Eberhard Karls Universität Tübingen

Learning to Control Emulated Muscles in Real Robots: Analysis and Application with the Unitree GO2 Quadruped

Essay Rotation Submitted as M.Sc. Degree Requirements in

Graduate Training Center of Neuroscience

Prepared By

Jan Kerner

Supervised By

Pierre Schumacher
Prof. Georg Martius

Rotation Length: 30.09.2024 - 06.12.2024

Submission Date: 16.12.2024

Abstract

Locomotion in quadrupedal robots has made significant advances in recent years, yet adapting to unpredictable real-world conditions remains a challenge. Unlike conventional electrical motors, biological muscles offer unique nonlinear properties that enhance stability and adaptability. This project explores the integration of emulated muscle models with the Unitree Go2 robot, building on previous work with the Solo8 robot platform. To achieve this, we developed a pipeline for training and deploying reinforcement learning policies on the Go2 robot. As a first step we utilized a PD controller-based policy to establish a reliable workflow for uploading the policies to hardware. Muscle actuators were then adapted and fine-tuned in simulation using the Isaac Lab environment based on their muscle force-length and force-velocity relationships. Finally the performance of of the muscle actuators was evaluated in a position control task alongside torque-based actuators. The results demonstrated that torque controllers reliably track the command position, while muscle controllers exhibit promising tracking capabilities, but lack precision in certain cases.

Introduction

Quadrupedal robotic systems and their surrounding ecosystems have seen significant advancements in recent years, particularly in locomotive capabilities. This progress has made them more suitable for a broader range of applications, including human-assistive technologies. The growing availability of affordable platforms, such as Unitree Robotics, combined with continued development of (partially) open-source software frameworks, such as ROS or Isaac Lab, has made research more cost-efficient and continues to incentivize future innovation in this field [1] [2] [3].

Despite that, current robotic systems continue to face challenges in adapting to real-world scenarios that exceed the controlled conditions of their training. In contrast, animals can rapidly adapt to uncertainties within complex and changing environments. A comparison of the actuation mechanisms shows that robots usually use linear electrical motors, whereas animals utilize muscles with nonlinear properties. These intrinsic muscle properties provide stability and robustness under perturbations [4], while also facilitating more robust, less constrained and data efficient learning, compared to electrical motors [5] [6].

To demonstrate the properties of emulated muscles on real robotic systems, previous work conducted on the "Solo8" robot [7] showed that utilizing muscle controllers offers advantages in locomotion and hopping tasks, particularly in terms of stability and adaptability on unseen terrain [6]. As the "Solo8" robot platform features 8 degrees of freedom (DOF) for locomotion, comparisons with current state-of-the-art quadrupedal systems are not entirely accurate. Therefore, the aim of this project is to build on these findings, by migrating the muscle models to the Unitree Go2 robot, which utilizes 12 DOF for locomotion, aligning more closely with current quadrupedal systems. This process involves developing a pipeline for model training and hardware deployment on the new platform, enabling direct and fair comparison with other state-of-the-art quadrupedal systems.

The project focuses on addressing challenges that include adapting the muscle model to the new Go2 platform in both simulation and hardware, ensuring consistent performance across different environments. To achieve this, we first train a locomotion task using a conventional PD controller for the Go2 platform to establish a workflow for uploading trained policies to the robot. This step ensures policy functionality using known methods, avoiding the added complexity introduced by the muscle controller. Once the policies are successfully uploaded we investigate the muscle actuators in simulation by adapting the MuJoCo-based muscle from [6] to the Isaac Lab simulation environment. In the simulation we apply the muscle actuators to the robot, fine-tune their parameters and evaluate its functionality compared to torque controlled actuators in a position control task.

Material and Methods

Simulation Environment

We use Isaac Lab as our simulation environment, which is built on top of Isaac Sim [8]. Its modular and extensible design simplifies common workflows in robotics research, enabling the easy creation of task-specific reinforcement learning (RL) environments. The GPU-based physics engine also allows for highly efficient parallel simulations, allowing multiple environments to run concurrently, see Figure 1. Additionally, its ecosystem provides predefined environments, sensors and robotic formats for the Go2.

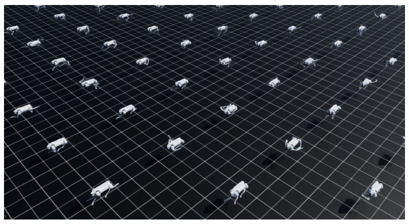

Figure 1: Screenshot of Isaac Lab showing multiple environments. This allows for simultaneous and fast training.

Robot Specifications

As robot platform, we are using the EDU version of the Go2 from Unitree Robotics, which is a quadruped robot with 12 DOF. Each joint can exert a maximum torque of 45Nm. Furthermore it offers an SDK for low-level hardware control (i.e. DC motors) that allows direct torque control (f.e. via ROS2). This is important for our task, in order to apply the muscle controller output directly to the robot. The onboard Jetson Orin NX enables autonomous policy execution, without the need of a physical connection to extern devices. The individual range of motion and the naming convention for every joint can be found in Figure 2.

Joint	**Min Angle** ϕ_{min}	**Max Angle** ϕ_{max}
Hip	-60°	60°
Calf	-180°	-45°
Thigh Front	-90°	220°
Thigh Back	-30°	290°

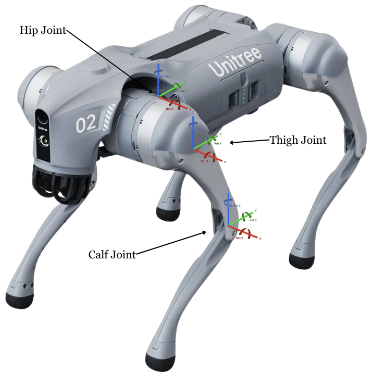

Figure 2: **Top**: Naming convention for every joint and corresponding minimal and maximal angle of rotation. **Bottom**: Positions and naming of joints. Here, the hip joint is rotating along the y-axis (green), the thigh joints are rotating along the x-axis (red), and the calf joints are rotating around the x-axis (red) [9].

Policy deployment

For deploying a policy, we use the walk-these-ways library [10]. It uses the Lightweight Communications and Marshaling (LCM) library to exchange information with the robot. LCM is a set of libraries for message passing targeted for real-time systems, providing low latency and high-bandwidth. This enables fast data transfer of the robot's observation data to the device running the policy network for inference, as well as fast transmission of action outputs from the policy network back to the robot.

Actuator Types

With the use of direct-drive electric motors in our robot, we can mimic the behavior and dynamics of muscles in real time. This is achieved by forwarding the torque outputs of a computational model of our muscle dynamics and applying them directly to the hardware.

To compare the behavior of our emulated muscles with an already known type of actuator, we train a policy with direct torque-controlled actuators in addition to our muscle controller. Their output is described as:

$$\tau_{applied} = clip(\tau_{computed}, -\tau_{max}, \tau_{max}) \tag{1}$$

where $\tau_{applied}$ is the torque applied to the joints, $\tau_{computed}$ the applied torque from our policy and τ_{max} the maximal possible joint torque.

For the muscle actuator we use the muscle controller defined in [6], which emulates the force-length and force-velocity characteristics of a muscle. The output torque can then be calculated as:

$$\tau_i = f_{max} \left[\sum_{k=1}^{2} (-1)^{k+1} FL(l_k) FV(\dot{l_k}) m_{act,k} + FP(l_k) \right] \tag{2}$$

where FL describes the force-length, FV the force-velocity relationship (see Figure 3), FP the passive force, l the muscle length and m_{act} the applied muscle activity.

Since we have an agonistic-antagonistic muscle configuration for every joint, the sum is taken over two muscles in opposing directions. τ_i then describes the torque at joint i. In our cases the passive forces are practically 0.

Muscle Parameter Tuning

As described in [6] and [12] the FV-relationship of the muscle primarily contributes to muscle damping. This property allows the tuning of our muscle actuators similarly to a PD controller, as described in [13]. This is achieved by setting $FL = 1$ and gradually changing the parameter $fvmax$ (see [6]), until stable movement with little oscillations is obtained. The parameters lce_{min} and lce_{max} were chosen to ensure that the generated muscle force falls within the linear region of the FL curve (see Figure 4).

Position Control Task

To test the trained policy for the torque and muscle actuators, we employ a position control task. In this task, the robot's body is fixed above the ground,

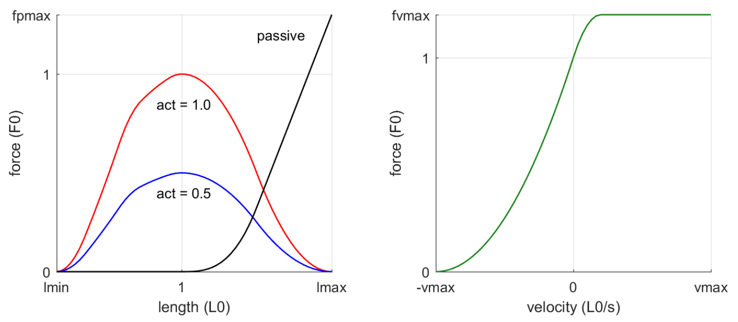

Figure 3: **Left**: Force-Length (red/blue) and Passive-Force (black) relationship curve for muscles with 2 different activation levels. Here L0 describes the length at which the muscle can exceed its maximum force (optimal muscle length). lmin and lmax describe the minimal and maximal muscle length **Right**: Force-Velocity relationship of muscles. [11]

while the joints have to reach a command position. The reward for the task is described as

$$r_{pos} = \omega_{pos} \cdot exp(-(a_{goal} - a_{curr})^2/\sigma) \tag{3}$$

where a_{goal} describes the target angle and a_{curr} the current angle of the joints, σ is the sensitivity factor and ω_{pos} the weighting coefficient. Additionally we also penalize high joint torques (for all 12 joints) by

$$r_\tau = -\omega_\tau \cdot \sum_{i=0}^{12} \tau_i^2 \tag{4}$$

where τ_i describes the torque applied at joint i and ω_τ the weighting coefficient and use action regularization

$$r_{act} = -\omega_{act} \cdot (a_{t-1} - a_t)^2 \tag{5}$$

where a_t describes the action at time step t and ω_{act} is the weighting coefficient The total reward is

$$r = r_{pos} + r_\tau + r_{act}. \tag{6}$$

We reset the task every 2 seconds with new joint positions.

Policy Training

For training our RL policy, we are using PPO [14], implemented by [15]. The multi layer perceptron(MLP) networks used by the actor and critic both contain 3 hidden layers with 512, 256 and 128 neuron and use ELU as activation function. The remaining hyperparameters can be found in Table 1

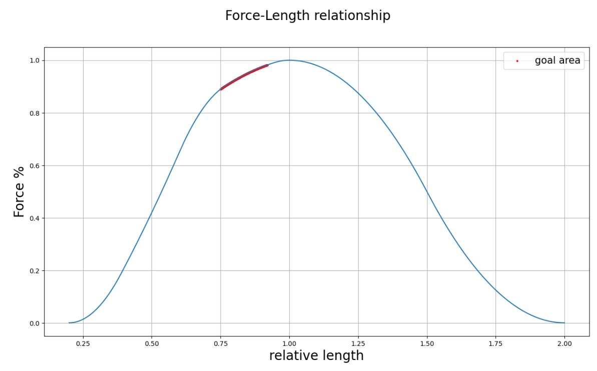

Figure 4: The FL curve is defined between l_{min} and l_{max}. We choose lce_{min} and lce_{max} such that the muscle length gets mapped to the part of the FV curve, which is linear (red).

Results

Policy Deployment

Before deploying the PD-controller policy to the robot, we analyzed its walking pattern. As shown in Figure 5, a regular walking pattern was obtained, closely resembling the walking pattern of the PD controller pattern described in [6]. After verifying this behavior, we successfully uploaded this locomotion policy to the robot. The robot was able to follow directional commands and showed stable locomotion on hardware.

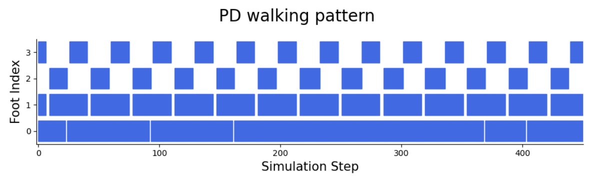

Figure 5: The policy for the PD controller was able to learn a regular walking pattern. Foot index 2 and 3 show a consistent and regular pattern, while foot indices 0 and 1 appear to lift less and slide closer to the ground during motion.

Parameter	**Value**
lr π	10^{-3}
ϵ	0.2
γ	0.99
λ	0.95
c_1	1.0
c_2	0.02
ω_τ	$-1.0 \cdot 10^{-2}$
ω_{act}	$-1.0 \cdot 10^{-1}$
ω_{pos}	2.0
σ	4

Table 1: Hyperparameters for PPO. Here ϵ is the PPO clipping parameter, γ the discount, λ the advantage estimation, c_1 the value loss coefficient and c_2 the entropy coefficient

Muscle Parameter Tuning

As seen in Figure 6 we increased the value for $fvmax$ stepwise by 0.1 until we found a value with little oscillation. Furthermore, we set the remaining parameters as shown in Table 2. The minimal and maximal joint angles were taken from Figure 2. After tuning the parameters and maintaining the natural dynamics of the FV and FL relationships, we get the motion dynamics, depicted in Figure 7. In addition we also put the f_{max} parameter to 20Nm, rather than the maximum of 45 Nm that the robot can generate, as this leads to a more stable behavior in training. The shown fine-tuning procedures are demonstrated on the rear right calf muscle, but are applied to the other joints in the same way.

Position Control Task

To investigate our results we compared the goal positions of the joints with the current position over time (see Figure 8). The command positions are updated every 2 seconds. The trained policy for the torque actuators sufficiently tracked the command positions of the environment.

The trained policy for the muscle actuator was not able to precisely follow the command positions in all joints. However, the muscle policy reacts to changes of the command position into the right direction with the hip and calf joints, but appears to overshoot until reaching the joint extrema. The thigh joints on the other hand were able to sufficiently follow the trajectory, while also maintaining intermediate angles and react to changes.

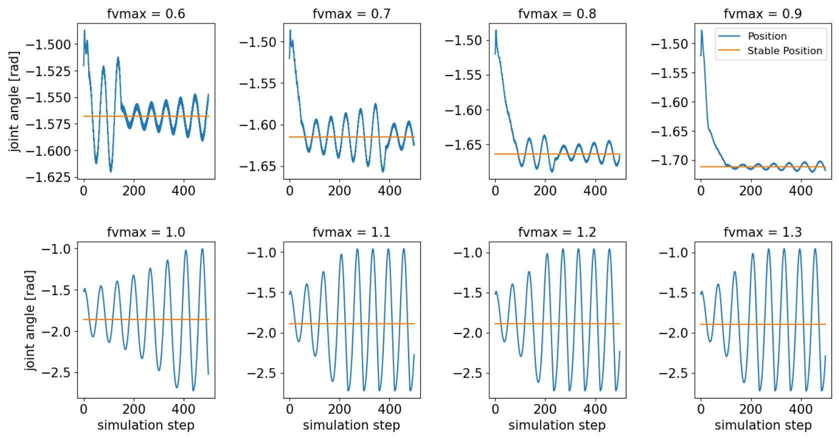

Figure 6: Fine tuning of the parameter $fvmax$. By increasing the parameter in steps of 0.1 we can see that the oscillationns for 0.9 are minimal. For values bigger or smaller than 0.9 the oscillations got bigger and made the position control less stable

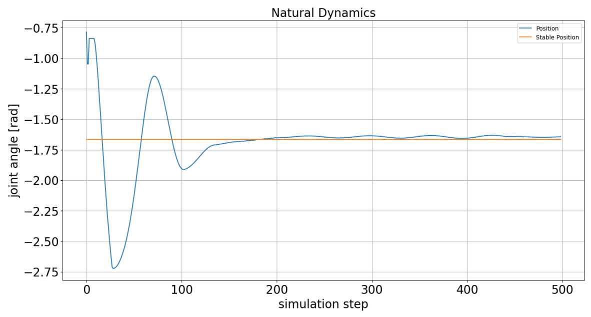

Figure 7: Muscle dynamics with free running force relationship functions and fixed parameters. A stable dynamics close to the stable position, without a lot of oscillations can be seen

Parameter	**Value**
l_{min}	0.2
l_{max}	2.0
fvmax	0.9
fpmax	2
lce_{min}	0.75
lce_{max}	0.92
f_{max}	20.0
ϕ_{min}	(*)
ϕ_{max}	(*)

Table 2: The muscle parameters after parameter tuning. (*): ϕ_{min} and ϕ_{max} are taken from Figure 2 for the respective muscle joints.

Discussion

During this project, we were converting the emulated muscle model controller from [6] to the Go2 robot platform and compared the performance of a learned policy for torque and muscle control in the joint position task in simulation. Our findings were that despite the fact that the muscle policy did not match the commanded positions perfectly for the calf and hip joints, promising tracing was done by the thigh joints. Therefore, further refinement of the muscle parameter could lead to more accurate results. Furthermore, we also successfully analysed and afterwards exported a locomotion policy to the Go2. With these findings and some more tuning on the muscle side, deploying policies on the Go2 seems appropriate.

Future Work

Parameter Tuning

As seen in our position control task, the muscle parameters, especially for the hip and calf joints, are not set sufficiently. The next steps would include optimization using more sophisticated methods for hyperparameter tuning such as random search or Bayesian optimization methods [16].

Task Extension

The only task on which the policy was trained was the position control task. To determine whether the policy network could learn how to control the muscles in more difficult environments and if it behaves similar in terms of stability to [6], a hopping and walking task need to be added. Furthermore no testing with external forces or perturbations were made to investigate the muscles stability.

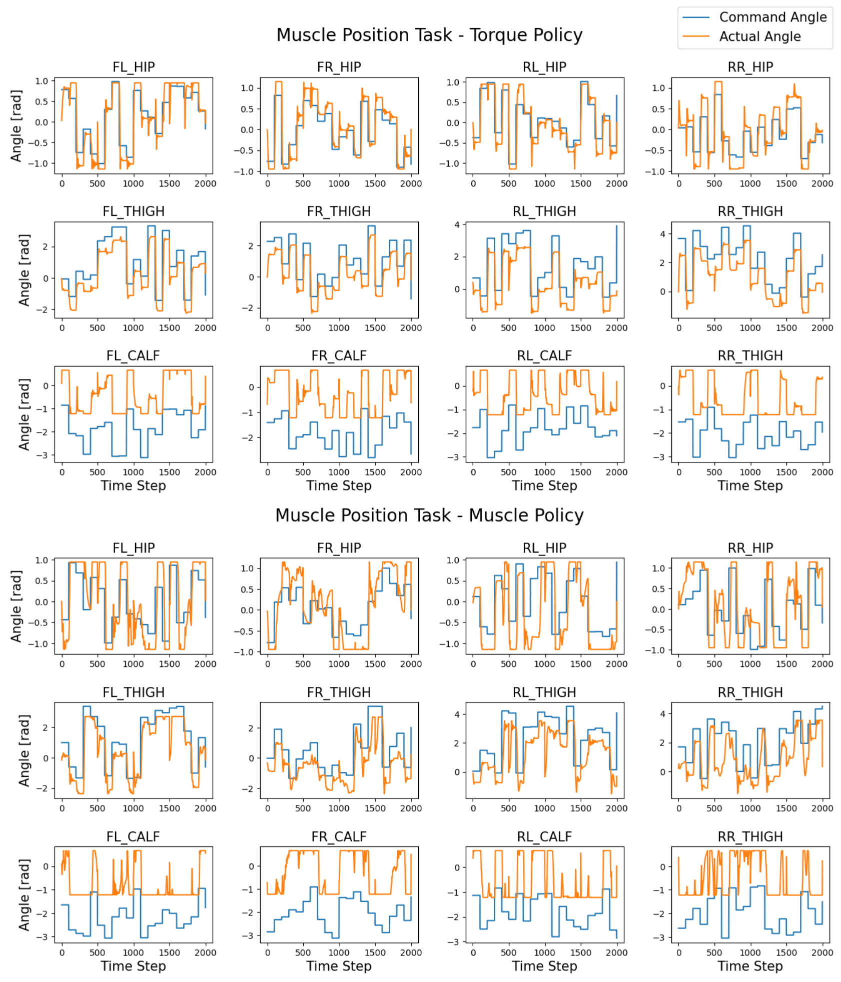

Figure 8: **Top**: Trajectory tracing of torque controller policy **Bottom**: Trajectory tracing of muscle controller policy. **Explanation**: The abbreviations correspond to the position of the joints. FL=Front Left, FR=Front Right, RL=Rear Left, RR=Rear Right.

Comparing these results would show the feasibility of converting the muscles to the Go2.

Policy Deployment

In this project, we use LCM for data transmission between the robot and the policy network hosting device. Because the Go2 robot natively supports ROS2, changing to ROS2 comes with benefits. Even if ROS2 is slower and less suitable for real-time systems, its data tranfer is more reliable. This is because the ROS2 data transfer is build on TCP, while LCM is using UDP. Taking into consideration, that the largest bottleneck of our control loop is the policy network inference itself, the data transfer time can be neglected.

Acknowledgements

I would like to express my deepest gratitude to my supervisors, Pierre Schumacher and Georg Martius, for their guidance, support, and encouragement throughout the course of this work. Their expertise and mentorship has always given me the little push in the right direction.

I am also grateful to Daniel Häufle for his insightful feedback and generous support on muscle modeling, which has greatly supported this project.

Additionally, I would like to thank the members of the Neuromech group, as well as the Autonomous Learning group in Tübingen, for their valuable input as well as the Graduate Training Centre for Neuroscience for providing me with the opportunity to conduct my research and studies in a supportive academic environment.

Finally, I am very thankful to everyone who has supported and encouraged me during this project.

References

[1] Telusma Mackenson Dwayne McDaniel Abderrachid Hamrani, Md Munim Rayhan and Leonel Lagos. Smart quadruped robotics: a systematic review of design, control, sensing and perception. *Advanced Robotics*, 0(0):1–27, 2024.

[2] NVIDIA. Fast track robot learning in simulation using nvidia isaac lab. https://developer.nvidia.com/blog/fast-track-robot-learning-in-simulation-using-nvidia-isaac-lab/, 2024. Accessed: 2024-12-28.

[3] Hamid Taheri and Nasser Mozayani. A study on quadruped mobile robots. *Mechanism and Machine Theory*, 190:105448, 2023.

[4] H. Wagner and R. Blickhan. Stabilizing function of skeletal muscles: an analytical investigation. *Journal of Theoretical Biology*, 199(2):163–179, 1999.

[5] Isabell Wochner, Pierre Schumacher, Georg Martius, Dieter Büchler, Syn Schmitt, and Daniel F. B. Haeufle. Learning with muscles: Benefits for data-efficiency and robustness in anthropomorphic tasks, 2023.

[6] Pierre Schumacher, Lorenz Krause, Jan Schneider, Dieter Büchler, Georg Martius, and Daniel Haeufle. Learning to control emulated muscles in real robots: Towards exploiting bio-inspired actuator morphology, 2024.

[7] Felix Grimminger, Avadesh Meduri, Majid Khadiv, Julian Viereck, Manuel Wüthrich, Maximilien Naveau, Vincent Berenz, Steve Heim, Felix Widmaier, Thomas Flayols, Jonathan Fiene, Alexander Badri-Spröwitz, and Ludovic Righetti. An open torque-controlled modular robot architecture for legged locomotion research. *IEEE Robotics and Automation Letters*, 5(2):3650–3657, April 2020.

[8] Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. Gpu-accelerated robotic simulation for distributed reinforcement learning. *CoRR*, abs/1810.05762, 2018.

[9] Unitree Robotics. Unitree robotics go2 robot image, 2024. Accessed: December 30, 2024.

[10] Teddy Liao. walk-these-ways-go2: Deploy walk-these-ways project on unitree go2. https://github.com/Teddy-Liao/walk-these-ways-go2, 2024. Accessed: 2024-12-29.

[11] DeepMind Technologies Limited. Muscles - mujoco documentation, 2024. Accessed: December 30, 2024.

[12] Fabio Izzi, An Mo, Syn Schmitt, Alexander Badri-Spröwitz, and Daniel FB Haeufle. Muscle prestimulation tunes velocity preflex in simulated perturbed hopping. *Scientific Reports*, 13(1):4559, 2023.

[13] Vishakha Vijay Patel. Ziegler-nichols tuning method. *Resonance*, 25(10):1385–1397, 2020.

[14] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[15] David Hoeller and Nikita Rudin. RSL RL: Fast and simple implementation of rl algorithms, designed to run fully on gpu. https://github.com/leggedrobotics/rsl$_{r}l, 2024. Accessed : 2024 - 12 - 29.$

[16] Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, and Si-Hao Deng. Hyperparameter optimization for machine learning models based on bayesian optimizationb. *Journal of Electronic Science and Technology*, 17(1):26–40, 2019.