Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Improving Spot's Walking Abilities in Simulation

Jakub Jon

Supervisor: Mgr. Martin Pecka, PhD.
January 2024

Acknowledgements

Declaration

Abstract

Keywords: ROS, Reinforcement Learning

Supervisor: Mgr. Martin Pecka, PhD.

Abstrakt

Klíčová slova: ROS, Posilované Učení

Překlad názvu: Vylepšení pohybových schopností Spota v simulaci

Contents

Figures

Tables

Chapter 1

Introduction

Over the last few years, many new machines have entered the quadrupedal robot market. There are both hobby versions of these machines as well as robots applicable in factories and other professional facilities.

There are two major approaches to controlling four-legged robots. First, model predictive control (MPC) is a popular computational approach to generating movements. Second, reinforcement learning (RL) has shown a lot of promise recently and has been applied to generate robust walking policies.

MPC controllers leverage the knowledge of the robot's dynamics by simulating the robot forward in time and optimizing the control input, usually represented as joint torques. The result of the optimization process is a function f(t) mapping time t to optimized torques τ. As these control signals are being applied to the individual motors, the robot's state evolves. However, since the model is not perfect due to model uncertainties or unmodeled dynamics in the forward simulation part of the MPC algorithm, the optimized state and the actual state diverge, making it necessary to optimize the control input trajectory again. This re-optimization is usually performed with a frequency of 100 Hz [4].

For RL algorithms, the most heavy-lifting calculations take place during a training phase when a policy is being searched for. Then, once a policy has been found, using it for generating limb movements is computationally much cheaper as compared to the MPC approach. Moreover, reinforcement learning policies tend to be much more robust that MPC controllers.

The goal of the presented project was to implement methods described in [5] and [6] and use them to control a model of Spot in simulation. All the code necessary for both training and deployment is available online [1]. Differences between training Anymal C, the robot for which the methods in [5] and [6] were originally designed, are briefly described. Finally, we talk about training and deployment of robots with additional payload on their back where an estimate of the payload's inertial properties is known.

[1]See bobnet and bobnet_isaac projects

Chapter 2
Simulator

Reinforcement learning (RL) algorithms are built to solve the Markov Decision Process, both fully observable and partially observable. A typical representation of a Markov Decision Process is depicted in figure 2.1. An agent takes actions in an environment, getting rewards and new observations along the way. The agent's goal is to find a policy that maximizes the expected discounted return.

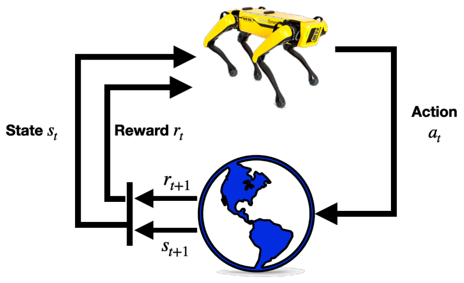


Figure 2.1: Markov Decision Process [1]

Typically environments are some kind of a simulated world with an agent in it. Through actions, the agent can interact with the environment. These interactions usually influence future observations and rewards, thus making RL a hard problem, as future rewards potentially depend on past decisions made. An example of a simulated environment can be seen in figure 2.2.

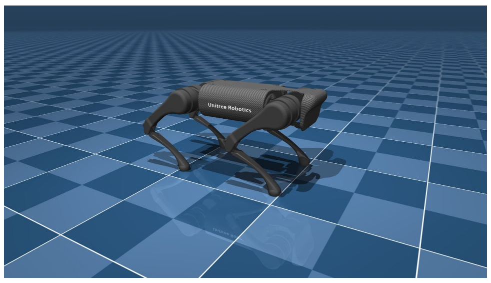


Figure 2.2: Agent - A1 robot - inside the Mujoco simulator [2]

There are many different physics simulators with the ability to accurately simulate highly dynamic machines, such as walking robots or drones. Examples are Gazebo, Mujoco[2], Raisim[7] and more. However, since many reinforcement learning algorithms use deep neural networks for decision making, with training on a GPU, and the above-mentioned simulators can only run their simulation code on a CPU, there is constant data transfer between the GPU and the CPU. This is arguably the biggest bottleneck and slows the training process dramatically.

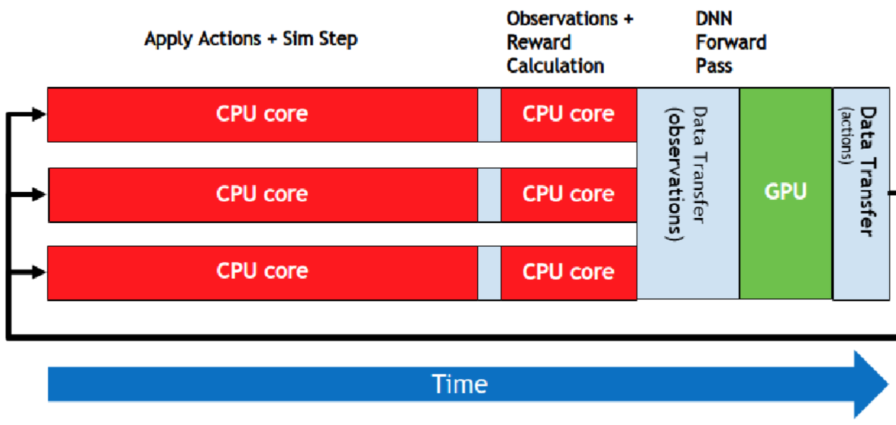


Figure 2.3: End-to-end simulation on a GPU [3]

Isaac Gym [3], a high performance GPU-based physics simulator for robot learning, solves this issue. As can be seen in figure 2.4, Isaac Gym's approach to simulating environments is to run all the code on a GPU, removing the need for any GPU2CPU or CPU2GPU data transfers.

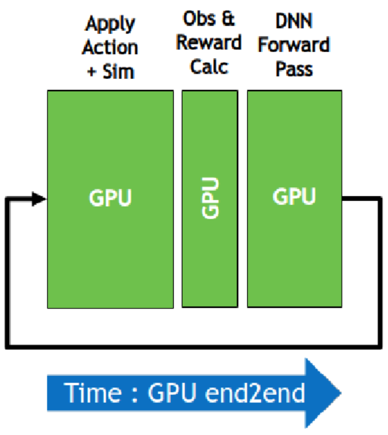


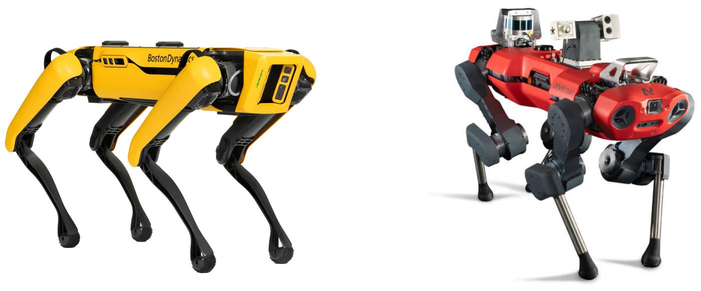
Figure 2.4: Typical simulation and training pipeline [3]

Therefore, in this work, we will use Isaac Gym for it's end to end GPU simulation approach, as it, in conjuction with RL training, can reduce training times from hours to mere minutes [8].

Chapter 3

Problem definition

Before we can start training an agent to solve an MDP, we first need to define it. The goal is to teach two quadrupedal robots walk on a wide variety of terrains. The two robots in question are Anymal C [9] and Spot [10].



(a) : Spot **(b) :** Anymal C

Figure 3.1: Quadrupedal robots

These robots will act in an environment where there are different terrain types with various properties. These terrains include stairs, noisy terrain and flat terrain with random blocks scattered around. Terrain properties which are randomly altered are friction coefficients, stairs sizes and more. The training environment inside the simulator is in figure 3.2.

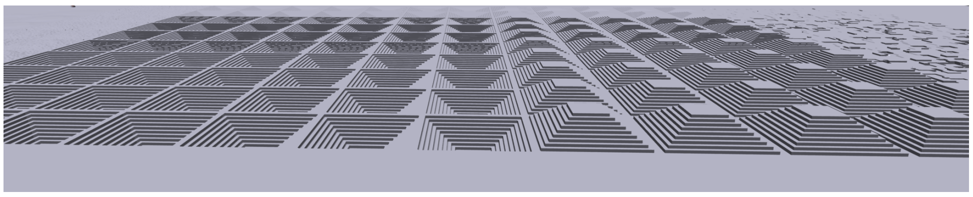


Figure 3.2: Environment

There are many reward terms included. The total reward is calculated as a clipped weighted average of these individual rewards:

$$r_{\mathrm{t+1}} = \max\{0, \sum_{i=0}^{N} w_i \cdot R_i(s_t, a_t, s_{t+1})\} \tag{3.1}$$

where N represents the total number of rewards, w_i represents the i-th reward term's weight and $R_i(s_t, a_t, s_{t+1})$ is the i-th reward term. Since the environment is reset whenever the robot's body touches the ground, we clip the total reward term at zero, so as to make sure the robot does not minimize "time till reset".

Robots are given positive reward signals whenever their velocity is close to the commanded velocity. Also, to generate more visually pleasing gaits, each foot's time off the ground is rewarded as well. Negative reward signals include punishments for high joint velocities and accelerations, high torques, undesired contacts with the ground and others.

Observations and training are described in chapters 4 and 5.

Chapter 4

Teacher model

There are many different ways how one can train a model to generate walking motions for quadrupedal robots. We will take the approach described in [4], which is a two-stage training pipeline.

4.1 Teacher model inputs

First, a teacher model is trained. The model is a combination of encoders and multi-layer perceptrons, accepting proprioceptive information, exteroceptive information and privileged information as an input. The model's output is an action, a vector of joint angles, that can directly be used to control our robot.

Input	**Dimension**
command	3
projected gravity	3
base linear velocity	3
base angular velocity	3
joint angle residuals	12
joint velocities	12
joint angle residual history	3x12
joint velocity history	2x12
action history	2x16
CPG information	8

Table 4.1: Proprioceptive information

Proprioceptive information includes desired base velocities v_x and v_y as well as desired yaw rate ω_z, all expressed in the base frame. These three values are what we use to steer the robot.

Information about the robot's base orientation w.r.t. the world frame is encoded in a projected gravity vector $\mathbf{g}_B$ with norm one. This vector always points in the negative direction of the Z axis in the world frame but changes in the base frame. As the subscript suggests, $\mathbf{g}_B$ is expressed in the robot's base frame.

Next, we include base linear and angular velocites, once again, expressed w.r.t. the base frame.

To give the machine some information about its limbs' state, we include joint angles and joint velocities together with a history of these values from the past few observations. This allows the network to infer other variables, such as joint accelerations or potentially detect foot slipping.

A few actions from the past are present. The model is punished for abrupt changes on its output, therefore we need to include it here, such that the model can take last actions into account.

Input	Dimension
height measurements	4x52

Table 4.2: Exteroceptive information

Height measurements are an extremely important part of the input to the neural network. Thanks to them, the model can essentially see what's around the robot and move its feet accordingly. There are 52 height samples per leg. These heights are sampled on a star-like pattern around the foot's current position, see figure 4.1.

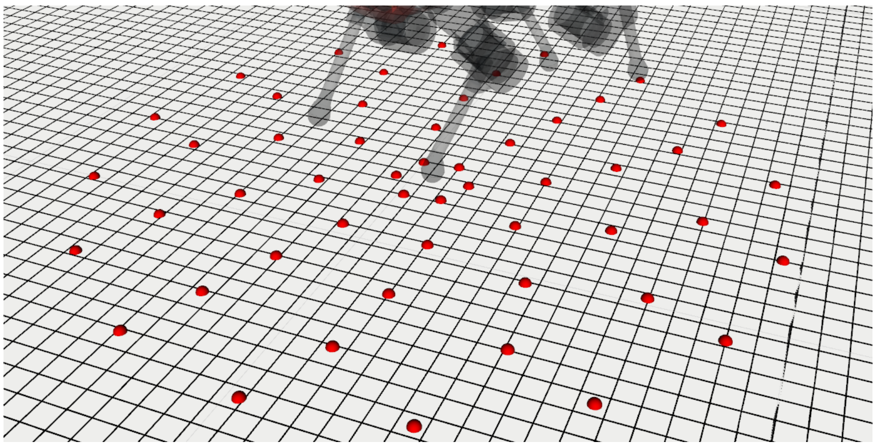


Figure 4.1: Sapling pattern around a single foot

Input	Dimension
foot contacts	4
reaction forces	4x3
surface normals	4x3
friction coefficient	1
upper leg contact	4
lower leg contact	4
airtime	4

Table 4.3: Privileged information

Privileged information includes quantities not accessible the robot in the real world. However, to make training for our teacher model a tiny bit easier, we include information about the presence or absence of a foot contact, current ground reaction forces, surface normals, ground friction coefficient (same around the entire robot), upper leg and lower leg contacts and lastly airtime - information about how long a given leg has been off the ground for. Figure 4.2 gives more clarification on some of these terms.

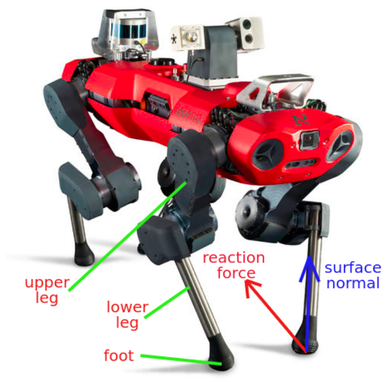


Figure 4.2: Anymal C annotated

4.2 Training

To train the teacher model, we use an open-source implementation of the Proximal Policy Optimization (PPO) algorithm [11] for PyTorch [1].

The training process is depicted in figure 4.3.

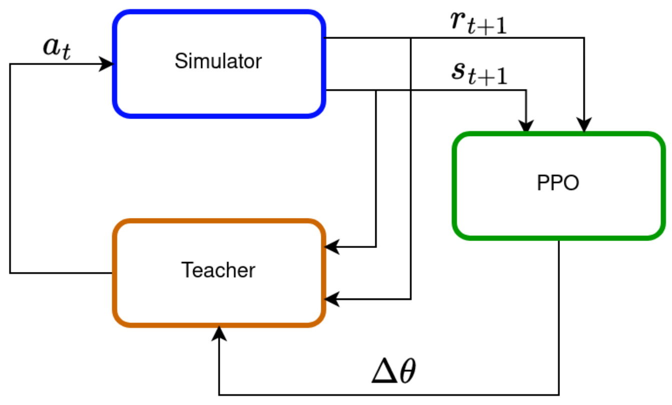


Figure 4.3: Teacher model training

Our model interacts with the simulated environment through its actions a_t. The PPO module processes subsequent observations and rewards, generating an update vector to the model parameters θ. New model parameters are calculated as $\theta_{t+1} = \theta_t + \Delta\theta$.

[1]https://github.com/leggedrobotics/rsl_rl

Chapter 5
Student policy

In this seconds stage of the training pipeline, a student model is trained. The student model is neural network that, after training, is applicable on the real robot. The input to the student is proprioceptive information in combination with exteroceptive information. Privileged information is excluded.

The student interacts with the environment and tries to immitate the teacher. For each state, both the teacher and the student generate an action. Ideally, these should be the same. Additionally, the student model tries to estimate the privileged information based on observations from the past. An error between privileged values and estimated privileged values is to be minimized.

Application of the trained student policy in the real worlds requires that artificial noise is added to the observations it receives. This makes the height measurements different from the ones the teacher uses. It is desirable that the student also tries to remove this noise.

Total loss for the student network is defined as

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{action}} + 0.5 \cdot \mathcal{L}_{\text{privileged}} + 0.5 \cdot \mathcal{L}_{\text{heights}} \tag{5.1}$$

with

$$\mathcal{L}_{\text{privileged}} = \|\mathbf{p}_{t+1} - \hat{\mathbf{p}}_{\mathbf{t+1}}\| \tag{5.2}$$

$$\mathcal{L}_{\text{action}} = \|\mathbf{a_t} - \hat{\mathbf{a}}_{\mathbf{t}}\| \tag{5.3}$$

$$\mathcal{L}_{\text{heights}} = \|\mathbf{h_t} - \hat{\mathbf{h}}_{\mathbf{t}}\| \tag{5.4}$$

where $\mathbf{a_t}$ is the action produces by the teacher, $\hat{\mathbf{a}}_{\mathbf{t}}$ is the action produced by the student, $\mathbf{p}_{t+1}$ is actual privileged information, $\hat{\mathbf{p}}_{\mathbf{t+1}}$ is estimated privileged information, $\mathbf{h_t}$ is noise-less heights information and lastly $\hat{\mathbf{h}}_{\mathbf{t}}$ is noisy heights information.

Students parameters are updated every iteration via the update rule $\phi_{t+1} = \phi_t + \Delta\phi$.

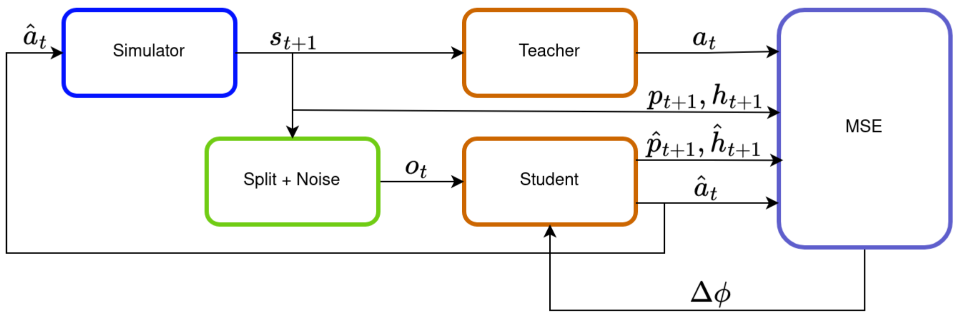
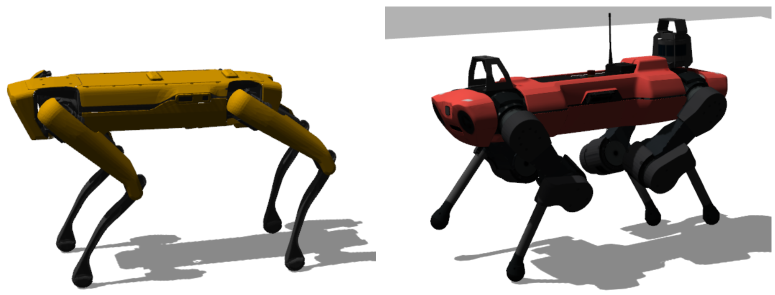


Figure 5.1: Student model learning

Chapter 6

Results

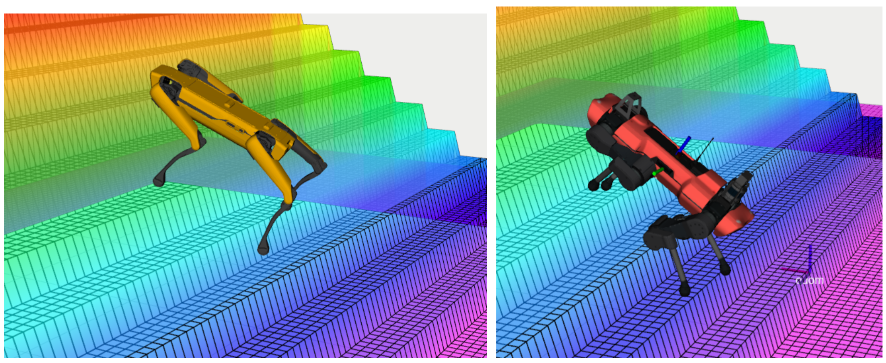
Two student policies have successfully been trained for both robots, Spot and Anymal C. Even though they've been trained in a completely different simulator, they are applicable inside of the Gazebo simulator.



(a) : Spot **(b) :** Anymal C

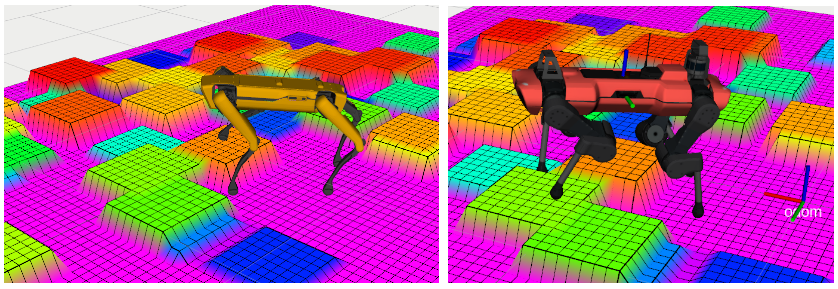
Figure 6.1: Spot and Anymal C inside of Gazebo controlled with their corresponding student networks

Testing included walking up a staircase and traversing a terrain with random blocks scattered around.



(a) : Spot **(b) :** Anymal C

Figure 6.2: Spot and Anymal C walking up a staircase



(a) : Spot **(b) :** Anymal C

Figure 6.3: Spot and Anymal C walking across a terrain littered with random blocks

Both robots can easily traverse these terrains with not much difficulty[1].

6.1 Differences between training Spot and Anymal

Ideally, it should be possible to change the robot's URDF and the inverse kinematics calculation to train a walking policy for a different quadrupedal robot. While this is the case, training Spot turned out to be more difficult a problem, since there is less symmetry present in Spot's model than there is symmetry present in Anymal's model. For example, due to its morphology, walking forwards and backwards for Anymal is very much the same task. That is not the case for Spot though.

For the same hyperparameters, training Spot takes longer, however once converged, the mean accumulated reward for both Spot and Anymal is the same. Lowering training convergence times would require reward shaping specific to each individual robot.

6.2 Training with additional payload

Experiments were also carried out where a payload was placed on Anymal's back and a noisy estimate of its inertial properties was added to the proprioceptive information. It was four values: mass m and diagonal elements of the inertia tensor I_{xx}, I_{yy}, I_{zz}. The mass was ranged 0 to 100 kilograms and all the inertia tensor elements ranged from 0.001 to 0.01 $kg \cdot m^2$. Training a student policy was successful, however using the policy in Gazebo proved difficult and made the robot shake violently. We believe it was caused by the policy's overfitting to the Isaac Sim's parameters. Futher adjustments to the student policy training procedure are necessary to improve generalization.

[1]Videos available here

Chapter 7

Conclusion

In this work, a two-stage training pipeline for training a walking policy for quadrupedal robots was introduced and used to train walking controllers for two robots: Anymal C and Spot. They were afterwards used in Gazebo, a different simulator than the one used for training, which was Isaac Sim. The aforementioned robots are capable of traversing a wide variety of terrains, including stairs and terrains with random obstacles in the way.

The possibilities of reinforcement learning in control are endless and many new applications are still waiting to be uncovered.

Bibliography

[1] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning.* Cambridge University Press, 2023, https://D2L.ai.

[2] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems.* IEEE, 2012, pp. 5026–5033.

[3] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac gym: High performance gpu-based physics simulation for robot learning," 2021.

[4] R. Grandia, F. Jenelten, S. Yang, F. Farshidian, and M. Hutter, "Perceptive locomotion through nonlinear model predictive control," 2022.

[5] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning robust perceptive locomotion for quadrupedal robots in the wild," *Science Robotics*, vol. 7, no. 62, Jan. 2022. [Online]. Available: http://dx.doi.org/10.1126/scirobotics.abk2822

[6] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, no. 47, Oct. 2020. [Online]. Available: http://dx.doi.org/10.1126/scirobotics.abc5986

[7] J. Hwangbo, J. Lee, and M. Hutter, "Per-contact iteration method for solving contact dynamics," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 895–902, 2018. [Online]. Available: www.raisim.com

[8] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," 2022.

[9] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch, R. Diethelm, S. Bachmann, A. Melzer, and M. Hoepflinger, "Anymal - a highly mobile and dynamic quadrupedal robot," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 38–44.

[10] “Boston dynamics,” Jan 2024. [Online]. Available: https://bostondynamics.com/

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” 2017.