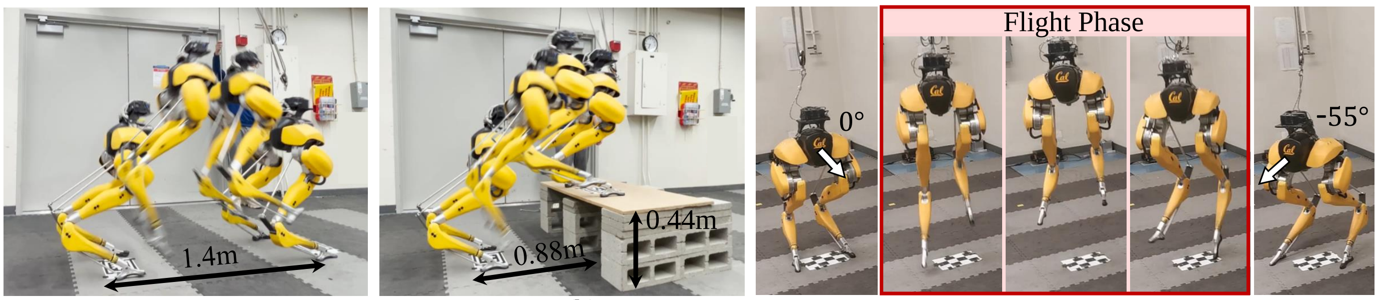
Robust and Versatile Bipedal Jumping Control through Multi-Task Reinforcement Learning

Zhongyu Li[1], Xue Bin Peng[2], Pieter Abbeel[1], Sergey Levine[1], Glen Berseth[3,4], and Koushil Sreenath[1]
[1]University of California, Berkeley, [2]Simon Fraser University, [3]Université de Montréal, [4]Mila
Email: zhongyu_li@berkeley.edu, xbpeng@sfu.ca, pabbeel@cs.berkeley.edu, svlevine@eecs.berkeley.edu, glen.berseth@mila.quebec, koushils@berkeley.edu



(a) (b) (c)

Fig. 1: Representative dynamic jumping maneuvers performed by a bipedal robot Cassie using the proposed multi-task control policies. From left to right: (a) the robot jumps over 1.4 m and lands at the given target; (b) the robot jumps to a target that is 0.88 m in front of the robot and 0.44 m above the ground, and (c) the robot jumps in place while turning 55° with a command to turn 60° in place. The policies are trained in simulation and deployed on the hardware without further tuning. Video is at: https://youtu.be/aAPSZ2QFB-E.

***Abstract*—This work aims to push the limits of agility for bipedal robots by enabling a torque-controlled bipedal robot to perform robust and versatile dynamic jumps in the real world. We present a multi-task reinforcement learning framework to train the robot to accomplish a large variety of jumping tasks, such as jumping to different locations and directions. To improve performance on these challenging tasks, we develop a new policy structure that encodes the robot's long-term input/output (I/O) history while also providing direct access to its short-term I/O history. In order to train a versatile multi-task policy, we utilize a multi-stage training scheme that includes different training stages for different objectives. After multi-stage training, the multi-task policy can be directly transferred to Cassie, a physical bipedal robot. Training on different tasks and exploring more diverse scenarios leads to highly robust policies that can exploit the diverse set of learned skills to recover from perturbations or poor landings during real-world deployment. Such robustness in the proposed multi-task policy enables Cassie to succeed in completing a variety of challenging jump tasks in the real world, such as standing long jumps, jumping onto elevated platforms, and multi-axis jumps.**

I. Introduction

One question that has been lingering since the first creation of bipedal robots is [22, 51]: how can we enable such complex robots to traverse complex environments using agile and robust maneuvers? For example, creating agile controllers that enable bipedal robots to jump over a given distance or onto different elevations can enable greater mobility in unstructured environments. However, jumping is a challenging skill to control for bipedal robots. During a standing jump, the robot needs to push its body off the ground and break contact, leap for a flight phase where the robot is underactuated due to conservation of angular momentum, and then make contact again when it lands on its legs while adjusting its body pose to not only recover from the large impact impulse but also stick the landing and remain standing. All of these events occur within a very short amount of time (typically less than 2 seconds). These events lead to a hybrid system that switches between modes with different contact configurations (*e.g.*, taking-off, flight, and landing). Planning and controlling for such discontinuous dynamics, especially for a bipedal robot with high dimensionality, nonlinearity, and underactuation, presents a very challenging task [50]. This challenge is further compounded when the robot needs to accurately land on a given target since the robot will now have to produce a precise translational and angular momentum at take-off in order to land at the desired location [71].

Model-based optimal control (OC) frameworks have made notable progress in controlling bipedal robots, including jumping, but they depend on carefully crafted models of the robot and the complex contact dynamics [29, 38, 55], and typically require manual design of task-specific control structures [8, 19, 74]. This often necessitates a simplified dynamics model, which only approximates the robot's full-order model, and pre-defined or pre-computed contact sequences [44, 45, 69] to reduce online optimization complexity. Such drawbacks result in previous model-based methods only being able to perform a fixed hop on torque-controlled bipedal robots like Cassie [69, 70]. By leveraging the robot's full-order dynamics, model-free reinforcement learning (RL) has shown some success in highly-dynamic locomotion control on quadrupedal

TABLE I: Benchmark with previous work tackling jumping control using optimal control (OC) or reinforcement learning (RL) on the bipedal robot Cassie in the real world.

	Controlled Landing Pose	Apex Foot Clearence	Longest Flight Phase	Maximum Leap Distance				
				Forward	Backward	Lateral	Turning	Elevation
Aperiodic Hop by OC [69]	No	0.18m	0.42s	∼0.5m*	0m	0m	0°	0m
Aperiodic Hop by OC [70]	No	0.15m	0.33s*	∼0.3m*	0m	0m	0°	0m
Periodic Hop by RL [58]	No	∼0.16m*	0.33s*	∼0.5m*	0m	0m	0°	∼0.15m*
Ours (Aperiodic Jump by RL)	**Yes**	**0.47m**	**0.58s**	**1.4m**	**-0.3m**	**±0.3m**	**±55°**	**0.44m**

*Not provided in the paper and the listed value is roughly estimated based on the comparison with the background environment in the accompanying video.

robots [25, 41, 47]. However, compared to quadrupeds that are inherently more stable, RL-based methods still struggle when applied to bipedal robots for more dynamic and aggressive maneuvers in the real world. Given that the most relevant prior RL-based system is only able to perform periodic hopping on Cassie [58], it remains an open question how more dynamic bipedal skills can be achieved in the real world, such as standing long jumps that could be more challenging than periodic motions [19, Sec. I].

A. Goal

In this paper, our goal is not limited to performing a single locomotion skill, but to exploring the possibility of creating a robust and versatile aperiodic jumping controller for a bipedal robot that enables it to land at different target locations, as shown in Fig. 1, and validating the advantages brought by learning with different skills using reinforcement learning. We hypothesize that, by exploring different jumping tasks, a multi-task jumping controller can be more robust as it allows the robot to leverage more diverse skills to maintain stability during dynamic maneuvers. For example, in order to recover from a large ground impulse on landing, the robot can quickly switch to another learned skill, such as a hop, which can allow for a more graceful recovery than simply continuing with the original behavior. We denote each jump to a different target configuration as a *task*. We use the term *multi-task* to refer to a policy that can perform a variety of jumping tasks, such as jumping over various desired distances and/or directions.

B. Contributions

The core contribution of this work is the first system that enables a life-sized torque-controlled bipedal robot to perform versatile jumping maneuvers with controlled landing locations in the real world. The robot is first trained in simulation with reinforcement learning and domain randomization. Training does not require an explicit contact sequence, and the learning algorithm automatically develops different contact sequences for different jumping tasks. In order to successfully transfer the learned skills for such dynamic maneuvers from simulation to the real world, we utilize two new design decisions. First, we present a new policy structure that encodes the robot's long-term Input-Output (I/O) history while also providing direct access to a short-term I/O history. By training the model in an end-to-end manner, we show that such a structure can outperform previously proposed architectures (Fig. 5). Second, we demonstrate that training the controller for a versatile range of tasks improves the robustness of the controller by using different learned jumping skills to recover from unstable states. We show that this robustness cannot be easily obtained by training only for a single task (Fig. 6). Combining these two techniques, we enable the bipedal Cassie robot to perform (1) several record-making jumps, such as a long jump (1.4m ahead) and a high jump onto a 0.44m-tall platform (Fig. 1), (2) various agile multi-axes bipedal jumps (Fig. 7, Fig. 9), and (3) to utilize diverse learned jumping skills to recover from external perturbations or impacts (Fig. 9), in the real world. We hope this paper can serve as a step forward for enabling more diverse, dynamic, and robust legged locomotion skills.

II. Related Work

Prior works tackling dynamic locomotion skills such as jumping with legged robots can be broadly categorized as corresponding to model-based optimal control (OC) and model-free reinforcement learning (RL). Table I compares our work with the most related prior efforts on the bipedal robot Cassie.

1) Model-based optimal control for legged jumping: Prior model-based methods for legged jumping control usually build up a layered optimization scheme, which includes offline trajectory optimization with detailed models of the robot's dynamics and ground contacts [9, 11, 44, 61], and online controllers that leverage simplified models of the robot's dynamics [43, 53, 63, 70]. In order to optimize trajectories for jumping, which needs to switch among modes with different underlying dynamics, there are two commonly employed solutions: relying on human-tuned predefined contact sequences [8, 18, 28, 45, 69], which is not scalable to different jump distances and/or directions, or leveraging contact-implicit optimization [7, 13, 32, 50, 75] which plans through contacts to avoid breaking the trajectory or using computationally expensive mixed-integer programming [1, 10, 11]. However, due to the computational challenges of optimization, both of the above-mentioned methods are still limited to offline computation for legged robots. As we show in this work, by training with different jumping skills offline, Cassie can automatically generate the appropriate contact sequences during online execution.

In the case of controllers for aperiodic dynamic jumps, many previous efforts require a separate landing controller to stabilize the robot from the large landing impacts [19, 24, 44, 69]. However, this approach requires estimating the contact events for legged robots, which on its own can be a challenging problem [21, 37, 49]. Furthermore, while there are a few prior attempts addressing *precise* jumping control on low-dimensional single-legged robots [64, 71] and a quadrupedal robot [43], mostly in simulation [43, 64], most prior work on bipedal jumping only focuses on the single vertical jump with the landing location not controlled [19, 63, 69, 70]. In this work, the proposed jumping controller demonstrates the capacity to control the landing pose of the bipedal robot without position feedback or explicit contact estimation.

2) Model-free RL for legged locomotion control: Recent years have seen exciting progress on using deep RL to learn locomotion controllers for quadrupedal robots [4, 15, 33, 39] and bipedal robots [6, 36, 52, 57, 68, 73] in the real world. Since it is challenging in general to learn a single policy with RL to perform various tasks [27], many prior works focus on learning a single-task policy [5, 41, 47, 72] for legged robots, such as just forward walking [15, 30, 67]. There have been efforts to obtain a multi-task policy, such as walking at different velocities using different gaits, conditioned only on variable commands [16, 17, 34, 52], which requires more extensive tuning due to the lack of a gait prior. Providing the robot with different reference motions for different skills can be helpful, but requires additional parameterization of the reference motions (*e.g.*, a gait library) [2, 23, 26, 36], policy distillation [68], or a motion prior [14, 48, 65]. There is also a line of research to explicitly provide contact sequences for legged robots [3, 40, 56, 58]. However, such methods are prescriptive and provide little opportunity for the robot to deviate from the contact plan, limiting the flexibility with which it can respond to perturbations. In our work, we show that a multi-task policy can enhance the robustness of a jumping policy by intelligently employing a variety of skills to react to perturbations.

3) Sim-to-Real Transfer for Legged Robots: To tackle sim-to-real transfer for RL-based methods, some works have sought to directly train policies directly in the real world [20, 60, 66], but most of the prior work, especially for dynamic skills, leverages a simulator to train the legged robot with extensive dynamics randomization [46] and then zero-shot transfer to the real world [15, 30, 33, 36, 58] or finetune with real-world data [26, 47, 59]. Since performing rollout on the hardware of human-scale bipedal robots is expensive, we use the zero-shot transfer method. In order to realize this, there are two widely-adopted techniques: (i) end-to-end training a policy by providing the robot with a proprioceptive short-term history [15, 23, 36] or long-term history [46, 56, 57], (ii) teacher-student training that first obtains a teacher policy with privileged information of the environment by RL, then uses this policy to supervise the training of a student policy that only has access of onboard-available observations [17, 25, 30, 33, 39, 73], which shows advantages over the end-to-end training method [30, 31, 33]. However, here we show that, for the dynamic control of bipedal robots, by training the robot in an end-to-end way with a newly-proposed policy structure, we can realize a better learning performance over the teacher-student method which separates the training process and requires more data.

III. Background and Preliminaries

In this section, we provide a brief introduction to our experimental platform, Cassie and the background of multi-task reinforcement learning.

A. Floating-base Model of Cassie

We use Cassie as the experimental platform in this work. Cassie (see Fig. 1) is a life-sized bipedal robot and is around 1.1 meter tall, with a weight of 31 Kg. It is a dynamic and underactuated system, with 5 actuated motors (abduction q_1, rotation q_2, thigh q_3, knee q_4, and toe q_7) and 2 passive joints (shin q_5 and tarsus q_6) connected by leaf springs on its left and right leg. We denote the motor positions as $\mathbf{q}_m = [q^{L/R}_{1,2,3,4,7}] \in \mathbb{R}^{10}$. The 6 degree of freedom (DoF) floating base (pelvis) may be represented with translational positions (sagittal q_x, lateral q_y, vertical q_z) and rotational positions (roll q_ψ, pitch q_θ, and yaw q_ϕ). In total, the robot has 20 DoFs $\mathbf{q} \in \mathbb{R}^{20}$. For more details about Cassie's configuration, we refer readers to [69, Fig. 2]. The observable joint positions on the Cassie are denoted as $\mathbf{q}^o = [q_{\psi,\theta,\phi}, \mathbf{q}_m, \dot{q}_{x,y,z}, \dot{\mathbf{q}}_m] \in \mathbb{R}^{26}$, which can be obtained from onboard joint encoders and IMUs, while the base linear velocity $\dot{q}_{x,y,z}$ can be estimated with an extended Kalman filter [68].

B. RL Background and Multi-Task Policy

We formulate the locomotion control problem as a Markov decision process (MDP). At each timestep t, the agent (*i.e.*, the robot) observes the environment state $\mathbf{s}_t$, and the policy π produces a distribution over the actions, $\pi(\mathbf{a}_t|\mathbf{s}_t)$, conditioned on the state. The agent then executes the action $\mathbf{a}_t$ sampled from the policy, interacts with the environment, makes an observation of the environment's new states $\mathbf{s}_{t+1}$, and receives a reward r_t. The objective of RL is to maximize the expected accumulative reward (return) the agent received over the course of an episode $\mathbb{E}[\Sigma_{t=0}^{T}\gamma^t r_t]$ where γ is a discount factor and T is the episode length. In order to obtain a policy that can perform multiple tasks, we provide a goal $\mathbf{c}$ which parameterizes the task. The policy $\pi(\mathbf{a}_t|\mathbf{s}_t, \mathbf{c})$ is then also conditioned on the given goal $\mathbf{c}$ to perform different tasks.

Task Parameterization: In our jumping task, the goal $\mathbf{c}$ specifies target commands for a desired jump $\mathbf{c} = [c_x, c_y, c_z, c_\phi]$, which consists of the target location $c_{x,y}$ on the horizontal plane, elevation c_z in the vertical direction, and turning direction c_ϕ after the agent lands, calculated based on the robot's pose before the jump, *i.e.*, in the local frame of robot's starting pose. Please note that the change in elevation c_z is defined as the change of the floor height, instead of the change of the robot's base height.

IV. Multi-Stage Training for Versatile Jumps

We now describe our multi-stage training framework for acquiring multi-task jumping policies. The training environment is developed in a simulation of Cassie using MuJoCo [12, 62].

A. Overview of the Multi-Stage Training Schematic

Our goal is to develop a locomotion control policy for jumping skills that can perform targeted jumps to different

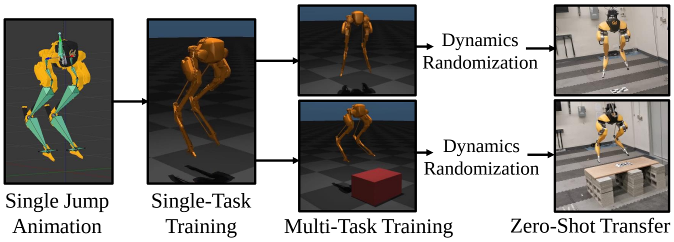


Fig. 2: The schematic to train the robot to perform versatile jumping skills in the real world starting with a single jumping animation. This framework consists of three stages. In the first stage, we focus on training the robot to imitate the animation while performing a single jump from scratch. After the robot is good at the single task, we randomize the task (to land at different locations and different turning directions/elevations) that is assigned to the robot during each training episode. After these two stages of training, we extensively randomize the dynamics properties of the environment in simulation in order to improve the robustness of the robot during the zero-shot transfer from sim to real.

locations. However, due to the challenging nature of jumping, it can be difficult to directly train a policy to perform a large variety of jumps from scratch. We observed that training policies from scratch to perform a large variety of jumps tends to lead to the robot adopting very conservative behaviors or even failing to learn to jump. Therefore, we use a multi-stage training scheme that consists of 3 stages, as illustrated in Fig. 2: (1) single-task training, (2) multi-task training, and (3) dynamics randomization. All stages of training are performed in simulation, but as we show in our experiments, the resulting models can then be directly deployed on a real Cassie robot. In *Stage 1*, the model is trained on a single task $\mathbf{c}$, *i.e.*, jumping in place. This stage of training results in a policy that is trained from scratch to specialize in a single task in simulation. Next, in *Stage 2*, the task $\mathbf{c}$ is randomized every episode to train the robot to jump to different targets. In this stage, the focus is primarily on performing the commanded task in the simulated environment. Finally, in *Stage 3*, we introduce extensive domain randomization on the simulation environment, while also randomizing the tasks, in order to improve the robustness and generalization of the policy for sim-to-real transfer. At each stage, the reward and episode designs of the MDP may vary in order to produce more effective policies for the objectives of the given stage.

In the rest of this section, we focus on explaining the details of Stages 2&3 in training, which share the same reward and episode design and cover both multi-task training and domain randomization. Stage 1 training, which is different in the choice of hyperparameters, is detailed in Appendix A.

B. Reference Motion

To initialize the training process, we provide a *single* jumping reference motion. The reference motion is a human-authored animation of Cassie jumping in place, created in a $3D$ creation suite [35], as presented in Fig. 2. This animated jump has an apex foot height of 0.5 m and an apex pelvis height of 1.1 m and has a timespan T_j of 1.66 second ($T_j = 1.66$).

TABLE II: A list of components of reward r_t which is a weighted summation of the listed items. The weight of each term is scheduled based on the jumping phase and training stage.

Reward Component r	**Weight w**			
	Stage 1		**Stage 2, 3**	
	$t \leq T_j$	$t > T_j$	$t \leq T_j$	$t > T_j$
Reference Motion Tracking				
Motion position: $r(\mathbf{q}_m, \mathbf{q}_m^r(t))$	15	15	7.5	15
Pelvis height: $r(q_z, q_z^r(t) + c_z)$	5	5	3	3
Foot height: $r(e_z, e_z^r(t) + c_z)$	10	10	10	10
Task Completion				
Pelvis position: $r(q_{x,y}, c_{x,y})$	12.5	12.5	15	15
Pelvis velocity: $r(\dot{q}_{x,y}, \dot{q}_{x,y}^d)$	0	3	12.5	12.5
Orientation: $r(q_{\psi,\theta,\phi}, [0, 0, c_\phi])$	12.5	12.5	10	10
Angular rate: $r(q_{\psi,\theta,\phi}, [0, 0, \dot{q}_\phi^d])$	3	3	10	10
Smoothing				
Ground Impact: $r(F_z, 0)$	5	0	10	0
Torque Consumption: $r(\boldsymbol{\tau}, 0)$	3	3	3	15
Motor velocity: $r(\dot{\mathbf{q}}_m, 0)$	0	15	0	25
Joint acceleration: $r(\ddot{\mathbf{q}}, 0)$	3	10	0	5
Change of action: $r(\mathbf{a}_t, \mathbf{a}_{t+1})$	0	0	10	10

This reference motion is only a kinematically-feasible motion for the agent and is not optimized to be dynamically feasible. After the end of the jumping animation, the reference motion will be set to a fixed standing pose for the robot.

C. Reward

The design of the reward function is important to encourage the robot to jump with agility. We further spilt the reward within a stage into two phases: before landing ($t \leq T_j$) and after ($t > T_j$), and the reward needs to vary based on these phases because the desired robot's behavior is different: performing aggressive jump versus stationary standing.

We here define a function:

$$r(u, v) = \exp(-\alpha ||u - v||_2^2) \tag{1}$$

where $r(u, v) \in (0, 1]$ defines a reward component that encourage the two vector u and v to be as close as possible, scaled by $\alpha > 0$ that balance the units. The reward r_t the agent receives at each timstep is a weighted summation of different components, $r_t = (\mathbf{w}/||\mathbf{w}||_1)^T \mathbf{r} \in [0, 1]$. The component vector $\mathbf{r}$ and weight vector $\mathbf{w}$ are detailed in Table II. The reward used here consists of three main components: reference motion tracking, task completion, and smoothing term.

The agent is encouraged to track the reference motor position by $r(\mathbf{q}_m, \mathbf{q}_m^r(t))$, pelvis height by $r(q_z, q_z^r(t) + e_z^d)$, and foot height $r(e_z, e_z^r(t) + e_z^d)$ at current timestep t. However, as recorded in Table II, the reference motion tracking term has a relatively small weight during the multi-task training because we want the agent to infer diverse skills, such as jumping to different locations, and the jumping-in-place reference motion may be no longer reasonable.

The task completion reward, on the contrary, is designed to dominate others during multi-task training. We first include the $r(q_{x,y}, c_{x,y})$ and $r(q_{\psi,\theta,\phi}, [0, 0, c_\phi])$ to encourage the agent to reach the desired location and orientation and stay there after it lands in order to accomplish the assigned task $\mathbf{c}$. Furthermore, pelvis linear velocity tracking $r(\dot{q}_{x,y}, \dot{q}_{x,y}^d)$ and angular rate tracking $r(\dot{q}_{\psi,\theta,\phi}, [0, 0, \dot{q}_\phi^d])$ are introduced to shape the sparse task reward, where $\dot{q}_{x,y}^d = c_{x,y}/T_j$ and $\dot{q}_\phi^d = c_\phi/T_j$. Moreover, although the task does not include the pelvis roll and pitch angle $q_{\psi,\theta}$, minimizing them to zero can help to stabilize the robot's pelvis.

We further introduce a smoothing term that is less important than task completion but overwhelms the motion tracking term. For example, we encourage the robot to produce less ground impact force F_z during its jump by $r(F_z, 0)$, to damp the body's oscillation after it lands by motor velocity reward $r(\dot{\mathbf{q}}_m, 0)$ and joint acceleration reward $r(\ddot{\mathbf{q}}, 0)$, and to be more energy efficient by $r(\boldsymbol{\tau}, 0)$. Moreover, the importance of having a stationary standing pose is highlighted by having a relatively large weight on torque consumption and motor velocity reward after the robot lands ($t > T_j$). This is because the introduction of dynamics randomization in Stage 3 will cause oscillation in body pose, by making the environment noisy, during standing. We also introduce an additional component in Stages 2&3, the change of action reward $r(\mathbf{a}_t, \mathbf{a}_{t+1})$, to further smooth the aggressive maneuver the robot may conduct to jump over a long distance.

D. Episode Design

Having a careful design of the reward may not be enough since it is challenging to encourage the agent to jump. The robot may keep failing to stabilize itself while learning to jump. Therefore, the robot may easily adopt very conservative but stable behaviors because it can quickly improve the return in this way. For example, the robot may just stand or just jump in place without completing the task, and can still have some suboptimal return. To prevent this, we note that a cautious design of the episode can also facilitate the training of dynamic jumping maneuvers.

In the stages for multiple tasks training, the maximum episode length is set to 2500 timestep which lasts 76 second. During such an episode, the robot is commanded to jump to a random target after a random time interval of standing, and these random values are uniformly sampled. The task is sampled from $c_x \sim U(-0.5, 1.5)$ m, $c_y \sim U(-1.0, 1.0)$ m, $c_z \sim U(-0.5, 0.5)$ m, and $c_\phi \sim U(-100°, 100°)$, and standing phase distribution is $U(1, 15)$ second. Such a "jump $\leftrightarrow$ stand" switch is repeated. Such a design can improve the robustness of the learned policy by performing repeat jumps over an episode. Moreover, compared to Stage 1 where the agent is asked to jump at $t = 0$, starting from Stage 2, there is a high probability the robot will start with a standing skill at each episode.

The episode will be terminated earlier if the robot falls over ($q_z < 0.55$ m or the tarsus joints hit the ground) to prevent it from having further rewards. We also emphasize the importance of foot height tracking and task completion to the robot by terminating the episode earlier if: (i) the foot height tracking error $|e_z - e_z^d|$ is larger than the bound E_e which is set at 0.32 m, or (ii) the robot does not arrive at the given target after it lands ($t > T_j$) when $[||q_{x,y} - q_{x,y}^d||_2, |q_\phi - q_\phi^d|] > E_t$ where $E_t = [0.35, 35°]$. Please note that we have a relatively small task completion error bound E_t while we have a large tolerance on the foot height tracking error E_e. Using such a design, the robot is allowed to deviate from the reference foot trajectory to find a better foot height trajectory for different tasks. The robot will also have more incentive to complete the task by landing close to the target and having more rewards in a longer episode.

Remark 1: The larger choice of the foot tracking error tolerance E_e also allows the robot to perform small hops after it lands. The robot is encouraged to stand by the foot height tracking reward but can dynamically switch to hop, including hopping to different places as long as staying within E_t, for better robustness. We do not specifically train or encourage the agent to deviate from the assigned task for robustness.

E. Dynamics Randomization

In order to succeed during the sim-to-real transfer, we introduce extensive randomization on dynamics parameters of the environment in Stage 3. The dynamics properties that are randomized are listed completely in Table III in Appendix B. During training at this stage, at each episode, the value of each dynamics parameter is uniformly sampled from the range listed in Table III. We consider three sources that cause the sim-to-real gap: (1) modeling errors, (2) sensor noise, and (3) communication delay between the high-level computer running the RL policy and the robot's low-level computer.

In order to robustify the policy to the modeling errors, we randomize the floor friction, robot's joint damping, link mass and inertia, and the position of the link's Center of Mass (CoM). Specifically, to deal with the error of motor dynamics between the simulation and hardware, we have a larger upper bound of the joint damping (4 times the default value) to approximate the motor aging issues on the hardware. We also randomize the PD gains used in the joint-level PD controllers (since our policy outputs target motor positions). The range is $\pm 30\%$ of the default value. Such a change is able to diversify the motor responses the robot is trained on and enhance the robustness to the change of motor dynamics during hardware deployment. Furthermore, specific to Cassie whose leg has leaf springs to connect the passive joints $q_{5,6}$, the parameters of the springs are important because they will have significant displacement during the taking-off and landing phases. Therefore, we introduce a 20% uncertainty on spring stiffness during training. We empirically found that the randomization of the motor dynamics and spring stiffness has a non-trivial effect to succeed during the sim-to-real for the bipedal jumping skills.

The sensor noise from joint encoders, IMU, and estimation error of the base linear velocity are simulated as a Gaussian noise whose mean is sampled in Table III at each episode.

V. Training Setup

We build up our control policy by optimizing reinforcement learning through the multi-stage training pipeline.

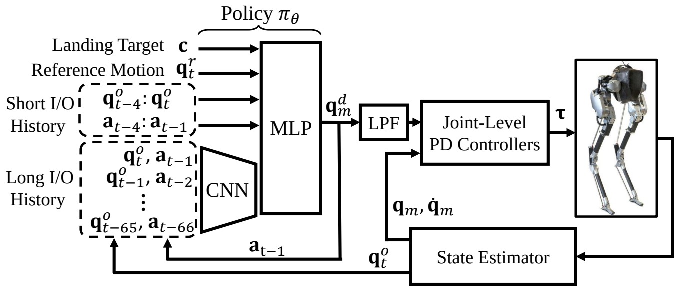


Fig. 3: The architecture of the multi-task jumping policy π_θ. The policy outputs the desired motor positions $\mathbf{q}_m^d$, which are used by joint-level PD controllers to generate the motor torques $\boldsymbol{\tau}$ on the robot. The input to the policy includes the task $\mathbf{c}$, which specifies the landing targets, the reference motion $\mathbf{q}_t^r$, which provides the robot a short preview of the reference trajectory, and a short 4-timestep history of the robot's input (robot's action $\mathbf{a}_{t-1}$) and output (robot's feedback $\mathbf{q}_t^o$). The policy is also provided with a long-term 2-second I/O history, which is first encoded by a 1D CNN. The policy updates at 33 Hz while the rest runs at 2 kHz.

A. *Policy Architecture*

Our policy π_θ is represented by a deep neural network with parameters θ. As shown in Fig. 3, it has two components, a base network represented by a multilayer perceptron (MLP), and a long-term history encoder represented by a 1D convolutional neural network (CNN). The policy operates at 33 Hz. Each action $\mathbf{a}_t$ specifies the target motor positions $\mathbf{q}_m^d$ for the robot. The action is first passed through a Low Pass Filter (LPF) [47], which smooths the motor targets before being applied to joint-level PD controllers. The PD controllers operates at 2 kHz, to generate motor torques $\boldsymbol{\tau} \in \mathbb{R}^{10}$ for driving the movements of the joints.

The input to the policy at timestep t contains four components: the task $\mathbf{c}$ introduced in Sec. III-B, a preview of the reference trajectory $\mathbf{q}_t^r$, a short-term history of previous actions and states (robot's Input/Output), and a long-term I/O history of the last 2-second. The preview of the reference trajectory $\mathbf{q}_t^r = [q_z^r(t), \mathbf{q}_m^r(t+1), \mathbf{q}_m^r(t+4), \mathbf{q}_m^r(t+7)]$ provided in the robot's observation contains the current reference pelvis height $q_z^r(t)$ and reference motor positions $\mathbf{q}_m^r$ sampled at 1, 4, and 7 future timesteps. To close the control loop, we provide the robot direct access to a short-term I/O history of the robot $(\mathbf{q}_{t-4:t}^o, \mathbf{a}_{t-4:t-1})$ in the previous 4 timesteps (about 0.12 second). The I/O history enables the policy to infer the dynamics of the system from past observations. The task $\mathbf{c}$, reference motion $\mathbf{q}_t^r$, and the short-term I/O history at current timestep t are directly passed as inputs to the base MLP.

For sim-to-real transfer, a short-term history may not be enough to provide adequate information to control dynamic maneuvers on a high-dimensional system. For example, during a jump, the landing event is affected by the angular momentum gained before the take-off, and the interval between these two events can be much longer than the 0.12 second. Therefore, we include an additional input in the form of a long-term I/O history of the past 2 seconds, which contains 66 timesteps of past observations and actions measurements $(\mathbf{q}_{t-65:t}^o, \mathbf{a}_{t-66:t-1})$. The timespan of this long I/O history is designed to cover the duration of a jump to help the policy implicitly infer the robot's dynamics, traveled trajectory, and contacts. To encode this long sequence of observations, we use a 1D CNN to compress it into a latent representation before providing it as an input to the base MLP. As we will see in Fig. 5, both the long-term and short-term history is needed for better learning performance.

In this work, the CNN encoder consists of 2 hidden layers whose [kernel size, filter size, stride size] are $[6, 32, 3]$ and $[4, 16, 2]$ with *relu* activation and no padding, respectively. The result of the CNN is flattened and concatenated into the inputs of the base MLP. The MLP has two hidden layers with 512 *tanh* units, followed by an output layer representing the mean of a Gaussian action distribution with a fixed standard deviation of $0.1I$.

B. *Training Details*

Empirically, we found that performing both turning and jumping to different elevations is very difficult for Cassie, which does not have a torso. Due to this hardware limitation, we choose to train two separate multi-task policies: a *flat-ground policy* that is specialized for jumping without elevation changes, *i.e.*, $e_z^d = 0$, and a *discrete-terrain policy* that is trained to jump onto platforms with different elevations without turning ($q_\phi^d = 0$).

Proximal Policy Optimization (PPO) [54] is used to train all policies π_θ in simulation, with a value function represented by a 2-layered MLP, which has an access to the ground truth observations. Due to the differences in the complexity of the different training stages, the three stages are trained with 6k, 12k, and 20k iterations, respectively. Each iteration collects a batch size of 65536 samples.

VI. Simulation Validation

Having introduced our methodology for training multi-task jumping policies, we will next validate the proposed method in simulation (MuJoCo). In this section, we aim to address two questions: (1) what are the advantages of the proposed policy architecture compared to models used in prior work, (2) whether training with multiple tasks can further improve the robustness of the policy over single-task training, by allowing the robot to utilize more diverse skills to recover from unstable states or unknown perturbations.

A. *Baselines*

To answer the first question, we benchmark our proposed policy architecture with several baselines illustrated in Fig. 4. All policies are trained with multiple tasks, *i.e.*, jumping to different landing locations and turning directions with no change of elevation, using the training schematic shown in Fig. 2, and are trained with 3 different random seeds. The details of baseline models are described in Appendix C.

To address the second question, we obtained two single-task policies using the proposed policy structure, as detailed below:

- **Single Task**: a policy that is trained on a single jumping-in-place task and extensive dynamics randomization as listed in Table III.
- **Single Task w/ Perturbation**: a policy similar to the single-task policy but is also trained with a randomized perturbation wrench (6

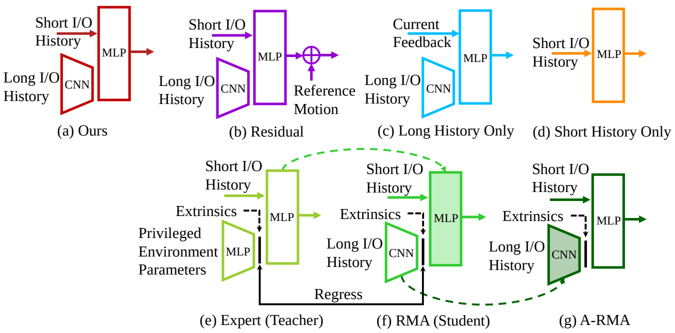


Fig. 4: Illustration of the baseline policy structures used to train the multi-task policy for bipedal jumping. (a) Ours: proposed structure as discussed in detail in Fig. 3. (b) Residual policy that has the same input structure as our method but outputs a residual term adding to the reference motor position [33, 68]. (c) Long History Only policy that only has the access to a long-term I/O history (we still provide robot immediate feedback to the base, as suggested by Peng et al. [46]). (d) Short History Only policy that is only provided with a short-term I/O history [36]. We also compare with the RMA [30]/Teacher-Student [33] training strategy where an (e) expert policy with access to privileged environment information (Table III) is first trained by RL and is later utilized to train (f) RMA (student) policy by supervised learning. The RMA can be further finetuned using by (g) A-RMA [31] by RL. The blocks are shaded if their parameters are not updated. The dash lines indicate that parameters are copied.

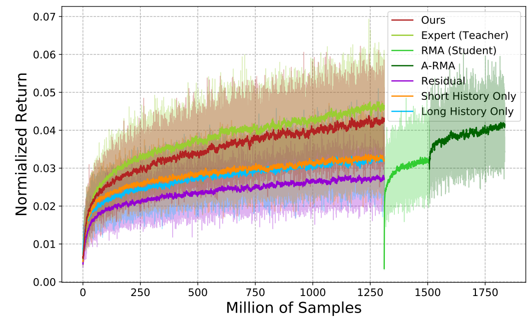


Fig. 5: Benchmark of learning curves trained by different policy structures in Stage 3 (multi-task with dynamics randomization). The curves are the average normalized returns trained with 3 random seeds while the colored areas enclose the min and max values obtained among different seeds. The normalized return is calculated by the return divided by the max episode length and in the range of $[0, 1]$. Our method shows similar performance as the expert policy which is used to supervise RMAs and has access to the privileged environment parameters. The A-RMA shows the second-best performance but it requires significantly more samples compared to the proposed methods, followed by RMA. The policies with short history only or long history only show a similar learning performance but are a bit worse than RMA in terms of returns. The residual policy shows the worst performance because the reference motion added to the policy's action prevents the agent from exploring more diverse maneuvers.

DoF) applied on the robot pelvis. The external forces and torques are sampled uniformly from $[-20\text{N}, -5\text{Nm}]$ to $[20\text{N}, 5\text{Nm}]$ and are applied on the robot's pelvis for a random time interval ranging from $[0.1, 2.0]$ second.

We compare these baselines based on two metrics: 1) learning performance in Sec. VI-B and 2) the ability to generalize to dynamics parameters that lie outside of the training distributions in Sec. VI-C. These two metrics are important for the sim-to-real transfer because the first one shows how well the policy can perform during training and the second evaluates robustness to changes in the environment, which are not considered during training, as can be the case during sim-to-real transfer.

B. Policy Structure Choice

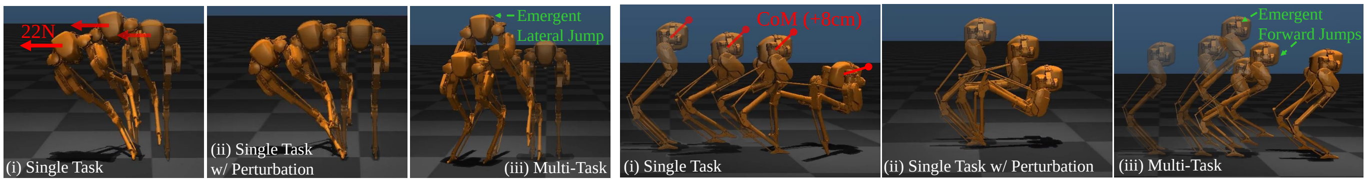
The learning curves from Stage 3 (multi-task learning with dynamics randomization) using our policy structure and baselines are presented in Fig. 5. The learning curves at early training stages (learning a single task in Stage 1 and multiple tasks in Stage 2) are available in Fig. 10 in Appendix D. The same hyperparameters and reward functions are used for every training stage.

According to Fig. 5 (and Fig. 10), the residual structure drawn as the purple curve shows the worst learning performance over all the training stages. The reason is the reference motion we provided is a dynamically-infeasible animation, which may cause the robot to spend more effort learning to correct these default motions, and prevents it from exploring more diverse trajectories and inferring the motion that is outside of the range of the reference motion.

The baselines using short history only (orange curve) and long history only (blue curve) show a similar learning performance. But if we combine these two by providing the policy with a long history encoder and direct access to short history, which results in our method, the learning performance can be enhanced to a large extent, as drawn as the red curves in Fig. 5. This showcases that, providing the policy with a long history is not enough because the robot may need immediate feedback which may be hidden from the long-history encoder. Providing the policy with direct access to the short history can address it and the agent can learn to utilize both information.

Remark 2: We note that there is other work using RNNs with LSTM, to encode the long-term I/O history [46, 56, 57]. We hypothesize that providing the policy direct access to the *short history* is not limited to the 1D CNN encoder but also to other neural network structures that capture temporal information such as LSTM, GRU, and Transformer. We choose 1D CNN in this work because it is easier to train.

The comparison between our method and RMA/Teacher-Student policies (green curves) is also interesting. During the training of the multi-task policy with dynamics randomization, our method only shows a little degradation compared to the expert policy. This actually showcases the advantages of our method because it can be zero-shot transferred to the real world while the expert policy that requires privileged information cannot. After training the expert policy, RMA and A-RMA have trained with 3k iterations and 5k iterations respectively, as shown in Fig. 5. We found that RMA has a large degradation compared to the expert policy due to the regression loss, and A-RMA is necessary to finetune the base policy in order to futher improve the return. RMA shows a better return than the policy with only short history or only long history, which is aligned with the finding from previous



(a) With Consistent Unknown Lateral Perturbation Force

(b) With Errors in Center of Mass Positions of All Links

Fig. 6: Robustness comparison among three policies which are: (i) trained with a single task (jumping in place) with dynamics randomization, (ii) trained with a single task with dynamics randomization and random perturbation, and (iii) trained with multiple tasks with dynamics randomization but without random perturbation (proposed). The testing scenarios are outside the training setting for all three policies. The single-task policies fail to stabilize the robot, even the one trained with extensive perturbations. The multi-task policy which is trained with diverse jumping tasks but without perturbation succeeds to stabilize the robot by exploiting the learned skills. The multi-task policy is able to deviate from the commands (jumping in place) and utilize a lateral jump to stay robust to the lateral external force and two forward jumps to adapt to the forward CoM offset.

work [30, 33]. However, even after A-RMA converged, the return is a bit worse than our method, while RMA and A-RMA require additional training and significantly more samples.

Remark 3: The long-term I/O history encoder used in RMA and A-RMA is the same as the one used in the proposed method, which shows a better learning performance than the original encoder proposed in [30, 31] as shown in Fig. 11b.

Summary of the Result: By the ablation study above, we can summarize three factors that can improve the learning performance in our case for dynamic locomotion control: (1) using desired motion positions as the action space (in contrast to the residual), (2) providing the policy with direct access to the short-term I/O history in addition to a long-term robot's I/O history, and (3) training the policy in an end-to-end way instead of separating the training process into teacher and student. This combination leads to our proposed method.

C. Advantages of the Multi-task Policy

In order to validate the advantages brought by multi-task training, we further compare our multi-task policy with the single-task policies. These two single-task policies are trained with the same amount of samples with dynamics randomization as the proposed one, whose learning curves are recorded in Fig. 11a. During the test in simulation, we command the robot to perform an in-place jump in an environment that the robot has not been trained on. As presented in Fig. 6, we conducted two tests where 1) a consistent lateral perturbation force is applied on the robot pelvis, and 2) the CoM of *all* links are set to be $+8$ cm off to the default position in all dimensions, while other dynamics parameters are set to the default values.

During these two tests, both of the single-task policies fail to control the robot, while the multi-task policy succeeded to stabilize the robot and perform a jump. Specifically, the policies trained with a single task directly fail during standing, even in the case where one is trained with extensive external perturbations that "force" the robot to explore more maneuvers by perturbing it from a nominal jump. On the contrary, the policy trained with multiple tasks, such as jumping forwards and lateral, without perturbations during training, is able to generalize the learned skills, exploit them to stabilize the robot, and pick the best jumping maneuver even if it is not commanded. For example, while being commanded to jump in place, the multi-task policy utilizes a lateral jump that it learned to stabilize the robot with the presence of lateral force (Fig. 6a(iii)) and two emergent forward jumps to adapt to the CoM errors in the forward direction (Fig. 6b(iii)). Such a benchmark highlights the advantages of learning with multiple tasks which makes the policy more robust.

Having conducted an extensive ablation study in simulation, we show that the proposed policy structure and multi-task training significantly improve the robustness of the policy over other policy structures or single-task policies.

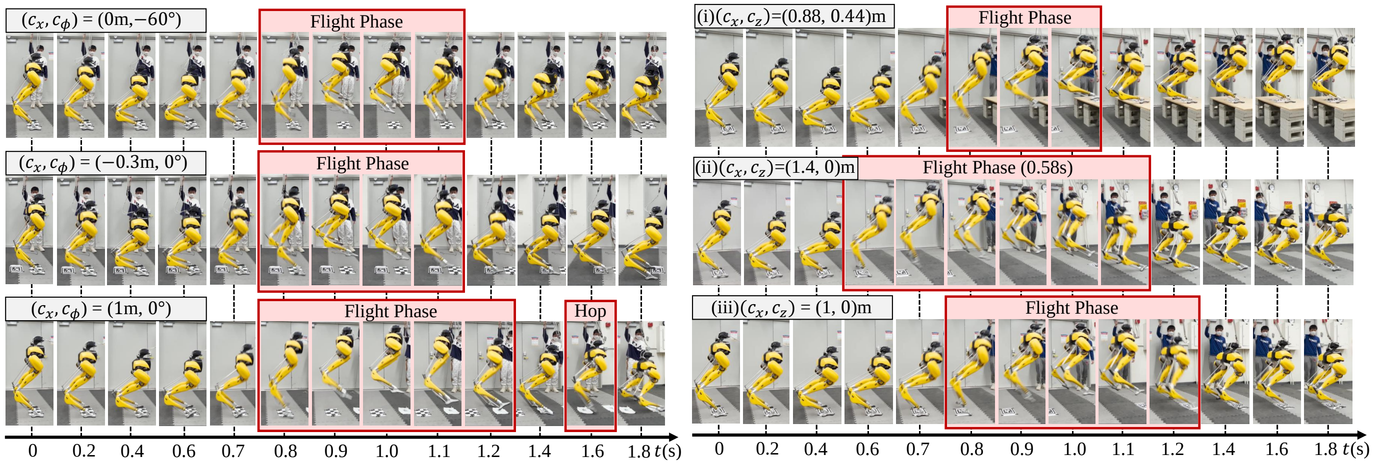
VII. Experiments

We now deploy the multi-task policies obtained in simulation, the *flat-ground policy* that is trained on different skills to jump to various locations and turning directions, and the *discrete-terrain policy* that is specialized in jumping to variable locations and elevations, on the hardware of Cassie. As shown in Fig. 1, both policies can successfully control the robot in the real world, without tuning.

Besides the ability to succeed in sim-to-real, in this section, we aim to validate two hypothesizes: 1) whether the policy trained in simulation can complete the same task in the real world, and 2) whether the multi-task policy is still able to exploit the learned skills to stabilize the robot after being transferred to the real world. The experiments can be best seen in the accompanying video (video_link). Please note that in all of the experiments, the robot does not have global position feedback, *i.e.*, once it starts to move, it does not know the distance to the landing target nor the distance to the ground.

A. Task Completion in the Real World

We first test the flat-ground policy on the robot in three distinct tasks: jumping in place while turning to negative 60 degrees, jumping 0.3 m backward, and jumping forward to land at a 1 m ahead target. As recorded in Fig. 7a, controlled by this multi-task policy, the robot is able to complete all three tasks. For example, during the turning task, the robot rotates to $-55°$ while jumping in the air, and lands at the same place where it took off (marked as a tag on the ground in Fig. 7a(i)). During the backward jump, different from the



(a) Different Jumps using the Flat-ground Policy

(b) Different Jumps using the Discrete-terrain Policy

Fig. 7: Snapshots of Cassie performing different jumps using the proposed multi-task policies. The snapshots are aligned with timestamps. The tags in the figures indicate the given landing targets. (a) Using a single policy that is specialized on flat ground, the robot is able to (i) jump in place while turning to -60°, (ii) jump 0.3 m backward, and (iii) jump 1 m forward, respectively. During the 1 m jump, the robot utilizes a forward hop to reach the goal after it lands on 0.5 m after the first jump. (b) The robot utilizes a single discrete-terrain policy to jump to different locations and elevations. The single policy can change the contact plan for different tasks. For example, the flight phase of the 1.4 m forward jump (ii) is the longest while being the shortest when the robot jumps onto the 0.44 m high elevation (i). The robot can land at the target (tag) with insignificant errors among all of these jumps without having global position feedback.

previous task, the robot leans backward before taking off (0.6-0.7 sec in Fig. 7a(ii)), and lands accurately at the target tag on the ground. To jump 1 m forward, the robot adopts a different maneuver where it leans forward before the flight phase and pushes itself off the ground with a larger strength, which results in a longer flight phase and travel distance than the previous two tasks. We also observe that the robot first lands at a 0.5 m landmark, but quickly conducts a forward hop when legs touch the ground (1.6 sec in Fig. 7a(iii)), and lands at the 1 m target tag in the last. We note that such a consecutive jumping maneuver does not happen during the same task in the simulation with the robot's nominal dynamics model. Such an experiment highlights two favorable features of the proposed policy: it can (1) adapt to different system dynamics (from sim to real) and (2) deviate from the reference motion and utilize multiple contacts to complete the given task (jumping to the target).

We then validate the discrete-terrain policy with three tasks: jumping 1 m ahead, 1.4 m ahead, and to a target that is 0.88 m ahead and 0.44 m above the ground, as presented in Fig. 7b. It shows the capacity to control the robot to jump over a distance/elevation to land on the given target accurately. We notice that the policy is able to adjust the robot's maneuvers/contact plan to jump over different distances/elevations. For example, compared to the 1 m jump (Fig. 7b(iii)), the robot takes off earlier while landing later during the 1.4 m jump (Fig. 7b (ii)). Such a change is reasonable as the robot needs a longer flight phase/larger take-off velocity in order to land at a farther location. Furthermore, when the robot is commanded to jump onto a 0.44 m table, the policy jumps more vertically (0.8 sec in Fig. 7b(i)) compared to the 1.4 m jump (0.6 sec in Fig. 7b(ii)) at the beginning of the flight phase while lifting the robot legs much higher in order to jump higher. Note that the robot makes contact with the platform much earlier than the other jumps on the ground, but this single policy is still able to stabilize the robot with different landing events. Furthermore, all these three experiments show that the robot controlled by the proposed policy can land accurately on the given target (land on the tags in Fig. 7b), which is a challenging task. Because the robot's motion is ballistic in the flight phase and a small error during taking-off may result in a large deviation from the landing target. In these experiments, the proposed policy is able to adapt to the dynamics of the robot hardware, adjust the robot's pose during the take-off, and accelerate to a velocity that can land the robot on the target.

Remark 4: During the long jump (like Fig. 7b(ii)), the robot leans its body forward at a large angle when it is pushing off from the ground and swings the legs forward during descending, and rotates its body forward w.r.t. contact points after it lands. Such a maneuver is very close to what we observed when a human athlete performs a standing jump [42, Fig. 1A]. Similar to humans, our robot's long jump skill is also learned during training, which is very different from the jumping-in-place reference motion we provided.

B. Sim-to-Real Gap

In order to further understand the difficulty to succeed in robot jumping experiments in Fig. 7, we take a close look at the sim-to-real gap. We record the robot's joint position profiles during a jump in the simulation with the robot's nominal dynamics parameters and on the robot's hardware. The profiles for a turning task (-60°, Fig. 7a(i)) using the flat-ground policy is presented in Fig. 8. According to the recorded profiles, the robot's actual joint position has a large deviation between the simulation (blue curves) and the real world (red curves). For example, the maximum error on the tarsus joint

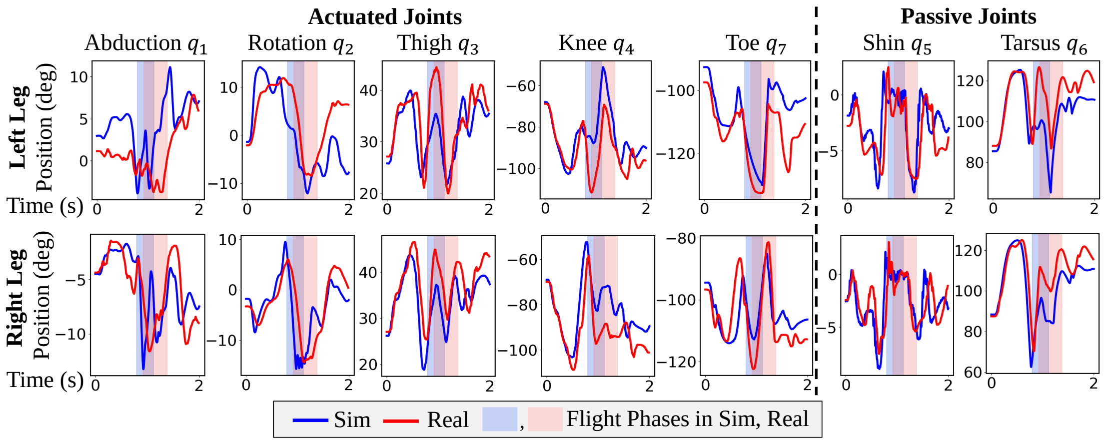


Fig. 8: The profiles of the robot's joint positions when it is commanded to jump and turn $-60°$ in place in simulation and the real world. We observe a large deviation of the joint profiles between sim and real, *e.g.*. the tarsus joints which are passive and driven by leaf springs show a significant difference during the sim-to-real transfer. Moreover, the flight phase in the real world is delayed compared with the one in the sim. Such errors highlight a big sim-to-real gap but our policy is robust to this and succeeds in controlling the robot to the given target.

position q_6 between sim and real is over 0.35 rad, which largely affects the robot's dynamics considering this joint is not actuated and is driven by a leaf spring whose nominal stiffness is 1250 Nm/rad. A similar deviation is observed in other joints, such as rotation joints q_2, thigh joints q_3 and knee joints q_4, which play a critical role during a jump and turning, and in other experiments using the discrete-terrain policy as recorded in Fig. 12. Such a discrepancy highlights the huge gap between the simulation and the real world but also showcases that, despite such a large gap, our methodology introduced in Sec. IV is able to stay robust and succeed in controlling the robot to accomplish the task.

C. Diverse and Robust Maneuvers by Multi-Task Policy

In order to push the limits of the proposed control policies, we further conduct more dynamic jumping experiments as presented in Fig. 9 and Fig. 13. As shown in Fig. 9a, using the single flat-ground policy, the robot performs a large repertoire of dynamic jumping maneuvers, such as jumping in place (Fig. 9a(i)), jumping to lateral (Fig. 9a(iii)), and multi-axes jumps such as blending lateral and forward jumps (Fig. 9a(iv)) and forward, lateral and turning (Fig. 9a(v)). In these multi-axes jumps, the robot demonstrates more complex maneuvers. For example, the robot leans in the lateral direction while jumping forward and turning to land on the target that is 0.5 m ahead, 0.2 m to robot's left, and turned $-45°$, as shown in Fig. 9a(v). During some challenging tasks, the robot is aware to utilize small hops to adjust its body pose after it lands with unstable states, such as demonstrated in Fig. 9a(iv)(v).

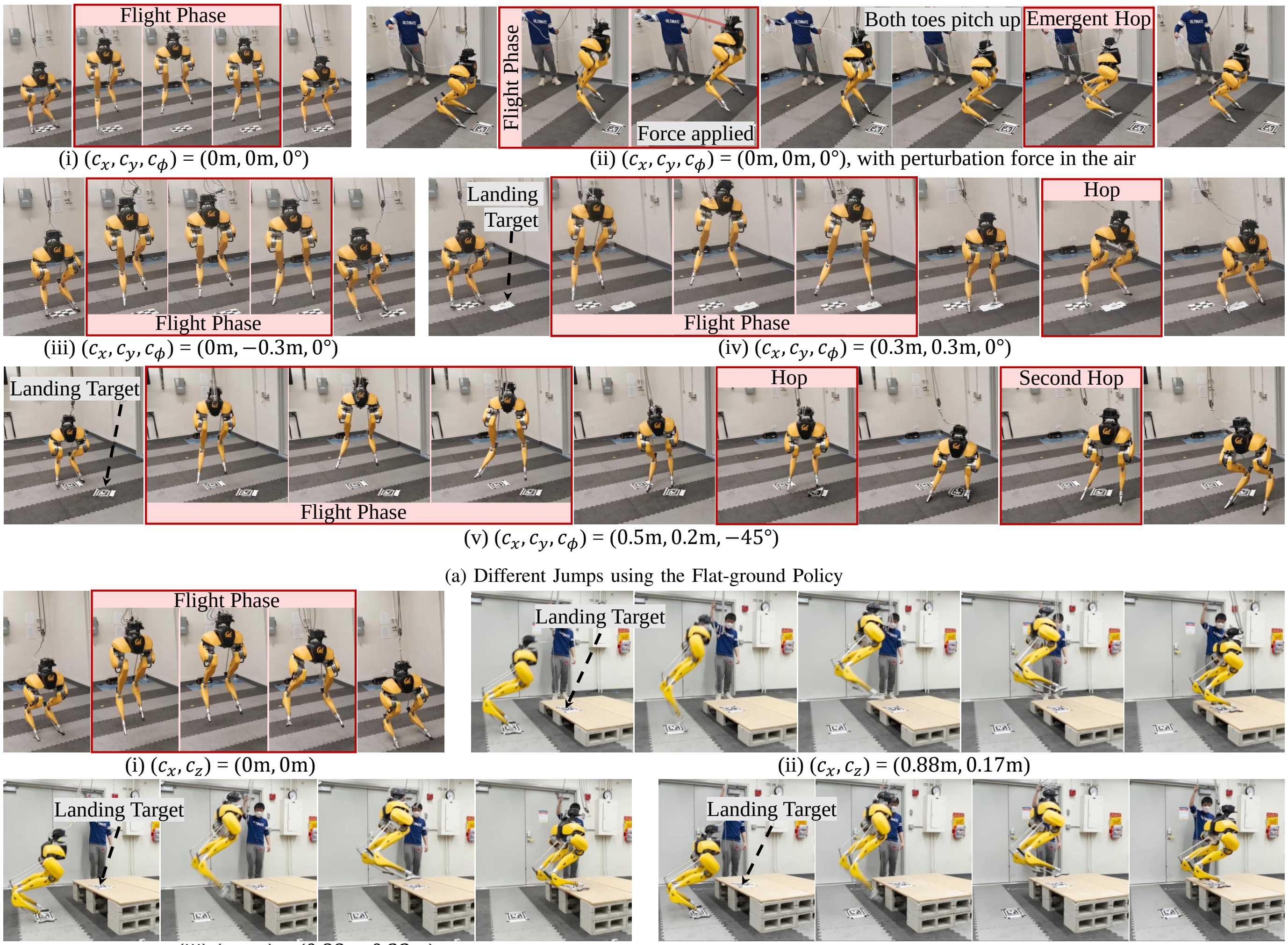
Moreover, in order to test the robustness of the policy, we applied a backward perturbation force on the robot's pelvis at its apex jumping height, as shown in Fig. 9a(ii). Due to such a perturbation, the robot leans backward during descending, and both of its toes pitch up after it lands, which makes the robot underactuated w.r.t contact points. However, the robot quickly exerts a backward hop, which is learned during the multi-task training, after it lands. By this hop, the robot can adjust its body pose during the flight phase and then land stably afterward. The task we gave to the robot in this test is to jump in place and it is interesting to see that the robot deviates from it in order to recover from falling over.

Remark 5: The robot, controlled by the proposed jumping policy, shows the ability to not rely on the pre-defined contact plan and can break the contact after it lands and make contact again when it needs to utilize impacts to stabilize itself. Such a capability is similar to contact implicit trajectory optimization [7, 13, 32, 50, 75]. While such optimization schemes still need to be computed offline for legged robots, our work achieves this online.

In the additional testing of the discrete-terrain policy demonstrated in Fig. 9b, the robot shows the ability to accurately land on different given targets. While the changes in the commanded distance and elevation are relatively small, the policy still demonstrates the ability to adjust the robot's take-off maneuvers in order to jump to the given targets.

VIII. Conclusion

In this work, we present an RL-based system for learning a large variety of highly-dynamic jumping maneuvers on real-world bipedal robots. In this work, we formulate the bipedal jumping problem as a parameterized set of tasks, and develop a task-conditioned policy that is trained in simulation but can then be deployed directly in the real world. In order to tackle the challenging multi-task learning problem, we utilized a multi-stage training scheme that divides the problem into three sub-problems and addresses each through different training stages. We showcase that by training with multiple tasks, the robot is able to generalize the learned skills to produce robust emergent recovery behaviors from large landing impact forces or unknown perturbations. The robustness acquired through multi-task training then also facilitates the sim-to-real transfer process, which can not be easily acquired through single-task training alone. Furthermore, we present a policy architecture that improves learning performance. Our framework enables a real Cassie robot to perform a suite of challenging jumping tasks, such as jumping to different locations, jumping onto



(i) (c_x, c_y, c_ϕ) = (0m, 0m, 0°)

(ii) (c_x, c_y, c_ϕ) = (0m, 0m, 0°), with perturbation force in the air

(iii) (c_x, c_y, c_ϕ) = (0m, −0.3m, 0°)

(iv) (c_x, c_y, c_ϕ) = (0.3m, 0.3m, 0°)

(v) (c_x, c_y, c_ϕ) = (0.5m, 0.2m, −45°)

(a) Different Jumps using the Flat-ground Policy

(i) (c_x, c_z) = (0m, 0m)

(ii) (c_x, c_z) = (0.88m, 0.17m)

(iii) (c_x, c_z) = (0.88m, 0.32m)

(iv) (c_x, c_z) = (0.64m, 0.32m)

(b) Different Jumps using the Discrete-terrain Policy

Fig. 9: Snapshots of various dynamic jumps performed by Cassie using the proposed policies. (a) The robot is able to perform a large repertoire of multi-axes jumps on flat ground. It shows the ability to stabilize the robot from a backward external perturbation (ii) by deviating from the commanded in-place jump and exploiting the learned backward jumping skills. The robot also leverages emergent hops after landing to stabilize it from a huge impact force, while being commanded to stand, like (iv) and (v). (b) Using a single discrete-terrain policy, the robot can not only jump in place (i), but also jump to different locations with different elevations (ii) (iii) (iv).

different evaluations, and blending multi-axis movements during a jump. A limitation we observe occasionally during some experiments is that the robot oscillates after a jump. This may be due to the challenges of having a single policy for both dynamic jumps and stationary standing. In the future, it will be interesting to combine this multi-task jumping policy with a more sophisticated perception system to traverse complex environments with greater mobility.

Acknowledgments

This work was supported in part by NSF Grant CMMI-1944722 and Canadian Institute for Advanced Research (CIFAR). The authors would like to thank Dr. A. Agrawal, X. Cheng, J. Chen, X. Huang, Y. Ni, L. Yang, and B. Zhang for their gracious help in the experiments.

References

[1] Bernardo Aceituno-Cabezas, Carlos Mastalli, Hongkai Dai, Michele Focchi, Andreea Radulescu, Darwin G Caldwell, José Cappelletto, Juan C Grieco, Gerardo Fernández-López, and Claudio Semini. Simultaneous contact, gait, and motion planning for robust multilegged locomotion via mixed-integer convex optimization. *IEEE Robotics and Automation Letters*, 3(3):2531–2538, 2017.

[2] Ryan Batke, Fangzhou Yu, Jeremy Dao, Jonathan Hurst, Ross L Hatton, Alan Fern, and Kevin Green. Optimizing bipedal maneuvers of single rigid-body models for reinforcement learning. In *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, pages 714–721. IEEE, 2022.

[3] Guillaume Bellegarda and Auke Ijspeert. Cpg-rl: Learn-

ing central pattern generators for quadruped locomotion. *IEEE Robotics and Automation Letters*, 7(4):12547–12554, 2022.

[4] Guillaume Bellegarda, Yiyu Chen, Zhuochen Liu, and Quan Nguyen. Robust high-speed running for quadruped robots via deep reinforcement learning. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10364–10370. IEEE, 2022.

[5] Miroslav Bogdanovic, Majid Khadiv, and Ludovic Righetti. Model-free reinforcement learning for robust locomotion using demonstrations from trajectory optimization. *Frontiers in Robotics and AI*, 9, 2022.

[6] Guillermo A Castillo, Bowen Weng, Wei Zhang, and Ayonga Hereid. Robust feedback motion policy design using reinforcement learning on a 3d digit bipedal robot. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5136–5143. IEEE, 2021.

[7] Iordanis Chatzinikolaidis, Yangwei You, and Zhibin Li. Contact-implicit trajectory optimization using an analytically solvable contact model for locomotion on variable ground. *IEEE Robotics and Automation Letters*, 5(4): 6357–6364, 2020.

[8] Hua Chen, Bingheng Wang, Zejun Hong, Cong Shen, Patrick M Wensing, and Wei Zhang. Underactuated motion planning and control for jumping with wheeled-bipedal robots. *IEEE Robotics and Automation Letters*, 6(2):747–754, 2020.

[9] Hongkai Dai, Andrés Valenzuela, and Russ Tedrake. Whole-body motion planning with centroidal dynamics and full kinematics. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 295–302. IEEE, 2014.

[10] Yanran Ding, Chuanzheng Li, and Hae-Won Park. Single leg dynamic motion planning with mixed-integer convex optimization. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–6. IEEE, 2018.

[11] Yanran Ding, Chuanzheng Li, and Hae-Won Park. Kinodynamic motion planning for multi-legged robot jumping via mixed-integer convex program. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3998–4005. IEEE, 2020.

[12] OSU DRL. cassie-mujoco-sim, 2023. URL https://github.com/osudrl/cassie-mujoco-sim.

[13] Luke Drnach and Ye Zhao. Robust trajectory optimization over uncertain terrain with stochastic complementarity. *IEEE Robotics and Automation Letters*, 6(2):1168–1175, 2021.

[14] Alejandro Escontrela, Xue Bin Peng, Wenhao Yu, Tingnan Zhang, Atil Iscen, Ken Goldberg, and Pieter Abbeel. Adversarial motion priors make good substitutes for complex reward functions. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 25–32. IEEE, 2022.

[15] Gilbert Feng, Hongbo Zhang, Zhongyu Li, Xue Bin Peng, Bhuvan Basireddy, Linzhu Yue, Zhitao Song, Lizhi Yang, Yunhui Liu, Koushil Sreenath, et al. Genloco: Generalized locomotion controllers for quadrupedal robots. *arXiv preprint arXiv:2209.05309*, 2022.

[16] Zipeng Fu, Ashish Kumar, Jitendra Malik, and Deepak Pathak. Minimizing energy consumption leads to the emergence of gaits in legged robots. *arXiv preprint arXiv:2111.01674*, 2021.

[17] Zipeng Fu, Xuxin Cheng, and Deepak Pathak. Deep whole-body control: learning a unified policy for manipulation and locomotion. *arXiv preprint arXiv:2210.10044*, 2022.

[18] Scott Gilroy, Derek Lau, Lizhi Yang, Ed Izaguirre, Kristen Biermayer, Anxing Xiao, Mengti Sun, Ayush Agrawal, Jun Zeng, Zhongyu Li, et al. Autonomous navigation for quadrupedal robots with optimized jumping through constrained obstacles. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pages 2132–2139. IEEE, 2021.

[19] Dip Goswami and Prahlad Vadakkepat. Planar bipedal jumping gaits with stable landing. *IEEE Transactions on Robotics*, 25(5):1030–1046, 2009.

[20] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018.

[21] Ross Hartley, Maani Ghaffari, Ryan M Eustice, and Jessy W Grizzle. Contact-aided invariant extended kalman filtering for robot state estimation. *The International Journal of Robotics Research*, 39(4):402–430, 2020.

[22] Masato Hirose and Kenichi Ogawa. Honda humanoid robots development. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 365(1850):11–19, 2007.

[23] 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.

[24] Se Hwan Jeon, Sangbae Kim, and Donghyun Kim. Online optimal landing control of the mit mini cheetah. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 178–184. IEEE, 2022.

[25] Gwanghyeon Ji, Juhyeok Mun, Hyeongjun Kim, and Jemin Hwangbo. Concurrent training of a control policy and a state estimator for dynamic and robust legged locomotion. *IEEE Robotics and Automation Letters*, 7 (2):4630–4637, 2022.

[26] Yandong Ji, Zhongyu Li, Yinan Sun, Xue Bin Peng, Sergey Levine, Glen Berseth, and Koushil Sreenath. Hierarchical reinforcement learning for precise soccer shooting skills using a quadrupedal robot. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1479–1486. IEEE, 2022.

[27] Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar,

Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *arXiv preprint arXiv:2104.08212*, 2021.

[28] Benjamin Katz, Jared Di Carlo, and Sangbae Kim. Mini cheetah: A platform for pushing the limits of dynamic quadruped control. In *2019 international conference on robotics and automation (ICRA)*, pages 6295–6301. IEEE, 2019.

[29] Daniel E Koditschek and Martin Buehler. Analysis of a simplified hopping robot. *The International Journal of Robotics Research*, 10(6):587–605, 1991.

[30] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021.

[31] Ashish Kumar, Zhongyu Li, Jun Zeng, Deepak Pathak, Koushil Sreenath, and Jitendra Malik. Adapting rapid motor adaptation for bipedal robots. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1161–1168. IEEE, 2022.

[32] Benoit Landry, Joseph Lorenzetti, Zachary Manchester, and Marco Pavone. Bilevel optimization for planning through contact: A semidirect method. In *Robotics Research: The 19th International Symposium ISRR*, pages 789–804. Springer, 2022.

[33] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5 (47):eabc5986, 2020.

[34] Quanyi Li, Zhenghao Peng, Haibin Wu, Lan Feng, and Bolei Zhou. Human-ai shared control via policy dissection. *arXiv preprint arXiv:2206.00152*, 2022.

[35] Zhongyu Li, Christine Cummings, and Koushil Sreenath. Animated cassie: A dynamic relatable robotic character. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3739–3746. IEEE, 2020.

[36] Zhongyu Li, Xuxin Cheng, Xue Bin Peng, Pieter Abbeel, Sergey Levine, Glen Berseth, and Koushil Sreenath. Reinforcement learning for robust parameterized locomotion control of bipedal robots. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2811–2817. IEEE, 2021.

[37] Tzu-Yuan Lin, Ray Zhang, Justin Yu, and Maani Ghaffari. Legged robot state estimation using invariant kalman filtering and learned contact events. *arXiv preprint arXiv:2106.15713*, 2021.

[38] Zachary Manchester and Scott Kuindersma. Variational contact-implicit trajectory optimization. In *Robotics Research: The 18th International Symposium ISRR*, pages 985–1000. Springer, 2020.

[39] Gabriel Margolis, Ge Yang, Kartik Paigwar, Tao Chen, and Pulkit Agrawal. Rapid locomotion via reinforcement learning. In *Robotics: Science and Systems*, 2022.

[40] Gabriel B Margolis and Pulkit Agrawal. Walk these ways: Tuning robot control for generalization with multiplicity of behavior. *arXiv preprint arXiv:2212.03238*, 2022.

[41] Gabriel B Margolis, Tao Chen, Kartik Paigwar, Xiang Fu, Donghyun Kim, Sang bae Kim, and Pulkit Agrawal. Learning to jump from pixels. In *5th Annual Conference on Robot Learning*, 2021. URL https://openreview.net/forum?id=R4E8wTUtxdl.

[42] Mark P Moresi, Elizabeth J Bradshaw, David Greene, and Geraldine Naughton. The assessment of adolescent female athletes using standing and reactive long jumps. *Sports Biomechanics*, 10(02):73–84, 2011.

[43] Chuong Nguyen, Lingfan Bao, and Quan Nguyen. Continuous jumping for legged robots on stepping stones via trajectory optimization and model predictive control. *arXiv preprint arXiv:2204.01147*, 2022.

[44] Quan Nguyen, Matthew J Powell, Benjamin Katz, Jared Di Carlo, and Sangbae Kim. Optimized jumping on the mit cheetah 3 robot. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7448–7454. IEEE, 2019.

[45] Hae-Won Park, Patrick M Wensing, Sangbae Kim, et al. Online planning for autonomous running jumps over obstacles in high-speed quadrupeds. *Robotics: Science and System*, 2015.

[46] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018.

[47] 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.

[48] Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. Amp: Adversarial motion priors for stylized physics-based character control. *ACM Transactions on Graphics (TOG)*, 40(4):1–20, 2021.

[49] Samuel Pfrommer, Mathew Halm, and Michael Posa. Contactnets: Learning discontinuous contact dynamics with smooth, implicit representations. In *Conference on Robot Learning*, pages 2279–2291. PMLR, 2021.

[50] Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33(1):69–81, 2014.

[51] M. H. Raibert, M. A. Chepponis, and H. Benjamin Brown. Experiments in balance with a 3d one-legged hopping machine. *International Journal of Robotics Research*, 3(2):75 – 92, June 1984.

[52] Diego Rodriguez and Sven Behnke. Deepwalk: Omnidirectional bipedal gait by deep reinforcement learning. In *2021 IEEE international conference on robotics and automation (ICRA)*, pages 3033–3039. IEEE, 2021.

[53] Martin Rutschmann, Brian Satzinger, Marten Byl, and Katie Byl. Nonlinear model predictive control for rough-terrain robot hopping. In *2012 IEEE/RSJ International*

Conference on Intelligent Robots and Systems, pages 1859–1864. IEEE, 2012.

[54] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[55] André Seyfarth, Hartmut Geyer, and Hugh Herr. Swing-leg retraction: a simple control model for stable running. *Journal of Experimental Biology*, 206(15):2547–2555, 2003.

[56] Yecheng Shao, Yongbin Jin, Xianwei Liu, Weiyan He, Hongtao Wang, and Wei Yang. Learning free gait transition for quadruped robots via phase-guided controller. *IEEE Robotics and Automation Letters*, 7(2):1230–1237, 2021.

[57] Jonah Siekmann, Srikar Valluri, Jeremy Dao, Lorenzo Bermillo, Helei Duan, Alan Fern, and Jonathan Hurst. Learning memory-based control for human-scale bipedal locomotion. *arXiv preprint arXiv:2006.02402*, 2020.

[58] Jonah Siekmann, Yesh Godse, Alan Fern, and Jonathan Hurst. Sim-to-real learning of all common bipedal gaits via periodic reward composition. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7309–7315. IEEE, 2021.

[59] Laura Smith, J Chase Kew, Xue Bin Peng, Sehoon Ha, Jie Tan, and Sergey Levine. Legged robots that keep on learning: Fine-tuning locomotion policies in the real world. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 1593–1599. IEEE, 2022.

[60] Laura Smith, Ilya Kostrikov, and Sergey Levine. A walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning. *arXiv preprint arXiv:2208.07860*, 2022.

[61] Zhitao Song, Linzhu Yue, Guangli Sun, Yihu Ling, Hongshuo Wei, Linhai Gui, and Yun-Hui Liu. An optimal motion planning framework for quadruped jumping. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11366–11373. IEEE, 2022.

[62] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[63] Barkan Ugurlu, Jody A Saglia, Nikos G Tsagarakis, and Darwin G Caldwell. Hopping at the resonance frequency: A trajectory generation technique for bipedal robots with elastic joints. In *2012 IEEE International Conference on Robotics and Automation*, pages 1436–1443. IEEE, 2012.

[64] Ivo Vatavuk and Zdenko Kovačić. Precise jump planning using centroidal dynamics based bilevel optimization. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3026–3032. IEEE, 2021.

[65] Eric Vollenweider, Marko Bjelonic, Victor Klemm, Nikita Rudin, Joonho Lee, and Marco Hutter. Advanced skills through multiple adversarial motion priors in reinforcement learning. *arXiv preprint arXiv:2203.14912*, 2022.

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

[67] Zhaoming Xie, Glen Berseth, Patrick Clary, Jonathan Hurst, and Michiel van de Panne. Feedback control for cassie with deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1241–1246. IEEE, 2018.

[68] Zhaoming Xie, Patrick Clary, Jeremy Dao, Pedro Morais, Jonanthan Hurst, and Michiel Panne. Learning locomotion skills for cassie: Iterative design and sim-to-real. In *Conference on Robot Learning*, pages 317–329. PMLR, 2020.

[69] Xiaobin Xiong and Aaron D Ames. Bipedal hopping: Reduced-order model embedding via optimization-based control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3821–3828. IEEE, 2018.

[70] William Yang and Michael Posa. Impact invariant control with applications to bipedal locomotion. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5151–5158. IEEE, 2021.

[71] Justin K Yim and Ronald S Fearing. Precision jumping limits from flight-phase control in salto-1p. In *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 2229–2236. IEEE, 2018.

[72] Fangzhou Yu, Ryan Batke, Jeremy Dao, Jonathan Hurst, Kevin Green, and Alan Fern. Dynamic bipedal maneuvers through sim-to-real reinforcement learning. *arXiv preprint arXiv:2207.07835*, 2022.

[73] Wenhao Yu, Visak CV Kumar, Greg Turk, and C Karen Liu. Sim-to-real transfer for biped locomotion. In *2019 ieee/rsj international conference on intelligent robots and systems (iros)*, pages 3503–3510. IEEE, 2019.

[74] Chi Zhang, Wei Zou, Liping Ma, and Zhiqing Wang. Biologically inspired jumping robots: A comprehensive review. *Robotics and Autonomous Systems*, 124:103362, 2020.

[75] Yifan Zhu, Zherong Pan, and Kris Hauser. Contact-implicit trajectory optimization with learned deformable contacts using bilevel optimization. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9921–9927. IEEE, 2021.

APPENDIX

A. Training in Stage 1

1) Reward: The reward design in Stage 1 is presented in Table II. In the first stage to initiate the training for a single jumping skill, we incentivize the robot to imitate the jumping-in-place animation. Therefore, the tracking rewards for motor position and foot height have overwhelming weights over others in order to accomplish a jump and stand still afterward. We also include the task completion term, but since the task is fixed in this stage, *i.e.*, $\mathbf{c} = \mathbf{0}$, this term is more to encourage the robot to jump in place and stabilize its pelvis orientation. We also have a smoothing term as a small fraction of the reward at this stage and do not have the change of action reward to prevent the robot from adopting a stationary behavior at the early stage of training.

2) Episode Design: In the initial training stage, the episode length is designed to have 750 timesteps corresponding to 23 seconds. The agent is asked to jump at $t = 0$ and to stand untill the end. If we allow the robot to stand at the beginning of the episode, the robot may focus on learning the easy standing skill and fail to explore the jumping maneuver. Furthermore, note that a jumping phase usually is less than 2 second but we have a 23-second episode. This is because the robot may learn jumping well but overlook the standing skill if the episode is short, which may result in the robot adopting an undesirable maneuver, such as continue hopping after landing. Having such a long episode can give the robot more incentive to learn a robust and stable standing skill in order to have a better return over the episode. The early termination conditions in Stage 1 are different than the multi-task training stage, except for the falling-over condition. In this stage, the foot height tracking error bound E_e is smaller (0.22 m) while the task completion error bound E_t is much larger ($[1.0, 45°]$). This is because we want to push the robot to jump by lifting its feet at the initial stage of training and completing the task is not a big concern at this stage.

B. Details of Dynamics Randomization

The details of the dynamics parameters and randomization range used in this paper are listed in Table III. Note that the range of the noise is relatively small (such as $0.1°$ in joint position measurement and $0.5°$ in joint velocity) because we found that the onboard sensors on Cassie are reliable and therefore we use a smaller bound to reduce the training complexity. For the robot that has larger sensor noises, a larger bound of the noise during training is recommended.

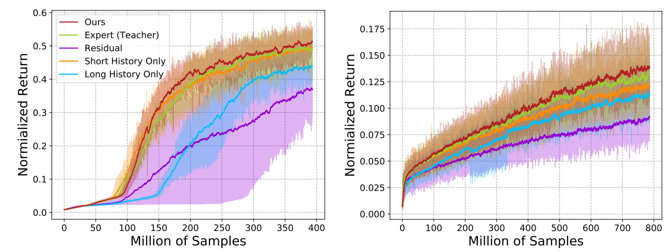
C. Details of Baseline Models

The details of the model structure we compared are listed as follow:

- **Ours** (Fig. 4a): the long-term I/O history is encoded with a CNN, while the short-term I/O history is provided directly as input to the base MLP. The policy direct outputs desired motion positions. The CNN encoder and the MLP base are jointly trained.
- **Residual** (Fig. 4b): the policy shares the same structure as the proposed one, but the policy output is a residual term added to the reference motor position at the current timestep, *i.e.*, $q_m^d = \mathbf{a}_t + q_m^r(t)$, which is used in [33, 68]. Please note that the policy has the reference motion as input.
- **Long History Only** (Fig. 4c): the policy only has a long-term I/O history encoded by a CNN, which is a baseline used in [30, 33]. Note that we still provide the robot feedback at the current timestep directly to the MLP base, as suggested by Peng et al. [46].
- **Short History Only** (Fig. 4d): the policy has short I/O history without the long-term I/O history CNN encoder, which is used in [36] and serves as a baseline in [31].
- **RMA/Teacher-Student**: an expert (teacher) policy (Fig. 4e) with access to privileged environment information (listed in Table III) is first trained using RL. The privileged information is encoded by an MLP into an 8D extrinsics vector. This expert policy is then used to *supervise* the training of an RMA (student) policy, which uses the base MLP copied from the expert policy, while using a long I/O history encoder to predict the teacher's extrinsic vector. This two-stage training scheme is used in [30, 33] and also adopted in other work such as [17, 25, 39].
- **A-RMA** (Fig. 4g): after the standard RMA training, the parameters of the long I/O history encoder are fixed, and the base MLP is further finetuned using RL as proposed by Kumar et al. [31]. Both RMA and A-RMA are also provided with a short I/O history.

TABLE III: Dynamics Randomization Range

Parameters	Range
Floor Friction Ratio	[0.3, 3.0]
Joint Damping	[0.3, 4.0] Nms/rad
Spring Stiffness	[0.8, 1.2] $\times$ default
Link Mass	[0.5, 1.5] $\times$ default
Link Inertia	[0.7, 1.3] $\times$ default
Pelvis (Root) CoM Position	[-0.1, 0.1] m in $q_{x,y,z}$
Other Link CoM Position	[-0.05, 0.05] m + default
Motor PD Gains	[0.7, 1.3] $\times$ default
Motor Position Noise Mean	[-0.002, 0.002] rad
Motor Velocity Noise Mean	[-0.01, 0.01] rad/s
Gyro Rotation Noise Mean	[-0.002, 0.002] rad
Linear Velocity Estimation Error	[-0.04, 0.04] m/s
Communication Delay	[0, 0.025] sec

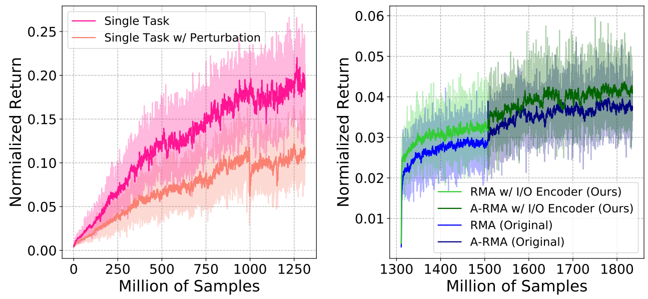


(a) Stage 1: Learning a Single Task (b) Stage 2: Learning Multiple Tasks

Fig. 10: Benchmark of learning curves trained by different policy structures trained with 3 random seeds in the early stages. The curves record the mean of normalized returns obtained using different seeds and the min and max among different seeds are the boundaries of the colored areas. The proposed method shows the best performance during the early stages of training including learning a single task from scratch (Stage 1) and multiple task in Stage 2.

D. Learning Performance in Early Stages

According to Fig. 5, at the training stages without dynamics randomization (Stage 1&2), our method shows similar, even a bit better, learning performance compared with the expert policy which has the access to the privileged environment



(a) Learning Single Task with Domain Randomization

(b) Learning with Different Memory Encoders for RMAs

Fig. 11: Additional Learning Curves.

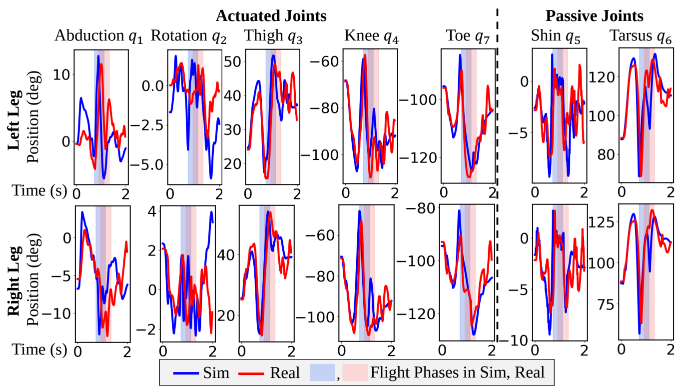


Fig. 12: The profiles of the robot's joint positions when it is commanded to jump to 0.44 m-tall elevation while forward 0.88 m in simulation and the real world, using the discrete-terrain policy.

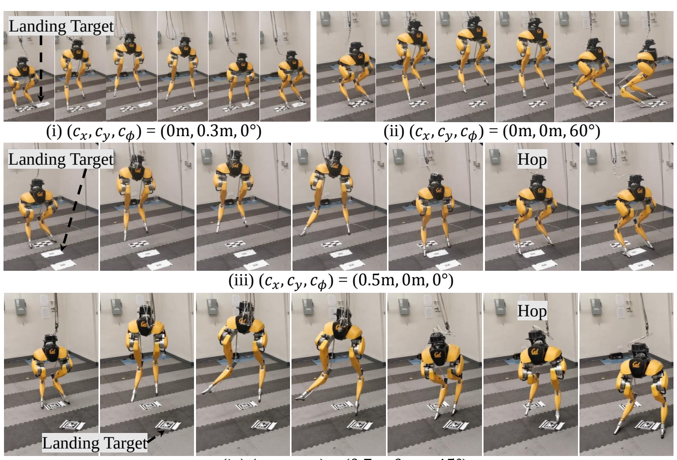


Fig. 13: Additional experiments shows Cassie jumping to different targets with the single flat-ground policy.

information. This is because the long-term I/O history is able to provide more information than the dynamics parameters used in the expert policy, such as the robot's take-off trajectory which will be useful to determine a better landing maneuver. Although the policy with short history only (orange curve) shows a faster learning curve at the initial stage of training (Stage 1, Fig. 10a), the learning performance using short history only and long history only (blue curve) show a similar learning performance in a more complex multi-task training stage (Stage 2).

E. Additional Learning Curves

The learning curves for single-task policies with dynamics randomization detailed in Sec. IV-E is recorded in Fig. 11a. Training RMAs with different long-history encoders are recorded in Fig. 11b. The RMA used in [30, 31] (Original) has a different structure of the long-term I/O encoder (1D CNN) than the one used in this work. It has 3 hidden layers and the [kernel size, filter size, stride size] of each layer is $[8, 32, 4]$, $[5, 32, 1]$, and $[5, 32, 1]$, with zero padding, respectively.

F. Additional Hardware Experiments

More experiment results are presented in Fig. 12 and Fig. 13. It shows the capacity of the flat-ground policy to accomplish more challenging jumping tasks on the real robot Cassie.