Learning for Dancing Quadruped Robot

Yaosheng Xu
yaoshengxu@g.harvard.edu

Hao Tang
haot@mit.edu

Shenzhe Yao
yaoshzh@mit.edu

Abstract

Controlling robots to imitate human dancing motions is a widely studied topic in robotics. Recent advances in generative machine learning enable the automatic generation of human rhythmic motions given an audio input. In this work, we developed a reinforcement learning method that controls a quadruped robot agent to realize the rhythmic motions generated from input music. The reinforcement learning model is deployed in both the simulation environment and a physical robot, providing reasonable dancing motions. We provide our code in this repository. [1]

1 Introduction

Dancing, the rhythmic motion of the human body with music, is an important part of multiple cultures in the world. Both designing and performing dancing are challenging tasks for humans that require long-term training and high-level skills. Therefore, realizing the autonomous dancing generation by robots is a desirable idea that attracts broad research interests [1-3]. In order to generate dancing actions from music as audio input, two models need to be realized (Fig. 1). The first is the audio model that generates target reference poses of the robot agent according to input music, and the second is the dancing agent controls the robot to follow the reference poses [4,5]. Previously, people have achieved the audio model using generative models trained on the AIST++ dataset that includes a large amount of human dancing data [6,7]. The capability of the audio model to generate new dancing motions cast challenges to the dancing agent, which needs to follow unseen motions instead of just conducting hard-coded actions. To the best of our knowledge, a reinforcement learning dancing agent that realizes the generated human dancing motion in quadruped robots remains lacking.

In this work, we developed a reinforcement learning method that controls a quadruped dancing agent to follow the dancing motions generated by the audio model. Combining our method with the audio model in the previous work, we trained the dancing agent in Isaac Gym virtual environment that performs rhythmic motion with music. We also deployed our model on a physical robot that appears reasonable dancing motion despite its deviation from the reference poses.

2 Related Work

Generative models for human dance poses conditioned on audio inputs are recently studied in Ref. [6,7], where various dancing motions are produced to match the input music. In Ref. [6], a probabilistic autoregressive architecture of a neural network model is designed to predict future poses based on the previous poses and audio input. Both AIST++ [8] and GrooveNet [9] human dancing datasets are used in their model training, yielding improved performance considering beat alignment and realism metrics compared to baseline agents. More recently, the Editable Dance Generation

[1] https://github.com/APM150/Dancing-Robot

6.8200 Computational Sensorimotor Learning Final Project (Spring 2023).

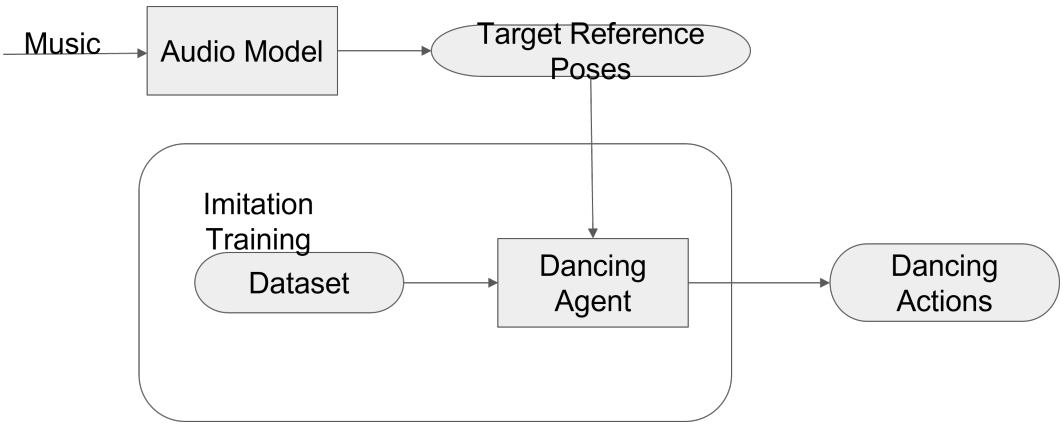

Figure 1: Overall Structure of the method to generate dancing actions from audio inputs.

(EDGE) model developed in Ref. [7] employs a transformer-based diffusion model combined with the Jukebox music feature extract. The EDGE model achieves state-of-the-art performance on multiple metrics including physical plausibility, beat alignment, and pose sequence diversity. In this work, we adopt the EDGE model to generate reference poses from music inputs.

On the other hand, controlling quadruped robots to imitate agile locomotion is studied in Ref. [4,5]. Ref. [4] employs proximal-policy optimization (PPO) algorithm to conduct reinforcement learning that matches the robot's poses to quadruped animals' poses. The reward function is defined as the similarity between the robot's poses and the animal's poses to imitate, which is measured by the difference between a set of key points on their bodies. More recently, a similar idea is employed to train a local motion policy that can generalize with the multiplicity of behaviors [5]. Diverse motions of the robots are realized that enable the agent to walk in different environments. We adjust the algorithm, reward, and state definition in Ref. [5] to train our dancing agent model, which imitates the human dancing motion.

3 Method

Our method aims to provide a controller policy $\pi(a_t|h_t, s_t)$, where a_t, h_t^{target}, and s_t specifies the action on robot's joints, reference poses to imitate, and state. The key points on the robot are trained to match the key points on the reference poses. For simplicity, we specify the key points just by the foot heights of the human body during dancing. As the robot has 4 foot heights, $\vec{h}_t$ is a four-dimensional vector. It is fitted to the target:

$$\vec{h} = (h^{fl}, h^{fr}, h^{rl}, h^{rr}) \rightarrow \vec{h}^{target} = (h^l, h^r, 0, 0) \tag{1}$$

where $h^{fl}, h^{fr}, h^{rl}, h^{rr}$ are the front-left, front-right, rear-left, rear-right foot heights, respectively, h^l and h^r are the targets left and right foot height of the human body extracted from the reference poses. That means we control the two rear feet to stay on the ground, and the two front foot heights are matched to the foot heights of human reference poses.

The state s_t is defined following the convention in Ref. [5], which specifies the current state including joint positions, velocities, gravitation vector in the body frame, and a behavior vector specifying task completion on the past 30 timesteps. The only difference is that we added the real foot heights $\vec{h}_t$ to the behavior parameters, which better specifies the state with respect to task completion. The action a_t is a vector of position targets of all joints of the robot, which is the input signal to control the physical robot. The robot has 12 joints controlled by a proportional-derivative controller with parameters $k_p = 20, k_d = 5$. The action is a zero vector when the position targets are at the nominal joint position, and $\vec{a}_t$ corresponds to the deviation of the position targets from the nominal joint position. The policy function $\pi_\theta(a_t|h_t, s_t)$ is realized by a multilayer perceptron (MLP) including three hidden layers with layer sizes of 512, 256, and 128, respectively. The value network in the PPO algorithm uses an MLP including two hidden layers with layer sizes of 256 and 128. The ELU activation function is used in both networks.

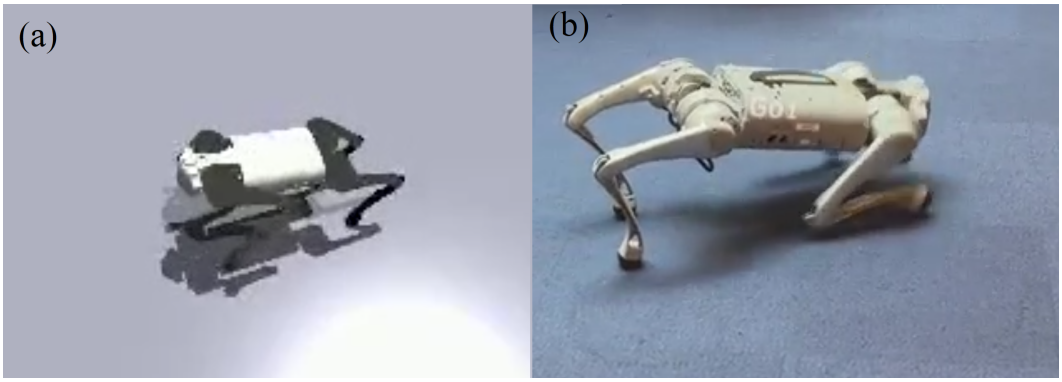

Figure 2: (a) Configuration of the robot in the simulation environment. (b) Image of the physical robot adopted in our study.

In the training process, the reward function includes both task rewards and auxiliary rewards. The auxiliary rewards aim to encourage the robot to move smoothly and naturally according to its body structure. The auxiliary rewards include augmented auxiliary and fixed auxiliary defined in the same way as Ref. [5], while we define the task reward according to the special feature of this project. The task reward from the front foot heights is given as:

$$r_f = c_f(e^{-\lambda_f|h^{fl}-h^{l,target}|} - 1) + c_f(e^{-\lambda_f|h^{fr}-h^{r,target}|} - 1) \tag{2}$$

where $\lambda_f = 10$ and $c_f = 30$. These rewards encourage the front foot heights to match the target foot heights. When the foot heights are perfectly matched, the reward equals zero. The deviation between the two sets of foot heights is negatively related to the rewards.

The two rear foot heights are defined as:

$$r_r = c_r(h^{rl})^2 + c_r(h^{rr})^2 \tag{3}$$

where the coefficients $c_r = -30$. That is a mean square loss that provides a penalty when the rear foot heights are not zero.

Combining each part of the rewards makes the model control the robot to move in a smooth, natural way and keep the front feet moving in a similar way with the reference pose generated from the music while keeping the rear feet on the ground. The motion of the front feet, together with the reference poses, then moves commensurate with the music.

The proximal policy optimization algorithm implemented in Isaac Gym [10] is employed to train the agent. The discount factor $\gamma = 0.99$, the GAE factor $\lambda = 0.95$, the clip range $\epsilon = 0.2$. An entropy bonus is added with $\alpha = 0.01$ to encourage exploration, and the training is implemented with Adam optimizer using a learning rate of 10^{-3}.

When converting the foot heights read from the reference poses of human dancing to robot foot heights, the value h^{target} is normalized to a given height:

$$\vec{h}^{target} = h_0 \frac{\vec{h}^{target}_{human}}{\max \vec{h}^{target}_{human}} \tag{4}$$

The maximal reachable foot height of the robot is $h_m = 0.5$ m. We tried a different setting of the normalization factor h_0, using either h_m or a larger value of 2 m.

4 Results

4.1 Training curve and foot heights comparison

The agent is trained in the Isaac Gym simulation environment to maximize the designed rewards. The presented training data is obtained via normalizing the foot heights to the range of $[0, 2]$ m. We first see the training loss and foot heights of the agent during and after the training process. The adaptation

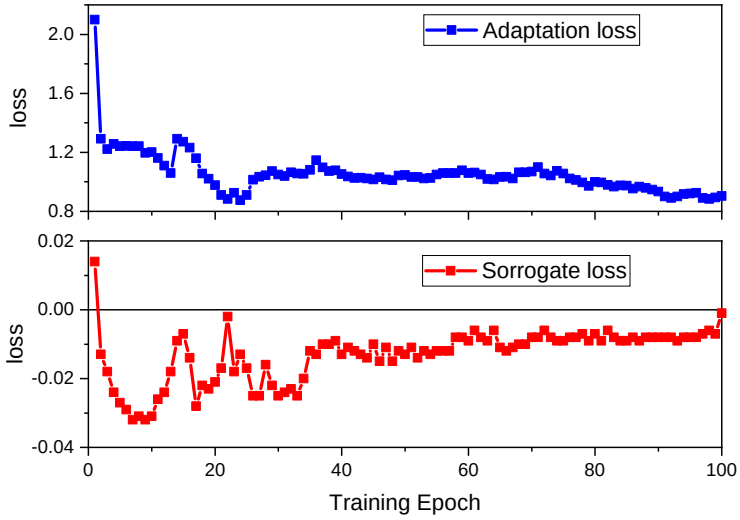

Figure 3: (a) Adaptation loss of the policy and (b) surrogate loss of the critic model during the PPO training process. The adaptation loss gradually decreases reduces and the surrogate loss converges to zero following the expectation.

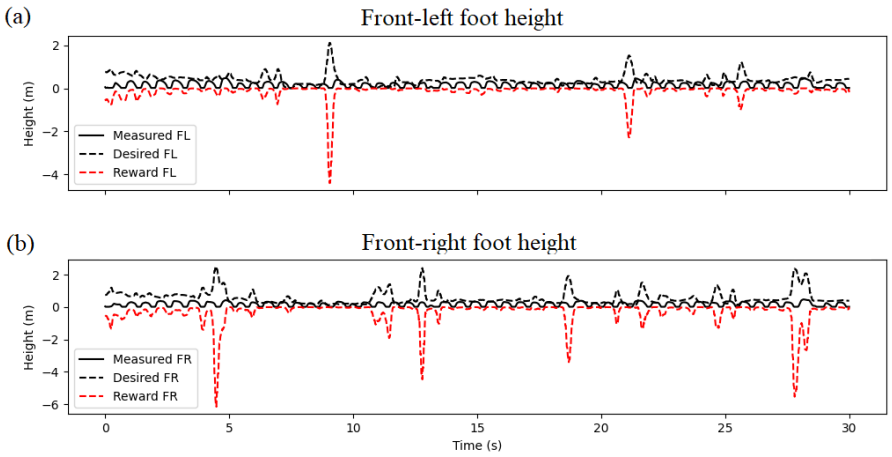

Figure 4: Target and observed (a) front-left and (b) front-right foot heights of the trained agent in the simulation environment with $h_0 = 2$ The black solid line, black dashed line, and red dashed line show the measured feet heights, target feet heights, and the reward from the front foot heights r_f.

loss and surrogate loss during training is shown in Fig. 3. The Adaptation loss gradually decreases and converges in the 100-epoch training process, and the surrogate loss oscillates and converges to zero. That follows the expectation of the PPO training, indicating a well-behaved training process.

We notice that the adaptation loss does not converge to zero but remains at a value of around 0.8. That's because the range of foot heights is beyond the capability of the robot's motion, so some of the targets are inaccessible. Setting a large scale of the foot heights is in order to encourage the robot to exhibit continuous motion, which will be discussed in section 4.3.

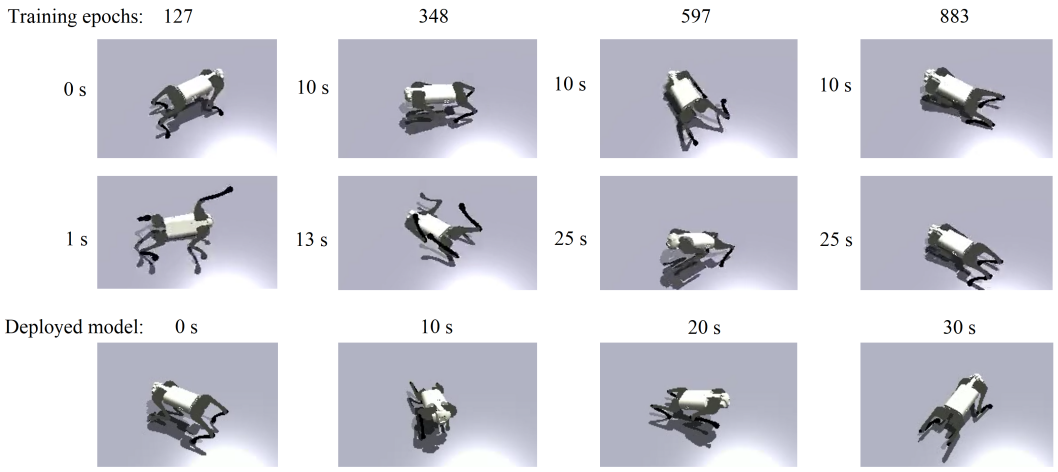

Figure 5: Model performance in the simulation environment during the training process. A robot dancing video is generated using the model after 127 (1st column), 348 (2nd column), 597 (3rd column), 883 (4th column), and 1000 (the final deployed version in the last row) training epochs.

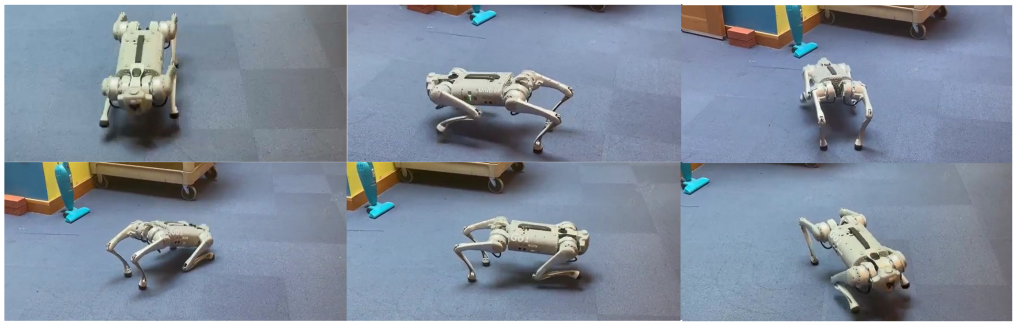

Figure 6: Model performance when deployed on a real robot. 6 snapshots in a dancing sequence is shown, including diverse poses and dancing motions.

After training, the model is deployed in the simulation environment, which provides time-dependent foot heights as shown in Fig. 4. During the 30 seconds of animation, the robot exhibits continuous, periodic motion approximately commensurate with the musical rhythm. The agent's foot heights still have a significant discrepancy with the target foot heights because there are a series of target poses from human foot heights that are inaccessible to the robot, which will be discussed in section 4.3.

4.2 Model performance

In this section, we visualize the performance of our agent during training and after deployment. In the training process, the model performance shows evident improvement, as shown in Fig. 5. At the beginning (epoch 127), the agent turns over in the first second. At training epoch 348, the agent maintains moving stably in the first 10 seconds but finally fails at 13 s. After 597 training epoch, the agent successfully keep balance in the 30s animation, but some distorted, unnatural poses still appear. After 883 training epoch, the agent moves stably and naturally, while a few unnecessary high-frequency oscillations still occur. The finally deployed version exhibits stable, natural motion following the rhythm of the music, and the high-frequency oscillation no longer appears. In the whole training process, we can observe a steady improvement of the agent in performing reasonable dancing motions.

Finally, we deployed our trained model to a physical robot, as shown in Fig. 6. The robot performs diverse motions that also lead to translation and reorientation of the robot, which are also key features of human dancing. Although the sim-to-real gap makes the physical robot's poses deviate from the robot's performance in the simulation environment, the motion still follows the input music and keeps high stability. This result examines that the model is robust against small variations of the environment parameters (like robot body parameters and friction of the floor).

4.3 Discussion: Influence of training parameters

4.3.1 Rear leg reward settings

The reward function for the rear legs was established during training as follows:

$$r_r = c_r(h^{rl})^2 + c_r(h^{rr})^2 \tag{5}$$

We introduced this reward function along with a tunable training hyperparameter c_r to penalize the robot's rear legs leaving the ground. When we set this parameter c_r to a smaller value, the robot's left or right rear legs might hop or even leap off the ground in order for the front legs to reach the target height and thus obtain a reward. In some cases, this resulted in the quadruped robot toppling over, as shown in Figure 7, leading to the failure of the dancing behaviour.

Figure 7: Four-legged robot toppled over during training due to improper hopping, caused by small c_r value

Upon increasing the absolute value of c_r during training, we finally settled on $c_r = -30$. With this value, our training effectively ensured that the robot's rear legs did not leap high off the ground, thereby ensuring the robot's safety and stability while learning and performing dance movements.

4.3.2 Task reward settings

The task reward function was defined as the similarity between the robot's pose and the target pose. We experimented with several ways to functionize the difference $|h^{fl} - h^{l,target}|$.

Initially, we used its L2 norm as the reward function: $r = c * (h^{fl} - h^{l,target})^2$, function curve is shown in the Figure 8. Through training, we obtained the following reward curve as Figure 9.

Despite reaching a convergent state after 1000 episodes of training, our quadruped robot remained in place and failed to exhibit the desired dance effect in sync with the music. We attribute this to the reward function being unable to provide the agent with sufficient motivation to take actions that would decrease gap between h^{fl} and $h^{l,target}$. And that is because L2 norm have a flat curve near goal point. So we needed to choose a reward function that has a steeper gradient when $h^{fl} - h^{l,target}$ is small.

Therefore, through iterative trials, we chose the following expression as our reward function:

$$r_f = c_f(e^{-\lambda_f|h^{fl}-h^{l,target}|} - 1) + c_f(e^{-\lambda_f|h^{fr}-h^{r,target}|} - 1) \tag{6}$$

This function's shape is shown in Figure 10, where we can see that it has a larger slope in region $|h^{fl} - h^{l,target}| < 0.2$. This property helps our agent better learn to follow the generated target pose. The reward curve in this case is shown as Figure 11.

Regarding the settings for the parameters c_f and λ_f in the reward function, we tested on different combinations of them as shown in Table 1.

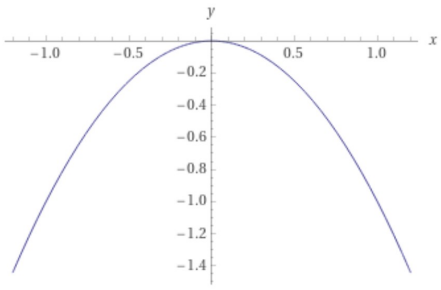

Figure 8: Illustration of the relationship curve between r (as y) and $|h^{fl} - h^{l,target}|$ (as x) in $L2$ norm reward function.

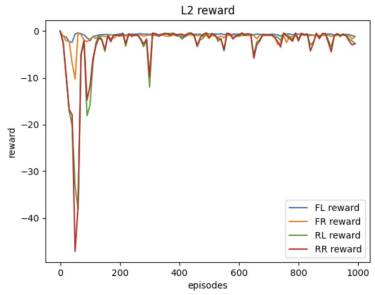

Figure 9: Illustration of the reward curve convergence as training episodes increase, in case of $L2$ norm reward function.

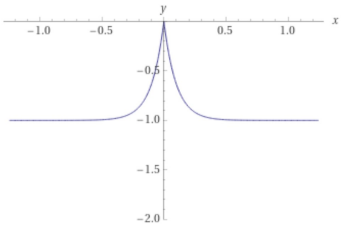

Figure 10: Illustration of the relationship curve between r (as y) and $|h^{fl} - h^{l,target}|$ (as x) in $L0$ reward function.

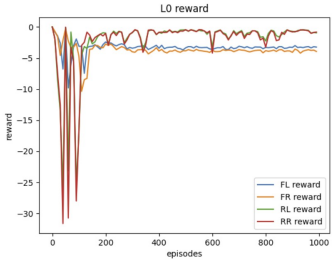

Figure 11: Illustration of the reward curve convergence as training episodes increase, in case of $L0$ reward function.

By comparing the reward and loss performance of the robot under different c_f and λ_f settings, we finally settled on $c_f = 30$ and $\lambda_f = 10$. This configuration yielded a relatively higher reward curve and faster convergence speed.

4.3.3 Choice of foot height normalization parameter h_0

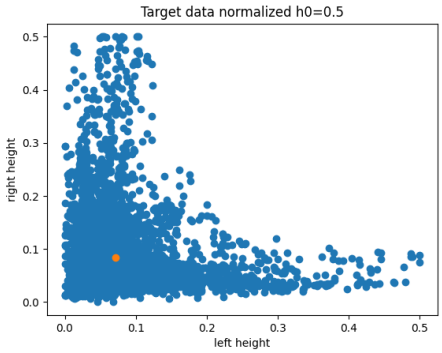

Figure 12: Distribution of normalized and scaled target data points, orange point shows the mean height of left and right font legs.

During experiments, we found that when h_0 was small or within the maximum reachable foot height of the robot ($h_m = 0.5$ m), robot's front foot always stayed at height 0 and did not hop. We consider this was because the robot needed a certain height to complete the entire hopping sequence. When the target height h^{target} was too low, in order to achieve hopping, the robot's front foot would exceed h^{target} , leading to penalty. As a result, during reinforcement learning, the robot tended to choose a conservative strategy, keeping its front foot still to maximize the reward, causing the reward hacking.

	Set 1	Set 2	Set 3	Set 4	Set 5
c_f	100	100	60	30	30
λ_f	100	10	100	100	10

Table 1: Tested sets of the reward training parameters c_f and λ_f in $L0$ reward function.

Therefore, we decided to increase the h_0 parameter so that the robot's front foot would not exceed the target height h^{target} during the entire hopping process. In this case, the robot will try to follow the rhythm of h^{target} as much as possible to reduce the accumulated penalty due to the position difference. In subsequent experiments, we tested cases with different h_0, Figure 13 shows the output results of $h_0 = 0.5, 2$, respectively. After tuning on h_0, we observed that when $h_0 = 2$, the motion of the quadruped robot is the most in line with the rhythm of the music and the generated posture, and has better reward and performance. Its output leg height curve is shown in Figure 4.

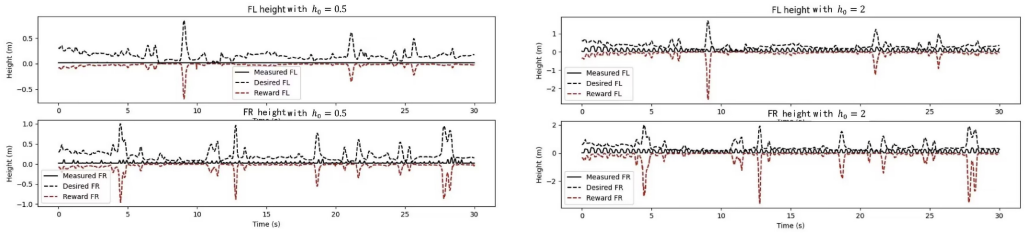

Figure 13: Target and observed front-left and front-right foot heights of the trained agent in the simulation environment with $h_0 = 0.5$ and 2. The black solid line, black dashed line, and red dashed line show the measured feet heights, target feet heights, and the reward from the front foot heights r_f.

4.4 Limitation in learning

A limitation on our robot's performance was the generated desired foot height h^{target}, which always stayed at a non-zero value, as the Desired FL/FR curves in Figure 4. This is difficult for a quadruped robot to achieve in reality because it cannot keep its front feet off the ground for an extended period; it needs contact with the ground for each hop. Therefore, this somewhat unrealistic desired height curve restricted our performance to a certain extent. It also explains why our achieved leg height curve repeatedly returns to the zero height in Figure 4.

5 Conclusion

In this study, we explored the application of Reinforcement Learning (RL) in teaching a quadruped robot to dance in rhythm with music. Our findings highlight the importance of a well-designed reward function and finely-tuned hyperparameters in achieving stable and rhythmic motion. Despite certain limitations, such as the constant non-zero target foot height, the robot exhibited significant performance improvement during training and maintained natural motion in response to music.

In terms of future work, there are several potential directions. One could be refining the reward function to better accommodate the physical limitations of the robot, such as its inability to maintain a constant non-zero foot height. This might enable the robot to achieve a wider range of dance movements without toppling over. Another could be incorporating the positional relationship between the four legs into the model to generate more coordinated and complex dance movements. Additionally, experimenting with more advanced learning algorithms might further improve the robot's performance. We hope that our work provides a valuable reference for further research in entertainment robotics and human-robot interaction.

Contributions

Yaosheng Xu: Design the architecture, write the training code, run the experiments, reward shape tuning, data visualization, and deploy to reality.

Hao Tang: Discuss ideas, generate plots, and write the report.

Shenzhe Yao: Discuss ideas, generate plots, and write the report.

Reference

[1] David Grunberg, Robert Ellenberg, Youngmoo Kim, and Paul Oh. 2009. Creating an autonomous dancing robot. In Proceedings of the 2009 International Conference on Hybrid Information Technology (ICHIT '09). Association for Computing Machinery, New York, NY, USA, 221–227.

[2] Grunberg, David, et al. "Development of an autonomous dancing robot." International Journal of Hybrid Information Technology 3.2 (2010): 33-43.

[3] Gkiokas, Aggelos, and Vassilis Katsouros. "Convolutional Neural Networks for Real-Time Beat Tracking: A Dancing Robot Application." ISMIR. 2017.

[4] Edward Lee Erwin Johan Coumans Jason Peng Jie Tan Sergey Levine Tingnan Zhang Robotics: Science and Systems 2020, RSS Foundation (2020)

[5] Margolis, Gabriel B., and Pulkit Agrawal. "Walk these ways: Tuning robot control for generalization with multiplicity of behavior." Conference on Robot Learning. PMLR, 2023.

[6] Guillermo Valle-Pérez, Gustav Eje Henter, Jonas Beskow, Andre Holzapfel, Pierre-Yves Oudeyer, and Simon Alexanderson. 2021. Transflower: probabilistic autoregressive dance generation with multimodal attention. ACM Trans. Graph. 40, 6, Article 195 (December 2021), 14 pages. https://doi.org/10.1145/3478513.3480570

[7] Tseng, Jonathan, Rodrigo Castellon, and C. Karen Liu. "EDGE: Editable Dance Generation From Music." arXiv preprint arXiv:2211.10658 (2022).

[8] Ruilong Li, Shan Yang, David A Ross, and Angjoo Kanazawa. 2021a. Learn to Dance with AIST++: Music Conditioned 3D Dance Generation. arXiv preprint arXiv:2101.08779 (2021).

[9] Omid Alemi, Jules Françoise, and Philippe Pasquier. 2017. Groovenet: Real-time musicdriven dance movement generation using artificial neural networks. networks 8, 17 (2017), 26.

[10] Makoviychuk, Viktor, et al. "Isaac gym: High performance gpu-based physics simulation for robot learning." arXiv preprint arXiv:2108.10470 (2021).