Abstract

Quadruped possess inherent advantages for locomotion, such as improved stability, robustness and adaptability, making them the ultimate choice in traversing challenging terrains and dynamic environments compared to other robotic forms. Developing controllers that effectively exploit quadrupeds' inherent advantages is a challenging task. Recent advances in employing Deep Reinforcement Learning (DRL) for quadruped locomotion have proven to be remarkably robust and promising. In this report, we present an end-to-end RL framework enabling a quadruped robot to achieve stable gait navigation on complex terrains. Trained initially on flat surfaces with fractal noise on simulation, our approach relies on proprioceptive inputs, eliminating the need for cameras or exteroception sensors. The framework exhibits robustness against external pushes and friction changes, promising potential applications in search and rescue, exploration, and in the defense sector. While focusing on flat surfaces, our future plans involve extending training to diverse terrains.

Prior Works

Recent developments have prominently employed end-to-end Deep Learning (DL) techniques, characteristic of learning-based approaches, to revolutionize quadruped locomotion across challenging and uneven terrains. These advancements have demonstrated notable robustness and adaptability, attesting to the efficacy of DL-based methodologies. By adopting end-to-end DL frameworks, which use sensory inputs to predict motor commands or forecast trajectory parameters, researchers tap into the stability and agility of quadrupeds, enabling them to navigate complex terrains more effectively. This shift avoids extensive manual parameter tuning which are needed in the case of conventional controllers for quadruped locomotion.

Some of the prominent works in quadrupedal locomotion using proprioception inputs on uneven terrains will be discussed in detail in this section.

1. DreamWaQ: Learning Robust Quadrupedal Locomotion With Implicit Terrain Imagination via Deep Reinforcement Learning

In this paper, the authors propose a framework that trains a robust locomotion policy for quadruped robots with only proprioception inputs using Deep RL algorithm. The contributions of this paper, as stated by the authors are threefold:

1) A novel locomotion learning framework via an asymmetric actor-critic architecture is proposed to implicitly imagine terrain properties using only proprioception.

2) A context-aided estimator network is proposed to estimate body state and environmental context jointly. Together with the policy, the proposed method outperforms existing learning-based methods.

3) A robustness and durability evaluation of the learned policy in the real world conducted through walking in diverse outdoor environments.

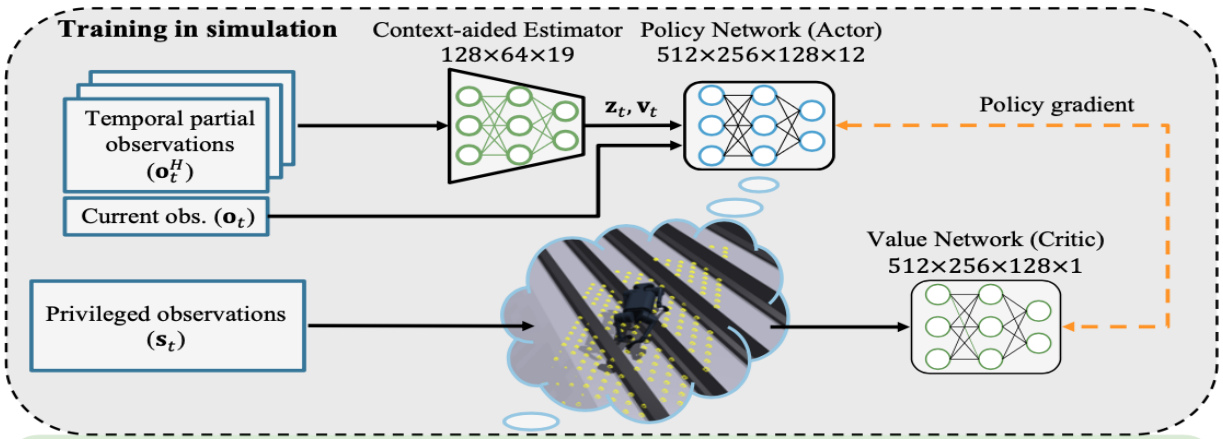

Fig. 1

The core concept underlying this method revolves around learning a good representation of the state which then serves as the foundation for predicting the joint angles directly from proprioceptive inputs. Fig. 1, shows the overview of DreamWAQ learning framework. Recent works have leveraged the teacher-student training paradigm. The teacher network deployed in simulation is presented with the full state inputs which includes privileged extrinsics information such as height map of terrain, friction coefficient, terrain normal, etc. The trained teacher policy then serves as the expert policy to train the student network to be deployed in real world

hardware which has access to only the observation/partial state inputs. However, as stated in the paper, Behaviour Cloning has certain limitations because the student policy learns from the good action supervision of the teacher policy and might be unable to learn the failure states encountered by the teacher policy in the early stages of learning. This serves as a motivation for the authors to propose a single learning framework having asymmetric actor-critic architecture for learning robust locomotion behavior on uneven terrains.

Terminologies

$\mathbf{o}_t^H = \begin{bmatrix}\mathbf{o}_t & \mathbf{o}_{t-1} \dots \mathbf{o}_{t-H}\end{bmatrix}^T$ - temporal observation at time t over the past H measurements

$\mathbf{o}_t = \begin{bmatrix}\boldsymbol{\omega}_t & \mathbf{g}_t & \mathbf{c}_t & \boldsymbol{\theta}_t & \dot{\boldsymbol{\theta}}_t & \mathbf{a}_{t-1}\end{bmatrix}^T$ - observation at time t (n x 1 vector)

$\mathbf{s}_t = \begin{bmatrix}\mathbf{o}_t & \mathbf{v}_t & \mathbf{d}_t & \mathbf{h}_t\end{bmatrix}^T$ - privileged observation

Zt - latent representation of world state

Vt - body linear velocity estimated by CENet

Policy Network

The observations to the policy network are

1. **O**t
2. **Z**t
3. **V**t

Zt and **V**t are estimated by the Context-Aided Estimator Network while **O**t is obtained from joint encoders and IMU. Since the policy network is provided only with the partial observations, it ensures seamless transition to hardware implementation, thus bypassing the usual method of training a student network architecture.

Value Network

The value network receives the full state of the world, which includes partial observation **Ot,** body velocity **Vt,** disturbance force **dt** and height map scan **ht** and is trained to output a single value that represents the value of the state.

Action Space

The action space is a 12 x 1 vector representing the target joint angles of the robot with respect to the robot's initial stand still pose.

$$\boldsymbol{\theta}_{\text{des}} = \boldsymbol{\theta}_{\text{stand}} + \mathbf{a}_t.$$

Reward Function

Reward	Equation (r_i)	Weight (w_i)
Lin. velocity tracking	$\exp\{-4(\mathbf{v}_{xy}^{\text{cmd}} - \mathbf{v}_{xy})^2\}$	1.0
Ang. velocity tracking	$\exp\{-4(\omega_{\text{yaw}}^{\text{cmd}} - \omega_{\text{yaw}})^2\}$	0.5
Linear velocity (z)	v_z^2	-2.0
Angular velocity (xy)	$\boldsymbol{\omega}_{xy}^2$	-0.05
Orientation	$\lvert\mathbf{g}\rvert^2$	-0.2
Joint accelerations	$\ddot{\boldsymbol{\theta}}^2$	-2.5×10^{-7}
Joint power	$\lvert\boldsymbol{\tau}\rvert\lvert\dot{\boldsymbol{\theta}}\rvert$	-2×10^{-5}
Body height	$(h^{\text{des}} - h)^2$	-1.0
Foot clearance	$(p_{f,z,k}^{\text{des}} - p_{f,z,k})^2 \cdot v_{f,xy,k}$	-0.01
Action rate	$(\mathbf{a}_t - \mathbf{a}_{t-1})^2$	-0.01
Smoothness	$(\mathbf{a}_t - 2\mathbf{a}_{t-1} + \mathbf{a}_{t-2})^2$	-0.01
Power distribution	$\text{var}(\boldsymbol{\tau} \cdot \dot{\boldsymbol{\theta}})^2$	-10^{-5}

Fig. 2

Context - Aided Estimator Network

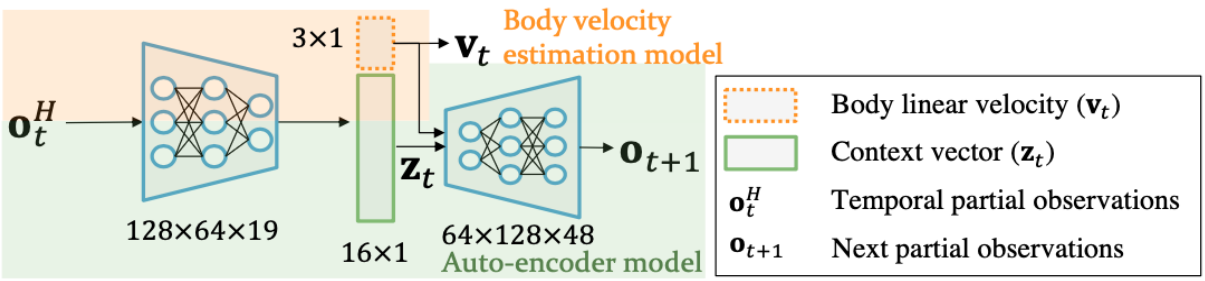

Fig. 3

The context-aided Estimator Network is the centerpiece of this paper. In a higher level, CENet predicts a context vector **Zt,** that intends to represent the terrain properties, and the body linear velocity **Vt,** which in turn helps to get a better understanding of the proprioceptive inputs. The latent vector **Zt,** is used to estimate **Vt** as well as to predict **Ot+1**. CENet architecture can be inferred from **Fig. 3** which depicts an auto-encoder architecture. The authors use a β- variational auto-encoder (β-VAE) as the auto- encoder architecture. CENet is optimized using a hybrid loss function, defined as follows:

$$\mathcal{L}_{\text{CE}} = \mathcal{L}_{\text{est}} + \mathcal{L}_{\text{VAE}}$$

where L_{est} and L_{VAE} are the body velocity estimation and VAE loss, respectively. The VAE network is trained with the standard B-VAE loss, which consists of reconstruction and latent losses. The authors employed MSE for the reconstruction loss and Kullback-Leibler (KL) divergence as the latent loss. The VAE loss is formulated as:

$$\mathcal{L}_{\text{VAE}} = MSE(\tilde{\mathbf{o}}_{t+1}, \mathbf{o}_{t+1}) + \beta D_{\text{KL}}(q(\mathbf{z}_t|\mathbf{o}_t^H) \parallel p(\mathbf{z}_t))$$

where **Ot+1** is the reconstructed next observation, $q(\mathbf{z}_t|\mathbf{o}_t^H)$ is the posterior distribution of the at given $\mathbf{o}_t^H$. $p(\mathbf{z}_t)$ is the context's prior distribution parameterized by a Gaussian distribution.

2. Minimizing Energy Consumption Leads to the Emergence of Gaits in Legged Robots

In this paper, the authors show that learning to minimize energy consumption plays a key role in the emergence of natural locomotion gaits at different speeds in real quadruped robots. The same approach leads to unstructured gaits in rough terrains which is consistent with the findings in animal motor control. Locomotion consumes a significant fraction of an animal's metabolic energy, suggesting that development of different gaits such as walk, trot, gallop, etc. are energy efficient at certain range of speeds. It also points out the fact that animals transition between different gaits at different speeds in order to minimize their energy consumption. In this work, the authors design an end-to-end learning framework to show how energy minimization leads to the emergence of structured locomotion gait patterns in flat terrains as well as unstructured gaits in complex terrains at different commanded speeds. This work leverages the use of the teacher-student training paradigm. The teacher network is provided with proprioceptive inputs along with privileged extrinsics information such as terrain height, terrain normal, gravity vector, etc at every time step and is trained in simulation. The student network has access to only the current proprioceptive inputs and history of proprioceptive inputs and predicted action outputs. The goal of the student policy is to mimic the behavior of the teacher policy. More importantly, the student policy must be able to infer the privileged information with the history of proprioceptive inputs and actions. The authors capitalize on their prior work, "RMA: Rapid Motor Adaptation for Legged Robots," to facilitate the adaptation of the policy learned in simulation onto physical hardware. This process is executed through the implementation of the student-teacher framework.

The main contributions of this paper, as stated by the authors include:

• Show that minimizing energy consumption plays a key role in the emergence of natural locomotion patterns in both flat as well as complex terrains at different speeds without relying on demonstrations or predefined motion heuristics.

• Show that the emergent gaits at different target speeds correspond to conventional animals in the similar Froude number range (sheep/horse) without any sort of pre-programming.

• Present a distillation-based learning pipeline to obtain velocity-conditioned policy that displays smooth gait transition as the target speed is changed.

• Demonstrate the emergent behaviors, robustness analysis, and gait patterns in simulation as well as a real-world budget quadruped robot.

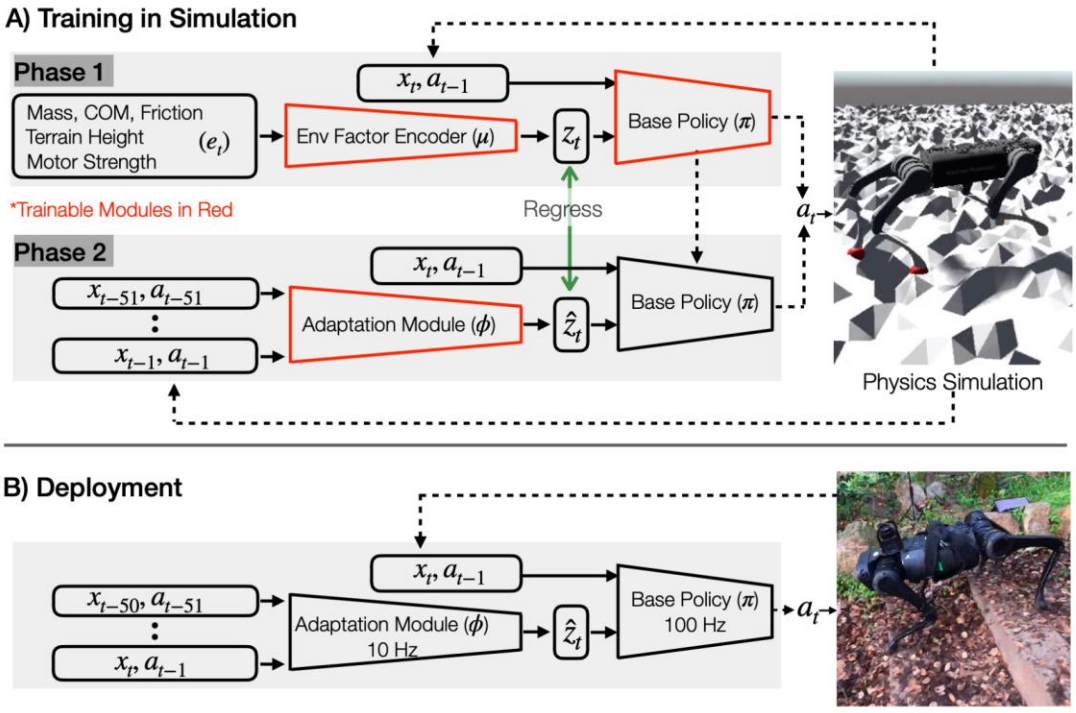

Fig. 2: RMA consists of two subsystems - the base policy π and the adaptation module ϕ. **Top:** RMA is trained in two phases. In the first phase, the base policy π takes as input the current state x_t, previous action a_{t-1} and the privileged environmental factors e_t which is encoded into the latent extrinsics vector z_t using the environmental factor encoder μ. The base policy is trained in simulation using model-free RL. In the second phase, the adaptation module ϕ is trained to predict the extrinsics $\hat{z}_t$ from the history of state and actions via supervised learning with on-policy data. **Bottom:** At deployment, the adaptation module ϕ generates the extrinsics $\hat{z}_t$ at 10Hz, and the base policy generates the desired joint positions at 100Hz which are converted to torques using A1's PD controller. Since the adaptation module runs at a lower frequency, the base policy consumes the most recent extrinsics vector $\hat{z}_t$ predicted by the adaptation module to predict a_t. This asynchronous design was critical for seamless deployment on low-cost robots like A1 with limited on-board compute. Videos at: https://ashish-kmr.github.io/rma-legged-robots/

Fig. 4

State Space

The network architectures of the policy network and the value network are symmetric. The state is 30 dimensional containing the joint positions (12 values), joint velocities (12 values), roll and pitch of the torso and binary foot contact indicators (4 values). The environment information as depicted in Fig. 4 includes center of mass position and the payload (3 dimensions), motor strength (12 dimensions), friction (1 dimension), linear speed in x direction vx (1 dimension), linear speed in y direction vy (1 dimension) and yaw speed ωyaw (1 dimension), making it a 19-dim vector.

Action Space

The action space is 12 dimensional corresponding to the target joint position for the 12 robot joints. The predicted joint angles are with respect to the robot's initial stand-still position.

$$\boldsymbol{\theta}_{\text{des}} = \boldsymbol{\theta}_{\text{stand}} + \mathbf{a}_t.$$

Reward Function

$$r = r_{\text{forward}} + \alpha_1 * r_{\text{energy}} + r_{\text{alive}}$$

$$r_{\text{forward}} = -\alpha_2 * \left|v_x - v_x^{\text{target}}\right| - \left|v_y\right|^2 - \left|\omega_{\text{yaw}}\right|^2$$

$$r_{\text{energy}} = -\boldsymbol{\tau}^T \dot{\boldsymbol{q}},$$

The total reward is the summation of three reward terms, namely, forward reward, energy reward and survival reward. The forward reward term rewards the agent for walking straight at the specified speed, energy reward term penalizes energy consumption and the survival reward term is the survival bonus.

The authors use the A1 URDF to simulate the A1 robot in the RaiSim simulator. They generate complex terrains using the inbuilt fractal terrain generator for flat and uneven surfaces. They claim that tra the policy on a completely flat surface results in unnatural gaits and leads to lesser foot clearance from the ground. Hence, they train the policies on simple fractal terrains with varying frequency of terrain heights instead of perfectly flat terrain. The policies are tested at 3 different target speeds, namely, 0.375 m/s, 0.9 m/s and 1.5 m/s. Walk gait is observed at 0.375 m/s, trot gait emerges at 0.9 m/s and gallop gait develops at 1.5 m/s.

Our Approach

Following the footsteps of recent end-to-end DL frameworks for quadruped locomotion on uneven terrains, our work builds on the idea of using proprioception to estimate target joint angles. For our training, we employed the Proximal Policy Optimization (PPO) algorithm. We put our policies to the test in the PyBullet simulation environment and apply them to navigate a flat surface with fractals. While we're primarily focused on simulation testing at the moment, our ultimate goal is to take these policies and apply them to our real-world robot.

To guide our learning process, we use a fixed curriculum strategy to adjust reward coefficients, manage responses to external forces, and handle friction changes. We're also planning to explore an adaptive curriculum approach in the future. Currently, our terrain model revolves around a flat fractal surface, while our robot policies are shaped by a fixed curriculum. This decision comes from a mix of careful tuning and methodical experimentation, leading to a well-performing control strategy.

Our in-depth policy evaluation revolves around a set speed of 0.9 m/s, with the robot's movement confined to the x-direction. This approach results in a robust trot gait that can handle varying friction conditions and unexpected pushes even beyond what it was trained on. Looking ahead, we're aiming to expand our capabilities by introducing direction tracking in our future work.

State Space

Our observation space assumes a vector size of 128 x 1. This space is categorized into three distinct categories: history (timestep t-1), proprioceptive inputs (timestep t), and extrinsic information.

In the history category, it encompasses joint action history, joint state history, and joint velocities.

The proprioception input at time t includes present joint states, present joint velocities, base orientation, base velocities, as well as the desired direction and turning direction.

Lastly, the extrinsic category entails various factors such as contact states, terrain height, terrain normal, friction coefficient, external forces, and contact forces.

Action Space

The action space is 12 dimensional corresponding to the target joint position for the 12 robot joints. The predicted joint angles are with respect to the robot's initial stand-still position.

$$\boldsymbol{\theta}_{\text{des}} = \boldsymbol{\theta}_{\text{stand}} + \mathbf{a}_t.$$

Reward Function

We adopt the reward function employed by Rapid Motor Adaptation (RMA), which is formulated as follows:

1) Forward: $\min(v_x^t, 0.35)$
2) Lateral Movement and Rotation: $-\|v_y^t\|^2 - \|\omega_{\texttt{yaw}}^t\|^2$
3) Work: $-|\boldsymbol{\tau}^T \cdot (\mathbf{q}^t - \mathbf{q}^{t-1})|$
4) Ground Impact: $-\|\mathbf{f}^t - \mathbf{f}^{t-1}\|^2$
5) Smoothness: $-\|\boldsymbol{\tau}^t - \boldsymbol{\tau}^{t-1}\|^2$
6) Action Magnitude: $-\|\mathbf{a}^t\|^2$
7) Joint Speed: $-\|\dot{\mathbf{q}}^t\|^2$
8) Orientation: $-\|\boldsymbol{\theta}^t_{\texttt{roll, pitch}}\|^2$
9) Z Acceleration: $-\|v_z^t\|^2$

Termination Condition

We terminate the episode if the episode completes 2000 steps, body roll exceeds 0.4 radians, body pitch exceeds 0.2 radians or body height drops below 0.327m.

Curriculum

We've implemented a fixed curriculum strategy for adjusting reward coefficients, external force magnitudes (for force test), and friction coefficient values (for friction variation test). This strategy is a result of thorough experimentation and follows a systematic approach.

At the outset, we set reward coefficients to very low values and assign maximum limits to each coefficient. Every 8 million timesteps, these coefficients gradually double until they reach their predetermined maximum limits. This strategy, honed through experimentation, ensures a controlled and steady training progression.

To test the robustness of our policy, we carry out force tests and change friction coefficients and observe the behavior of our policy. We apply random external forces along all 3 axes. We start with tiny magnitudes ranging from -5N to +5N along the x, y, and z axes. Throughout training, these magnitudes increase incrementally until they hit a maximum magnitude of 80N. Similarly, the friction coefficient undergoes a similar treatment. We begin with a steady coefficient of 0.6 and as training advances, the range of friction values widens. For example, in the initial 10 to 20 million timesteps, it varies between 0.55 and 0.65. Later, at around 50 million timesteps, this range expands further.

Notably, all our policy evaluations are performed on our custom quadrupedal model, "Stoch3," which is a medium-sized quadruped. The integration of direction tracking capabilities is reserved for our future work.

Learnings

> PyBullet Simulation

Here are few important details that I worked out while using the PyBullet simulator for RL training:

a. The frequency of applying actions to the motors is 20 Hz. This appears as a stable value while working with PyBullet simulation.
b. Certain elements inside the robot's URDF file contain references to their mesh files. To load the URDF file along with the meshes use:

   p.setAdditionalSearchPath(PATH_TO_MESH_FILES)

c. Loading the URDF file requires specifying the base position of the robot on the terrain. Reset the base height of the robot to a suitable value upon running GUI making sure that the robot is as close to the ground as possible. This is particularly important because, upon reset and while training, the robot makes joint motions immediately after the **step** function is called. If the robot is loaded at a height much above the ground surface, it starts to make the movements in mid-air leading to producing noisy data and unstable motions. Perform empty steps inside the **reset** function for suitable timesteps until the robot hits the ground surface and is stabilized.
d. The robot's feet getting stuck in the ground is a common problem faced in the PyBullet simulator. Hence, instantiate the robot's initial height such that it is above the ground surface terrain
e. When the quadruped's URDF file is loaded, the robot is always initialized at a full vertical position (at maximum height). Training with this configuration of full height with a single training environment does not lead to realistic gaits. Initialize the robot with an initial stance position.
f. For motor control, we used '**POSITION_CONTROL'** mode listed in the **'setJointMotorControl2'** method. We found that using a **positionGain** value of 0.5 was most effective while training. It is based on careful experimentation with various gains.

> Development of Environment script

In this section I will be describing the experiments I carried out while creating the environment script for RL training:

Initially we adopted an Incremental Strategy to predict target joint positions. In the papers listed in the **'Prior Works'** section, the authors predict joint angles with respect to the robot's initial stand-still position and can be described as:

$$\boldsymbol{\theta}_{\text{des}} = \boldsymbol{\theta}_{\text{stand}} + \mathbf{a}_t.$$

In our strategy,

Joint angle (t) = Joint angle (t-1) + at

Joint angle (t=0) = Theta (stand)

However, we found that training with this strategy does not lead to development of any gait motion and cannot be realized through hardware.

***** In some of our experiments, we trained our policies on a completely flat surface with no fractal variations. This led to the robot developing unrealistic gaits with very less foot clearance from the ground. This especially becomes a problem if this policy is transferred to hardware because some studies suggest that the robot usually sinks its base height in the real world, much more at higher speeds.

****** In our initial experimentations, we used different reward function terms and their coefficients for training. The total reward term was the summation of individual reward terms multiplied with their respective reward coefficients. We did not adopt any curriculum strategy at that time. Therefore, deciding a good value for the reward/penalty coefficient terms was a harder problem and the robot was unable to move forward and typically collapsed to the ground as training progressed. This is explained by the fact that the robot abandons its task or chooses an early termination when the task reward is overwhelmed by penalties from the auxiliary objectives such as energy minimization term, action rate term, z velocity term, etc. Therefore, we adopted a curriculum strategy for updating the reward/penalty coefficient terms which is explained in the **'Curriculum'** section of '**Our Approach'.**

******* We used StableBaselines3 as our RL Library. We found that normalization of observations and rewards is very essential during training of the policy. StableBaselines performs dynamic

normalization of rewards and observations. Normalization is particularly important for several reasons such as for stability of learning, gradient descent update (not normalizing rewards can lead to very large or very small gradient updates), robustness, exploration-exploitation trade-off, etc.