Hopping Control of a Monopod Robot: Reinforcement Learning vs Raibert Heuristic

Zhuochen Yuan
Tsinghua Shenzhen International Graduate School
Tsinghua University
Shenzhen, China
xx@xx

Linqi Ye
School of Future Technology
Shanghai University
Shanghai, China
yelinqi@shu.edu.cn

Houde Liu
Tsinghua Shenzhen International Graduate School
Tsinghua University
Shenzhen, China
liu.hd@sz.tsinghua.edu.cn

Bin Liang
Department of Automation
Tsinghua University
Beijing, China
bliang@tsinghua.edu.cn

***Abstract*—As a classic control problem, monopod hopping was traditionally solved using a state machine-based approach known as the Raibert Heuristic. Recently, reinforcement learning (RL) has emerged as a powerful alternative for robotic control. In this paper, we apply RL to monopod hopping control and compare its performance with the traditional Raibert Heuristic. Simulations demonstrate that RL achieves faster hopping speeds and better robustness than the Raibert method. Both control strategies were implemented on a simulated monopod robot and evaluated based on hopping speed and robustness. The results highlight RL's superiority in complex environments, while the Raibert approach remains a good solution for simpler scenarios.** (The attached video can be found in https://www.bilibili.com/video/BV1eTxgeuEjV/?vd_source=25bf190003ff1ebd36e7649d3641e141)

Keywords*—*Monopod hopping, reinforcement learning, motion control, Raibert Heuristic, Proximal Policy Optimization

I. INTRODUCTION

The control of legged robots poses significant challenges due to the need to balance dynamic stability, energy efficiency, and adaptability across diverse terrains. Monopod robots, with their single-leg locomotion, serve as simplified models for understanding legged motion and have been extensively studied as a platform for testing control algorithms. These robots allow researchers to explore the fundamental principles of balance, hopping, and dynamic locomotion, which can later be applied to more complex multi-legged systems [1].

Early research on monopod robots focused on the mechanics of hopping and balance, with Marc Raibert's pioneering work in the 1980s being one of the most influential in this field [2]. Raibert developed a heuristic control strategy for legged robots, breaking the hopping control problem into three independent tasks: regulating forward velocity, controlling hopping height, and maintaining body balance through foot placement [3]. His work led to the development of Raibert's Hopper, a single-legged robot capable of stable, dynamic hopping and running. This robot was among the first to demonstrate successful legged locomotion using a simple control framework based on a state machine, and it laid the foundation for further research in legged robotics [4].

Over the years, the Raibert Heuristic has been expanded and modified to control more complex legged systems, such as bipedal and quadrupedal robots, but the simplicity and effectiveness of this approach continue to make it a relevant method for monopod robots [5]. However, the limitations of the Raibert Heuristic become apparent when dealing with more complex and unpredictable environments, as it relies on pre-tuned parameters that lack adaptability to new terrains or dynamic disturbances [6].

In addition to Raibert's work, research on monopod robots has also explored passive dynamics and energy efficiency. For example, the Spring-Loaded Inverted Pendulum (SLIP) model has been widely used to study the dynamics of hopping and running in both animals and robots. The SLIP model simplifies legged locomotion by representing the leg as a massless spring, allowing researchers to focus on the mechanical principles underlying efficient movement. This model has inspired numerous monopod robot designs, demonstrating that passive dynamics can play a critical role in achieving energy-efficient locomotion.

Recent advancements have also included the use of nonlinear control techniques and optimization methods to improve the stability and efficiency of monopod robots. For example, trajectory optimization techniques have been used to compute energy-optimal gaits and jumping maneuvers for monopod systems. Additionally, model predictive control (MPC) has been applied to monopod robots to enhance real-time stability and adaptability, providing more responsive control in dynamic environments.

In recent years, reinforcement learning (RL) has emerged as a promising and adaptive alternative for robotic control, including monopod systems. RL enables robots to learn optimal control policies through interaction with their environment, continuously improving objectives such as stability and energy efficiency. Lillicrap et al.[14] introduced the Deep Deterministic Policy Gradient (DDPG) algorithm, which has been widely adopted for continuous control tasks, including legged locomotion. Schulman et al.[15] developed Proximal Policy Optimization (PPO), a computationally efficient and robust algorithm that has become a standard in RL-based robotic control . More recently, Miki et al.[16] integrated terrain sensing into reinforcement learning-based locomotion, significantly enhancing the robot's ability to adapt to varying environments .

These studies, along with others in the field, highlight the strengths of RL in improving the adaptability and performance of legged robots, offering greater flexibility in handling unpredictable environments. However, challenges such as high computational demands and policy generalization remain, limiting the widespread application of RL in real-world monopod robot systems.

This paper provides a comparative analysis between the traditional Raibert Heuristic and modern RL-based control methods for a monopod robot, focusing on stability, speed, and adaptability. Both control strategies are implemented in a simulated environment to evaluate their performance. The Raibert Heuristic demonstrates stable control in predictable conditions but struggles with adaptability in dynamic environments. In contrast, RL shows superior adaptability, allowing the robot to dynamically adjust to varying terrains while achieving better stability and energy efficiency.

The paper also indicates the idea of using hybrid approaches that combine the stability of heuristic methods with the adaptability of RL, which may offer a more versatile solution for complex environments. Additionally, this work presents a novel application of reinforcement learning to monopod hopping, demonstrating faster speeds and more adaptability compared to traditional methods, particularly in challenging, uneven terrains.

II. Methods

A. *Raibert Heuristic Control*

The Raibert Heuristic control system for the 3D monopod robot decomposes the hopping task into three relatively independent components: forward velocity control, body attitude control, and hopping height control [3], as shown in Fig. 1. By decoupling these controls, the system assumes weak coupling between them, which simplifies the overall control strategy. Each component independently manages its respective function—velocity, balance, and height—while the system as a whole maintains stable and effective hopping [4].

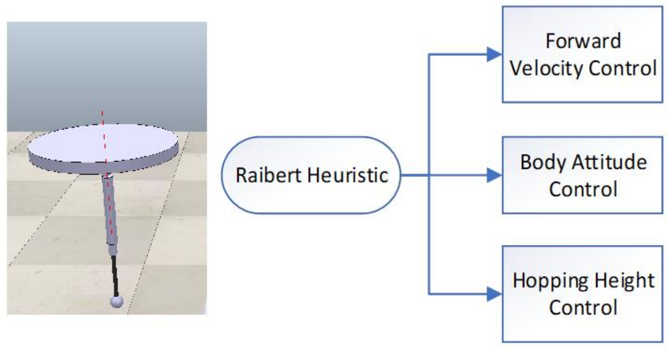

Fig. 1. Control Diagram of Raibert Heuristic Method

Forward Velocity Control. The control system stabilizes the robot’s speed by adjusting the foot’s position during the flight phase to ensure the necessary acceleration or deceleration occurs during the stance phase. The foot is extended forward to generate the required horizontal force upon ground contact, which allows for optimal velocity control during the stance phase [2].

The horizontal velocity $\dot{\boldsymbol{x}}$ determines the neutral foot position x_{f0}, which is calculated as:

$$\boldsymbol{x}_{f0} = \frac{\dot{x}T_S}{2} \tag{1}$$

where T_s is the duration of the stance phase. To correct for velocity errors, the foot's displacement from the neutral position $x_{f\Delta}$ is determined by the difference between the current velocity $\dot{x}$ and the desired velocity $\dot{x}_d$:

$$\boldsymbol{x}_{f\Delta} = k_x(\dot{\boldsymbol{x}} - \dot{x}_d) \tag{2}$$

where k_x is a gain constant. The actual foot placement during flight is then calculated as:

$$\boldsymbol{x}_f = \frac{\dot{x}T_S}{2} + k_x(\dot{\boldsymbol{x}} - \dot{\boldsymbol{x}}_d) \tag{3}$$

Body Attitude Control. The control system ensures the robot remains upright by adjusting the hip torque during the stance phase. The system uses gyroscopes to measure the pitch ϕ_P and roll ϕ_R angles and applies corrective torques τ_1 and τ_2 to the hip actuators [5]:

$$\tau_1 = -k_p(\phi_P - \phi_{P,d}) - k_v\dot{\phi}_P \tag{4.1}$$

$$\tau_2 = -k_p(\phi_R - \phi_{R,d}) - k_v\dot{\phi}_R \tag{4.2}$$

where k_p and k_v are proportional and derivative gains, and $\phi_{P,d}$ and $\phi_{R,d}$ are the desired pitch and roll angles. These adjustments help maintain balance during the hopping cycle [6].

Hopping Height Control. The hopping height control regulates the vertical amplitude of the robot’s jumps by modulating leg extension during the stance phase. It ensures that the robot reaches the desired height on each hop by controlling the thrust generated by the leg [7].

The synchronization of these three control components—forward velocity, body attitude, and hopping height—is managed by a finite state machine (FSM), which tracks the robot's hopping cycle and coordinates the control actions during different phases: Loading, Compression, Thrust, Unloading, and Flight [9], as shown in Fig. 2.

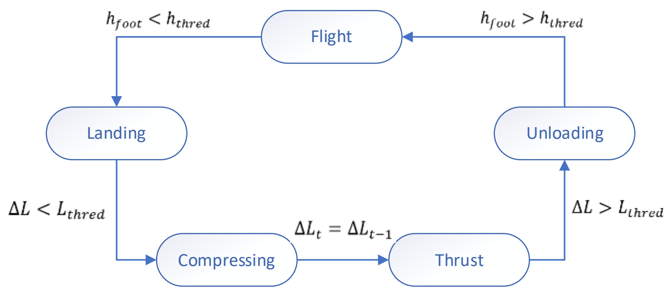

Fig. 2. State Machine Diagram of Periodical Hopping

- Loading begins when the foot touches the ground. The FSM signals the system to stop extending the leg and applies zero hip torque to maintain balance.
- In the Compression phase, as the leg shortens, the upper leg chamber is sealed, and the body attitude is adjusted through precise control of the hip servos.
- During the Thrust phase, as the leg lengthens, the leg is pressurized to generate upward force for the next hop, while body attitude control continues via the hips.
- As the leg approaches full extension, the Unloading phase begins, during which the leg thrust is stopped, and zero hip torque is applied.
- Finally, in the Flight phase, the leg is no longer in contact with the ground. The leg is depressurized, and the system repositions the leg for the next landing while maintaining control of the body attitude.

These phases are triggered by sensory inputs, such as foot-ground contact detection and leg compression sensors [8]. This ensures that control actions for forward velocity, body attitude, and hopping height are executed at the correct moments. The FSM facilitates smooth transitions between phases, maintaining overall stability throughout the hopping cycle and ensuring real-time coordination of the various control components [10].

B. *Reinforcement Learning Control*

In this work, we apply the Proximal Policy Optimization (PPO) algorithm to train the control policy for the monopod robot [15], see Fig. 3. The reinforcement learning framework is modeled as a Markov Decision Process (MDP), defined by the tuple (S, A, P_a, R_a), where [14]:

- S represents the set of all possible states,
- A represents the action space,
- $P(s_{t+1}|s_t, a_t)$ is the state transition function,
- $R(s_{t+1}|s_t, a_t)$ is the immediate reward function.

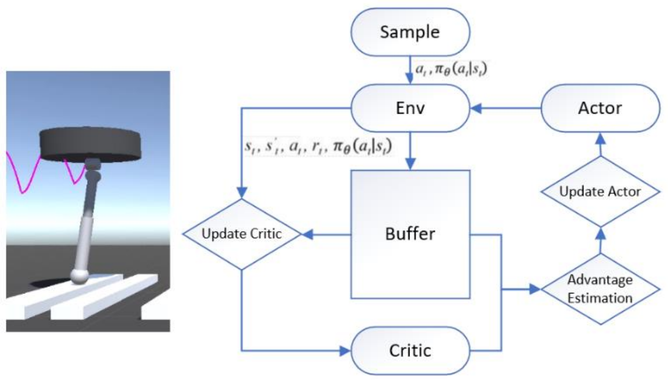

Fig. 3. Diagram of Reinforcement Learning Method

At each time step t, the agent selects an action a_t based on the current state s_t. The MDP then computes the next state s_{t+1} and the corresponding reward r_t, providing feedback to the agent. The objective is to learn a policy $\pi(a_t|s_t)$ that maximizes the cumulative discounted reward over time:

$$J(\pi) = E_\pi(\sum_{t=0}^{\infty} \gamma^t r_t) \tag{5}$$

where γ is the discount factor, determining how much future rewards are valued [17].

Policy Network: To train the policy $\pi_\phi(a_t|s_t)$, we use an actor-critic architecture with the Proximal Policy Optimization algorithm. PPO is well-suited for continuous control tasks like monopod hopping, as it maintains a balance between exploration and exploitation while ensuring stable and efficient learning. The actor network outputs the control actions, while the critic network estimates the value of each state, helping the agent evaluate the quality of its actions [16].

State Space: The state space consists exclusively of proprioceptive information related to the robot's internal state. This includes joint angles, joint velocities, body orientation (pitch, roll, and yaw), and the linear and angular velocities of the robot's body. This rich set of proprioceptive data enables the agent to effectively perceive the robot's posture and dynamics, making informed decisions to maintain stability and control. However, it does not include any external environmental data such as terrain features or obstacles, making the model entirely reliant on internal feedback for decision-making [13].

Action Space: The action space is continuous, controlling various parameters such as leg extension, which determines the hopping height, and joint torques, which adjust the orientation and stability of the body. These actions provide smooth control over the robot's movements, enabling it to adapt its internal configuration to maintain balance and achieve effective hopping [21].

Reward Function: The reward function r_{total} is designed to incentivize stability, minimize energy consumption, and encourage forward progression. It consists of several components [18]:

- Live Reward: A constant reward r_{live} is provided at each time step to encourage the agent to maintain balance and learn continuously.
- Orientation Reward: The orientation reward penalizes the robot for deviating from a stable posture. It is based on the body's pitch θ_x and roll θ_z:

$$r_{orix} = w_{ori1} \times \min(|\theta_x|, |360° - \theta_x|) \tag{6.1}$$

$$r_{oriz} = w_{ori2} \times \min(|\theta_z|, |360° - \theta_z|) \tag{6.2}$$

where w_{ori1} and w_{ori2} correspond to the weights of the two part rewards. In this case, both weights are set to -0.05.

- Velocity Reward: The velocity reward encourages the robot to achieve the desired forward and vertical velocities, which helps the robot maintain a steady forward movement while jumping. It is calculated as follows:

$$r_{vel} = w_h \times |v_y| + w_f \times v_f \tag{7}$$

where $v_f = \sqrt{v_x^2 + v_z^2}$ is the forward velocity and v_y is the vertical velocity.

- Position Reward: Due to the focus on the dynamic performance of hopping, the position reward is not constrained in this case:

$$r_{pos} = 0 \tag{8}$$

This comprehensive reward function ensures that the robot learns to maintain balance, move efficiently, and adapt its internal state based on proprioceptive feedback [22].

Policy Optimization Process: PPO optimizes the policy by repeatedly interacting with the environment, updating the policy network to maximize the expected cumulative reward. Futhermore, we set termination conditions that a learning episode terminates when the robot meets any of the following conditions [12]:

- Excessive Pitch Angle: If the body's rotation around the X-axis exceeds 15°:

$$\min(|\theta_x|, |360° - \theta_x|) > 15° \tag{9}$$

- Excessive Roll Angle: If the body's rotation around the Z-axis exceeds 15°:

$$\min(|\theta_z|, |360° - \theta_z|) > 15° \tag{10}$$

- Low Height: If the body's height drops below 0.5 meters:

$$y_{body} < 0.5(m) \tag{11}$$

When any of these termination criteria are met, the episode resets to its initial state, and a new training episode begins.

Through continuous interaction with the environment, the agent optimizes its policy by trial and error, progressively improving the robot's stability, energy efficiency, and adaptability to varying terrains [11].

III. Experiments

A. Experimental Setup

The experiments utilized two different simulation platforms to assess the performance of the control methods. The Raibert heuristic control method was implemented in CoppeliaSim, a simulation environment ideal for traditional control algorithms due to its robust physics engine and high-fidelity modeling capabilities. Conversely, the PPO-based reinforcement learning approach was implemented in Unity 3D using the ML-Agents toolkit, which offers an effective framework for training and deploying machine learning models in complex 3D environments.

A consistent 3D monopod robot model was employed across both platforms, with identical physical parameters—such as leg length, joint limits, and mass properties—ensuring that the comparative results were attributable solely to the control strategies rather than discrepancies in simulation settings.

The control loop in both environments operated at a frequency of 100 Hz, facilitating smooth and responsive control actions. All experiments were conducted on flat terrain in both CoppeliaSim and Unity 3D to maintain uniform testing conditions, allowing for a direct comparison of the two control methods under equivalent scenarios.

B. Control Methods Compared

Raibert Heuristic: Implemented in CoppeliaSim using a traditional state machine-based approach. The control system is manually tuned to achieve optimal performance on flat terrain. The Raibert heuristic method is tested under the same conditions as those used during training, focusing on evaluating the stability and effectiveness of the predefined control strategy.

Reinforcement Learning (PPO): The control policy is developed in Unity 3D using the ML-Agents toolkit. The model is trained on flat terrain over [number of episodes/steps], utilizing a reward function that encourages stability and forward velocity. The trained policy is then tested on flat terrain in Unity 3D to assess its performance under the same conditions as the Raibert heuristic method.

This setup ensures that both methods are evaluated under comparable scenarios, allowing for a clear comparison of the differences attributable to their respective control strategies.

C. Results and Analysis

Forward Velocity Comparison. The forward velocity comparison between the Raibert heuristic method and the PPO-based RL method is shown in the figure below. The results indicate that:

- RL Method: The PPO-based RL method achieves a higher and more consistent forward velocity over time, as seen from the orange curve in the graph. The average forward velocity is around 0.35 to 0.4 m/s, with minor fluctuations.

- Raibert Heuristic Method: The Raibert heuristic method, represented by the blue curve, shows significantly lower forward velocity, fluctuating around 0.05 to 0.1 m/s. This indicates a less effective forward movement compared to the RL method.

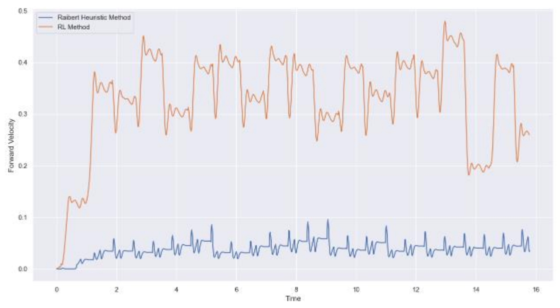

Fig. 4. Comparison of forward velocity

The comparison highlights the superior performance of the RL method in achieving higher and more stable forward velocity under similar conditions.

Position and Velocity Over Time For Raibert Heuristic Method. The position and velocity of the robot in the X, Y, and Z directions using the Raibert heuristic method are shown in the figures below.

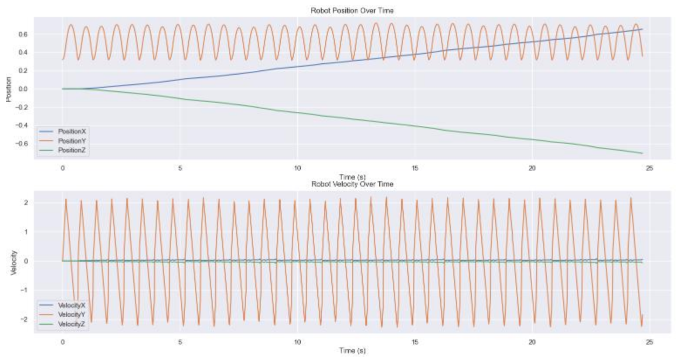

Fig. 5. Trajectory of Raibert Heuristic method

The position data shows that the robot maintains a relatively stable height during the hopping motion, while the forward movement is slow and steady. There is minimal displacement in the Z-axis. The velocity data indicates periodic oscillations corresponding to the hopping cycle, with the forward velocity remaining relatively low.

These results suggest that the Raibert heuristic method is effective at maintaining a stable hopping pattern but is limited in achieving significant forward movement.

Position and Velocity Over Time For PPO Method. The position and velocity of the robot in the X, Y, and Z directions using the PPO method are shown in the figures below:

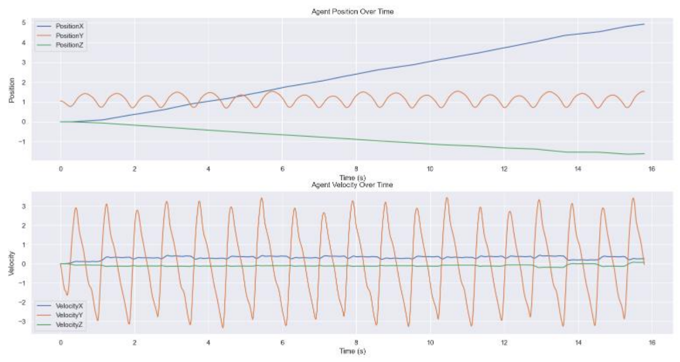

Fig. 6. Trajectory of Raibert Heuristic method

The results demonstrate that the PPO method not only achieves better forward movement but also maintains a stable hopping pattern, making it a more effective control strategy.

Training Performance of PPO. To better understand the training dynamics of the PPO-based reinforcement learning method, we analyze the changes in policy loss, value loss, and cumulative reward over the course of training. The following figures illustrate these metrics across episodes:

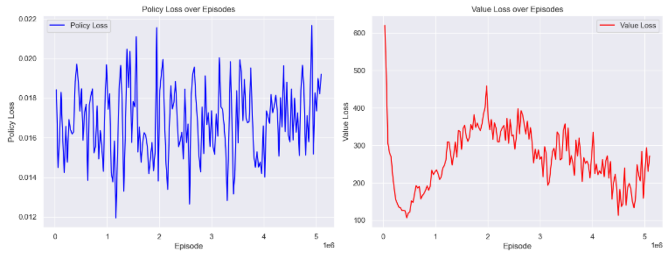

Fig. 7. Loss functions over training episodes

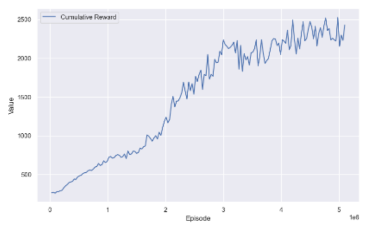

Fig. 8. Cumulative reward over training episodes

The upper left graph depicts the policy loss over training episodes. The policy loss fluctuates within a relatively stable range, indicating that the PPO algorithm maintains a consistent balance between exploration and exploitation during training. Despite some fluctuations, the policy loss remains relatively low, suggesting stable policy updates throughout the training process.

The upper right graph shows the value loss, which represents the error in the value function approximation. The value loss starts high and decreases significantly during the initial phase of training, indicating rapid improvement in value estimation. However, after the initial drop, the value loss shows more variability, reflecting the challenges in accurately predicting the expected returns as the policy evolves.

The cumulative reward shows a clear upward trend, indicating that the agent's performance improves steadily as training progresses. Initially, the cumulative reward increases rapidly, reflecting the agent's quick adaptation to the environment and the learning of basic hopping dynamics. As training continues, the reward growth slows down and exhibits occasional fluctuations, suggesting that the agent is refining its policy and adapting to more nuanced aspects of the task, such as optimizing energy efficiency and maintaining stability.

Around 2 million episodes, the cumulative reward stabilizes, indicating that the agent has converged to an effective policy. The slight variations in reward after convergence suggest ongoing fine-tuning and response to minor perturbations in the environment.

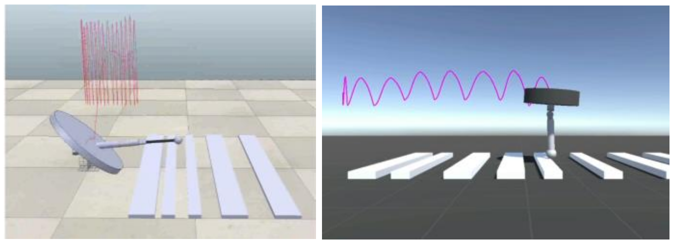

Fig. 9 . Rough Terrain Test. Left: Robot Falls with Raibert Heuristic; Right: Robot Hops Robustly with Reinforcement Learning.

Robustness of the Two Methods. We compare the robustness of the two methods through hopping on a rough terrain. As shown in Fig. 9, it can be seen that the robot falls with Raibert heuristic method as soon as the robot touches the edges, while the robot hops robustly all over the rough terrain with reinforcement learning method, which is also shown in the video attachment: https://www.bilibili.com/video/BV1eTxgeuEjV/?vd_source=25bf190003ff1ebd36e7649d3641e141.

IV. CONCLUSION

This study provides a comparative analysis of the traditional Raibert Heuristic and a modern reinforcement learning (RL)-based control method for a 3D monopod robot. While the Raibert Heuristic proves effective in stable environments due to its simplicity, it struggles with dynamic and unpredictable terrains, as it relies on pre-defined control parameters and a finite state machine. In contrast, the RL-based method, utilizing the Proximal Policy Optimization (PPO) algorithm, demonstrates superior adaptability and performance by continuously learning an optimal control policy through interaction with the environment.

The key findings of this paper include:

- Stability and Performance: The RL method outperforms the Raibert Heuristic in maintaining stability and achieving consistent hopping motions. It successfully adapts to varying terrain conditions by dynamically adjusting the control strategy based on proprioceptive feedback. In contrast, the Raibert Heuristic, while maintaining stability on flat terrain, struggles with even slight environmental variations.

- Hopping Speed: The RL-based approach achieves significantly higher and more stable forward velocities compared to the Raibert Heuristic. This is attributed to the RL agent's ability to optimize foot placement and body dynamics through trial and error, resulting in more efficient and powerful leg movements. The Raibert Heuristic, constrained by its static control parameters, is limited in achieving similar performance.

Overall, the study suggests that while the Raibert Heuristic is suitable for simple, controlled environments, RL-based methods excel in complex, dynamic scenarios. Future research could explore hybrid control strategies that combine the fast-response capabilities of heuristics with the adaptability of RL. Such approaches could leverage the robustness and computational efficiency of heuristics for basic stability control while employing RL for continuous adaptation and optimization in challenging environments.

In conclusion, RL-based methods, despite their higher computational demands, offer significant advantages in adaptability, stability, and performance for legged robot control. These methods provide a robust framework for advancing autonomous monopod robots, making them better equipped to handle diverse real-world terrains and dynamic challenges.

REFERENCES

[1] Pratt, G. A., & Williamson, M. M. (1995). Series elastic actuators. Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 1, 399-406.

[2] Raibert, M. H. (1986). Legged robots that balance. MIT Press.

[3] Raibert, M. H., & Hodgins, J. K. (1991). Animation of dynamic legged locomotion. ACM SIGGRAPH Computer Graphics, 25(4), 349-358.

[4] Hodgins, J. K., & Raibert, M. H. (1991). Biped gymnastics. The International Journal of Robotics Research, 10(4), 243-262.

[5] Wisse, M., & van Frankenhuyzen, J. (2006). Design and control of a walking robot with compliant ankles and segmented feet. Robotics and Autonomous Systems, 54(8), 625-631.

[6] Hurst, J. W., & Rizzi, A. A. (2008). Series compliance for an efficient running gait. IEEE Robotics & Automation Magazine, 15(3), 42-51.

[7] Blickhan, R. (1989). The spring-mass model for running and hopping. Journal of Biomechanics, 22(11-12), 1217-1227.

[8] McMahon, T. A., & Cheng, G. C. (1990). The mechanics of running and the efficiency of locomotion. Journal of Biomechanics, 23, 65-78.

[9] Farley, C. T., & González, O. (1996). Leg stiffness and stride frequency in human running. Journal of Biomechanics, 29(2), 181-186.

[10] Todorov, E. (2004). Optimality principles in sensorimotor control. Nature Neuroscience, 7(9), 907-915.

[11] Qian, X., & Su, Y. (2018). Robust model predictive control for underactuated balancing robots. IEEE Transactions on Control Systems Technology, 26(2), 564-571.

[12] Mayne, D. Q. (2014). Model predictive control: Recent developments and future promise. Automatica, 50(12), 2967-2986.

[13] Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: A survey. The International Journal of Robotics Research, 32(11), 1238-1274.

[14] Lillicrap, T. P., et al. (2016). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

[15] Schulman, J., et al. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

[16] Miki, T., et al. (2022). Terrain-adaptive locomotion skills using reinforcement learning and trajectory optimization. arXiv preprint arXiv:2202.02872.

[17] Levine, S., Finn, C., Darrell, T., & Abbeel, P. (2016). End-to-end training of deep visuomotor policies. The Journal of Machine Learning Research, 17(1), 1334-1373.

[18] Zhao, X., & Collins, S. H. (2020). An adaptive machine learning framework for controlling bipedal robots with a spring-mass model. IEEE Transactions on Robotics, 36(3), 767-781.

[19] Kumar, A., et al. (2021). Learning to Run with a Model Predictive Controller: Self-Supervised Visual MPC with Keypoint Detection. arXiv preprint arXiv:2107.09240.

[20] Gu, S., et al. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. 2017 IEEE International Conference on Robotics and Automation (ICRA), 3389-3396.

[21] Hutter, M., et al. (2017). Anymal: A highly mobile and dynamic quadrupedal robot. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 38-44.

[22] Peng, X. B., et al. (2018). DeepMimic: Example-guided deep reinforcement learning of physics-based character skills. ACM Transactions on Graphics (TOG), 37(4), 1-14.