Balancing is Boring, Let's Try Bouncing: Exploration of Ball Bouncing Task in Isaac Gym

Chaofan Lin [1]

Abstract

This paper modifies the built-in Ball Balancing task in the Isaac Gym benchmark environment into a brand new task called Ball Bouncing. Theoretically, we analyze emerging problems such as the sustainability of bouncing and the height control. Empirically, we train and test this task in different reinforcement learning libraries, including a lightweight self-implemented library called PIG. We show the designed reward function converges easily and it is convenient to control the bouncing behaviour by configuring the parameters. For PIG library, we also reproduce some other tasks based on it.

1. Introduction

Recently, Isaac Gym (Makoviychuk et al., 2021) becomes one of the most promising physics simulation environment for reinforcement learning (RL) due to its ability of efficient GPU sampling and massively parallel simulating.

Ball Balancing is an simple but interesting task provided in Isaac Gym benchmark environment (IsaacGymEnvs[1]). It serves as an introductory example for those new to Isaac Gym and makes users quickly understand the usage of force and torque sensors. In this environment, there is a three-legged table and a hard ball in each instance, as Figure 1 shows. The agent controls the table by applying different forces to the legs, trying to keep the ball balanced on the table.

But simply balancing the ball makes the task visually boring. Imagining that you have worked hard to train a high-scoring agent, only to have the ball sit still on the table. Curiosity naturally drives us to try a few different goals.

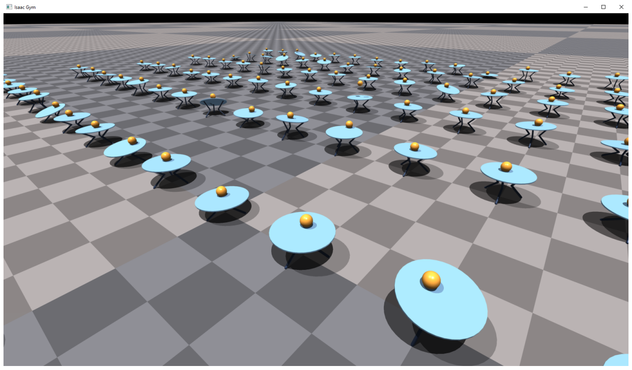

Figure 1. Ball Balancing task previewed in the official playground: `https://github.com/NVIDIA-Omniverse/IsaacGymEnvs/blob/main/docs/rl_examples.md`

Thanks to the amazing physical engine of the Isaac Gym simulator, we can do more under the same environment setting. We observe that the material of the ball and the table allows the ball to bounce off the table as long as we apply an upward force to the table, which makes it easier for us to achieve a new task called Ball Bouncing with only minimal changes in the environment settings, observation function and reward function. For those who want to quickly try it out and see the effect, we have an one-line hacking way to turn the built-in Ball Balance task in IsaacGymEnvs into our Ball Bounce task. We also develop an out-of-the-box codebase to train this task where you can configure the reward functions flexibly with Isaac Gym. (Section 3.1)

And after successfully making the ball bouncing on the table, we can go deeper and do some physical analysis of some interesting topics, such as the sustainability of bouncing (Section 3.2) and how to implement height control (Section 3.3) in the process of bouncing.

We also take this task as a workload and test it on the RL library RL Games (Makoviichuk & Makoviychuk, 2021) and our self-implemented PPO (Schulman et al., 2017) algorithm library PIG (Short for ”PPO for Isaac Gym”), which is a minimal library for IsaacGymEnvs. We also reproduce some other examples such as Cartpole in PIG.

*Equal contribution [1]Zhiyuan College, Shanghai Jiao Tong University, Shanghai, China. Correspondence to: Weinan Zhang <wnzhang@sjtu.edu.cn>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

[1] `https://github.com/NVIDIA-Omniverse/IsaacGymEnvs`

In short, our contribution in this paper can be summarized as follows:

- Propose a new task called Ball Bouncing and provide a simple way to turn the Ball Balancing task into the Ball Bouncing task.
- Analyze the sustainability and height control of ball bouncing from the perspective of RL and dynamics.
- Develop a lightweight RL library PIG and use it to train the Ball Bouncing task.

The code is published in Github[2]. The folder 'big' (Short for "Bouncing in Isaac Gym") contains the Ball Bouncing task settings in Isaac Gym and the folder 'pig' implements the lightweight library mentioned above.

2. Background

2.1. Isaac Gym

Isaac Gym is a physics simulation environment developed by NVIDIA for end-to-end GPU accelerated reinforcement learning research. Different from common CPU-based physics simulators such as Gym (Brockman et al., 2016), PyBullet (Coumans & Bai, 2016–2021) and MuJoCo (Todorov et al., 2012), Isaac Gym performs physics simulation in GPU side by leveraging NVIDIA PhysX and directly passes tensors from physics buffers, totally eliminating the overhead of communicating between CPU simulation and GPU neutral network policy training. It is very suitable for continuous physical simulation-based reinforcement learning scenarios such as Ball Bouncing, which requires sophisticated simulation in the process of collision.

Another outstanding feature of Isaac Gym is that it supports massively parallel deep reinforcement learning simulating. Thousands of identical environments can be run in the same time, which greatly accelerates the sampling process.

IsaacGymEnvs contains the official example RL environments. In the Preview 4 version, it is separated from the Isaac Gym source package. It wraps some common RL environments with Gym-style APIs. By default it leverages RL Games as the algorithm library to train the examples. In this paper we particularly focus on the Ball Balance task.

2.2. Environment Settings

Ball Bouncing shares the same setting with the Ball Balancing task in IsaacGymEnvs. In this task, a single environment consists of two interacting rigid bodies:

- A three-legged table made up of a tray and three identical legs located in $0, \frac{2\pi}{3}, \frac{4\pi}{3}$. Each leg is composed of two segments called upper leg and lower leg. The agent can control the angle between them to interact with the tray just like a man bending and stretching his elbow.
- A hard ball. It is thrown from a random height above the table and falls into it initially. Its radius is relatively small (But we can configure it).

Coordinate System. As is shown in Figure 2, we adopt right-handed system with the z axis as the vertical axis, which is decided by the normal vector configuration. The table is placed in $(0, 0, z_{tray})$ at first standing with all legs straightly. The z coordinate of the ground is 0.

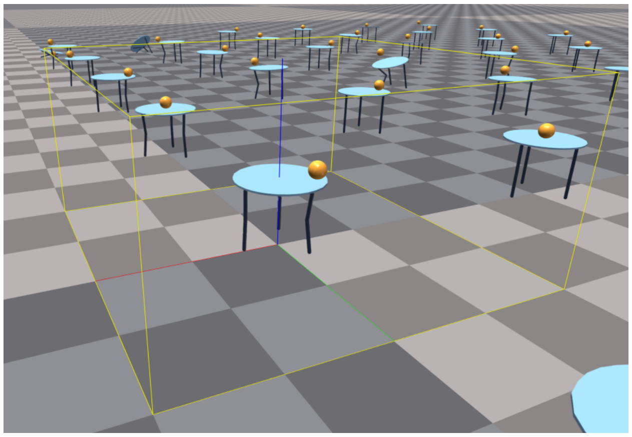

Figure 2. The coordinate system. Red line indicates the positive direction of x-axis; green line indicates the positive direction of y-axis; blue line indicates the positive direction of z-axis.

And there are some important environment settings that need to be listed (See Table 1). The unit of length is not specified in the simulator but can be view as "meter" in real world.

Table 1. Selected environment settings in Ball Bouncing task.

VARIABLE	VALUE	EXPLANATION
$\vec{e}_{norm}$	(0, 0, 1)	*Normal vector of the coordinate system*
g	-9.81	*The acceleration of gravity (directed)*
z_{tray}	0.772	*The height of the tray with straight legs*
r_{BALL}	0.1	*The radius of the ball*
r_{tray}	0.5	*The radius of the tray*

[2] https://github.com/SiriusNEO/GOODBOUNCE

3. Ball Bouncing Task

3.1. One-Line Hacking to Bounce the Ball

The original reward function of the Ball Balancing task in IsaacGymEnvs is:

$$\mathbf{r}(s) = \frac{1}{1+\mathbf{d}(s)} \times \frac{1}{1+\mathbf{v}(s)}, \quad s: \text{the state}$$

where

$$\mathbf{d}(s) = \sqrt{x_{\text{ball}}^2 + y_{\text{ball}}^2 + (z_{\text{ball}} - 0.7)^2},\ \mathbf{v}(s) = \|\vec{v}_{\text{ball}}\|$$

Therefore the lower the velocity of the ball is and the closer it is to the fixed point $P:(0,0,0.7)$, the higher rewarding the agent can get. According to the environment settings in Section 2.2, P is slightly under the center of the tray. Therefore the reward instructs the agent to keep the ball as still as possible in the center of the table.

Our hack just changes the reward to

$$\mathbf{r}(s) = \frac{1+\mathbf{v}(s)}{1+\mathbf{d}(s)}$$

And the ball starts to bouncing in the ball. The core idea is to change the negative correlation with velocity to a positive correlation. This hack is so simple that you can immediately try it in IsaacGymEnvs.

Although we have successfully bounced the ball, there is not a clear measurement for this new task. In other words, how do we evaluate the performance of a ball bouncing agent? Analogy to juggling Ping Pong in our daily life, we can start from those two requirements:

- The agent should be able to bounce the ball sustainably, which means each time the ball falls, the agent should be able to catch it and bounce it back.
- The agent should be able to bounce the ball around a specified height. That is, we can control the height of the bouncing process.

3.2. Sustainability of Bouncing

We call the table **missed the ball** if the bouncing environment is reset. In our settings, the environment is reset whenever $z_{\text{ball}} < 1.5 \cdot r_{\text{ball}}$ (Almost touch the ground).

Definition 3.1. A bouncing process is sustainable if the table never miss the ball.

We can judge the sustainability of a bouncing system by the average time interval between two misses. Physically, the more off-center the hit between the ball and the tray, the higher the chance of a miss. So in order to constrain the coordinates of the hit point, we can add weights to the coordinates:

$$\mathbf{d}(s) = \sqrt{\beta \cdot (x_{\text{ball}}^2 + y_{\text{ball}}^2) + \alpha \cdot (z_{\text{ball}} - 0.7)^2} \qquad (1)$$

This two parameters can not only scale $\mathbf{d}(s)$ comparing to $\mathbf{v}(s)$ (Considering that they have different unit dimensions) but also adjust the weights between x^2+y^2 and z^2. We can see in the following experiments that if we put more importance on the former, the sustainability will be improved in some extent.

3.3. Height Control

3.3.1. The Height of the Target Point

Recalling that in the reward function, we calculate the distance to a fixed point $P:(0,0,0.7)$. And the closer it gets, the more reward the agent can receive. So to control the average bouncing height, we can rise the target point P to a suitable position.

Theoretically, let $P:(0,0,h_{\text{target}})$. We can view the ball as a particle in our model and consider the bounce as a classical vertical projectile, as Figure 3. The green part indicates the reward during this process $\mathbf{r}(v_0)$.

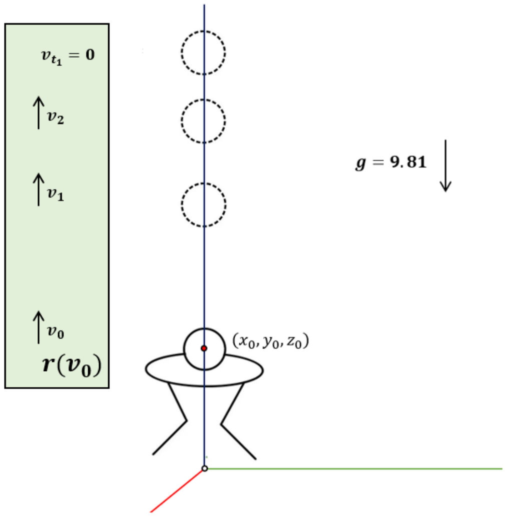

Figure 3. The classical vertical projectile physics model of bouncing process.

Suppose we bounce the ball in an initial velocity $\vec{v_0} = (0,0,v_0)$ and an initial position (x_0,y_0,z_0). Assuming the frequency of reward sampling is high enough. Then the total reward in one round can be approximated as an integral

$$
\begin{aligned}
\mathbf{r}(v_0) &= 2\int_0^{t_1} \frac{1+v(t)}{1+\sqrt{x_0^2+y_0^2+(z(t)-h_{\text{target}})^2}}\mathrm{d}t \\
&= \frac{2}{g}\int_0^{v_0} \frac{1+v}{1+\sqrt{x_0^2+y_0^2+(z_0+\frac{v^2-v_0^2}{2g}-h_{\text{target}})^2}}\mathrm{d}v
\end{aligned} \tag{2}
$$

Ideally, for a given parameter h_{target}, the agent will bounce the ball in an initial velocity $\arg\max_{v_0} \mathbf{r}(v_0)$.

Theorem 3.2. $\arg\max_{v_0} \mathbf{r}(v_0)$ *increases monotonically with* h_{target}.

The detailed proof is attached in Appendix A. Hence, by configuring h_{target}, we can approximately control the average height of the bouncing.

3.3.2. Height Punishment

The method in Section 3.3.1 is not exact because we use an ideal physics model by which we can not implement a precise height control. Let us switch from the physical point of view to one of RL and use a directly way: set a height limit such that if the ball exceeds this limit, it will get punishment.

Practically, we can design a punishment function as

$$
f_{\text{punish}}(z_{\text{ball}}) = \begin{cases} -c_{\text{punish}} & z_{\text{ball}} > h_{\text{limit}} \\ 0 & \text{otherwise} \end{cases} \tag{3}
$$

And add it in the reward. Note that strict height limit may affect sustainability, given that bouncing in a shorter height range is more difficult.

3.4. Final Reward

Combing the discussion in Section 3.2 and Section 3.3, we design the reward function as

$$
\mathbf{r}(s) = \frac{1+\mathbf{v}}{\sqrt{\alpha\cdot(z_{\text{ball}}-h_{\text{target}})^2+\beta\cdot(x_{\text{ball}}^2+y_{\text{ball}}^2)}} + f_{\text{punish}}(z_{\text{ball}}) \tag{4}
$$

Here we introduce four configurable parameters $\alpha, \beta, h_{\text{target}}$ and h_{limit}. The punishment factor c_{punish} is left as a preset constant.

4. The PIG Library

The PIG (PPO for Isaac Gym) library aims to use minimal codes to train and test the examples in IsaacGymEnvs, given that the default library RL Games is not clear enough for those new to RL and Isaac Gym. It is created to train the Ball Bouncing task at first but we also use it to reproduce other examples. It is very interesting to see the model trained successfully in our self-implemented RL library!

The Algorithm. As its name, PIG only adopts PPO as its training algorithm. The implementation mainly refers to the notebook assignment in our RL course and Hands-On-RL[3].

Trainer. Like IsaacGymEnvs, PIG also leverages Hydra (Yadan, 2019) as its configuration parsers and resolvers. And PIG takes a subset of information in the IsaacGymEnvs configuration files to initialize the network and prepare the environment. Therefore it is totally compatible with the original configuration files.

5. Experiments

5.1. Bouncing in Different RL Libraries

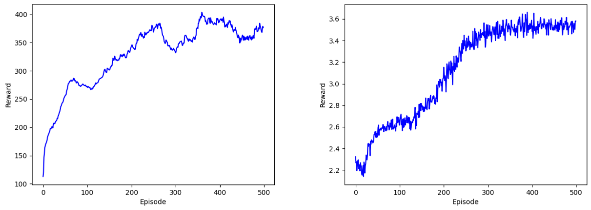

Figure 4. The reward during training in different libraries. Left: Trained in RL Games; Right: Trained in PIG.

Figure 4 shows the reward curves during training Ball Bouncing task. We can see that after tuning the hyper parameters, the reward function converges on both libraries. Note that the scales of the reward on these two libraries are different. In RL Games, the reward reaches an average of 350 after about 250 episodes while in PIG, it reaches an average of 3.5 in about 300 episodes.

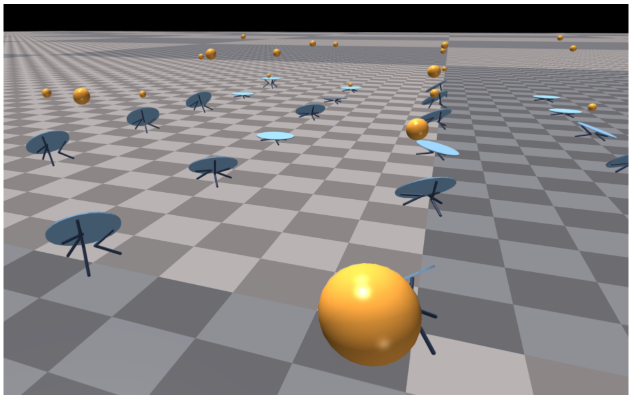

Figure 5. Snapshot of bouncing in Isaac Gym simulator.

[3] https://github.com/boyu-ai/Hands-on-RL

About the actual performance, with a reward of 350 in RL Games, the table is intelligent enough to stably bounce the ball. But in PIG, the visual effect in the simulator with reward score 3.5 is poor. This mainly blames to the implementation of PIG, which is too naive.

Figure 5 gives a snapshot of bouncing. In order to show it more vividly, we provide some GIFs of bouncing in Github[4] to show: (1) The interesting process of bouncing in Isaac Gym, which generates by a fine-tuned checkpoint; (2) The effect of heigh control.

5.2. Bouncing in Different Sustainability

In 3.2, we have roughly discussed the relation between the sustainability and the $beta$ parameter. We can visualize the sustainability by drawing the z-t diagram of a ball. As is shown in Figure 6, $\beta = 4$ shows better sustainability than $\beta = 1$, which confirms our conclusion that higher β leads to higher sustainability. The other parameters take default values: $\alpha = 1$, $h_{\text{target}} = 0.7$ without height punishment.

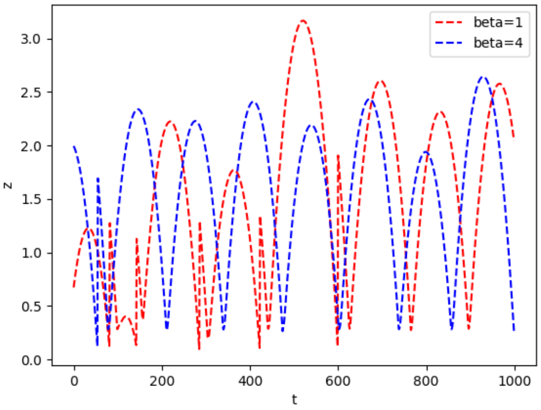

Figure 6. z-t diagram for two Ball Bouncing agents with different β. Red: the agent with $\beta = 1$; Blue: the agent with $\beta = 4$. The blue agent shows a higher stability and sustainability.

5.3. Bouncing in Different Height

Recall that we explore two ways to control the height. Figure 7 shows three z-t curves in different h_{target} settings. The other parameters are $\alpha = 2$, $\beta = 1$ without height punishment. Here we set $\alpha = 2$ such that the model pays more attention in h_{target}, which is beneficial for us to see its influence. The experiment result is consistent with Theorem 3.2 that higher h_{target} brings a higher average bouncing height.

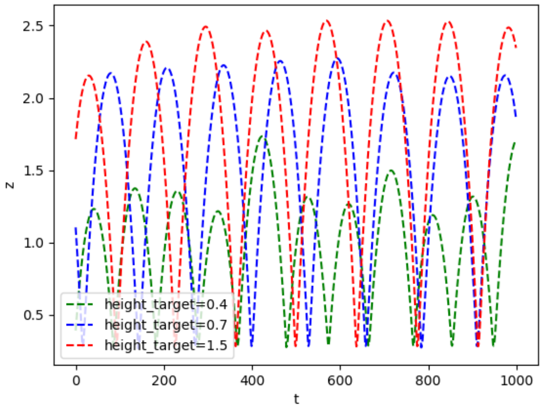

Figure 7. z-t diagram for two Ball Bouncing agents . Green: the agent with $h_{\text{target}} = 0.4$; Blue: the agent with $h_{\text{target}} = 0.7$; Red: the agent with $h_{\text{target}} = 1.5$.

Figure 8 illustrates the effect of height punishment. The other parameters are $\alpha = 1$, $\beta = 4$ and $h_{\text{target}} = 0.7$. We set $\beta = 4$ because with a height punishment, it is more difficult for an agent to maintain the sustainability. As we can see, the bouncing height of the purple agent is limited by the punishment mechanism.

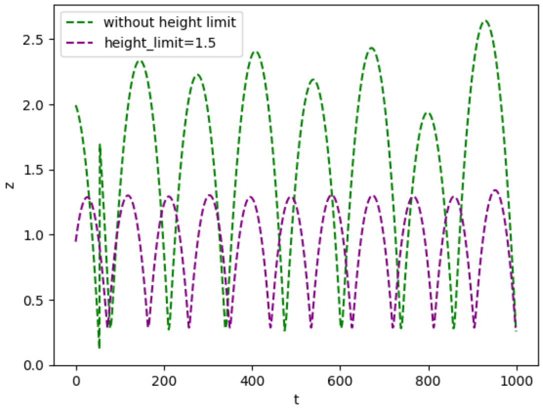

Figure 8. z-t diagram for two Ball Bouncing agents. The purple agent has a height limit of 1.5.

5.4. More Examples Trained in PIG

We also try other tasks provided by IsaacGymEnvs in PIG. Cartpole is a classical workload which is also provided by many other RL simulators and can be well trained by PIG. But for some complex tasks like Ant and Humanoid,

[4] https://github.com/SiriusNEO/GOODBOUNCE/blob/main/README.md

PIG is too naive to successfully train an agent with enough intelligence.

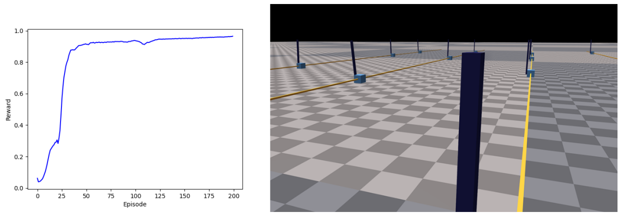

Figure 9. Cartpole task in PIG. Left: The reward curve in training; Right: The simulation result.

6. Conclusion

We propose a new visually interesting RL task called Ball Bouncing, with the aim of bouncing the ball sustainably and around a specified height. We implement this task by reusing the environment settings of the built-in Ball Balancing task in IsaacGymEnvs and designing a new parameterized reward function. We show this reward function is easy to converge (trainable) and can be used to control some behaviours in the bouncing process. We evaluate it in different RL libraries and also in a self-implemented lightweight library called PIG. As a byproduct, PIG can not only train Ball Bouncing but also other examples in IsaacGymEnvs.

Acknowledgements

This paper is the course project of CS3316 Reinforcement Learning, ACM Class 2020. The whole work is based on the NVIDIA Isaac Gym Preview Release 4. Many thanks to the open source reinforcement learning libraries used in this work. And thanks to Hands-On-RL for its code and materials which help me understand and implement the PPO algorithm in PIG. Thanks to the TAs and Prof. Weinan Zhang in this course for their kind help in my course project!

References

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

Coumans, E. and Bai, Y. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021.

Makoviichuk, D. and Makoviychuk, V. rl-games: A high-performance framework for reinforcement learning. https://github.com/Denys88/rl_games, May 2021.

Makoviychuk, V., Wawrzyniak, L., Guo, Y., Lu, M., Storey, K., Macklin, M., Hoeller, D., Rudin, N., Allshire, A., Handa, A., and State, G. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms, 2017.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

Yadan, O. Hydra - a framework for elegantly configuring complex applications. Github, 2019. URL https://github.com/facebookresearch/hydra.

A. Proof of Theorem 3.2

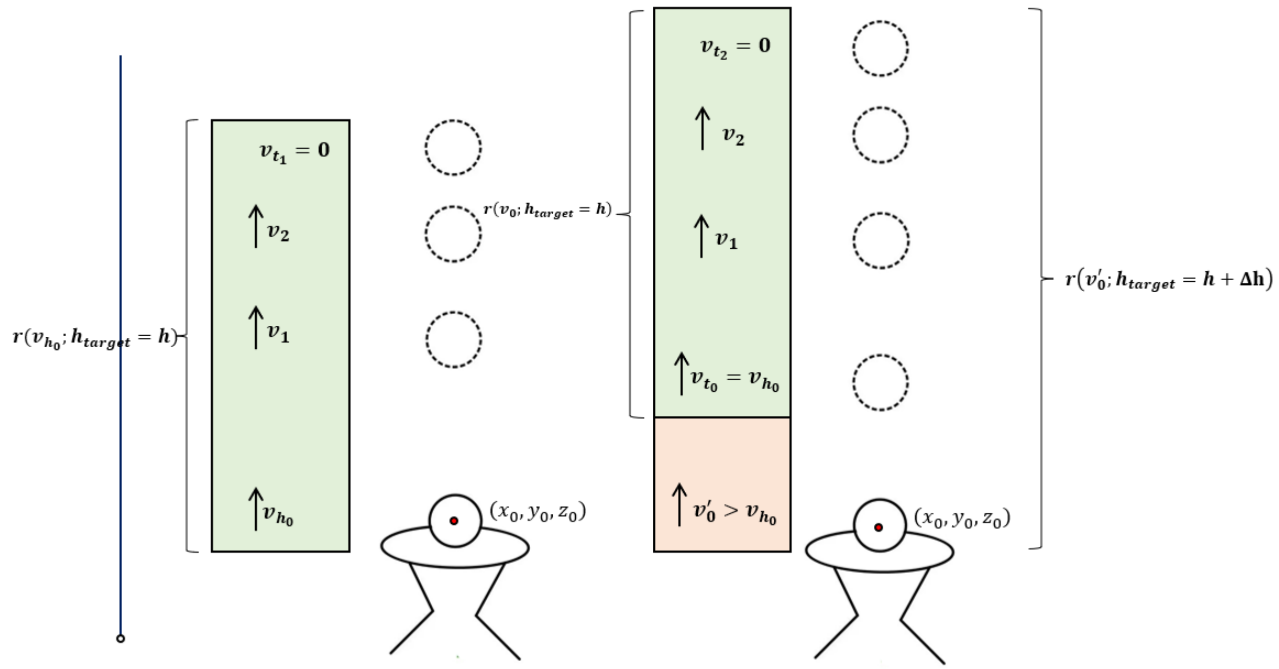

Figure 10. Schematic diagram of the proof.

Proof. Given a target height h, let $v_h := \arg\max_{v_0} \mathbf{r}(v_0)$ with $h_{\text{target}} = h$. For a fixed h_0, we only need to prove that for any $\Delta h > 0$, $v_{h_0+\Delta h} > v_{h_0}$. For convenience, we denote $\mathbf{r}(v; h_{\text{target}} = h)$ with the the meaning of $\mathbf{r}(v)$ under the setting that $h_{\text{target}} = h$.

The optimal bouncing process P_1 under $h_{\text{target}} = h_0$ is shown in the left of Figure 10. We first find a new initial velocity v_0' under $h_{\text{target}} = h_0 + \Delta h$ such that

$$\frac{1}{2}mv_0'^2 = \frac{1}{2}mv_{h_0}^2 + mg\Delta h$$

So using this initial velocity, we can divide the ball's motion into two parts. The first part P_2 is $z_0 \sim z_0 + \Delta h_0$ (The orange part in Figure 10) and the second part P_3 is the rest part (The green part in Figure 10). When the ball finishes the first part, according to the Kinetic Energy Theorem, it is in a velocity of v_{h_0}. Therefore P_3 is totally the same as the original process P_1, so they return the same reward $r(v_{h_0}; h_{\text{target}} = h_0)$, which is the optimal reward in this part by assumption. Next we want to demonstrate

$$\mathbf{r}(v_0'; h_{\text{target}} = h_0 + \Delta h) > \mathbf{r}(v_{h_0}; h_{\text{target}} = h_0 + \Delta h) \tag{5}$$

For the red part, the former can get more rewards because they have the same distance to the target point but the former has higher velocity. And for the reward in green part, the former is optimal. Therefore (5) is proved.

And because the green part is optimal, we have a corollary

$$\forall v_0 < v_0',\ \mathbf{r}(v_0'; h_{\text{target}} = h_0 + \Delta h) > \mathbf{r}(v_0; h_{\text{target}} = h_0 + \Delta h)$$

Therefore

$$v_{h_0+\Delta h} = \arg\max_{v_0} \mathbf{r}(v_0; h_{\text{target}} = h_0 + \Delta h) \geq v_0' > v_{h_0}$$

□