Continuous Versatile Jumping Using Learned Action Residuals

Yuxiang Yang YUXIANGY@CS.WASHINGTON.EDU

Xiangyun Meng XIANGYUN@CS.WASHINGTON.EDU

Wenhao Yu MAGICMELON@GOOGLE.COM

Tingnan Zhang TINGNAN@GOOGLE.COM

Jie Tan JIETAN@GOOGLE.COM

Byron Boots BBOOTS@CS.WASHINGTON.EDU

University of Washington
Robotics at Google

Editors: N. Matni, M. Morari, G. J. Pappas

Abstract

Jumping is essential for legged robots to traverse through difficult terrains. In this work, we propose a hierarchical framework that combines optimal control and reinforcement learning to learn continuous jumping motions for quadrupedal robots. The core of our framework is a stance controller, which combines a manually designed acceleration controller with a learned residual policy. As the acceleration controller warm starts policy for efficient training, the trained policy overcomes the limitation of the acceleration controller and improves the jumping stability. In addition, a low-level whole-body controller converts the body pose command from the stance controller to motor commands. After training in simulation, our framework can be deployed directly to the real robot, and perform versatile, continuous jumping motions, including omni-directional jumps at up to 50cm high, 60cm forward, and jump-turning at up to 90 degrees. Please visit our website for more results: `https://sites.google.com/view/learning-to-jump`.

Keywords: Legged Locomotion, Robot Agility, Optimal Control, Reinforcement Learning

1. Introduction

Jumping can greatly extend the capabilities of legged robots. Compared to walking, jumping exhibits a long ”air phase”, where all legs leave the ground at the same time. This air phase enables the robot to traverse through large areas without making contact, which is essential for difficult terrains with large gaps or abrupt height changes. While recent works have greatly improved the speed (Di Carlo et al., 2018; Kim et al., 2019; Margolis et al., 2022) and robustness (Kumar et al., 2021; Agarwal et al., 2022; Miki et al., 2022) of legged robots, most of them focused on standard walking behaviors with continuous foot contacts. Meanwhile, long-distance jumping is still a difficult task, and usually requires manual trajectory design (Li et al., 2022a; Park et al., 2021) as well as long periods of offline planning (Chignoli et al., 2022; Winkler et al., 2018).

In this work, we present a hierarchical learning framework for quadrupedal robots to jump continuously, where the jumping direction and distance can be specified online. Continuous jumping has long been a difficult task for legged robots due to complex robot dynamics and frequent, abrupt contact changes. On the one hand, while optimal control based controllers have achieved robust walking in many quadruped platforms (Di Carlo et al., 2018; Grandia et al., 2019), they usually assume a simplified dynamics model for computation efficiency (Di Carlo et al., 2018), and cannot control the robot pose precisely in highly dynamic motions like jumping (Chignoli et al., 2022). On the other hand, despite recent success in learning for locomotion (Kumar et al., 2021; Margolis et al., 2022; Miki et al., 2022), reinforcement learning (RL) based controllers still require careful reward tuning (Peng et al., 2020) and extended training times to learn jumping motions, due to the non-smooth reward landscape created by abrupt contact changes. Therefore, it can be difficult to use standard control or learning techniques for the jumping task.

Our framework addresses the challenges above, and learns continuous, versatile jumping motions that can be transferred directly to the real world. The core of our framework is a *stance controller*, which computes desired body pose by summing over the outputs from a manually-designed *acceleration controller* and a learned *residual policy*. Our design of the stance controller has two major benefits. Firstly, warm-starting the policy training with the acceleration controller reduces noises in the reward landscape, so that training process converges smoothly and efficiently. Secondly, with the residual policy trained, the robot performance is no longer limited by the simplified dynamics model used by the acceleration controller, and therefore can better stabilize the robot throughout the entire jumping episode. In addition to the stance controller, we also implemented a low-level *whole-body controller* to convert the body pose command to motor actions. By combining the acceleration controller, the residual policy and the low-level whole-body controller, our framework learns continuous, versatile jumping motions automatically.

We train our framework on a simulated environment of the Go1 quadruped robot from Unitree (Unitree), and test the trained policy directly on the real robot. The trained framework enables versatile jumping motions for the robot, including jumping at different directions and distances (up to 50cm high, 60cm forward), and jump-turning (up to 90 degrees). We then conduct detailed analysis on the behavior of our overall framework, and verify that the combination the acceleration controller and residual policy can learn more stable jumping motions than each individual method. Additionally, we compare our method to end-to-end RL and find that our method is at least 1 order-of-magnitude more data efficient, thanks to the hierarchical setup and the smooth reward landscape from the acceleration controller.

In summary, the contributions of this paper include the following:

1. We propose a hierarchical framework that combines optimal control and reinforcement learning to learn continuous, versatile jumping for quadrupedal robots.

2. The trained framework can be directly transferred to the real robot and achieves continuous jumping motions at substantial height (50cm) and distance (60cm).

3. Our experiments show that the combination of controller and residual policy can learn more stable jumping motions than using either method individually.

2. Related Works

Learning Agile Locomotion Recently, researchers have made significant progress in applying reinforcement learning (RL) for quadrupedal locomotion. Using reinforcement learning, the legged robots can adapt to the environment (Kumar et al., 2021; Miki et al., 2022) and learn diverse skills such as self-righting (Hwangbo et al., 2019; Wu et al., 2022), high-speed walking (Margolis et al., 2022), goal-keeping (Huang et al., 2022). However, for successful real-robot deployment, RL-based controllers usually require significant effort in imitation learning(Peng et al., 2020), reward shaping (Kumar et al., 2021) and sim-to-real (Tan et al., 2018; Hwangbo et al., 2019; Song et al., 2020), especially for more agile tasks such as jumping. Compared to the end-to-end RL approaches, we re-design the RL task to learn the residual action (Johannink et al., 2019) that adds to a manually-designed acceleration controller. As a result, the training process consumes significantly fewer data and does not require complex reward specification. More over, the learned policy can be deployed directly to the real world without additional fine-tuning.

Optimal Control for Locomotion With recent advancements in actuator design and numerical optimization, optimal-control based controllers have enabled high-speed and robust locomotion (Di Carlo et al., 2018; Kim et al., 2019; Grandia et al., 2019) for legged robots. For computation efficiency, these controllers usually simplify the robot dynamics model, such as the single rigid body model with massless legs (Di Carlo et al., 2018; Li et al., 2022a,b), so that the optimization can run in real-time. However, these simplified models usually cannot capture the robot's orientation dynamics accurately, especially when the robot is in the air. Therefore, it can be difficult to use them for jumping tasks with long periods of air time. To optimize for more precise jumps, controllers usually need to pre-compute the entire jumping trajectory offline using higher-fidelity models (Chignoli et al., 2022; Winkler et al., 2018). Compared to these approaches, our method does not require offline optimization, jumps continuously, and can respond to different landing positions and orientations.

Combining control and learning for legged locomotion Recently, a number of works have proposed to use reinforcement learning and optimal control jointly for robust and versatile locomotion. A common approach is to set up the controller hierarchically, where a high-level policy outputs intermediate commands, which are converted by a low-level controller into motor actions. While such hierarchical approaches have enabled the robot to walk more efficiently (Yang et al., 2022; Da et al., 2020) and conquer difficult terrains (Xie et al., 2021), they are not yet demonstrated on more agile tasks such as jumping. We use a similar hierarchical setup, but uses a different whole-body controller in the low-level for precise tracking of body acceleration, and achieves continuous, versatile jumping on the real robot. Another common approach is to use RL to finetune the controller's outputs. Recently, Bellegarda and Nguyen (2020) demonstrates that deep RL can improve the quality of a single jump in simulation. Our work extends their result by using RL to optimize for the controller's performance in the real world, and achieves continuous, omni-directional jumping on the real robot, as well as jump turns.

3. Overview

We design a hierarchical framework to learn continuous robot jumping (Fig. 1). The timing of each jump is regulated by an open-loop time-based *contact scheduler*, which subsequently specifies the desired state of each leg (*swing* or *stance*). We adopt the pronking gait, where all legs enter and leave the ground at the same time, and set the duration of each jump to be 1 second, which consists of 0.5s of stance time and 0.5s of swing time. Based on the output from the contact scheduler, we use separate control strategies for swing and stance legs, where the *stance controller* controls the desired body pose and *swing controller* controls the foot positions. At the low level, a whole-body

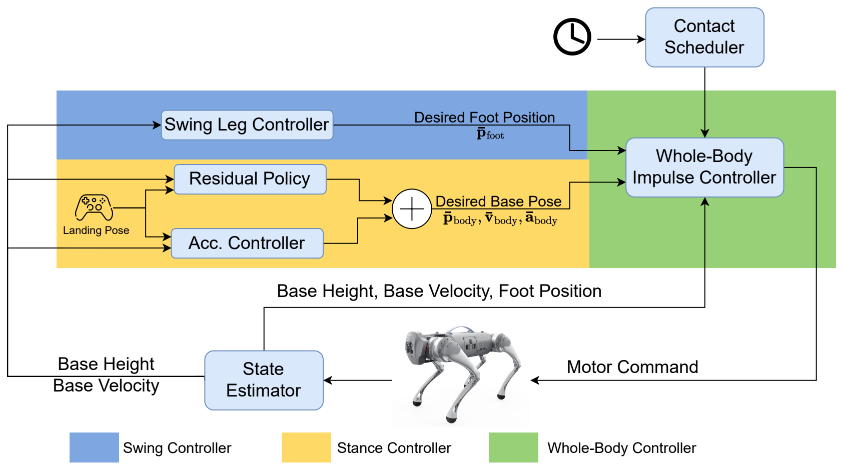


Figure 1: Our hierarchical learning framework controls the jumping of the Go1 robot from two different levels.

controller converts the pose command from the swing or stance controller to motor commands. We also implemented a Kalman Filter based state estimator to estimate the position and velocity of the robot. We run the entire pipeline at 500Hz so that the robot can respond quickly to external perturbations.

As a critical component of the entire control process, the stance controller needs to achieve sufficient lift-off speed for each jump, while maintaining body stability throughout the entire jump. To achieve that, we compute the pose command of the stance controller as the sum of a manually designed *acceleration controller* and a learned *residual policy*, where the acceleration controller computes base acceleration to ensure sufficient lift-off velocity based on simplified robot dynamics, and the residual policy fine-tunes the controller's action to ensure stability. Since the robot is under-actuated in the air, we use a simple trajectory controller for swing legs, which computes the desired landing position of each feet according to the Raibert Heuristic (Raibert, 1986).

4. Learning Residual Policy for Continuous, Versatile Jumping

4.1. Reinforcement Learning Preliminaries

The reinforcement learning (RL) problem is represented as a Markov Decision Process (MDP), which consists of a state space $\mathcal{S}$, an action space $\mathcal{A}$, transition probability $p(s_{t+1}|s_t, a_t)$, reward function: $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, an initial state distribution $p_0(s_0)$ and an episode length T. We aim to learn a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the expected cumulative reward over the entire episode, which is defined as:

$$J(\pi) = \mathbb{E}_{s_0 \sim p_0(\cdot), a_t \sim \pi(s_t), s_{t+1} \sim p(\cdot|s_t, a_t)} \sum_{t=0}^{T} r(s_t, a_t) \tag{1}$$

4.2. Environment setup

We design our environment so that the robot can jump in several directions within each episode. More specifically, each episode consists of 5 jumps, where the robot jumps in-place, 1m forward, 0.5m backward, 0.2m to the left, and 0.2m to the right, in that order. Before each jump, the environment records the robot's current position and computes the desired landing position based on the jumping directions. This information is then supplied to the residual policy to compute the desired pose commands, which is subsequently sent to the low-level whole-body controller. The details of our environment are as follows:

State Space The state space includes the robot state and task information. More specifically, the robot state includes the position, orientation, linear velocity, and angular velocity of the base, as well as the foot positions. The task information includes the robot's distance to the desired landing position, and the remaining time in the current locomotion cycle.

Action Space Since the low-level whole-body controller is effective for precise tracking of the body pose (Kim et al., 2019), we directly command the desired body pose in our environment. In the original work by Kim et al. (2019), the whole-body controller takes in an 18-dimensional vector specifying the position, velocity, and acceleration for each of the base's 6 DoFs. To reduce the search dimension, we design the action space to specify one command for each DoF of the base, and computes the other two dimensions of each DoF heuristically. More specifically, the action space specifies the desired linear *acceleration* (3-dimensional) and angular *acceleration* around the z axis (1-dimensional), so that the policy can directly control the planar and vertical movements of the body, as well as turning, which can change rapidly within each jump. The action space specifies the desired *position* for the remaining two DoFs, namely the body roll and pitch, to avoid unnecessary body oscillations. The remaining commands for each DoF is specified heuristically. See section. 5 for more details.

Reward We design the reward function as the linear combination of alive bonus, distance penalty, orientation penalty and contact consistency penalty:

$$r = r_{\text{alive}} + w_p \cdot r_{\text{position}} + w_o \cdot r_{\text{orientation}} + w_c \cdot r_{\text{contact}} \tag{2}$$

where the alive bonus, $r_{\text{alive}} = 4$, is a fixed constant that makes the total return positive. The position reward, r_{position} is the squared distance between the robot's current position and the desired landing position, normalized by the total desired jumping distance. The orientation reward, $r_{\text{orientation}} = -(\text{roll}^2 + \text{pitch}^2)$ encourages the robot to stay upright. The contact consistency reward $r_{\text{contact}} = \sum_{i=1}^{4} \mathbb{1}(c_i \neq \hat{c}_i)$ is the sum of 4 indicator functions about the contact situation of each leg, where $\hat{c}_i \in \{0, 1\}$ is the desired contact of foot i according to contact scheduler (Section. 3) and $c_i \in \{0, 1\}$ is the actual contact of foot i. Intuitively, the contact consistency reward ensures that the robot jumps according to the desired schedule without significant mismatch in lift-off and touch-downs. We use $w_p = 1, w_o = 5, w_c = 0.4$ in all our experiments.

Early Termination To prevent the learning algorithm from unnecessary explorations in suboptimal regions, we terminate an episode early if the robot falls (i.e. when the base height is lower than 8cm, when the cosine distance between body's upright vector and the gravity direction is less than 0.6, or when any body parts came in contact with the ground).

4.3. Manually-designed Acceleration Controller

Due to the discrete contact change, the reward landscape in the jumping environment can be highly non-smooth with local minima, which makes it challenging to learn using standard exploration

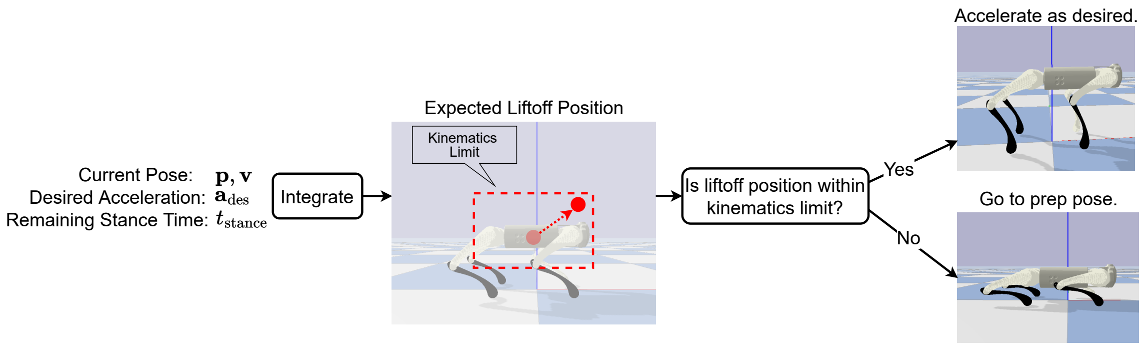

Current Pose: $\mathbf{p}, \mathbf{v}$
Desired Acceleration: $\mathbf{a}_{\text{des}}$
Remaining Stance Time: t_{stance}
Integrate
Expected Liftoff Position
Kinematics Limit
Is liftoff position within kinematics limit?
Yes
No
Accelerate as desired.
Go to prep pose.

Figure 2: The acceleration controller decides whether to execute the desired acceleration command $\boldsymbol{a}_{\text{des}}$ based on the estimated lift-off position, which is computed by numerical integration. If the lift-off position (dark red dot) is within the approximated kinematics limit (dashed square), the controller executes the $\boldsymbol{a}_{\text{des}}$. Otherwise the controller moves the robot to a low-standing preparation pose.

strategies in RL algorithms. To facilitate learning, we manually design an acceleration controller as the base policy for the environment, and uses reinforcement learning to learn *residual actions* to finetune the policy's performance. The acceleration controller models the robot body as a single point mass, computes the desired lift-off velocity based on contact timing, and tracks this lift-off velocity using simple heuristics.

Computing the Lift-off Velocity The acceleration controller computes the desired lift-off velocity $\boldsymbol{v}_{\text{liftoff}} = (v_x, v_y, v_z, v_{\text{yaw}})$ based on the desired landing displacement p_x, p_y, p_{yaw} and the swing time t_{swing}. The planar velocities, $v_x = \frac{p_x}{t_{\text{swing}}}, v_y = \frac{p_y}{t_{\text{swing}}}, v_{\text{yaw}} = \frac{p_{\text{yaw}}}{t_{\text{swing}}}$, is the average flying speed required for the robot to land at the desired position and orientation. The vertical velocity, $v_z = \frac{1}{2} g t_{\text{swing}}$, is the minimum vertical velocity for the robot to maintain the desired swing time, where g is the gravity constant.

Tracking the Lift-off Velocity To track the desired lift-off velocity $\boldsymbol{v}_{\text{liftoff}}$, the acceleration controller computes the desired acceleration $\boldsymbol{a}_{\text{des}} = \frac{\boldsymbol{v}_{\text{liftoff}} - \boldsymbol{v}}{t}$ based on the remaining time in the stance phase t, the desired liftoff velocity $\boldsymbol{v}_{\text{liftoff}}$ and the current CoM velocity $\boldsymbol{v}$. The acceleration controller then either executes this acceleration $\boldsymbol{a}_{\text{des}}$, or moves the robot to a preparation position, based on an estimation of the lift-off position (Fig. 2). More specifically, the acceleration controller assumes the robot as a point-mass, and computes the body's CoM position at lift-off time based on the current pose and desired accelerations. If the lift-off position violates kinematics limits (e.g. if the base is so high that foot contact cannot be maintained), the controller computes an alternative acceleration that moves the robot to a low-standing preparation position. Otherwise, the controller outputs the desired acceleration $\boldsymbol{a}_{\text{des}}$. To simplify computation, we approximate the feasible CoM positions as a bounding box around the robot's current CoM.

4.4. Training the Residual Policy

To improve the performance of the acceleration controller, we train a policy to add an action residual to the acceleration controller's outputs. We represent our policy as a neural network with 1 hidden layer of 256 units and tanh activation. We train our policy using Augmented Random Search (ARS) (Mania et al., 2018), a simple, parallelizable evolutionary algorithm that has been successfully applied to locomotion tasks (Iscen et al., 2018; Tan et al., 2011, 2016; Geijtenbeek et al., 2013). We chose ARS because it explores in the policy parameter space and does not directly inject noise in action outputs. Moreover, ARS evaluates policies require accurate estimation of the value function, which can be challenging for hierarchical tasks (Yang et al., 2022) due to its non-Markovian nature.

5. Low-level Whole-body Controller

At the low-level, we use a whole-body controller (WBC) to convert the body and foot pose commands into motor actions. Our implementation is based on the work by Kim et al. (2019) with a few modifications. We briefly summarize the controller design here. Please refer to the original work for further details.

Interface with Stance Controller In summary, WBC takes in a 18-dimensional vector that specifies the desired pose $\boldsymbol{p}_{\text{body}}$, velocity $\boldsymbol{v}_{\text{body}}$, and acceleration $\boldsymbol{a}_{\text{body}}$ for each of the 6 DoFs of the robot *body*, as well as the foot swing positions $\boldsymbol{p}_{\text{foot}}$. The foot swing positions $\boldsymbol{p}_{\text{foot}}$ is directly specified by the swing controller (Section. 3). The stance controller (Section. 4.2) specifies 4 out of the 6 dimensions for the base accelerations $\boldsymbol{a}_{\text{body}}$ (3 linear accelerations and angular accelerations around the z axis), as well as 2 out of the 6 dimensions of the base pose $\boldsymbol{p}_{\text{body}}$ (roll and pitch). For the remaining pose commands, we set the desired body pose p_{body} to be the current body pose, the desired linear body velocity to the current velocity, the desired angular body velocity to 0, and the desired angular acceleration around the x, y axis to 0.

Computation of Motor Commands WBC computes an impedance command that specifies the desired position $\bar{\boldsymbol{q}}$, velocity $\bar{\dot{\boldsymbol{q}}}$ and torque $\bar{\boldsymbol{\tau}}$ for each *motor*. The applied torque $\boldsymbol{\tau}$ is the sum of the desired torque plus the PD feedback:

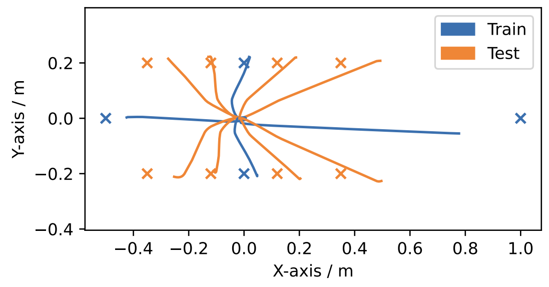
$$\boldsymbol{\tau} = k_p(\bar{\boldsymbol{q}} - \boldsymbol{q}) + k_d(\bar{\dot{\boldsymbol{q}}} - \dot{\boldsymbol{q}}) + \bar{\boldsymbol{\tau}} \tag{3}$$

where $\boldsymbol{q}, \dot{\boldsymbol{q}}$ is the current position and velocity of each motor, and k_p, k_d are fixed gains. To compute the motor command, WBC first applies an inverse kinematics algorithm, which computes the desired position $\bar{\boldsymbol{q}}$ and velocity $\bar{\dot{\boldsymbol{q}}}$ for each motor, in order to move the robot to the desired body position $\boldsymbol{p}_{\text{body}}$ and velocity $\boldsymbol{v}_{\text{body}}$. After that, WBC computes the additional motor torque $\bar{\boldsymbol{\tau}}$ required to achieve the desired base accelerations $\boldsymbol{a}_{\text{body}}$, based on the full rigid-body dynamics model of the robot.

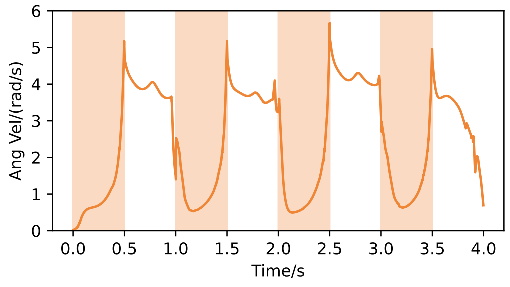
6. Results

6.1. Experiment Setup

We test our framework on a Go1 robot from Unitree (Unitree), which is a small-scale, 15kg quadrupedal robot with 12 degrees of freedom. We build the simulation environment of Go1 using Pybullet (Coumans and Bai, 2016–2020), and implement the entire control pipeline, including the the state estimator, low-level WBC, the jump controller and the residual policy in Python. We train the residual policy on a desktop computer with a 16-core CPU, where the training takes around 3 hours to complete.



(a) Omni-directional jumping: CoM trajectory



(b) Jump-turn: z-component of angular velocity

Figure 3: Different jumping skills learned by our framework. **Left**: Omni-directional jumping. Lines show birds-eye view of CoM trajectory, where the robot starts facing positive x direction. Crosses show desired landing positions. Colors show whether the direction is seen during training. **Right**: Continuous jump-turn. Plot shows the z-component of angular velocity (approximately the change rate of yaw angle). Shaded area indicates foot contact.

6.2. Continuous, Versatile Jumping

We test the performance of our learned framework on a series of jumping tasks, where each task specifies either a different desired jumping direction, or a desired turning rate (Fig. 7). Although we train the framework in only 4 jumping directions, the resulting controller interpolates between them and jumps in intermediate directions (fig. 3a). The framework also enables the robot to jump and turn *simultaneously*, and achieves an average turning rate of about 3.5 rad/s (fig. 3b). For omni-directional jumping, we also notice that the controller tends to overshoot for forward jumps, and undershoot for backward jumps. This is likely due to the asymmetric leg designs of the robot platform, which generates higher accelerations in the forward direction than in the backward direction.

6.3. Transfer to the real robot

We deploy the learned residual policy directly to the real world without additional finetuning. Thanks to the robustness of the low-level WBC, our framework can complete several high jumps in the real world, including jumping in multiple directions and turning (Fig. 4). Please visit the website for videos. The robot achieves a maximum jumping height of around 50cm, a forward jumping distance of around 60cm, and a maximum turning rate of 90 degrees per jump. Note that this jumping performance in the real world is slightly lower than the robot's performance in simulation. We hypothesize this as a result of unmodeled motor saturation, and plan to investigate further in future works.

6.4. Comparison with baseline policies

To further validate our design choices, we conduct an ablation study by removing either the residual policy or the acceleration controller from our pipeline. We also compare our framework with an end-to-end RL policy that directly outputs motor position commands. The result is summarized in figure. 5.

Acceleration Controller Only We test the performance of the manually designed acceleration controller without learned residual policy on the same jumping task. While the controller completes

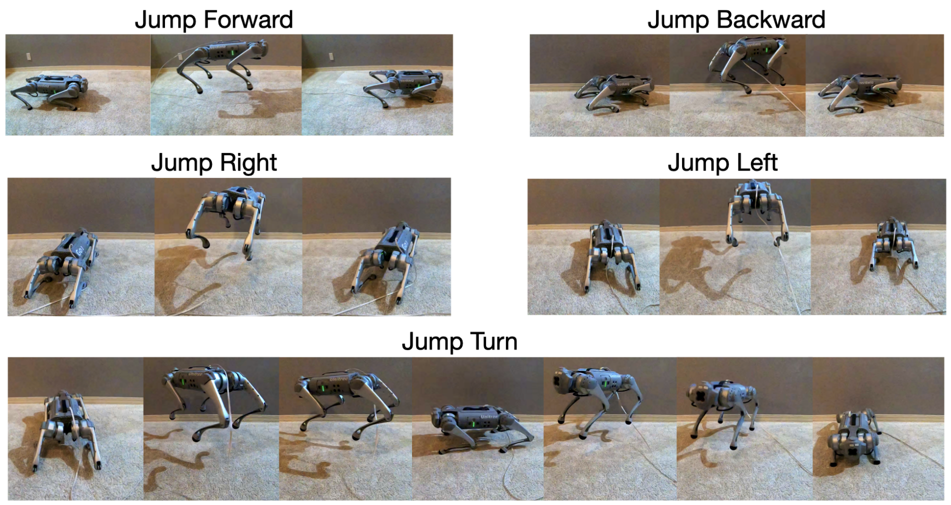


Figure 4: Footages of the Go1 robot completing several jumps in the real world.

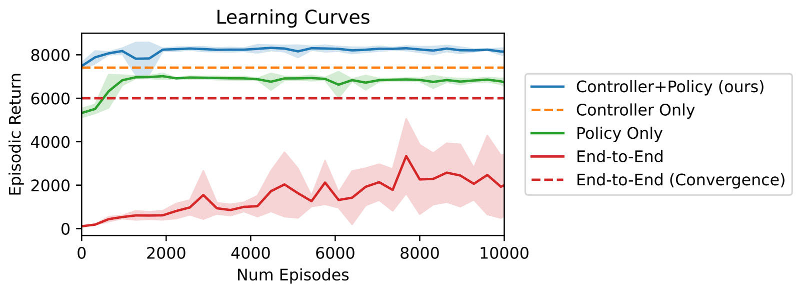


Figure 5: Learning curves of our framework compared with baseline methods. Error bar shows 1 standard deviation.

all the jumps without falling over, it achieves a lower reward without the residual policy (Fig. 5). We also find that the residual policy increases the overall success rate of the robot in the real world (Table. 1). To further understand the effectiveness of the residual policy, we perform a backward jump with and without the residual policy, and plot the base pitch angle in Fig. 6. While the acceleration controller maintains the body pitch closer to reference in the stance phase, it causes significantly larger pitch angle deviations in the swing phase. This is because the acceleration controller approximates the robot as a point-mass, and cannot account for changes in body orientations. In contrast, the residual policy learns to correct this pitch angle deviation by slightly shifting the body pitch in stance phase, which results in lower overall pitch deviations throughout the entire jump.

Policy Only Without the acceleration controller, the residual policy gets stuck in a local minima, and achieves a low reward (Fig. 5). The resulting policy does not achieve sufficient height for each jump, and keeps legs in contact most of the times (Fig. 7a). We hypothesize this as a result of the frequent contact changes in the environment, which can create a noisy reward landscape for the algorithm to optimize. In contrast, the policy trained with the controller achieves consistent, high flight times for each jump (Fig. 7b).

Task	Controller + Policy	Controller Only
Forward	100%	20%
Backward	80%	0%
Left	100%	80%
Right	100%	60%
Jump-Turn	100%	0%

Table 1: Success rates (over 5 trials) with or without the residual policy. A jump is successful if it does not trigger the early termination condition (Sec. 4.2).

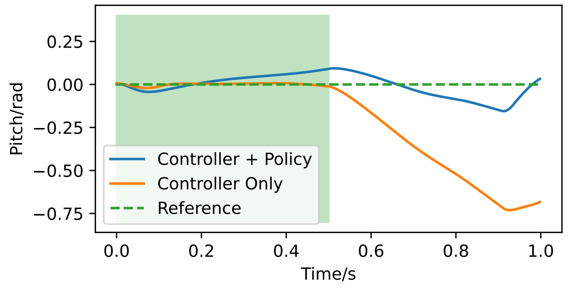
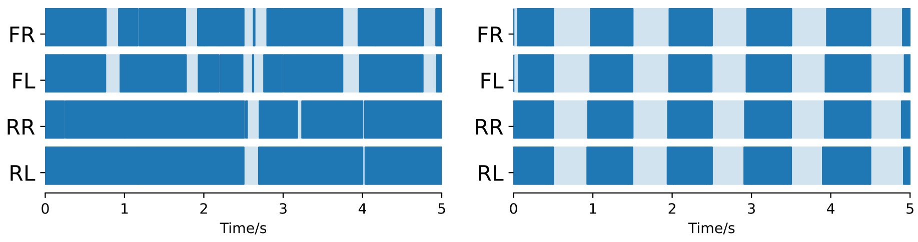


Figure 6: Robot pitch angle during a backward jump using different controllers. Shaded area shows stance phase.



(a) Policy Only.

(b) Controller + Policy (ours).

Figure 7: Foot contacts for controllers trained without and with the jump controller. Dark area indicates foot contact.

End-to-End RL Lastly, we compare our method to a baseline, where we train an end-to-end reinforcement learning policy that directly outputs motor position commands. Similar to previous works (Kumar et al., 2021; Miki et al., 2022; Tan et al., 2018), we reduce the control frequency to 50Hz to avoid high-frequency motor oscillations. We use the same state space and reward function in the environment design, and train the policy using the same ARS algorithm. We find that the E2E policy learns significantly slower, and requires around 10 times more training episodes (Fig. 5). The policy also achieves the lowest overall reward compared to the other policies. While additional efforts in reward shaping and imitation learning (Peng et al., 2020) can improve the performance of the E2E policy, our method creates a simple, data-efficient alternative that does not require pre-recorded demonstration trajectories.

7. Conclusion

In this work, we present a hierarchical framework to for continuous quadruped jumping, which consists of a manually designed acceleration controller, a learned residual policy, and a low-level whole-body controller. The trained framework can be transferred directly to the real world, and is capable of continuous jumping at arbitrary angle and distances according to user specification, as well as jump-turning. One limitation of our work is that it currently only supports the pronking gait, where all 4 legs leave or touch the ground at the same time. Supporting more versatile gaits such as bounding or galloping can potentially increase the height and distance of the jump, which we plan to investigate in future work. Another future direction is to integrate perception into our framework, so that the robot can use its jumping skills to traverse through difficult terrains autonomously.

References

Ananye Agarwal, Ashish Kumar, Jitendra Malik, and Deepak Pathak. Legged locomotion in challenging terrains using egocentric vision. *arXiv preprint arXiv:2211.07638*, 2022.

Guillaume Bellegarda and Quan Nguyen. Robust quadruped jumping via deep reinforcement learning. *arXiv preprint arXiv:2011.07089*, 2020.

Matthew Chignoli, Savva Morozov, and Sangbae Kim. Rapid and reliable quadruped motion planning with omnidirectional jumping. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6621–6627. IEEE, 2022.

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2020.

X Da, Z Xie, D Hoeller, B Boots, A Anandkumar, Y Zhu, B Babich, and A Garg. Learning a contact-adaptive controller for robust. *Efficient Legged Locomotion. arXiv*, 200910019, 2020.

Jared Di Carlo, Patrick M Wensing, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. Dynamic locomotion in the mit cheetah 3 through convex model-predictive control. In *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 1–9. IEEE, 2018.

Thomas Geijtenbeek, Michiel Van De Panne, and A Frank Van Der Stappen. Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics (TOG)*, 32(6):1–11, 2013.

Ruben Grandia, Farbod Farshidian, René Ranftl, and Marco Hutter. Feedback mpc for torque-controlled legged robots. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4730–4737. IEEE, 2019.

Xiaoyu Huang, Zhongyu Li, Yanzhen Xiang, Yiming Ni, Yufeng Chi, Yunhao Li, Lizhi Yang, Xue Bin Peng, and Koushil Sreenath. Creating a dynamic quadrupedal robotic goalkeeper with reinforcement learning. *arXiv preprint arXiv:2210.04435*, 2022.

Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

Atil Iscen, Ken Caluwaerts, Jie Tan, Tingnan Zhang, Erwin Coumans, Vikas Sindhwani, and Vincent Vanhoucke. Policies modulating trajectory generators. In *Conference on Robot Learning*, pages 916–926. PMLR, 2018.

Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6023–6029. IEEE, 2019.

Donghyun Kim, Jared Di Carlo, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control. *arXiv preprint arXiv:1909.06586*, 2019.

Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. 2021.

He Li, Tingnan Zhang, Wenhao Yu, and Patrick M Wensing. Versatile real-time motion synthesis via kino-dynamic mpc with hybrid-systems ddp. *arXiv preprint arXiv:2209.14138*, 2022a.

He Li, Tingnan Zhang, Wenhao Yu, and Patrick M Wensing. Zero-shot retargeting of learned quadruped locomotion policies using hybrid kinodynamic model predictive control. *arXiv preprint arXiv:2209.14123*, 2022b.

Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018.

Gabriel B Margolis, Ge Yang, Kartik Paigwar, Tao Chen, and Pulkit Agrawal. Rapid locomotion via reinforcement learning. *arXiv preprint arXiv:2205.02824*, 2022.

Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62):eabk2822, 2022.

Hae-Won Park, Patrick M Wensing, and Sangbae Kim. Jumping over obstacles with mit cheetah 2. *Robotics and Autonomous Systems*, 136:103703, 2021.

Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. *arXiv preprint arXiv:2004.00784*, 2020.

Marc H Raibert. *Legged robots that balance*. MIT press, 1986.

Xingyou Song, Yuxiang Yang, Krzysztof Choromanski, Ken Caluwaerts, Wenbo Gao, Chelsea Finn, and Jie Tan. Rapidly adaptable legged robots via evolutionary meta-learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3769–3776. IEEE, 2020.

Jie Tan, Yuting Gu, Greg Turk, and C Karen Liu. Articulated swimming creatures. *ACM Transactions on Graphics (TOG)*, 30(4):1–12, 2011.

Jie Tan, Zhaoming Xie, Byron Boots, and C Karen Liu. Simulation-based design of dynamic controllers for humanoid balancing. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2729–2736. IEEE, 2016.

Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

Unitree. Go1 Website.

Alexander W Winkler, C Dario Bellicoso, Marco Hutter, and Jonas Buchli. Gait and trajectory optimization for legged systems through phase-based end-effector parameterization. *IEEE Robotics and Automation Letters*, 3(3):1560–1567, 2018.

Philipp Wu, Alejandro Escontrela, Danijar Hafner, Ken Goldberg, and Pieter Abbeel. Daydreamer: World models for physical robot learning. *arXiv preprint arXiv:2206.14176*, 2022.

Zhaoming Xie, Xingye Da, Buck Babich, Animesh Garg, and Michiel van de Panne. Glide: Generalizable quadrupedal locomotion in diverse environments with a centroidal model. *arXiv preprint arXiv:2104.09771*, 2021.

Yuxiang Yang, Tingnan Zhang, Erwin Coumans, Jie Tan, and Byron Boots. Fast and efficient locomotion via learned gait transitions. In *Conference on Robot Learning*, pages 773–783. PMLR, 2022.