Abstract—Deep reinforcement learning (deep RL) has emerged as an effective tool for developing controllers for legged robots. However, a simple neural network representation is known for its poor extrapolation ability, making the learned behavior vulnerable to unseen perturbations or challenging terrains. Therefore, researchers have investigated a novel architecture, Policies Modulating Trajectory Generators (PMTG), which combines trajectory generators (TG) and feedback control signals to achieve more robust behaviors. In this work, we propose to extend the PMTG framework with a finite state machine PMTG by replacing simple TGs with asynchronous finite state machines (Async FSMs). This invention offers an explicit notion of contact events to the policy to negotiate unexpected perturbations. We demonstrated that the proposed architecture could achieve more robust behaviors in various scenarios, such as challenging terrains or external perturbations, on both simulated and real robots. The supplemental video can be found at: [http://youtu.be/XUTISZaM8f0](http://youtu.be/XUTISZaM8f0).

Index Terms—Finite State Machine, Reinforcement Learning, Quadrupedal Locomotion

I. INTRODUCTION

Locomotion has been one of the most challenging problems in the robotics domain. A controller must achieve its primary task of moving its body toward the desired direction while maintaining the balance with limited sensing and actuation capabilities. One popular approach is model-based control that leverages identified dynamics and control principles, demonstrating effective locomotion on quadrupedal [5], [16] and bipedal robots [17], [26]. However, model-based control often requires considerable manual effort to develop a proper model for each task. On the other hand, deep reinforcement learning (deep RL) has emerged as a promising approach to learn a robust policy from simple reward descriptions [4], [6], [22]. However, common feed-forward network architectures often make learned policies less effective for extrapolated tasks.

Researchers have attempted to develop a hybrid technique to take the best from both model-based and learning-based approaches. For instance, a few prior works proposed hierarchical controllers where a policy outputs high-level parameters to low-level model-based controllers that are typically implemented with the Model Predictive Control strategy (MPC) [10], [25], [27]. Although this approach is known to be sample-efficient and robust, its performance highly depends on the design of the low-level controller.

Policies Modulating Trajectory Generators (PMTG) [8] is a novel architecture that embeds prior knowledge into neural network policies. The predefined trajectory generator offers both memory and prior knowledge to the control policy. Researchers of the previous study have successfully trained robust walking policies using PMTG. However, the original design of trajectory generators employs a phase-based piecewise function controlled by a shared system clock, which is more vulnerable to external perturbations compared to asynchronous control strategies [24].

In this work, we propose a novel policy representation that is explicitly aware of contact events that are important to locomotion tasks. We propose a novel policy architecture, Finite State Machine Policies Modulating Trajectory Generator (FSM-PMTG), which extends the trajectory generators in the original PMTG. Our key idea is to combine trajectory generators with finite state machines (FSM), which run asynchronously across different legs. This simple extension of asynchronous FSM allows a robot to be aware of contact events explicitly and effectively adapt its behaviors to perturbed scenarios. We also propose a few implementation details of FSM that further boost the performance of the proposed architecture.

We evaluate the proposed FSM-PMTG on the locomotion task of a quadrupedal robot, A1, in both simulated and real-world environments. Our results suggest that our architecture shows much more robust behaviors than the original PMTG.
in various scenarios, such as rough terrains or external perturbations. We also demonstrate that FSM-PMTG is better at the sim-to-real transfer.

II. RELATED WORKS

It has been a very long time for researchers to use finite state machines (FSMs) to make robots walk. Mcghee et al. [13], [14], [24] are some of the earliest studies to prove that simple FSMs can accomplish the coordination of effective joint movements. Mcghee et al. [14] defined each leg of the quadrupedal as a two-state automata, and the quadrupedal locomotion gait as a matrix $G$. $G_{ij}$ stands for the state of $j$-th leg automata in $i$-th state of the gait. They also developed a hierarchical structure where the states of the hip and knee joints act as a sub-automata of the leg belonged to. Although Mcghee et al. [14] verified that both synchronous and asynchronous finite state machines are capable of producing connected gaits in either machines or animals, Tomovic et al. [23] pointed out that asynchronous mode of locomotion finite state machines is highly insensitive to disturbances because its state transitions are independent of shared clock frequencies, which makes the locomotion highly reliable. Since then, more complicated controllers based on finite state machines have been developed for various applications related to robotic locomotion. Park et al. [15] designed an asynchronous finite state machine to manage unexpected large ground-height variations in bipedal robot walking. And Lee et al. [9] showed that asynchronous finite state machines can also be used in a data-driven biped control to generate robust locomotion. Recently, Bledt et al. [1] showed that locomotion context like contact states of each leg can be used in finite state machines to help to generate stable locomotion and to assist gait switching. However, the design of effective FSMs requires a lot of time-consuming trials and errors based on the prior knowledge if the goal is to overcome challenging terrains or dynamics environments.

The recent advances of deep reinforcement learning enable a more convenient approach for developing control policies by leveraging simple reward descriptions. Various policy gradient methods, such as Deep Deterministic Policy Gradient (DDPG) [11], Trust Region Policy Optimization (TRPO) [20], and Proximal Policy Optimization (PPO) [21], have been widely adopted to train effective control policies, particularly in the context of robotic locomotion. As these methods usually learn policies in simulated environments, a lot of efforts to overcome the sim-to-real gap have been made, such as domain randomization [2], [22], [28], meta learning [3], [29], and compact observation space [22]. Recently, a few learning-based approaches including training an extra network to model actuator dynamics [6] or training a Cycle-GAN that maps the images from the simulator to corresponding realistic images [18] have also reached great performance in real robot experiments.

Although simple MLPs have been used as policy network in a large number of studies [2], [22], [28], PMTG [8] offers an effective approach to improve the performance of this simple policy architecture by taking an advantage of predefined trajectory generators (TGs). In this architecture, the policy can directly modulate the behavior of the predefined trajectory generators while generating additional feedback control signals. It has been shown that PMTG architecture can produce robust locomotion policies in both simulated and real-world environments.

In this paper, we propose a new policy architecture, FSM PMTG, by combining all three directions: FSMs, deep RL, and TGs. The prior knowledge is presented as the trajectory generator based on an asynchronous finite state machine, of which the state transition conditions are influenced by the robot proprioceptive sensory information. This asynchronous TG is modulated in the PMTG architecture to get a robust control policy using deep RL. We show that our novel policy architecture allows us to develop robust locomotion that can overcome challenging terrains.

III.FINITE STATE MACHINE PMTG

A. Background: The Original PMTG

In this work, our goal is to develop a finite state machine-policies modulating trajectory generator (FSM-PMTG) to obtain a more robust policy for challenging or dynamic environments. The original PMTG [8] is a policy representation that allows reinforcement learning (RL) to find more smooth behaviors with fewer samples. In short, PMTG divides policy outputs into the trajectory generator modulating parameters and the feedback terms ($u_{fb}$). Then the trajectory generators (TG) convert the parameters to the smooth control signals $u_{tg}$ and add them with the feedback terms. Therefore, PMTG allows a policy to explore smooth behaviors by leveraging TGs, while fine-tuning the behaviors with the feedback terms. Typically, PMTG is optimized with on-policy RL algorithms, such as PPO [21] or ARS [12]. For more details, please refer to the original paper.

B. Overview of FSM-PMTG

Our key idea is to further extend the trajectory generator (TG) of the original PMTG with an asynchronous FSM. The top-level architecture for FSM-PMTG is shown in Figure 2. Compared with the basic framework of PMTG [8], we made two adjustments in the FSM-PMTG:

1) Extend TG with asynchronous FSM.

![Fig. 2. Overview of FSM-PMTG: The output (actions) $u_{tg}$ of a predefined Trajectory Generator (TG) is combined with that of a learned policy network ($u_{fb}$). The learned policy also modulates the parameters of the TG at each time step and observes its state. Note that we have additional context variables from the robot to Async-FSM TG for state transitions, which are not the part of the observations.](image)
2) Add a data path to transfer locomotion contexts from the robot to the asynchronous FSM.

Note that locomotion contexts, such as contact flags or joint angles, are directly read from the robot sensors and does not need to be the part of the policy inputs.

We believe that two extensions, considering locomotion contexts and being aware of their asynchrony, are important prior knowledge that can dramatically improve locomotion performance. These two adjustments make a shortcut for TGs to perceive first-hand measurements from sensors on the robot without delay and add asynchrony to the whole controller.

C. Interfaces to FSM TG

In this section, we will describe the inputs and outputs to FSM-TG, which is a core module of the proposed architecture. In our system, the policy generates the policy modulating term (along with the feedback term $u_{fb}$), which consists of three TG parameters $(f_{tg}, A_{tg}, h_{tg})$ and passes them to FSM-TG. Then FSM-TG combines TG parameters and locomotion contexts to compute control signals $u_{tg}$. In addition, FSM-TG will return the TG status to the policy. Let us elaborate more details here.

TG Parameters: Our model adopts the idea from Iscen et al.’s work to use frequency $f_{tg}$, amplitude $A_{tg}$, and height $h_{tg}$ as parameters of our trajectory generator. In the original PMTG, the frequency $f_{tg}$ will influence the speed of joint rotation. However, because our asynchronous controller does not have a fixed cycle time, we use $f_{tg}$ to determine the ideal cycle step $S$ when the robot walks on the flat terrain without external perturbations. With control time interval as $dt$, we define the ideal cycle step $S$ as:

$$S = \left\lfloor \frac{1}{f_{tg} \cdot dt} \right\rfloor.$$  \hspace{1cm} (1)

The amplitude $A_{tg}$ and height $h_{tg}$ will decide the termination position tuple $\epsilon$, which does not require a well-tuned configuration.

Locomotion Context: Locomotion contexts are additional data from the robot to the trajectory generator, which helps the TG to decide when to transfer to the next state. We relay two types of locomotion contexts, joint positions and contact signals, to evaluate the state transition conditions.

TG Output: There are two parts of the output from the trajectory generator. The first is the target joint positions which will be adjusted by the feedback term $u_{fb}$. The second is the state of FSMs, such as the state index.

D. Async-FSM TG Design

In this section, we will describe the design of our asynchronous FSM-TG (Figure 3). We will first describe three possible states for each leg and then combine them into one automaton to describe the locomotion behavior.

Joint Sub-Automata: Let us assume that each leg has two pitch joint angles: $\alpha$ are for hip joints, and $\beta$ are for knee joints. We adopt a simple 3-state design (Figure 4) of the joint sub-automata according to Iscen et al. [14], which have the following states:

1) Lift up-forwards[$S_1$]: In this state, the leg moves from the most rear position $(\alpha_1, \beta_1)$ to the most front position $(\alpha_2, \beta_2)$ position. Put on the ground[$S_2$]: The raised leg will be moved to $(\alpha_3, \beta_3)$ to contact with the ground. Move backwards[$S_3$]: The leg will be moved backwards to the most rear position $(\alpha_1, \beta_1)$.

Therefore, we can parameterize the given states using a 6-tuple $\epsilon = (\alpha_1, \alpha_2, \alpha_3, \beta_1, \beta_2, \beta_3)$ to describe nominal poses of each state. Note that we will interpolate these poses to produce the final PD targets rather than directly using these poses.

Asynchronous FSM: As also suggested by Mcghee et al. [14], we can define a gait matrix $G$ with $k$ columns is introduced to describe the locomotion automation composed of $k$ legs, where $G_{ij}$ means in the gait state $i$ for the leg $j$. In the original paper, $G_{ij}$ can be either 0 (in the support phase) or 1 (in the transfer phase). In our case, each value corresponds to each leg’s state. We describe a trotting gait matrix using the joint sub-automata as follows:

$$G_{SA} = \begin{bmatrix}
S_1 & S_3 & S_3 & S_1 \\
S_2 & S_3 & S_3 & S_2 \\
S_3 & S_1 & S_1 & S_3 \\
S_3 & S_2 & S_2 & S_3
\end{bmatrix},$$  \hspace{1cm} (2)

where its columns refers to the front left, front right, rear left, and rear right leg. We can consider $S_1$ as the transfer
phase while $S_2$ and $S_3$ as the support phase. For any phase of this gait, two legs on the same diagonal move in the same way.

Other essential components of FSM are transition conditions. In our design, transition conditions typically consider either contact event or joint tracking status. It is worth noting that there exist two kinds of state transition lines in the figure. The solid line means a hard transition where the state transition happens only all transition conditions are satisfied. The dashed line means a soft transition where if one of the legs in the current state reaches its transition condition, its state will automatically transfer to the next state, instead of waiting until all other legs are ready. We believe this asynchronous design is essential for obtaining a robust locomotion policy, especially in challenging terrains and dynamic environments with unexpected random perturbations.

IV. IMPLEMENTATION DETAILS

A. Kinematic Controller with Desired Durations

In our design, we parameterize joint sub-automata with three target poses with six variables. When we want to control the robot, we do not want to use the given pose directly as PD target because it will cause unnecessarily abrupt movements. Instead, our goal is to apply exponential interpolation for achieving smoother motions. To this end, we designed the following kinematic controller. For a $k$-state finite state machine, the duration distribution $\tau$ is defined by a $k$-dim vector

$$\tau = [d_1, ..., d_k], \text{ where } \sum_{i=0}^{k} d_i = 1.$$  

and the termination position for each state is defined by the $k$-tuple of vectors $\epsilon = (e_1, ..., e_k)$.

Let us assume that the current state is $i$, the current time is $t$ steps after entering the state $i$, and the current position is $p(t)$, a simple exponential model is adopted to calculate the expected action $a_i$, which suggests the increment from the current joint angles to the next angles:

$$a(t) = K_i \cdot (e_i - p(t)), \quad (3)$$

where $K_i$ is a coefficient. We can solve $K_i$ to achieve the desired durations:

$$K_i = 1 - \frac{(S \cdot d_i + 0.5)}{\sqrt{|e_i - e_{i-1}|}} \quad (4)$$

where $S$ is the total steps in the ideal situation.

In our joint sub-automata, the duration distribution is set as a constant vector $\tau = [0.34, 0.16, 0.50]$, which is similar to the $1:2$ duration distribution for transfer phase and support phase mentioned in the work of Mcghee et al. [13].

B. Contact Correction

For the joint sub-automata mentioned above, $S_3$ requires making contacts during the entire state to produce sufficient friction forces. A robot can easily achieve this goal on flat ground without any obstacles, but it will be much harder if we deploy the robot on complicated and dynamic terrains. We try to mitigate this issue using our contact correction algorithm, which is designed to guarantee contacts during the state $S_3$.

The contact correction before entering $S_3$ helps the leg automata to decide when to transfer to the support phase. As shown in Figure 5, there are generally two basic cases that requires contact correction: Early Contact and Over Extension. These cases happen especially when the upcoming and the current grounds are not on the same flat plane. Therefore, we invent two simple rules to correct these behaviors. For Early Contact situation, our approach is an early transfer of the leg automata’s state to support phase. For Over Extension case, we keep attempting to extend legs with the cached action $\alpha(t - 1)$ until the legs make contact.

![Fig. 5. General situations that require contact correction. A: expected leg position (dashed blue) from current position (black) on ideal flat terrain. B: Before reaching the idea expected position, the leg contact the ground at an earlier step. Then the leg should stop moving down (dashed red) and transits to the S3. C: After reaching the idea expected position, the leg has not contacted with the ground. Then the leg should extend a little (dashed red).](image)

V. EXPERIMENTS

We conducted both simulated and real robot experiments to show that our FSM PMTG allows us to learn robust quadrupedal locomotion policies on challenging terrains. Particularly, we designed our experiments to verify the following research questions:

1) Can our FSM PMTG policy be transferred to difficult terrains without additional training?
2) Can our FSM PMTG show more robust behaviors than the original PMTG?
3) Can our FSM PMTG overcome a sim-to-real gap?

A. Simulated Experiments

Experimental Setup. We took the RaiSim [7] as our training environment. The training was done solely on flat terrain with a friction coefficient of 0.5. We also added random directional perturbation forces, up to $20N$ horizontally and $40N$ vertically, to the robot multiple times at random timing to obtain more stable behaviors. To fit the given specification of AlienGo and A1 Explorer robots from Unitree [19], we set the maximum velocity in the task $v_{max}$ as $0.6 \text{ m/s}$. Other experiment settings, like the duration of each episode and the testing velocity profile, were the same as those mentioned in [8].

We learned a policy with an actor-critic on-policy learning method, Proximal Policy Optimization. Both the actor and
critic networks were represented as simple 3-layer fully connected neural networks with 128 and 64 hidden nodes, respectively. The input dimension for all policies was 12, including z-axis of the robot frame in the world frame, the current linear velocity of the robot along the target direction, the current angular velocities of the robot (roll, pitch, yaw), the sub-automata’s current state of each leg, and the target velocity generated by the velocity controller. The output dimension was 11: 8 dimensional $u_{f0}$ for pitch joint angles and 3 dimensional ($f_{tg}, A_{tg}, h_{tg}$) for trajectory generators.

We trained three agents for comparison: 1) Original PMTG, 2) Synchronous FSM PMTG, and 3) Asynchronous FSM PMTG (ours). The synchronous FSM PMTG was designed to evaluate the importance of asynchronous control and contact awareness. We implemented it to generate the same trajectory as the Async FSM on the ideal flat terrain, but will not take the extra locomotion context. Because the original PMTG for locomotion was designed for the Minitaur robot, we designed a very similar version for the A1 robot.

![Graph showing learning curves](image)

**Fig. 6. Learning Curve:** The model is trained on flat terrain with randomized external forces. Our designs of FSM PMTG (both Sync and Async control) can reach a stable policy less than 1000 rollouts on the flat terrain with high reward obtained ($\geq 0.58/0.6$).

**Results.** First, we compared the learning curves of three agents, Async FSM PMTG (Ours), Sync FSM PMTG, and PMTG, in Figure 6. The results indicate that both FSM PMTG agents achieved much higher rewards than the original PMTG by closely following the given desired velocity profile. In this experiment, a synchronous version of FSM PMTG showed a slightly better performance than an asynchronous version. The reasons can be that the Sync FSM PMTG has a simpler structure to be well-tuned, and there are no external perturbations in this scenario. Then, we will evaluate their robustness against challenging and dynamic environments.

We evaluated the robustness of all the three agents by deploying them to more challenging and dynamic scenarios, which are unseen during training. We applied two types of randomization to the testing scenario: random terrain generation and random external perturbation. First, we pre-generated ten random terrains with the given maximum height, in our case, 30 cm. We also randomized their initial locations for generating a wider range of situations. We also set different friction coefficients $\mu$ and random control latency because they are common factors that will influence the robustness of the locomotion policies in practice, as suggested in the work of Tan et al. [22]. We also applied random perturbations, which are popular testing scenarios in legged robot research. In summary, we tested three types of scenarios: a flat terrain without perturbations (Flat/Stationary), a random terrain without perturbations (Random/Stationary), and a flat terrain with perturbations (Flat/Perturb).

Table I presented four evaluation criteria. $\bar{R}$ was the average reward where the maximum possible value $R_{max}$ is 0.6. $\bar{D}$ was the average travel distance where its target value $D_{target}$ was 16.9 m. $\bar{E}_v$ was the average velocity error and $S$ was the success rate among ten trials. The results showed that our Async FSM PMTG produced a more robust locomotion policy to cope with both challenging scenarios, Random/Stationary and Flat/Perturb. Particularly, the Async FSM PMTG showed significantly better robustness against external perturbations scenarios (Flat/Perturb) with 70 to 80 percent higher success rates. For more details, please refer to the generated motions in Figure 7 and the supplemental video.

For a more detailed analysis, we plotted one of the hip joint angle trajectories when an external perturbation causes the leg to get stuck. Figure 8 showed that both the Original PMTG and Sync FSM PMTG kept their stable action frequency as long as the target speed remained unchanged, even if the robot lost its balance in the dynamic environment. On the other hand, our Async FSM PMTG sensed the moment when the leg gets stuck from the extra locomotion context. To cope with this, the Async FSM PMTG slowed down the action frequency to keep balance and waited for the end of the accidental events.

| Scenarios                | PMTG Agent              | Measurements |
|-------------------------|-------------------------|--------------|
|                         | Original [8]            | $\bar{R}$ | $\bar{D}$ | $\bar{E}_v$ | $S$ |
| Flat/Stationary         | Sync FSM                | 0.563    | 16.4    | 0.0816    | 1.00 |
|                         | Async FSM               | 0.587    | 19.0    | 0.0599    | 1.00 |
|                         | Original                | 0.423    | 6.52    | 0.218     | 0.68 |
| Random/Stationary       | Sync FSM                | 0.467    | 13.5    | 0.127     | 0.85 |
|                         | Async FSM               | 0.482    | 13.4    | 0.120     | 0.95 |
| Flat/Perturb            | Original                | 0.292    | 8.27    | 0.183     | 0.20 |
|                         | Sync FSM                | 0.333    | 9.67    | 0.193     | 0.10 |
|                         | Async FSM               | 0.361    | 14.3    | 0.117     | 0.90 |

**B. Real Robot Experiments**

We also deployed the policy learned in the simulated environments to the real A1 robot to test whether our approach is robust enough to handle the sim-to-real gap. The main challenges for the sim-to-real transfer are control latency and observation noise. To deal with the challenges, we import domain randomization in the training process by randomizing some “missing” control steps, randomizing the position of the center of mass, and randomizing the external perturbation. In
Fig. 7. Generated motions for our model on Random/Stationary terrains. The top row (A) shows the case that the terrain blocks the path in the way of the robot. The bottom row (B) suggests that the robot can cope with the imbalance about the roll axis caused by the complex terrain. We use red dots to show contacts for the FL/RR legs and blue dots for the FR/RL legs.

Fig. 8. Front right leg hip joint angle ($\alpha$ / rad) control when front left leg gets stuck: Different from synchronous controller like Sync FSM PMTG and Original PMTG, action frequencies became much lower after entering the Leg Stuck period (the red region) for Async FSM PMTG.

Fig. 9. Result of Simulated and Real Robot Experiments. Time tickets at 0.0 s, 2.4 s, 4.8 s, 7.2 s, and 9.6 s are chosen for recording the robots’ positions.

shows the comparison between the real robot experiments and the corresponding simulated experiments. From the results, we observed that the original PMTG model failed to produce a successful forwarding policy. And our asynchronous model moved to the furthest, for asynchronous controller can be more robust against unpredictable mechanical latency.

Please note that this experiment does not disprove the sim-to-real transferability of the original PMTG. If we carefully tune DR parameters, it would be possible to obtain a reasonable PMTG policy that works on the hardware. However, we observed that our FSM-PMTG could better bridge the sim-to-real gap without careful domain randomization tuning.

VI. CONCLUSION

In this work, we propose an extended version of Policies Modulating Trajectory Generator, Finite State Machine PMTG, that takes advantage of the prior knowledge of locomotion contexts and the associated walking automata. Our key idea is to extend the trajectory generator with a finite state machine, as its name suggests. This simple invention allows us to learn a robust asynchronous policy by being aware of the desired event sequences. We evaluate the proposed method on various challenging locomotion tasks with randomized terrains and perturbations. We demonstrate that our method can outperform the original PMTG and the synchronous version of our FSM PMTG. We also succeed in the sim-to-real transfer from the RaiSim environment to the A1 robot.

Based on our current results, we plan to extend the current FSM design to learn robust quadrupedal locomotion. In this work, our Async FSM trajectory generator is based on simple ideas proposed by Mcghee [14]. In future work, we are looking forwards to importing advanced design of asynchronous control models to our architecture as more powerful trajectory generators, for asynchronous control models have been widely used in relative domains like robot imaging, robot formation, and robot navigation. We believe this extension will give a policy the ability to sense a broad range of events more explicitly, eventually leading to better robust behaviors.

ACKNOWLEDGEMENTS

We would like to thank Visak C.V. Kumar for his help during the real world deployment.
REFERENCES

[1] Gerardo Bledt, Patrick M Wensing, Sam Ingersoll, and Sangbae Kim. Contact model fusion for event-driven locomotion in unstructured terrains. In 2018 IEEE International Conference on Robotics and Automation (ICRA), pages 4399–4406. IEEE, 2018.

[2] Yevgen Chebotar, Ankur Handa, Viktor Makovickych, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience, 2019.

[3] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks, 2017.

[4] Tuomas Haarnoja, Jie Tan, Tingnan Zhang, Erwin Coumans, Yoonsang Lee, Sungeun Kim, and Jehee Lee. Data-driven biped locomotion. arXiv preprint arXiv:1812.11103, 2018.

[5] Marco Hutter, Hannes Sommer, Christian Gehring, Mark Hoepflinger, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Quadrupedal locomotion using hierarchical operational space control. The International Journal of Robotics Research, 33(8):1047–1062, 2014.

[6] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Atil Iscen, Ken Caluwaerts, Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Policies modulating trajectory generators, 2019.

[7] Yoonsang Lee, Sungeun Kim, and Jehee Lee. Data-driven biped control. In ACM SIGGRAPH 2010 papers, pages 1–8, 2010.

[8] Atil Iscen, Ken Caluwaerts, Jie Tan, Tingnan Zhang, Erwin Coumans, Vikas Sindhwani, and Vincent Vanhoucke. Deepgait: Planning and control of quadrupedal gaits using hierarchical operational space control. Science Robotics, 4(26):eaau5872, Jan 2019.

[9] Jemin Hwangbo, Joonho Lee, and Marco Hutter. Per-contact iteration method for solving contact dynamics. IEEE Robotics and Automation Letters, 3(2):895–902, 2018.

[10] Chao Li, Xin Min, Shouqian Sun, Wenqian Lin, and Zhichuan Tang. Deepgait: A learning deep convolutional representation for view-invariant gait recognition using joint bayesian. Applied Sciences, 7(3):210, 2017.

[11] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[12] Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search for fast adaptation of deep networks, 2017.

[13] Robert B McGhee. Finite state control of quadruped locomotion. Simulation, 93(3):135–140, 1967.

[14] Robert B McGhee. Some finite state aspects of legged locomotion. Mathematical Biosciences, 2(1-2):67–84, 1968.

[15] Hae-Won Park, Alireza Ramezanli, and Jessy W Grizzle. A finite-state machine for accommodating unexpected large ground-height variations in bipedal robot walking. IEEE Transactions on Robotics, 29(2):331–345, 2012.

[16] Hae-Won Park, Patrick M Wensing, and Sangbae Kim. High-speed bounding with the mit cheetah 2: Control design and experiments. The International Journal of Robotics Research, 36(2):167–192, 2017.

[17] Alireza Ramezanli, Jonathan W Hurst, Kaveh Akbari Hamed, and Jessy W Grizzle. Performance analysis and feedback control of atrias, a three-dimensional bipedal robot. Journal of Dynamic Systems, Measurement, and Control, 136(2):021012, 2014.

[18] Zhaoming Xie, Xingye Da, Michiel van de Panne, Buck Babich, and Mohi Khansari. Rl-cyclegan: Reinforcement learning aware simulation-to-real, 2020.

[19] Unitree Robotics. Unitree official website: https://www.unitree.com/.

[20] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017.

[21] Marco Hutter, Hannes Sommer, Christian Gehring, Mark Hoepflinger, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. Science Robotics, 4(26):eaau5872, Jan 2019.

[22] 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, 2018.

[23] R Tomović, R Anastasijević, J Vučo, and D Tepavac. The study of locomotion by finite state models. Biological cybernetics, 63(4):271–276, 1990.

[24] R. Tomović and R.B. McGhee. A finite state approach to the synthesis of bioengineering control systems. IEEE Transactions on Human Factors in Electronics, HFE-7(2):65–69, 1966.

[25] Vassilios Tsounis, Mitja Alge, Joonho Lee, Farbod Farshidian, and Marco Hutter. Deepgait: Planning and control of quadrupedal gait using deep reinforcement learning. IEEE Robotics and Automation Letters, 5(2):3699–3706, 2020.

[26] Eric R Westervelt, Jessy W Grizzle, Christine Chevallereau, Jun Ho Choi, and Benjamin Morris. Feedback control of dynamic bipedal robot locomotion. CRC press, 2018.

[27] 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:2104.09771, 2021.

[28] Zhaoming Xie, Xingye Da, Michiel van de Panne, Buck Babich, and Animesh Garg. Dynamics randomization revisited: A case study for quadrupedal locomotion. arXiv preprint arXiv:2011.02404, 2020.

[29] Wenhao Yu, Jie Tan, Yunfei Bai, Erwin Coumans, and Sehoon Ha. Learning fast adaptation with meta strategy optimization, 2020.