Lowering reinforcement learning barriers for quadruped locomotion in the task space

Lauren Cooke[1*], and *Callen* Fisher[1]

[1]Department of Electrical and Electronic Engineering, Stellenbosch University, South Africa

Abstract. In contrast to traditional methods like model predictive control (MPC), deep reinforcement learning (DRL) offers a simpler and less model-intensive option to develop quadruped locomotion policies. However, DRL presents a steep learning curve and a large barrier to entry for novice researchers. This is partly due to research that fails to include comprehensive implementation details. Moreover, DRL requires making numerous design choices, such as selecting the appropriate action and observation spaces, designing reward functions, and setting policy update frequencies, which may not be intuitive to new researchers. This paper aims to facilitate entry into reinforcement learning simulations by illuminating design choices and offering comprehensive implementation details. Results demonstrate that training a quadruped robot in the task space yields natural locomotion and increased sample efficiency compared to conventional joint space frameworks. Furthermore, the results highlight the interdependence and interrelation of the action space, observation space, terrain, reward function, policy frequency, and simulation termination conditions.

1 Introduction

Developing robust and adaptive locomotive control policies for legged robots is a well-established and challenging task. Traditional control methods, such as trajectory optimisation [1], state estimation [2], and model-predictive control (MPC) [3] have produced locomotion in varying environments. However, these techniques often require precise dynamic and kinematic modelling which can result in control policies that are arduous to develop. Furthermore, these methods often fail to generalize effectively to unfamiliar environments that were not considered during their development.

In contrast, deep reinforcement learning (DRL) has demonstrated the ability to create complex robotic control policies without the need for rigorous modelling [4, 5]. When successfully applied, reinforcement learning can automate parts of the controller design, while producing robust policies that can generalise to environments not experienced during training [6, 7]. DRL has been used to perform locomotion for various types of legged robots like bipeds [8, 9], and quadrupeds [10-13] as demonstrated in Fig 1. Furthermore, DRL has

* Corresponding author: 22556303@sun.ac.za

achieved numerous control tasks such as navigating uneven terrain [7, 14], performing parkour [15], and real-time obstacle avoidance [16].

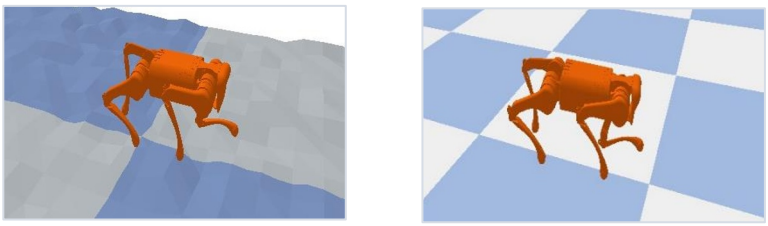

Fig 1. An agent being trained and evaluated in a simulation environment. The image on the left shows the agent being trained on uneven terrain. The image on the right depicts the trained policy being deployed on the robot on flat terrain which proves useful for evaluation purposes.

Despite performing impressive control tasks and reducing the complexity of traditional methods, DRL presents a substantial barrier to entry for new researchers [17]. Current algorithms often require considerable training times [10] and extensive hyperparameter tuning [18]. Results are also heavily linked to the environment's design whereby factors like choice of state space, action space [19, 20] and termination conditions can cause substantial differences in a policy. Moreover, the design of a reward signal contributes significantly to the behaviour of the developed policy [10, 15, 16, 21].

The term 'environment' in this context refers to the simulated setting in which the quadruped robot operates and interacts. This includes the physical layout, obstacles, surface properties, the simulator parameters and functions that fall under the OpenAI gym environment standard [22]. Typically, these parameters are selected through a trial-and-error approach or based on intuition. Such intuition can be developed by replicating the results of other research, which requires authors to thoroughly document their implementation details, experimental setups, and hyperparameters [17]. While some researchers provide open-source code, such as Zhuang et al. [15] and Tan et al. [4], adapting these principles to different environments, tasks, and robots requires deeper domain knowledge.

This work focuses on the challenges that new researchers face when developing a DRL framework for quadruped locomotion. It aims to facilitate researchers in developing their intuition for this control problem by motivating design choices, highlighting failure points, and providing comprehensive implementation details. By simplifying the problem and detailing design choices, the intention is to prevent wasted effort that originates from research that is easily misinterpreted or non-reproducible. Furthermore, this paper advocates for the use of the task space as the choice of action space for quadruped locomotion tasks. The choice of action space is task-dependent, and has been explored for tasks involving manipulators [19, 23], bipeds [8], and quadrupeds [20, 21]. Ultimately, this work aims to contribute to the growing body of knowledge in DRL research for quadruped locomotion, while also providing a solid foundation for future investigations in this domain.

1.1 Related work

Numerous works leverage reinforcement learning to develop quadruped locomotion control policies; however, each implementation differs substantially. Some research, such as Hwangbo et al. [5] use reference trajectories to develop control policies. This work utilises a physical and simulated ANYmal [25] robot to develop control policies that can respond to high-level velocity commands as well as recover from a fall. The work by [4] employs a user-

specified combination of training from scratch and using reference trajectories. However, this work produced trotting and galloping motions on flat ground in simulation as well as on a physical Minitaur [26] without the use of reference trajectories. Haarnoja et al. [6], and Ha et al. [24] also utilised a Minitaur robot, although, both works train control policies directly on the physical robot and without any reference trajectories. Training directly on physical robots streamlines the training process. However, in the case of [6], it enforces the need for manual, time-consuming, resetting of the training environment. To address this shortcoming, [24] presents a multi-task learning procedure that embeds safety constraints within the DRL framework.

Current research also seeks to push the boundaries of quadruped locomotion by training more complex robotic platforms in environments with uneven terrain [7, 10, 11, 14], and challenging environments where jumping [21], or a combination of dynamic behaviours, is required [15, 16, 27]. Locomotion on various forms of uneven terrain is demonstrated by Lee et al. [7] using the ANYmal robot. This work utilises a two-stage DRL framework whereby the first-stage learner has access to privileged information about the environment. Uneven terrain is explored further without the use of privileged information by [10] and [11]. Both works utilise a Unitree A1 robot [28] trained in a simulation environment that features uneven terrain. The work by [10] emphasises the use of minimising energy to produce more efficient and natural gaits. While [11] prioritises the choice of action space, specifically the use of the task space, to generate efficient and natural policies. The limits of a Unitree A1 robot's capabilities are explored by Zhuang et al. [15] whereby a two-stage DRL framework is utilised to produce agile and dynamic motions, like leaping, climbing, and crawling.

While all the above authors achieve their respective locomotion goals, some of the intricacies of the training process are not detailed. This includes details such as hyperparameters, simulation time step, termination conditions, and terrain details. Furthermore, work that has a more discrete focus, tends to neglect the synergy between various components that enables the achievement of the desired behaviour. Although this list of authors is not exhaustive, it indicates that there is a problem describing implementation details. Often the significance of framework features can be misinterpreted or misunderstood. Hence, not having all implementation details can reduce progress in the field and lead to wasted research effort. This work addresses the obstacles of previous research by providing all implementation details as well as addressing the significance and effect of various DRL framework components for quadruped locomotion.

The remainder of this paper is organised as follows: Section 2 presents background information about reinforcement learning and the choice of algorithm. In Section 3 the simulation framework and setup are explored. Further investigation into the design decisions and framework component choices is presented and evaluated in Section 4. Finally, conclusions are drawn in Section 5.

2 Reinforcement learning

Using the reinforcement learning problem formulated by Sutton and Barto [29], quadruped locomotion can be modelled as a Markov Decision Process (MDP). Formally MDPs consist of four components; a state space, an action space, state transition probabilities and a reward. This is typically specified by the tuple (S, A, R, P); where S represents the state or observations of the environment, A represents the quantity selected by the agent that is used to control the environment, $P(s_{t+1}|s_t, a_t)$ is the probability of the next state, (s_{t+1}),

occurring for a given state-action pair, (s_t, a_t), while $R(s_t, a_t, s_{t+1})$ is the expected reward for transitioning from state s_t to state s_{t+1} due to action a_t.

The goal of an agent is to select actions that will maximize future rewards given a particular environment [29]. A policy (π), which dictates action selection, is determined using appropriate learning algorithms. Several state-of-the-art learning algorithms exist, however, one of the most prevalent in quadruped locomotion literature is Proximal Policy Optimisation (PPO) [18]. PPO [30] is an on-policy, model-free algorithm that optimises the following clipped surrogate objective:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[min\left(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$

where $\hat{A}_t$ is the advantage function [31] at time t, ϵ is a hyperparameter that sets the clip range, and $r_t(\theta)$ is the probability ratio. This ratio can be expressed as:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

where π_θ is a stochastic policy, and θ_{old} and θ are the parameters before and after the update, respectively. The surrogate objective attempts to keep the old and new policies close together, thereby penalising large policy updates.

3 Experiment setup and training details

This section details the reinforcement learning framework and design decisions. The overall control diagram is presented in Fig 2. The elements of this diagram are discussed below. Furthermore, the framework utilises the Unitree A1 robot [28]. A simplified representation of the robot is presented in Fig 2.

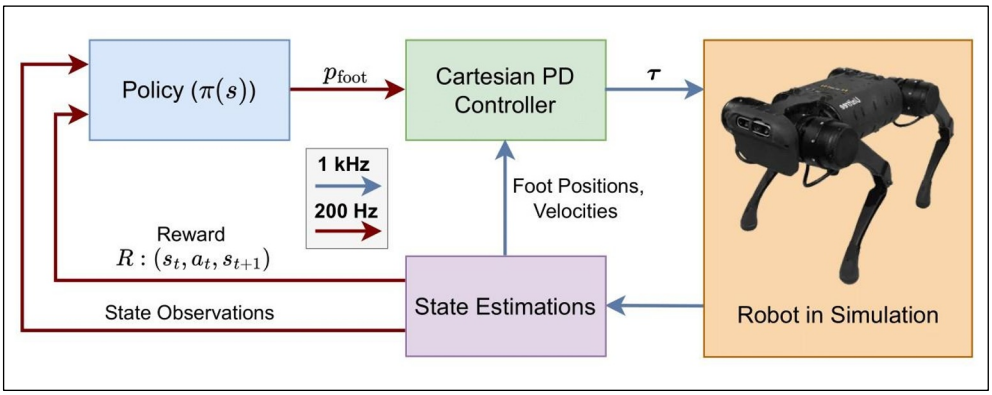

Fig 2. The policy (π) outputs desired foot positions (p_{foot}) at 200 Hz. The desired foot positions are scaled to coordinates in the robot's leg frame. They are then converted to torques using Cartesian PD control, which is updated at 1 kHz. The Unitree A1 robot image featured in the diagram is from [28].

3.1 Action space

We propose the utilisation of the task space or Cartesian space, where desired foot positions are generated by the agent $(A \in \mathbb{R}^{12})$†. Unlike learning in the joint space, the task space

†$A = \{a_1, a_2, \dots, a_{12}\}$, which are all real numbers

imposes intrinsic constraints wherein, the agent's actions are confined to possible leg positions. Consequently, the agent's exploration is limited without unnatural bias.

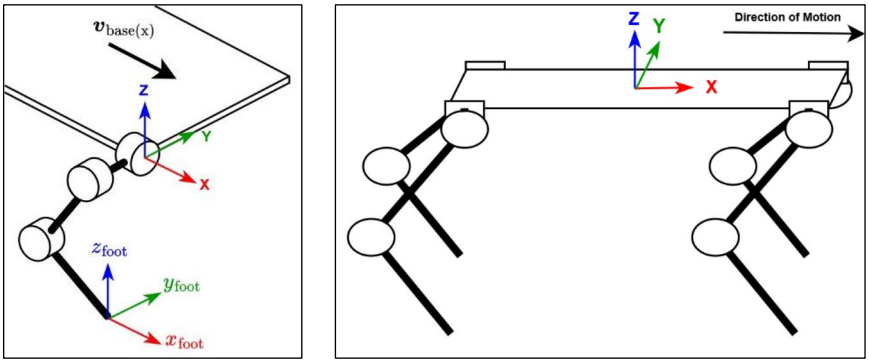

Fig 3. A simplified representation of the robot coordinate frames. Left depicts the front right leg of the robot. Right shows the full body of the robot.

The agent selects normalised foot positions between -1 and 1 as a product of the neural network structure described in Section 3.6. These foot positions are scaled to coordinates in the robot's leg frame, specifically from each hip location. This concept is illustrated in Fig 3. The scaling factors for the agent's output are:

$$x_{foot} \in [-0.2, 0.2] \ \ m$$
$$y_{foot} \in [-0.05, 0.05] \ \ m$$
$$z_{foot} \in [-0.33, -0.15] \ \ m$$

where the scaling factors are defined for all the robot legs and all combinations of these coordinates result in achievable foot positions. The coordinate frame of the robot's hip and foot is illustrated in Fig 3. The scaled foot positions for each leg are then tracked using Cartesian PD control with:

$$\boldsymbol{\tau} = \boldsymbol{J}(\boldsymbol{q})^T \left[\boldsymbol{K}_p(\boldsymbol{p}_d - \boldsymbol{p}) - \boldsymbol{K}_d(\boldsymbol{v})\right] \quad (1)$$

where $\boldsymbol{J}(\boldsymbol{q})$ is the foot Jacobian at joint configuration $\boldsymbol{q}$, $\boldsymbol{K}_p$ and $\boldsymbol{K}_d$ are the diagonal matrices of proportional and derivative gains respectively, $\boldsymbol{p}_d$ is the desired foot location provided by the DRL agent, and $\boldsymbol{p}$ and $\boldsymbol{v}$ are the current foot locations and velocities in the leg frame. The values of $\boldsymbol{K}_p$ and $\boldsymbol{K}_d$ were selected to be 500 and 10 $\left(\boldsymbol{K}_p = 500\,\boldsymbol{I}_3, \boldsymbol{K}_d = 10\,\boldsymbol{I}_3\right)$ in agreement with similar research [11]. The foot positions are selected by the policy at 200Hz; however, they are tracked by the PD control at 1 kHz. This ensures that the robot has sufficient time to move its feet into the desired positions before a new position is supplied. Furthermore, the slower foot position selection aids in smoothing the robot's motions.

3.2 Observation space

The observation space consists of feedback from the robot's proprioception as well as the reinforcement learning command from the previous time step ($S \in \mathbb{R}^{59}$). The agent's action from the previous time step is provided before scaling. Proprioception includes the body's height, orientation quaternion, linear and angular velocities, joint positions and velocities, as well as the foot positions. This information can be measured or estimated from the robot's onboard sensors and is provided in the simulation.

3.3 Reward function

The reward function forms a crucial part of the performance of an agent. Additionally, it can be laborious to design since seemingly small changes can potentially result in large behavioural consequences [18]. The reward function is designed to facilitate straight locomotion at specific speeds while promoting energy efficiency and preventing falls. The reward components are detailed in Table 1 where the total reward is denoted as follows:

$$r = r_{\text{velocity}} + r_{\text{orientation}} + r_{\text{energy}} + r_{\text{alive}} \tag{2}$$

where r_{velocity} rewards forward movement at designated velocities, $r_{\text{orientation}}$ penalises skewness, r_{energy} discourages energy expenditure and r_{alive} provides survival incentive. The energy component r_{energy} is calculated as the instantaneous power output of the 12 motors, determined by summing the product of the torque (τ) and joint velocity $(\dot{q})$ for each motor. This energy reward component is inspired by the work of Fu et al. [10]. Upon each environment reset, the target velocity, (v_{target}), is varied between 0.35 and 0.4 m/s. This velocity represents the desired velocity that the robot should reach to achieve a walking gait. Finally, $\boldsymbol{\omega}$ is the robot body's orientation represented as a quaternion. The robot body's coordinate frame as well as the direction of $v_{base(x)}$ is illustrated in Fig 3.

Table 1. Reward function terms.

Reward Name	Formula	Weight
r_{velocity}	$\lvert v_{base(x)} - v_{target} \rvert$	- 1
$r_{\text{orientation}}$	$\lVert \boldsymbol{\omega} - (0,0,0,1) \rVert$	-0.3
r_{energy}	$\sum \tau^T \dot{q}$	-0.0001
r_{alive}	v_{target}	1.3

3.4 Simulation termination conditions

The termination of episodes helps prevent an agent from exploring and exploiting undesirable states. The most logical termination state that is commonly mentioned in literature is once a number of time steps are exceeded [4, 7, 10, 11]. During training an episode is terminated after 10 seconds, therefore after 2000 time steps. Additionally, an episode is terminated if the robot body exceeds a certain roll or pitch. The roll and pitch were chosen to encourage achievable body positions that prevent falling over while still encouraging exploration. The acceptable roll range was chosen to be [-23°, 23°], while the acceptable pitch range was chosen to be [-30°, 30°] with respect to the horizontal axis.

3.5 Dynamics randomisation

Dynamics randomization, although not the exclusive approach for enhancing sim-to-real transfer, is the primary method employed in research to increase sim-to-real capabilities. This method involves the deliberate variation of environmental conditions to produce policies that are more robust to model discrepancies and external disturbances [12]. In the framework, the

parameters are perturbed upon an environment reset using the ranges supplied in Table 2. As seen in the table, the additional mass corresponds to 20% of the robot base's mass and is applied to the center of mass of this component.

Table 2. Randomised physical parameters and their ranges.

Parameter	Range
Friction coefficient	[0.8, 1]
Maximum motor torque	[28.5, 33.5] (N.m)
Additional body mass	[-0.94, 0.94] (kg)
Center of mass	[-0.15, 0.15] (m)

3.6 Simulation environment

The popular simulation environment, Pybullet [32], along with the Unitree A1 robot platform [28] were used to create a simulation environment for training. To learn a policy, the Stable-Baselines3's [33] implementation of PPO was used with the hyperparameters found in Table 3. The initial values were selected using Optuna [34], an open source hyperparameter optimisation framework. The values generated by Optuna, were then tested using a parametric study and adjusted if the average reward of a trained policy increased. As indicated in the table, both the actor and critic deep neural networks (DNNs) have two hidden layers of 128 hyperbolic tangent (tanh) activated neurons each. This ensures that the actor's output is normalised in the range [-1, 1]. Additionally, tanh is proven to improve PPO performance [35].

The total policy training time was approximately 3 million steps and 16 hours using a Dell Precision 3640 tower with one GPU. The results of this policy are presented in **Fig 4**, which shows the leg swing angles as well as the foot positions.

Table 3. PPO hyperparameters.

Parameter	Value	Parameter	Value
Horizon (T)	4096	GAE parameter (λ)	0.95
Optimiser	Adam	Clipping parameter (ϵ)	0.2
Learning rate	$1 \cdot 10^{-4}$	Value function coefficient	1
Minibatch size	128	Network architecture	[128, 128]
Number of epochs	10	Activation function	tanh
Discount factor (γ)	0.99	-	-

4 Results

This section presents the results of the simulation framework for learning locomotion policies. Furthermore, the effect of various components on the development of policies is

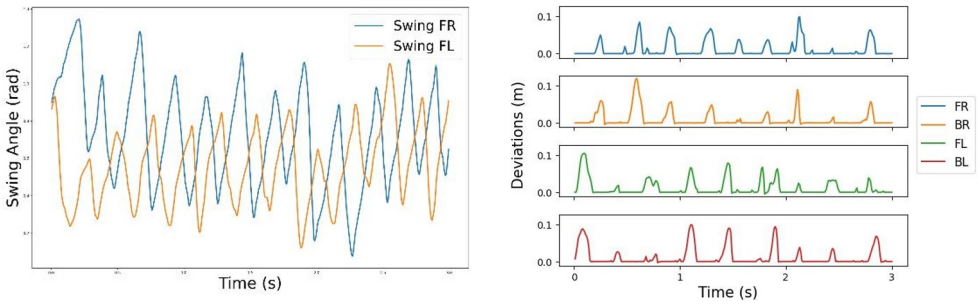

Fig 4. Results of the trained policy. Left shows the front right and front left thigh angles for 3 seconds of walking. Right shows the foot heights for 3 seconds of walking.

discussed to highlight the intricacies of DRL and aid in informing future design decisions. This discussion incorporates the successes of the policy that produced a walking gait as well as insights gained from countless failures.

4.1 Action space choice

To analyse the effectiveness of the task space, training in the task space is compared with training in the joint space. Current research interrogates the optimality and efficiency of the locomotion policies developed using the joint space [8, 11, 20]. These works suggest that adopting the task space for training not only enhances sample efficiency but also yields robust locomotion.

To facilitate a comprehensive comparison, two simulation frameworks that utilise the joint space as the action space were developed. Excluding the actions space, the joint space frameworks are identical to the tasks space framework. In the first joint space framework, the action generated by the agent consisted only of torque commands for each of the motors. The resulting policies performed poorly. The agent selected near-maximum torques that resulted in irregular motion and did not generate a large reward. Ultimately the performance of this framework was so poor that it necessitated the usage of another joint-space framework for a more complete comparison.

A joint-space framework whereby the agent selects target joint angles as its action, is a common, high-performing action space [5, 15, 20]. In the second joint space framework, the agent generates desired joint angles, ($\boldsymbol{q}_d$), which are then tracked using a low-level controller with the following equation:

$$\boldsymbol{\tau} = \boldsymbol{K}_p(\boldsymbol{q}_d - \boldsymbol{q}) - \boldsymbol{K}_d(\dot{\boldsymbol{q}}) \tag{3}$$

where $\boldsymbol{q}$ is the current joint positions, $\dot{\boldsymbol{q}}$ is the current joint velocities, $\boldsymbol{K}_p$ and $\boldsymbol{K}_d$ are the diagonal matrices of proportional and derivative gains respectively, $\boldsymbol{q}_d$ is the desired joint angles provided by the DRL agent. The values of $\boldsymbol{K_p}$ and $\boldsymbol{K_d}$ were selected to be 55 and 0.8 $\left(\boldsymbol{K}_p = 55\,\boldsymbol{I}_3, \boldsymbol{K}_d = 0.8\boldsymbol{I}_3\right)$ in agreement with similar research [10]. This framework performed significantly better than the first joint space framework. However, it performed poorly when compared to the task space framework. Not only was it unable to reach the same

reward after double the number of samples, but it also produced unnatural looking locomotion. A comparison of the training curves is presented in Fig 5. Furthermore, the joint space policy exhibited torques consistently approaching the upper limits of the motor's

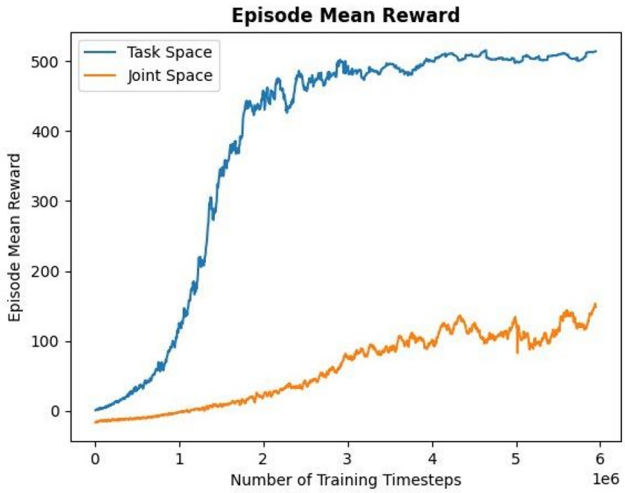

Fig 5. Episode mean reward when training in the proposed action space, the task space, compared to training in the joint space.

capabilities. Conversely, the task space framework produced significantly lower torques. Ultimately the task space framework was able to produce more natural looking locomotion, with more realistic torques in less samples than both joint space frameworks.

4.2 Observation space discussion

The design of the observation space in reinforcement learning is a trade-off: too few observations result in a partially observable controller, while too many observations can lead to an overfitted and brittle controller. Research that utilises the joint space can produce motion with relatively small observation spaces [5, 7, 10]. However, due to training in the joint space, this study required the observation space to be augmented by adding foot positions. This enables the agent to create relationships between the robot's actual foot positions and its outputs, i.e. the desired foot positions. Unlike similar research by [11], the framework was able to achieve locomotion without the addition of the foot velocities. It is hypothesized that this is because the agent can map its output, the scaled foot positions, to the actual foot positions without needing to understand the torque relationship expressed in Equation (1).

To determine the appropriateness of the observation space, two additional policies were trained with varying observation spaces. The first framework included the same observation space as Section 3.2 but additionally included foot contact indicators ($\boldsymbol{s} \in \mathbb{R}^{63}$). The foot contact indicators are Boolean values that signify when each foot is in contact with the ground, with a value of 1 indicating contact. The second framework also utilised the same observation space as Section 3.2. However, the orientation quaternion was substituted for body roll and pitch ($S \in \mathbb{R}^{57}$). All other parameters of the two policies remained consistent with policy described in Section 3. Furthermore, each additional policy was trained five times.

The policy that included the foot contact indicators, resulted in the quadruped attempting to stabilize itself for extended periods before leaning forward and terminating the training episode. This behaviour could be attributed to the specific neural network structure and hyperparameters chosen, suggesting that altering these might yield better performance. Interestingly, training this framework on uneven terrain with varied terrain amplitude, did not yield better behaviour. The agent was trained on fractal terrain similar to the terrain described in Section 4.4. The terrain amplitude was varied between [0, 0.04] m, and all variations caused similar behaviours of the quadruped leaning forward without attempting to walk or step. This leads to the notion that perhaps the addition of foot contact indicators cannot aid in enforcing foot-lifting behaviour but rather the terrain and reward are responsible for the learning of those behaviours in a task space framework.

The second framework was an attempt at reducing the observation space, while leveraging other research, specifically [5] and [10]. This framework led to policies where the robot moved forward for a few steps before its heading became unstable, or it exhibited erratic motions with excessive roll or pitch. This instability might be due to the direct correlation between the orientation quaternion and the reward. Consequently, additional training sessions were conducted by modifying the $r_{\text{orientation}}$ component described in Section 3.3 to:

$$r_{\text{orientation}} = -\alpha \left| {v_{\text{base(y)}}}^2 \right| - \alpha \left| {\omega_{\text{base}(z)}}^2 \right| \tag{4}$$

Where $v_{\text{base(y)}}$ is the robot body's velocity in the y-direction, $\omega_{\text{base}(z)}$ is the body's yaw and α is the weight of the reward. This component was implemented by [10] and [15]. Despite varying (α) to optimize performance, the policy consistently reached a local minimum where the robot pitched downward slowly. The best performance occurred with α = 0.2, where the robot took two steps before quickly pitching downward and terminating the episode. These results suggest that this reward and observation space component may be better suited to joint-based action spaces.

4.3 Reward parameter influence

As described in Section 3.3, each component of the reward function is designed to promote different desirable behaviours. To evaluate their individual contributions, certain elements were tested in isolation. Initially policies were trained only using r_{velocity}. Without r_{alive}, the agent struggles to learn any form of locomotion and typically terminates an episode by tilting forward quickly. This tilting behaviour yields a high reward and therefore there is no incentive for the agent to develop long-term behaviours. Additional policies were trained using the complementary pair of r_{velocity} and r_{alive}. While this combination produced unnatural-looking motion, it did translate to long-term behaviours. Hence, this pair of reward elements perform well in combination to reach long-term velocity goals.

When r_{energy} is added to the reward, the gaits become more stable and natural-looking. However, the robot tends to deviate from a straight path and can exhibit unusual behaviours. These behaviours are discussed further in Section 4.4. Finally, $r_{\text{orientation}}$ was added to keep the robot moving in a straight line. During the tuning of the weights for each of the components, a common local minimum that was encountered was the robot pitching slowly forward. This behaviour is illustrated in Fig 6. Often the robot would pitch forward, attempt to regain stability to prevent episode termination, then pitch forward again to exploit the reward.

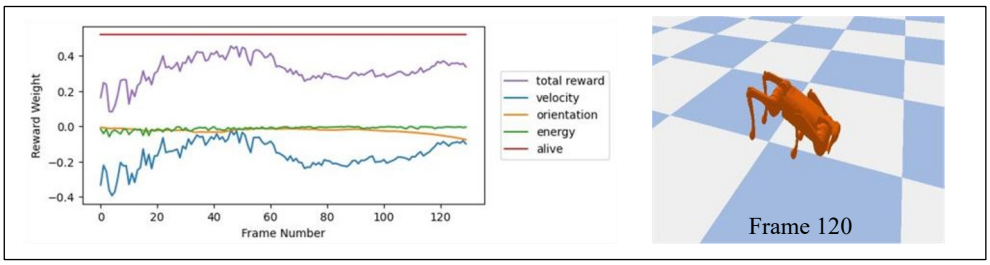

Fig 6. The contribution of each of the reward component as the trained agent performs undesirable behaviour. The image on the right shows the robot pitching drastically forward. This image correlates to frame 120 of the graph. The image on the left shows the variation of the total reward and its various components as the trained agent interacts with the environment. Notably this behaviour still receives a large total reward over the duration of the episode. This is due to the desired speed being achieved by the robot pitching forward, while energy is minimised by the robot not moving its legs. Similar behaviour occurred in frames 35 to 45, as seen by the peak in total reward in the graph.

4.4 Influence of training with fractal terrain

Numerous policies trained on flat terrain produced shuffling motion, where the robot moved forward by moving its feet quickly back-and-forth. This is a result of the energy minimisation reward component. The agent attempts to minimise energy, while still moving forward and staying alive, therefore the quick back-and-forth motion is a local minimum that is easy to exploit while still receiving a large reward. Some policies on flat terrain produced foot-lifting behaviours. However, this resulted in the robot using one or two of the front feet to propel itself forward with large steps, while the back legs simply dragged behind. This behaviour is illustrated in Fig 7. Again, these agents exploited the energy minimisation reward by completely minimising the energy on the back legs. Ultimately, when even terrain and r_{energy} are used in conjunction for training, this can result in leg dragging or high frequency oscillations that are undesirable outside of a simulated environment.

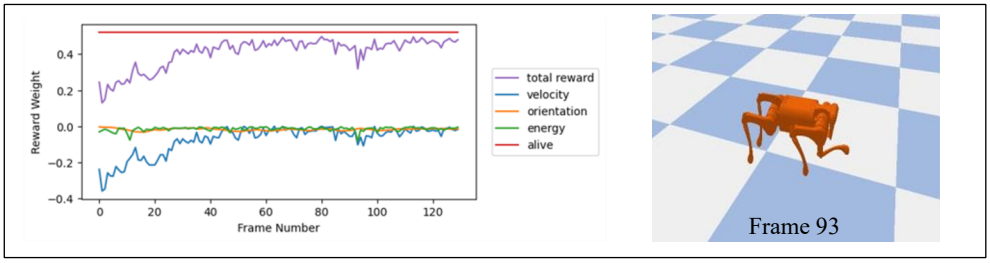

Fig 7. The contribution of each of reward component as the trained agent performs undesirable behaviour. The image on the right shows the agent dragging its back legs while using its front legs to move forward. This image correlates to frame 93 of the graph. The image on the left shows the variation of the total reward and its various components as the trained agent interacts with the environment. Notably this behaviour still receives a large total reward, due to the desired speed being achieved by the robot moving using its front legs, while energy is minimised by dragging its back legs.

The framework results were generated by training on fractal terrain with 2 octaves, an amplitude of 0.02m, a frequency of 10, a gain of 0.25, and a lacunarity of 2. The terrain was randomised upon each environment reset. Incorporating uneven terrain is crucial for enforcing foot clearance behaviours and has played a similar role in other work [10, 11, 15]. The agent was able to lift all of its feet when fractal terrain was introduced into the simulation.

This ensured that the resulting motion appeared more natural and would not cause motor burnout if transferred to real hardware.

4.5 Effect of termination conditions

Terminating episodes when unwanted conditions have been reached can prevent an agent from exploring undesirable behaviours. A common termination condition is when the quadruped's body is below a certain height as implemented by [9] and [14]. This condition was tested with several agents that were trained with constant body height termination conditions between 22 and 28% of leg length. The resulting policies performed poorly as they fell into local minima where each episode terminated quickly due to the agent exploiting a high reward from forward velocity, i.e. the robot tilted forward quickly. Additionally, this prevented the agent from exploring any long-term behaviours. This issue could also be attributed to the choice of algorithm; as PPOs approach of keeping policy updates close together can lead to convergence on local minima. To improve results, the constraint's effect could be incremented over time or integrated into the reward function. A few policies were tested with a height reward instead of a height termination condition. These policies no longer exploited the same local minima but instead had longer-term locomotion and were able to prevent the robot from crouching. However, as evident in Section 3.3, the height reward was removed in future iterations of the final policy as it was no longer found necessary to achieve desired behaviours.

Termination conditions referencing the robot body's roll and pitch are abundant in research. This condition is relatively intuitive as the most undesirable behaviour of an agent would be falling, i.e. excessively rolling, or pitching. Determining the ranges of allowable roll and pitch is challenging. The range of motion should be large enough that a natural amount of pitching can occur, but small enough to prevent the exploration of undesirable states, however, not too small as to result in the agent exploiting local minima. We found that when the roll and pitch constraints were small ($\pm 10°$ and $\pm 20°$) the agent often got stuck in unusual local minima where the robot either pitched quickly forward or performed unnatural motion. The tight constraint on roll and pitch causes an episode to terminate quickly before an agent has a chance to explore and exploit long term reward. Increasing the bounds of both the roll and pitch to the amount described in Section 3.4 helped to prevent the exploitation of these local minima. In particular, the larger ranges provide the agent more time to stabilise itself during early training episodes. Consequently, this enables the agent to be able to explore long-term behaviours in later training episodes once it has learnt how to stabilise itself.

4.6 Influence of dynamics randomisation

Results were generated by fixing the dynamics present in Table 2 to the lowest value in each range. The resulting policies fell into local minima similar to the one shown in Fig 6. With identical dynamics for each episode, the agent exploited these dynamics, leading to an undesirable local minimum of pitching forward slowly. Several other agents were trained with one or two dynamics randomized, but their performance did not improve. Dynamics randomization was included in the simulation primarily to facilitate easier sim-to-real transfer for future work. However, it prevented the agent from exploiting local minima, thereby promoting desirable behaviour in the simulation, and making it essential to produce a good policy.

4.7 The role of policy frequency

When using PyBullet, the frequency at which the agent selects actions and the frequency at which those actions are applied can significantly impact the policy. If the agent selects actions too quickly, it typically results in extremely jerky policies. This is because there is insufficient time for the joints to move into the desired positions before a new action is supplied. While this may not apply to other physics simulators, it is worth noting that work by [10] and [15] also report different frequencies for action selection and action application. The framework performed best when the actions were applied every 1 kHz but they were only selected every 100 – 200 Hz as demonstrated in Fig 2. Moreover, this means that the simulation had a time step of 0.001 which was run 5 - 10 times before an action was selected. Overall, this resulted in smoother motions as well as behaviours that seemed more realistic. The final policy is illustrated in Fig 8 and was generated using the linked Github code.

Fig 8. The final policy walking on flat terrain.

5 Conclusion

The task space is an appropriate choice of action space for locomotion tasks, as it achieves lower torques, and improved sample efficiency when compared to typical joint space DRL frameworks. The detailing of all design parameters can also expedite other research and enable further progress in more complex locomotion tasks. Moreover, the interdependence of details such as observation space, action space, reward function, termination conditions, terrain selection and policy frequency can provide further research questions to advance the study of quadruped locomotion with DRL. Ultimately, numerous factors within a DRL framework contribute to the final policy and no seemingly small details of a framework should be overlooked, as each can potentially provide a meaningful contribution. As part of future work, the policy will be deployed on a physical quadruped. Additionally, the aim is to training future policies in simulation that focus on complex motions such as turning, jumping, climbing stairs, and navigating various terrains.

References

1. M. Posa, C. Cantu, and R. Tedrake, *A direct method for trajectory optimization of rigid bodies through contact*, Int. J. Robot. Res., **33**, pp. 69–81, (2014).
2. M. Bloesch, M. Hutter, M.A. Hoepflinger, S. Leutenegger, C. Gehring, C.D. Remy, and R. Siegwart, *State estimation for legged robots-consistent fusion of leg kinematics and IMU*, Robotics, **17**, pp. 17–24, (2013).
3. T. Erez, K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov, *An integrated system for real-time model predictive control of humanoid robots*, in 2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids), Atlanta, pp. 292–299, (2013).

4. J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, V. Vanhoucke, *Sim-to-Real: Learning Agile Locomotion For Quadruped Robots*, (2018). Accessed: Apr. 10, 2024. [Online]. Available: http://arxiv.org/abs/1804.10332
5. J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, *Learning agile and dynamic motor skills for legged robots*, Sci. Robot., **4**, (2019).
6. T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, *Learning to Walk via Deep Reinforcement Learning*. (2019). Accessed: Feb. 21, 2024. [Online]. Available: http://arxiv.org/abs/1812.11103
7. J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, *Learning quadrupedal locomotion over challenging terrain*, Sci. Robot., **5**, (2020).
8. H. Duan, J. Dao, K. Green, T. Apgar, A. Fern, and J. Hurst, *Learning Task Space Actions for Bipedal Locomotion*, in 2021 IEEE International Conference on Robotics and Automation, ICRA, pp. 1276–1282, (2021).
9. A. Kumar, N. Paul, and S. N. Omkar, *Bipedal Walking Robot using Deep Deterministic Policy Gradient*, (2018). Accessed: Feb. 21, 2024. [Online]. Available: http://arxiv.org/abs/1807.05924
10. Z. Fu, A. Kumar, J. Malik, and D. Pathak, *Minimizing Energy Consumption Leads to the Emergence of Gaits in Legged Robots*, (2021). Accessed: Oct. 23, 2023. [Online]. Available: http://arxiv.org/abs/2111.01674
11. G. Bellegarda, Y. Chen, Z. Liu, and Q. Nguyen, *Robust High-Speed Running for Quadruped Robots via Deep Reinforcement Learning*, in 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, pp. 10364–10370, (2022).
12. W. Zhao, J.P. Queralta, and T. Westerlund, *Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey*, in 2020 IEEE Symposium Series on Computational Intelligence, SSCI, pp. 737–744, (2020).
13. W. Zhu and A. Rosendo, *PSTO: Learning Energy-Efficient Locomotion for Quadruped Robots*, Machines, **10**, p. 185, (2022).
14. A. Kumar, Z. Fu, D. Pathak, and J. Malik, *RMA: Rapid Motor Adaptation for Legged Robots*, (2021). Accessed: Apr. 30, 2024. [Online]. Available: http://arxiv.org/abs/2107.04034
15. Z. Zhuang, Z. Fu, J. Wang, C. Atkeson, S. Schwertfeger, C. Finn, and H. Zhao, *Robot Parkour Learning*, (2023). Accessed: Oct. 23, 2023. [Online]. Available: http://arxiv.org/abs/2309.05665
16. T. He, C. Zhang, W. Xiao, G. He, C. Liu, and G. Shi, *Agile But Safe: Learning Collision-Free High-Speed Legged Locomotion*, (2024). Accessed: Apr. 30, 2024. [Online]. Available: http://arxiv.org/abs/2401.17583
17. P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, *Deep Reinforcement Learning That Matters*, In Proceedings of the AAAI conference on artificial intelligence, **32**, (2018).
18. H. Zhang, L. He, and D. Wang, *Deep reinforcement learning for real-world quadrupedal locomotion: a comprehensive review*, Intell. Robot., **2**, pp. 275–297, (2022).
19. P. Varin, L. Grossman, and S. Kuindersma, *A comparison of action spaces for learning manipulation tasks*, in 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, (2019).

20. X. B. Peng and M. Van De Panne, *Learning locomotion skills using DeepRL: does the choice of action space matter?*, in Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation, pp. 1–13, (2017).
21. G. Bellegarda, C. Nguyen, and Q. Nguyen, *Robust Quadruped Jumping via Deep Reinforcement Learning*, (2023). Accessed: Feb. 21, 2024. [Online]. Available: http://arxiv.org/abs/2011.07089
22. G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, *OpenAI Gym*, (2016). Accessed: Jan. 21, 2024. [Online]. Available: http://arxiv.org/abs/2401.17583
23. R. Martín-Martín, M. A. Lee, R. Gardner, S. Savarese, J. Bohg, and A. Garg, *Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks*, in 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, pp. 1010–1017, (2019).
24. S. Ha, P. Xu, Z. Tan, S. Levine, and J. Tan, *Learning to Walk in the Real World with Minimal Human Effort*, (2020). Accessed: May 06, 2024. [Online]. Available: http://arxiv.org/abs/2002.08550
25. M. Hutter *et al.*, *ANYmal - a highly mobile and dynamic quadrupedal robot*, in 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, pp. 38–44, (2016).
26. G. Kenneally, A. De, and D. E. Koditschek, *Design Principles for a Family of Direct-Drive Legged Robots*, IEEE Robot. Autom. Lett., **1**, pp. 900–907, (2016).
27. L. Han, Q. Zhu, J. Sheng, C. Zhang, T. Li, Y. Zhang, H. Zhang, Y. Liu, C. Zhou, R. Zhao, and J. Li, *Lifelike Agility and Play on Quadrupedal Robots using Reinforcement Learning and Generative Pre-trained Models*, (2023). Accessed: May 06, 2024. [Online]. Available: http://arxiv.org/abs/2308.15143
28. Unitree, *A1 - highly integrated, pushing* limits, (2016 – 2024). Accessed: Mar. 08, 2024. [Online]. Available: https://m.unitree.com
29. R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*. in Adaptive computation and machine learning. Cambridge, Mass: MIT Press, (1998).
30. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal Policy Optimization Algorithms*, (2017). Accessed: Feb. 05, 2024. [Online]. Available: http://arxiv.org/abs/1707.06347
31. J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, *High-Dimensional Continuous Control Using Generalized Advantage Estimation*, (2018). Accessed: Feb. 05, 2024. [Online]. Available: http://arxiv.org/abs/1506.02438
32. E. Coumans and Y. Bai, *PyBullet, a Python module for physics simulation for games, robotics and machine learning*, (2016 - 2024). [Online]. Available: http://pybullet.org
33. A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, *Stable-Baselines3: Reliable Reinforcement Learning Implementations*, J. Mach. Learn. Res., **22**, pp. 1–8, (2021).
34. T. Akiba, S. Sano, T. Yanase, T Ohta, and M. Koyama, *Optuna: A next-generation hyperparameter optimisation framework*, (2019). [Online]. Available: http://arxiv.org/abs/1907.10902
35. Y. Duan, X. Chen, R. Houthooft, J. Schulman, P. Abbeel, *Benchmarking Deep Reinforcement Learning for Continuous Control,* (2016). Accessed: May 03, 2024. [Online]. Available: arXiv:1604.06778