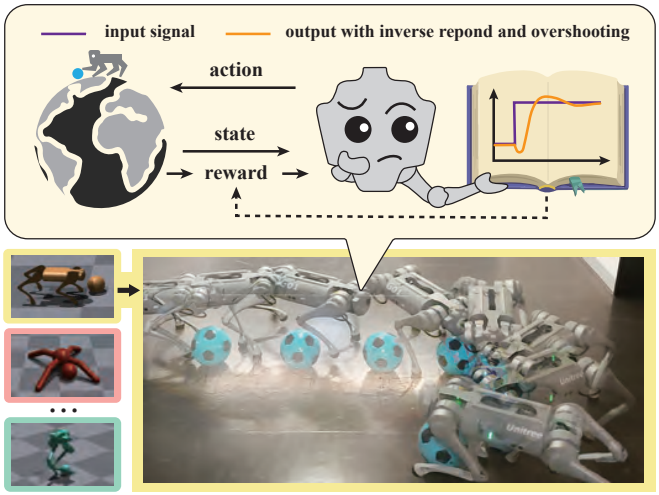


Fig. 1. **Demonstration of a DexDribbler**. Guided by a feedback controller, the robot learns pinpoint coordination between body movement and feet motion in simulator. This enables it to execute "deliberate overshooting" — a critical technique for performing sharp cuts and turns while dribbling on flat and smooth surfaces in real world.

This work was conducted by the authors while they were affiliated with the VIS Group, Computer Vision Lab, ETH Zurich, 8092 Zurich, Switzerland.
*Corresponding Author: huyuto@ethz.ch / email@youtoo.hu

projection model, and further facilitate the complex dynamic manipulation of the ball in real-world deployment.

Our main experiments are conducted on quadrupedal dribbler for robot soccer. Besides the improved results in quantitative evaluations, we find that our learning policy can perform sharp cuts and turns on flat and smooth surfaces. Our method is not only effective in the case of dynamic ball dribbling. In scenarios where the dimension of an agent's low-level action space is large, it can offer a way to integrate high-level priors into the learning process. It results in a globally more optimal policy while preserving the accuracy of low-level maneuvers. We also test our methods on multiple types of quadrupedal and bipedal robots and various terrains in simulator, to test the generalizability and adaptability of our method. Our dynamic supervision can be easily adapted to those scenarios and the resulting models perform significantly better than the state-of-the-art.

II. Related Work

A. Dynamic Object Manipulation

Reaction to an externally moving object, e.g. catching a moving ball, is the most studied form of dynamic manipulation for robot. Such tasks require the collaboration of perception, prediction, planning and control. Solutions vary from cases where fully observed object trajectories using motion capture systems [10] to partially observed ones by ego-cameras [11]. Successful demonstrations are showcased on a variety of robotics platforms ranging from robotic arms [12], legged robots [13] to drones [11].

Subsequent exploration delves into cases where the dynamic motion of the object comes from the repeated internal imposition given by robots, such as juggling balls [14], throwing objects into a box [15], pushing a block following a given path [16], and coordinated loco-manipulation [17]. However, when the manipulator is the leg instead of the arm and hand, the manipulation needs to achieve both supporting the body and actuating the targets, which leads to a multi-task policy problem.

B. Legged Locomotion

A classical approach to control quadrupedal and bipedal robots is using a model-based controller [18] to plan the joint motion while minimizing the tracking error and energy consumption. Recently, deep reinforcement learning methods [3]–[8] showed a great capability in complex and agile maneuvers without compromising real-time performance. With a carefully designed neural context estimator [4], [6]–[8], the robot can traverse multiple kind of terrain blindly with a single learned policy. Utilizing ego-centric or exteroceptive visual information [19], [20] can further improve the performance. Further, model-based approaches can be combined with learning-based methods, as a system identification wrapper for safe navigation [21], as a back-mounted block to improve action implementation accuracy [22], or as intermediate block to provide feed-forward signal to reduce the delay and error [23]. Recently, DTC [2] employs guidance from traditional controllers to enhance learned policy's precision and robustness, a concept we also embrace. However, unlike DTC's reliance on a traditional trajectory planner during deployment, our method has engraved motion references only during training, thus eliminating the need for runtime guidance.

C. Legs as Manipulators: Robot Soccer

Achieving human-level soccer skills with robots remains an enduring goal in the robotics community [24]. But most presented soccer skills by legged robots, such as kicking [25] and goalkeeping [26], use rule-based motion primitives due to the complex dynamics. Recently, by leveraging deep reinforcement learning, quadrupedal robots demonstrate the capacity to perform multiple skills separately such as dribbling a ball to a target [27] on grass and continuously dribbling a ball on multiple rough terrain [1]. Kicker [28] and Goal Keeper [29] quadruped robots can also be trained to precisely shoot a soccer ball to a target or jump to block a shoot. A series of works [9], [30], [31] successfully transfer a soccer play skills from bipedal players in simulators to simplified yard in real world. The agent can combine kicking, fall-recovering and intercepting skills to score a goal, yet can not actively keep the ball within the playing area. The demands for robots to continuously adapt to the ball's movement and strategically position themselves for redirection inspire us to propose our approach: incorporating body-level moving guidance into a learning based joint-level control policy, rather than doing policy optimization purely replying on carefully designed reward [1], [9].

III. Overview

We introduce a novel framework for teaching robots to perform dynamic ball manipulation tasks efficiently and effectively. Sec.IV presents the training phase in simulator: we take vectorized robot's pro-prioceptive results and ball's position as state observation, and using manufacturer-provided URDF model to simulate action execution result. Apart from rewards, explicit body motion guidance is calculated in a feedback formulation that takes the ball's and robot's current and target states at each timestep to direct the training of our policy network. In addtion, domain randomization are applied jointly with a context-aided estimator network to improve learning efficiency and sim-to-real transfer quality.

Upon completing training in simulator, limb articulation policy is fully embedded within the neural network, enabling a zero-shot sim-to-real transfer without the need for further motion reference or policy adjustments. A crucial component for real-world deployment is an additional measurement module that provides the ball's position from the robot's ego-perception (two onboard 210° field-of-view fisheye cameras, one facing forward and one facing downward), ensuring state input consistent with how the policy is trained. Sec.V introduces the deployment phase, including how to get the ball state estimation by combining the detected bounding boxes, the neural estimator, geometric prior and dynamic models. Sec.VI gives evaluation results in both simulated and

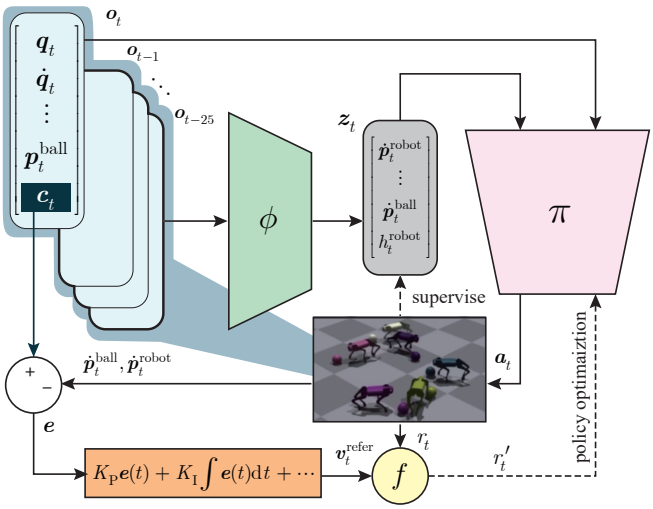


Fig. 2. **Training pipeline during learning phase in simulator.** Ground truth states can be obtained from simulator, and are used both for estimator network supervision, and for a "PID" style body speed feed-back reference computation. The other PD controller for action-to-motor control is omitted.

real-world environments showing effectiveness and overall improvements from our method.

IV. Training Phase in Simulator

A. Environment Design

1) State Definition: The whole policy network Π contains two separate blocks: the context estimator ϕ and the actor π. The input to the whole network Π is a observation set $[\boldsymbol{o}_t, \boldsymbol{o}_{t-1} \cdots, \boldsymbol{o}_{t-24}]$ consisting of the 25-step history of robot joint positions $\boldsymbol{q}$ and velocities $\dot{\boldsymbol{q}}$, ball position in body frame $\boldsymbol{p}_t$, gravity unit vector in the body frame $\boldsymbol{g}_t$, global body yaw ψ_t, and timing reference variables τ_t as defined in [3]. The commands $\boldsymbol{c}_t$ consist of the target ball velocities v_x^{cmd}, v_y^{cmd} in the global frame. The action space $\boldsymbol{a}_t$ is the target position of the twelve joints $\boldsymbol{q}_t^{cmd}$, which will be taken as input a low-level PD controller with $\boldsymbol{k}_p = 20.0, \boldsymbol{k}_d = 0.5$, adjusting the power of joint motor to perform given movement in the simulator and real world.

2) Ball-Terrain Interaction Model: The drag force on the ball as it rolls on the ground is different from the sliding friction force on the robot's foot as it comes into contact with the ground. Different from DribbleBot's approach to assume drag force proportional to the square of the ball velocity which is similar to aerodynamic drag, we follow the Relative Velocity Dependent (RVD) rolling friction model [32] where rolling friction is directly proportional to the relative angular velocity. In our case, if the ball is pure-rolling, then the relative angular velocity is directly proportional to the velocity of the ball. If the ball is not rolling purely, then the ground will be sufficiently smooth that there will be very little friction. Therefore, we can further approximate the drag effect as

$$\ddot{\boldsymbol{p}}_{ball} = \frac{\boldsymbol{F}_{drag}}{m} = \frac{-C_{drag}}{m} \cdot \dot{\boldsymbol{p}}_{ball} = -C_D \cdot \dot{\boldsymbol{p}}_{ball}. \quad (1)$$

During the simulation, we perform domain randomization to use different values of C_D within the range $[-0.1, 0.5]$ to emulate terrains with various resistance forces, such as a field with tall grass (high), pavement (low), wooden floor (close to zero) and possible inclination and unevenness (temporary negative).

TABLE I

PARAMETERS FOR DOMAIN RANDOMIZATION

Dynamics Parameter	Range	Units
Payload Mass	[-1.0,2.0]	kg
Motor Damping	[90,110]	%
Motor Stiffness	[95,105]	%
Joint Calibration	[-0.02,0.02]	rad
Robot-Terrain Friction	[0.10,2.00]	-
Robot-Terrain Restitution	[0.00,2.00]	-
Robot Center of Mass Displacement	[-0.15,0.15]	m
Mass	[0.20,0.40]	kg
Camera Frame Arrival Rate	[0.3,0.7]	-
Teleporting Position	[0.0,1.0]	m
Perturbation Velocity	[0.0,0.3]	m/s
Ball-Terrain Drag Coefficient	[-0.1,0.5]	-
v_x^{cmd}	[-1.5,1.5]	m/s
v_y^{cmd}	[-1.5,1.5]	m/s

Considering the ball alone as a dynamical system, the state-space equation of the system is

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\boldsymbol{p}\\ \dot{\boldsymbol{p}}\end{bmatrix} = \begin{bmatrix}0 & 1\\ 0 & -C_D\end{bmatrix}\begin{bmatrix}\boldsymbol{p}\\ \dot{\boldsymbol{p}}\end{bmatrix}, \quad (2)$$

where C_D determines the matrix's Eigen Value. When $C_D > 0$, the system is stable and the ball will always stop, so the robot just needs not to fall down while waiting for the ball to stop. However, for the cases of asymptotic stable and unstable when $C_D \leq 0$, the ball can not stop itself, the robot needs to make the right move at the right time in order to maintain control of the ball. This explains why although a rough terrain seems challenging for locomotion, controlling the ball on a smooth terrain turns out to be more difficult.

3) Context Definition: Prior works [4], [6]–[8] found some contextual information that is not accessible from robot's onboard sensors can be inferred explicitly or implicitly from historical action-observation responds. Assisting the network with those estimated context can be useful for sim-to-real transfer. Motivated by those prior works, we build a context-aided estimator network ϕ to better adapt the environment. We focus on three parts of the value: i) The environment parameter that changes during domain randomization, as in Table I. ii) The robot body's velocity and the body height that can't be obtained from on-board sensors. iii) The ball's true relative position and velocity, as well as the prediction of its position in the coming two time step, as a partial understanding of the ball's forward dynamic.

All the values are estimated in a explicit way $\boldsymbol{z}_t = \phi(\boldsymbol{O}_t)$ because some of the value can be useful for other module mentioned in V-B. And ϕ can be learned supervised by the ground-truth states $\boldsymbol{s}_t$ obtained from the simulator. Getting actions from a network conditioned on those estimated context $\boldsymbol{a}_{t+1} = \pi(\boldsymbol{O}_t, \boldsymbol{z}_t)$ can better cope with sim-to-real gap.

4) Reward Design: Our reward function closely follows the original DribbletBot [1], to highlight the effect of the

modules added in the pipeline instead of reward tuning. The whole reward function is consisted of: i) a Task reward for tracking the commanded ball velocity in the global reference frame. ii) Auxiliary terms encouraging the robot to be close to the ball to make interaction between the robot and the ball possible. And reward when the robot is directly facing the ball, to promote ball visibility in the camera. iii) A set of gait reward terms[1] and standard safety reward terms to induce a well-regulated gait pattern, penalize dangerous angular joint and facilitate the sim-to-real transfer.

B. Feedback Controller in the Learning Loop

Behavior to bring the ball to a stop on a rough terrain was more of the robot waiting for the ball to decelerate, rather than actively stopping the ball with the feet. However, to actively stop the ball or make the ball heading the inverse direction on a smooth terrain, robot needs to surpass the ball by a short distance first, providing proper position and enough time to take action to stop the ball from continuing its movement — an "overshoot" that would sacrificing the short-term reward, as the robot accelerates in the opposite direction from $\boldsymbol{c}_t$, but allows for greater cumulative returns.

The existing RL framework should have cope with such problem theoretically. But for this dribbling case study, the behavior failed to emerged by itself. This required some careful design of a guidance mechanism, to make the policy informed of the higher-body-level suggestion whereas the action space controlled by the policy is the lower-motor-level angle, leading the way robot controlling its own body aligned with our expecting behavior.

We note that the traditional feedback controller such as a PID controller can generate manageable overshoots. And particularly in non-minimum-phase systems, there are situations where the initial response to a control input is in the opposite direction of the desired outcome. This phenomenon, which is typically referred to as "inverse response", is inevitable in some systems and a necessary process to achieve the final goal. Inspired by this, we integrate a feedback block in the framework of RL to generate such behavior and to guide the learning of the policy, as shown in Fig. 2,

$$\begin{aligned}\boldsymbol{v}_t^{\text{refer}} = K_{\text{P}}\left(\dot{\boldsymbol{p}}_t^{\text{ball}} - \dot{\boldsymbol{p}}_t^{\text{robot}}\right) + K_{\text{I}}\int\left(\dot{\boldsymbol{p}}_t^{\text{ball}} - \dot{\boldsymbol{p}}_t^{\text{robot}}\right)\\ + K_{\text{CMD}}\left(\dot{\boldsymbol{p}}_t^{\text{ball}} - \boldsymbol{c}_t\right),\end{aligned} \tag{3}$$

where $K_{\text{P}} = 0.5, K_{\text{I}} = 4.0, K_{\text{CMD}} = 1.0$, and all other variables used for computing is obtainable from states $\boldsymbol{s}_t$. Having the reference speed given by the feedback controller, we need to integrate such guidance into the policy to make the it able to generate low level joint motion to meet the requirement of the high-level guidance. Here we shape the total reward with regard of how well current action align to the high-level speed guidance.

$$r'_t = f(r_t, \boldsymbol{s}_t) = e^{-\Delta p_t^{\text{feet}}/\sigma}\left(r_t + e^{-\left|\boldsymbol{v}_t - \boldsymbol{v}_t^{\text{refer}}\right|}\right), \tag{4}$$

[1]for the formulation of each term please refer to envs/dribble_rewards.py, all terms are also listed in the appendix of DribbleBot [1]

where $\sigma = 0.02$ and

$$\Delta p_t^{\text{feet}} = \sum_{i=1}^{4}\left(1 - \mathbb{1}_i^{\text{near}}\right)\cdot\left|\boldsymbol{p}_t^{\text{foot-}i} - \boldsymbol{p}_{\text{RH}}^{\text{foot-}i}(\boldsymbol{v}_t^{\text{refer}})\right|. \tag{5}$$

In Eq.(5), the suggested kinematic motion of every foot $\boldsymbol{p}_{\text{RH}}^{\text{foot-}i}(\boldsymbol{v}_t^{\text{refer}})$ is calculated from the traditional Raibert Heuristic Gait Generater [33] given the body velocity and time reference, and $\mathbb{1}_i^{\text{near}}$ is an indicator function which becomes 1 once the distance of the ball relative to the i^{th} foot below a predefined threshold of 10 cm. It guarantees that when the ball is close to a particular leg, the flexibility of agile maneuver is not limited.

C. Neural Network Architecture and Optimization:

We use Proximal Policy Optimization (PPO) [34] in an asymmetric way to train our soccer dribbling policy. the policy (actor) receives temporal partial observations $\boldsymbol{O}_t$ and the context vector $\boldsymbol{z}_t$ estimated from context network detached from the gradient propagation, while the value network (critic) receives the full state $\boldsymbol{s}_t$. The actor and critic are separate neural networks having three fully-connected hidden layers of the same sizes [512, 256, 128]. The context network is a neural network having two fully-connected hidden layers of sizes [512, 256], and the parameters are trained according to the MSE loss between the estimate states, with the corresponding ground truth state abtained from the simulator. As the three networks are independent of each other, the parameter sets can be jointly optimized within one gradient descent step. We use ELU as activation function for all networks.

V. Deployment Phase in Real World

During the real world deployment, the whole control pipeline can be transferred zero-shot. As the network parameters no longer require update, the policy network π and estimator network ϕ are set to inference mode with no grad propagation, and the critic network can be dropped. Most of the element in the observation vector $\boldsymbol{o}_t$ can be provide from the onboard IMU and motor encoders. Command $\boldsymbol{c}_t$ can be read from joysticks for real time control or from a predefined script for reproducible benchmark. We choose to command the robot in the local body frame of the robot at the first time step, which makes the same command direction constant and easy to interpret from the recording. The global orientation is given by the yaw ψ_t obtained from the 9-DOF IMU. Due to random magnet disturbances and accumulated shift, absolute positional drift is unavoidable, but short-term direction consistency can be guaranteed.

The only element in the observation vector that can not be obtained directly from on-board sensors is the three dimensional ball position $\boldsymbol{p}_t^{\text{ball}}$. What we can get from the onboard cameras are RGB pixel tensors $\boldsymbol{I}$. Therefore, a ball detector is required to get the position of the ball in the image. Then the detection measurement needs to be further converted to vectorized position in robot body frame — the same formulation as we get in the simulation phase.

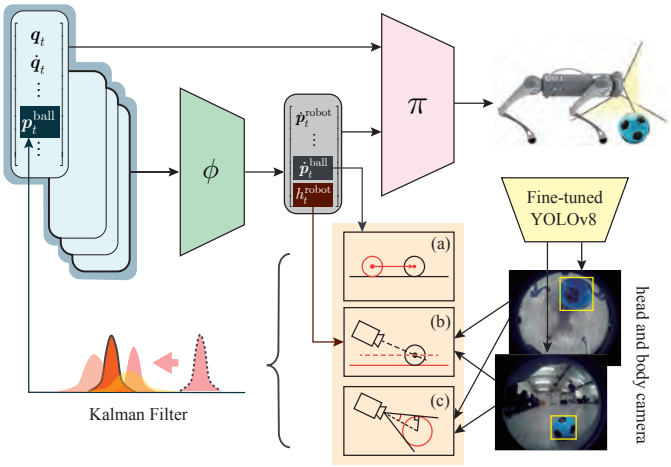


Fig. 3. **Deployment pipeline in real world.** The ball position vector is calculated by a kalman filter combining: (a) Constant Velocity Model (b) Projection-Intersection Model (c) Viewing Angle Model

A. Detection network for Ball Perception

In our project, we used a specialized ball-detection system using the YOLOv8 object detection model [35], initially trained on the Coco [36] image dataset. Due to the wide-angle fisheye lens of the camera, and drastic changes in occlusion and shading when the ball is at different positions around the body, off-the-shelf models trained on standard rectified internet images faced challenges in accurately detecting the spherical soccer ball, especially when it is at the edges of the field of view — making it easily distracted and occluded by legs. To overcome this issue, we finetune the YOLOv8 model with a dataset of 1000 manually annotated fisheye images of soccer balls captured from our robot's perspective. This dataset included various scenarios, containing images where the ball was positioned at the periphery of the frame and right under the body. We further enhance the network's robustness by employing standard image augmentation techniques, including horizontal and vertical flipping, HSV value shifting and blur effects. Finally, we are able to get detection boxes $\boldsymbol{B} = \text{YOLO}(\boldsymbol{I}) = [x_{\text{min}}, y_{\text{min}}, x_{\text{max}}, y_{\text{max}}]^{\text{T}}$ in pixel space, having 0.948 mAP@0.5.

B. Fusion of Perception result

In the equidistant fisheye model, the distance between a pixel in the image and the principal point is directly proportional to the angle of incidence: $r = f \cdot \theta$.

1) Viewing Angle Model: Given the bounding box of the ball in pixel coordinates, we first compute the approximate ball diameter in pixel by calculating the Geometric Mean of the box edge $\Delta d = \text{sqrt}\left((x_{\text{max}} - x_{\text{min}}) \cdot (y_{\text{min}} - y_{\text{min}})\right)$. This pixel distance corresponds to the angle formed between the two sides of the ball and the camera center in the world frame $\Delta\theta = \Delta d / f$, as in Fig. 3(c). Further, knowing the ball size $D_{\text{ball}} = 18\text{cm}$ in the real world, we can calculate the distance between the ball and the camera center $D = D_{\text{ball}}/2 \sin(\Delta\theta/2)$. With distance scale fixed, the three dimensional position vector of the ball in robot frame can be calculated.

2) Projection-Intersection Model: The center pixel of the ball's bounding box $P_{\text{x}} = (x_{\text{max}} - x_{\text{min}})/2, P_{\text{y}} = (y_{\text{min}} - y_{\text{min}})/2$ is corresponding to a ray that starts from the camera center having an angle $\theta = \text{sqrt}(P_{\text{x}}^2 + P_{\text{y}}^2)/f$ with camera optical axis, and an angle $\alpha = \arctan(P_{\text{x}}/P_{\text{y}})$ with the camera horizon axis. Knowing that ball only moves on the ground and ignoring the swing during the locomotion, we can get the ball's center by calculating the intersection point between the center pixel project ray and the plane of a known height, as in Fig. 3(b).

3) Constant Velocity Model: As the context estimator also output the estimated speed of the ball $\dot{\boldsymbol{p}}_t^{\text{ball}}$, we can calculated the position of the ball in the coming time step simply applying a constant velocity model $\boldsymbol{p}_t^{\text{ball}} = \boldsymbol{p}_{t-1}^{\text{ball}} + \dot{\boldsymbol{p}}_{t-1}^{\text{ball}} \Delta t$. By defining the ball state maintained by the filter (different with the "state" taken by the policy) as $\boldsymbol{x} = [\boldsymbol{p}, \dot{\boldsymbol{p}}]^{\text{T}}$. We can formulate a vectorized representation of the dynamical model:

$$\boldsymbol{x}_t = \begin{bmatrix} \boldsymbol{I}_{2\times2} & \boldsymbol{I}_{2\times2} \cdot \Delta t \\ O & \boldsymbol{I}_{2\times2} \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_{t-1} \\ \dot{\boldsymbol{p}}_{t-1} \end{bmatrix} + \boldsymbol{w} = \boldsymbol{F} \cdot \boldsymbol{x}_{t-1} + \boldsymbol{w}, \quad (6)$$

where $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{O}, \boldsymbol{Q})$ accounts for the uncertainty of the dynamic model, we set $\boldsymbol{Q}_{4\times4} = \text{diag}(0.01, 0.01, 0.2, 0.2)$

4) Kalman filter combining three models: Having four position measurements from two different cameras using two separate observation models, and one velocity measurement from the context estimation network, we can formulate the measurement process as $\boldsymbol{m} = [\boldsymbol{p}_{\text{angle}}^{\text{cam1}}, \boldsymbol{p}_{\text{angle}}^{\text{cam2}}, \boldsymbol{p}_{\text{center}}^{\text{cam1}}, \boldsymbol{p}_{\text{center}}^{\text{cam2}}, \dot{\boldsymbol{p}}_{\phi}]^{\text{T}} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{\vartheta}$, where

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{I}_{2\times2} & \boldsymbol{I}_{2\times2} & \boldsymbol{I}_{2\times2} & \boldsymbol{I}_{2\times2} & \boldsymbol{O}_{2\times2} \\ \boldsymbol{O}_{2\times2} & \boldsymbol{O}_{2\times2} & \boldsymbol{O}_{2\times2} & \boldsymbol{O}_{2\times2} & \boldsymbol{I}_{2\times2} \end{bmatrix}^{\text{T}}$$

is the measurement matrix and $\boldsymbol{\vartheta} \sim \mathcal{N}(\boldsymbol{O}, \boldsymbol{R})$ is the measurement noise, we set $\boldsymbol{R}_{10\times10} = \text{diag}(0.01, 0.01, \cdots 0.01, 0.1, 0.1)$. Considering we need the ball state before the context estimator having output, we calculated the initial ball position using only the *Viewing Angle Model* and the detection box with higher confidence from two images. After setting the initial ball speed as zero and $\boldsymbol{P}_0 = 0.01 \cdot \boldsymbol{I}_{4\times4}$, we can calculate ball state within each step using Kalman update procedure:

$$\begin{aligned} \tilde{\boldsymbol{x}}_t &= \boldsymbol{F}\hat{\boldsymbol{x}}_{t-1}, \qquad \tilde{\boldsymbol{P}}_t = \boldsymbol{F}\boldsymbol{P}_{t-1}\boldsymbol{F}^T + \boldsymbol{Q} \\ \boldsymbol{K}_t &= \tilde{\boldsymbol{P}}_t\boldsymbol{H}^T\left(\boldsymbol{H}\tilde{\boldsymbol{P}}_t\boldsymbol{H}^T + \boldsymbol{R}\right)^{-1} \\ \hat{\boldsymbol{x}}_t &= \tilde{\boldsymbol{x}}_t + \boldsymbol{K}_t\left(\boldsymbol{z}_t - \boldsymbol{H}\tilde{\boldsymbol{x}}_t\right), \ \boldsymbol{P}_t = \left(I - \boldsymbol{K}_t\boldsymbol{H}\right)\tilde{\boldsymbol{P}}_t \end{aligned} \quad (7)$$

Notice that all matrixes are partitionable and that velocity estimation $\dot{\boldsymbol{p}}_{\phi}$ is always available. When some perceptual sources are not available because of missed detection, out of view or occlusion, we can simply disable the corresponding block in matrix calculation. Therefore, the proposed Neural aided Kalman filter is a practical and convenient approach for merging multi-source estimations. However, most measurement processes are simplified to a unit mapping. Further improvements could be made by accurately representing the measurement process and the uncertainty of neural estimators.

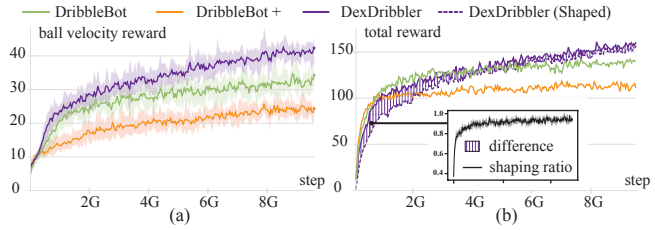


Fig. 4. **Reward curve during training phase.** (a) Task related reward term (b) Summation of all reward terms

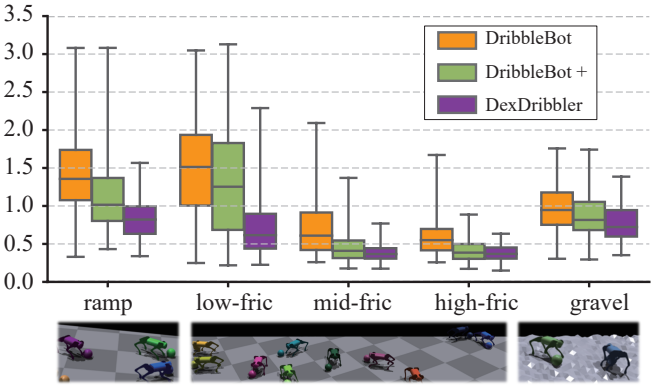


Fig. 5. **Absolute ball velocity tracking error on different terrain.**

VI. Experiments

For a comparative evaluation for solving the dribbling task as a dynamic object manipulation problem, we compared the following algorithms with access to proprioceptions only:

1) DribbleBot [1]: The policy was optimized only through roll-out returns, but during domain randomization we extend the ball-drag coefficient below zero to make sure the agent has been exposed to such (smooth and unstable terrain) cases.

2) DribbleBot+: We extend **DribbleBot**'s existing context estimator network to output all estimate-able random parameters, as an enhancement. Other parts remains the same.

3) DexDribbler: Our proposed method. The network architecture is exactly the same as **DribbleBot+**, but during the trainning phase the reward is further shaped by reference body movement generate by the feedback controller.

All the methods above were trained using the same actor and critic network architecture, same domain randomization range and fixed the initial random seeds.

A. Simulation Performance

1) Learning Performance: We used the Isaac Gym simulator [36] based on a open-source implementation of PPO [12] to synchronously train the policy, value, and estimator networks. We trained 8k domain-randomized agents in parallel. All training was conducted on a single NVIDIA TITAN XP GPU. The expected return from the critic network stabilized after 2 billion global steps, yet the reward curve continued to grow slowly with ongoing training. We documented the training curve over 10 billion global steps, equivalent to approximately 48 hours, to align with the baseline settings (7 billion global steps) and ensure that our improvements were not temporary throughout the whole training phase.

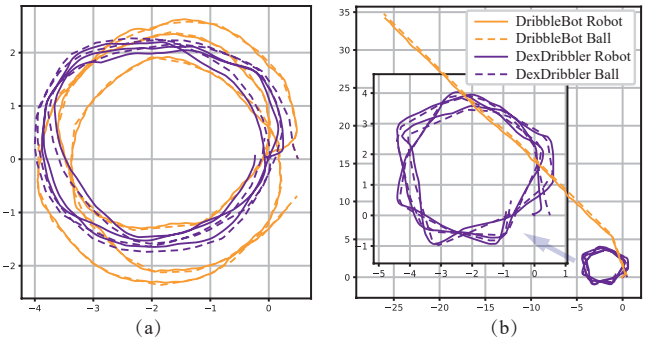


Fig. 6. **Circular trajectory following test in simulator.** (a) middle-drag terrain where $C_D = 0.2$ (b) low-drag terrain where $C_D = 0$

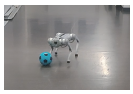
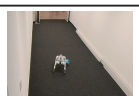
As in Fig. 4, our method always obtains the largest reward regarding the ball velocity error within the same number of iteration steps, meaning that the agent can control the ball the most precisely in the environment compared to others. As for the total reward, we plot the reward before and after guidance shaping. Our method does not get the maximum total reward at the beginning, but as the body movement created by our behavior gradually matches the movement supervision from the feed-back controller, the agent can find a overall more optimal policy, and makes total reward eventually exceeds that of the other methods.

2) Ball Controllability on Different Terrain: We evaluate the command tracking performance in the same simulation environment to evaluate the final performance in with-in distribution cases. The ball-drag coefficient C_D is sample between [-0.1,0.1), [0.1,0.3) and [0.3,0.5] to simulate low, mid and high drag terrain respectively. We add two more environments: a 3° ramp and a gravel environment to simulate the out-of-distribution cases. The robot was given random commands for 40 seconds, and the commands were uniformly sampled from [-1.0m/s, 1.0m/s] every ten seconds. For fair comparison, random commands were generated using the same random seed for each policy. Robot controlled by each policy runs 1000 times with different random seeds to verify repeatability. We measured Absolute Tracking Error (ATE) as the performance metric and constructed as shown in Fig. 5. Our method consistently outperforms the baselines. Moreover, We can see that as the ball-drag coefficient increases, several methods can perform dribbling more accurately and the difference between them reduces. This confirms our analysis mentioned in Sec. IV-A that high-drag terrain actually makes the task easier.

3) Trajectory Following Test: We generate the velocity commands to follow a circle trajectory with diameter of 5 meters. Notice that the commands are applied in velocity space, and in position space the system can be considered open-loop, so drift always exists. Nevertheless, the size of the drift error in position space can reflect the following accuracy in velocity space. On high-drag terrain as shown in Fig.

TABLE II

REAL-WORLD DRIBBLING PERFORMANCE EVALUATION.

Smooth			Tile		
DribbleBot	0/5		DribbleBot*	4/4	
DexDribbler	**5/5**		DexDribbler	5/5	
Grass			Gravel		
DribbleBot*	4/4		DribbleBot*	4/4	
DexDribbler	5/5		DexDribbler	5/5	
Ramp-Up			Ramp-Down		
DribbleBot*	0/4		DribbleBot*	0/4	
DexDribbler	**4/5**		**DexDribbler**	**1/5**	

* cited from [1]

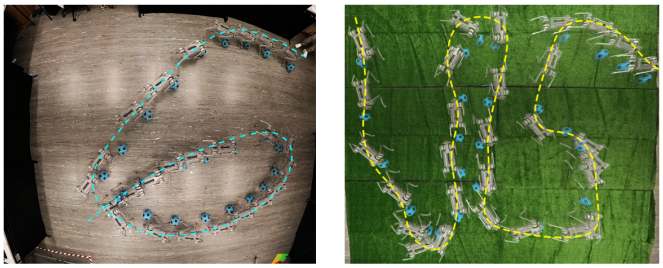


Fig. 7. **Real-world trajectory following test.** The robot is controlled under teleportation to create some recognizable trajectory, "6" on smooth terrain and "VIS" on grass terrain.

6(a), both method can enable robot successfully follow the trajectory, but our method's trajectory has larger overlapping part. However, on low-drag terrain, the baseline method lose the control of the ball. As a result, the ball keeps moving in one direction. Our method, although becoming less accurate, still keeps the ball rolling within a circular trajectory.

B. Real World Performance

We use the Unitree Go1 robot and a size 3 soccer ball for all realworld experiments. We zero-shot transfer the policy learned in simulator to the real world, as metioned in Sec.V-B. While Dribbling in the real world, the locomotion policy must adapt when the terrain causes the feet to slip or stumble. To keep the ball within control, it additionally needs to do running and kicking adjustment depending on how the ball interacts with the terrain, E.g., on grass, high drag tends to slow down the rolling ball; on smooth floor, low drag may cause it to speed away from the robot; on gravel, the ball changes direction unpredictably as it impacts the terrain surface.

1) Quantitative Results on Diverse Terrains: We quantitatively evaluate the success rate of the fully autonomous behavior of the dribbling policies while executing a scripted trajectory across diverse terrains: the set of [Tile, Grass, Gravel] terrains that Dribblebot already have been tested on, a [Slope] case that have not been solved, and a [Smooth floor] case that is not included in their test. The robot is commanded with a predetermined trajectory: dribble forward at 1.0 m/s for 5 s, then stop the ball. As we don't integrate a auto-recovery maneuver, the failure cases can be easily judged: if the distance between robot and ball exceed 0.5m, or the robot falls. This is a slightly different testing method than the Dribblebot's "turn-back" testing. We did this because the "stopping state" is crucial, as switching to any dribbling direction is easy when the ball stops, and the ability to stop the ball reflects the overall ball controllability. Additionally, dribbling uphill and downhill are two cases, each presenting their unique challenges, thus seperating them into two dribble-and-stop process allow for a more fine-grained evaluation of performance.

As in Table II, in scenarios where Dribblebot has been tested, our method performed the same as they did. On the smooth terrain, Dribblebot can only follow the ball and run behind it, but cannot make it stop. Our method can make a quick cut-off to bring the ball to stop. When dribbling up-hill a ramp, our method can respond quickly to ball shift, blocking the ball from rolling back to its starting point. However, when dribbling down-hill, our method often pitches forward and fall with face when trying to intercept a ball that is accelerating and rolling down. We find that covering random gravity direction during training can address this issue, but for a fair comparison we still report the results of zero-shot deployments under the uniform setting.

2) Trajectory Following Test: We qualitatively evaluate our dribbling controller under teleoperation on diverse terrains with different ball-terrain dynamics. Because our system operates without a tether or external sensing, long-term safety could not be guaranteed when following only open-loop speed command. Therefore, we make robot receive commands from a joystick manipulate by human as a close-loop spotter. Yet, the policy still needs to be accurate enough and make agile large-angle turns at corners to create recognizable trajectory graphics, especially on smooth and uneven surfaces. Fig. 7 shows stitched overhead photos to illustrate the real-world dribbling performance.

C. Generalizability on similar tasks

By simply adapting the action dimension and specifying the foot index number, our training pipeline and the original Dribblebot's approach are both naturally compatible for ball dribbling tasks across different legged robot configurations. When we replaced the Unitree Go1 robot with Cassie, a popular large-size bipedal robot, and NAO, a smaller bipedal robot used in the RoboCup Standard Platform League, our method demonstrated a 39.2% and 11.6% higher final task-return than Dribblebot's, respectively, after 10 billion training steps in a simulator. These results highlight that the improvements from our method are not limited to a single robot model and show our method's potential for training soccer robots that are ready to compete under the official rules and field conditions of RoboCup.

VII. CONCLUSION AND FUTURE WORKS

We propose a framework that allows high-level dynamic supervision to guide complex limb articulation policy learning, enabling robot to learn rapid turning responses for real-world dribbling tasks and become a DexDribbler. It shows improved

performance compared to existing learning-based approaches, and shows capability to keep a even naturally unstable ball-surface system within control. The integration of feedback control within RL framework not only enhances specific skills learning like dribbling, but also has potential broader applications across various complex robotic tasks.

However, it still has a number of limitations which we hope to explore and improve upon in future work: (1) Deeper integration between model-based and data-driven method, as discussed in V-B. (2) Multi-task soccer player: recovering [1] shooting [28] and goalkeeping [29] skills could be integrated to create a fully autonomous soccer agent. (3) Awareness of other object or agent: Future work could incorporate more information about the environment geometry as well as realize high-level counter-play or cooperation with other agents. Ultimately, our aim is to develop robots that could potentially compete with humans in the near future.

REFERENCES

[1] Y. Ji, G. B. Margolis, and P. Agrawal, "DribbleBot: Dynamic Legged Manipulation in the Wild," *arXiv*, Apr. 2023.

[2] F. Jenelten, J. He, F. Farshidian, and M. Hutter, "Dtc: Deep tracking control," *Science Robotics*, vol. 9, no. 86, 2024.

[3] G. B. Margolis and P. Agrawal, "Walk these ways: Tuning robot control for generalization with multiplicity of behavior," in *Conference on Robot Learning*. PMLR, 2023.

[4] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, Jan. 2019.

[5] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Conference on Robot Learning*. PMLR, 2022.

[6] I. M. A. Nahrendra, B. Yu, and H. Myung, "Dreamwaq: Learning robust quadrupedal locomotion with implicit terrain imagination via deep reinforcement learning," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[7] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "RMA: Rapid Motor Adaptation for Legged Robots," in *Robotics: Science and Systems XVII*, Jun. 2021.

[8] G. Ji, J. Mun, H. Kim, and J. Hwangbo, "Concurrent Training of a Control Policy and a State Estimator for Dynamic and Robust Legged Locomotion," *IEEE Robotics and Automation Letters*, vol. 7, Apr. 2022.

[9] T. Haarnoja, G. Lever, S. H. Huang, D. Tirumala, J. Humplik, M. Wulfmeier, S. Tunyasuvunakool, N. Y. Siegel, R. Hafner, M. Bloesch, K. Hartikainen, A. Byravan, L. Hasenclever, Y. Tassa, F. Sadeghi, N. Batchelor, F. Casarini, S. Saliceti, C. Game, N. Sreendra, K. Patel, M. Gwira, A. Huber, N. Hurley, F. Nori, R. Hadsell, and N. Heess, "Learning agile soccer skills for a bipedal robot with deep reinforcement learning," *Science Robotics*, vol. 9, no. 89, p. eadi8022, 2024.

[10] K. Dong, K. Pereida, F. Shkurti, and A. P. Schoellig, "Catch the Ball: Accurate High-Speed Motions for Mobile Manipulators via Inverse Dynamics Learning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020.

[11] K. Su and S. Shen, "Catching a Flying Ball with a Vision-Based Quadrotor," in *2016 International Symposium on Experimental Robotics*, ser. Springer Proceedings in Advanced Robotics, 2017.

[12] S. Kim, A. Shukla, and A. Billard, "Catching Objects in Flight," *IEEE Transactions on Robotics*, vol. 30, Oct. 2014.

[13] B. Forrai, T. Miki, D. Gehrig, M. Hutter, and D. Scaramuzza, "Event-based Agile Object Catching with a Quadrupedal Robot," *arXiv*, Apr. 2023.

[14] K. Ploeger, M. Lutter, and J. Peters, "High acceleration reinforcement learning for real-world juggling with binary rewards," in *Conference on Robot Learning*. PMLR, 2021.

[15] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "TossingBot: Learning to Throw Arbitrary Objects With Residual Physics," *IEEE Transactions on Robotics*, vol. 36, Aug. 2020.

[16] A. Heins, M. Jakob, and A. P. Schoellig, "Mobile Manipulation in Unknown Environments with Differential Inverse Kinematics Control," in *2021 18th Conference on Robots and Vision (CRV)*, May 2021.

[17] Z. Fu, X. Cheng, and D. Pathak, "Deep whole-body control: learning a unified policy for manipulation and locomotion," in *Conference on Robot Learning*. PMLR, 2023.

[18] G. Bledt, M. J. Powell, B. Katz, J. Di Carlo, P. M. Wensing, and S. Kim, "MIT Cheetah 3: Design and Control of a Robust, Dynamic Quadruped Robot," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2018.

[19] A. Loquercio, A. Kumar, and J. Malik, "Learning visual locomotion with cross-modal supervision," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[20] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, Oct. 2020.

[21] Z. Li, J. Zeng, A. Thirugnanam, and K. Sreenath, "Bridging Model-based Safety and Model-free Reinforcement Learning through System Identification of Low Dimensional Linear Models," *arXiv*, May 2022.

[22] S. Lyu, H. Zhao, and D. Wang, "A composite control strategy for quadruped robot by integrating reinforcement learning and model-based control," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

[23] S. Zhou, M. K. Helwa, and A. P. Schoellig, "Deep neural networks as add-on modules for enhancing robot performance in impromptu trajectory tracking," *The International Journal of Robotics Research*, vol. 39, Oct. 2020.

[24] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa, "RoboCup: The Robot World Cup Initiative," in *Proceedings of the First International Conference on Autonomous Agents*, Feb. 1997.

[25] M. Veloso, W. Uther, M. Fijita, M. Asada, and H. Kitano, "Playing soccer with legged robots," in *1998 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 1998.

[26] M. Friedmann, J. Kiener, S. Petters, D. Thomas, O. Von Stryk, and H. Sakamoto, "Versatile, high-quality motions and behavior control of a humanoid soccer robot," *International Journal of Humanoid Robotics*, vol. 05, Sep. 2008.

[27] S. Bohez, S. Tunyasuvunakool, P. Brakel, F. Sadeghi, L. Hasenclever, Y. Tassa, E. Parisotto, J. Humplik, T. Haarnoja, R. Hafner, M. Wulfmeier, M. Neunert, B. Moran, N. Siegel, A. Huber, F. Romano, N. Batchelor, F. Casarini, J. Merel, R. Hadsell, and N. Heess, "Imitate and Repurpose: Learning Reusable Robot Movement Skills From Human and Animal Behaviors," *arXiv*, Mar. 2022.

[28] Y. Ji, Z. Li, Y. Sun, X. B. Peng, S. Levine, G. Berseth, and K. Sreenath, "Hierarchical Reinforcement Learning for Precise Soccer Shooting Skills using a Quadrupedal Robot," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2022.

[29] X. Huang, Z. Li, Y. Xiang, Y. Ni, Y. Chi, Y. Li, L. Yang, X. B. Peng, and K. Sreenath, "Creating a dynamic quadrupedal robotic goalkeeper with reinforcement learning," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

[30] A. Byravan, J. Humplik, L. Hasenclever, A. Brussee, F. Nori, T. Haarnoja, B. Moran, S. Bohez, F. Sadeghi, B. Vujatovic, and N. Heess, "NeRF2Real: Sim2real Transfer of Vision-guided Bipedal Motion Skills using Neural Radiance Fields," *arXiv*, Oct. 2022.

[31] S. Liu, G. Lever, Z. Wang, J. Merel, S. M. A. Eslami, D. Hennes, W. M. Czarnecki, Y. Tassa, S. Omidshafiei, A. Abdolmaleki, N. Y. Siegel, L. Hasenclever, L. Marris, S. Tunyasuvunakool, H. F. Song, M. Wulfmeier, P. Muller, T. Haarnoja, B. Tracey, K. Tuyls, T. Graepel, and N. Heess, "From motor control to team play in simulated humanoid football," *Science Robotics*, vol. 7, Aug. 2022.

[32] Y. C. Zhou, B. D. Wright, R. Y. Yang, B. H. Xu, and A. B. Yu, "Rolling friction in the dynamic simulation of sandpile formation," *Physica A: Statistical Mechanics and its Applications*, vol. 269, Jul. 1999.

[33] M. H. Raibert and E. R. Tello, "Legged Robots That Balance," *IEEE Expert*, vol. 1, Nov. 1986.

[34] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv*, Aug. 2017.

[35] J. Terven and D. Cordova-Esparza, "A Comprehensive Review of YOLO: From YOLOv1 and Beyond," *arXiv*, Oct. 2023.

[36] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," in *ECCV 2014*, 2014, vol. 8693.