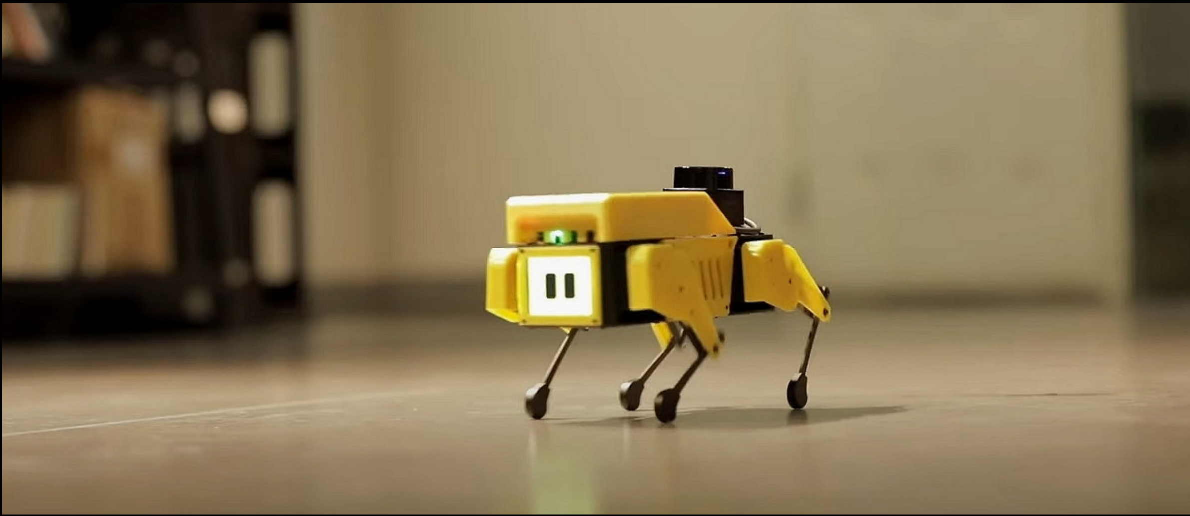
A Soft-Actor-Critic approach to quadruped locomotion



Gijs de Jong

Layout: typeset by the author using LaTeX.
Cover illustration: Mangdang Robotics

A Soft-Actor-Critic approach to quadruped locomotion

Gijs de Jong
13130811

Bachelor thesis
Credits: 18 EC

Bachelor *Kunstmatige Intelligentie*

University of Amsterdam
Faculty of Science
Science Park 900
1098 XH Amsterdam

Supervisor
dr. A. Visser

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 900
1098 XH Amsterdam

June 30, 2023

Abstract

Quadruped robots offer a promising alternative to their wheeled counterparts due to their ability to navigate diverse terrains and overcome various obstacles. However, the control problem for legged robots is challenging as it requires precise control of actuators and good coordination between all four legs. With the advent of machine learning, learning-based methods have shown promising results in both simulation and real-world robots. This work investigates the application of learning-based methods, specifically the Soft Actor-Critic (SAC) and Augmented Random Search (ARS) algorithms, for gait generation. The study focuses on the mini-Pupper 2 robot, a recently developed platform by Mangdang Robotics. By replicating experiments performed in a previous study, we demonstrate the generalisability of the framework and learning methods. Our results show that these techniques successfully generalise to different robot platforms and both SAC and ARS can effectively be used to learn legged locomotion that provides a robust and versatile solution for navigating diverse terrains.

Contents

1 Introduction **3**

2 Related Work **6**

3 Theoretical Background **7**
3.1 Quadruped Gait Generators 7
3.2 Reinforcement Learning Preliminaries 8
3.2.1 Markov Decision Process 8
3.3 Soft Actor-Critic 9
3.3.1 Maximum Entropy Reinforcement Learning 9
3.3.2 Soft Actor-Critic 10

4 Method **12**
4.1 Problem statement 12
4.2 Dynamics and Domain Randomized Gait Modulation with Bezier Curves 12
4.2.1 Bezier Curve Gait 13
4.2.2 Gait Modulation 13
4.2.3 Dynamics and Domain Randomisation 14
4.3 Augmented Random Search 14
4.3.1 Random Search 15
4.3.2 Augmentations 15
4.4 SAC for learning legged locomotion 16
4.5 The robot 17
4.5.1 Physical robot 17
4.5.2 Simulation 17

5 Experiments and Results **18**
5.1 Experiment setup 18
5.1.1 Simulated environment 18
5.1.2 Reward function 20

5.1.3 Parameters 20
5.2 Experiments 20
5.2.1 Stable gait 20
5.2.2 Fast gait 21
5.3 Results 22
5.3.1 Stable gait 22
5.3.2 Fast gait 23

6 Conclusion **26**

7 Discussion **27**

Chapter 1

Introduction

Legged robots are a promising alternative to wheeled robots due to their distinct advantages [1]. These robots, with the small contact area of their feet, are better at navigating diverse terrains and overcoming various obstacles. However, locomotion for legged robots is a challenging control problem as it requires precise control of actuators combined with good coordination between all four legs. With the recent advancements in machine learning, learning-based methods have shown promising results in both simulation and real world robots [2]–[4].

Research into legged robotic locomotion has been primarily done in simulation, because of the high cost and complexity associated with real-world electronics. Training with a real-world robot is also significantly slower as simulations can simply reset whenever the robot falls over, but real robots need to be manually reset. Additionally, locomotion learned in simulation does not necessarily transfer to a real robot seamlessly due to aspects of reality that are difficult to model accurately in simulation. For instance, real-world physics are comprised of complex interactions between objects such as friction, collisions, deformations and contact forces making them hard to simulate accurately. Secondly, actuator dynamics are also difficult to model due to latency in the physical response, non-linear behaviour meaning their responses are not proportional to the applied input. Lastly, performance of actuators can change over time due to wear. Therefore, to bridge the gap between simulation and reality, researchers often use robot platforms equipped with state-of-the-art actuators and sensors. Examples of such platforms include the ANYmal robot by ANYbotics (Figure 1.1a) and the Spot robot by Boston Dynamics (Figure 1.1b) [5], [6].

(a) ANYmal

(b) Spot

Figure 1.1: Two examples of high-end robots with state-of-the-art actuators and sensors. Left is the ANYmal quadruped robot, developed by ANYbotics. Right is the Spot quadruped robot, developed by Boston Dynamics.

However, these high-end robots tend to be expensive, limiting access primarily to well-funded research groups. Therefore recent developments have introduced more accessible platforms, such as the Stanford Pupper robot released by Stanford in 2021 [7] as seen in Figure 1.2a. This low-cost open-source platform is designed to reduce complexity and cost in order to facilitate undergraduate students in robotics research. Another more accessible platform is the SpotMicroAI[1] robot as seen in Figure 1.2b, this quadruped was designed by Deok-yeon as a low-cost alternative to the Spot robot. Due to the lower cost, these robots have significantly lowered the barriers to testing machine-learning-based control approaches on real world robots [7].

(a) Stanford Pupper

(b) SpotMicroAI

(c) mini-Pupper

Figure 1.2: Three low-cost open-source robot platforms. Left is the Stanford Pupper developed by Kau [7]. Center is the SpotMicroAI robot developed by Deok-yeon and further improved by Rahme *et al.* [8]. Right is the mini-Pupper robot developed by Mangdang Robotics based on the Stanford Pupper.

While these platforms do not come equipped with state-of-the-art actuators and sensors, Rahme *et al.* [8] demonstrated that state-of-the-art actuators and

[1]https://spotmicroai.readthedocs.io/en/latest/

sensors are not required when applying learning-based methods to robot research. By applying the Augmented Random Search algorithm to learn a policy and randomising the domain and dynamics during the training process, Rahme achieved promising results in dynamic quadruped gaits on the SpotMicroAI robot [8]. Building on this study Eshuijs [9] applied a Soft Actor-Critic learning approach to the framework proposed by Rahme, also showcasing promising results.

In this thesis, we explore whether the D^{2}GMBC techniques explored by Eshuijs [9] generalise to other quadruped robots. Specifically, we will apply these techniques to the recently released mini-Pupper 2 robot[2], developed by Mangdang Robotics. By replicating the experiments performed in the study by Eshuijs [9], we show the generalisability of the framework and learning methods.

[2]https://github.com/mangdangroboticsclub/mini_pupper_2_bsp

Chapter 2

Related Work

Over the last three decades with advancements in computing and mechanics, the quadruped robots have been getting increasingly more advanced [10]. Concurrently the rapid advancements in Artificial Intelligence (AI) have given rise to exciting new methods of learning robot gaits.

In 1999, Hornby *et al.* [11] introduced a technique for the autonomous evolution of a dynamic gait in the Sony Quadruped Robot. The robot's gait was evaluated using its digital camera and infrared sensors. It was discovered that the evolved gaits exhibited a higher traversal speed compared to those developed manually.

A few years later Kohl *et al.* [2] presented a machine-learning approach to optimising a quadruped gait. They demonstrated that by using policy gradient reinforcement learning to find a set of parameters, they significantly outperformed a variety of existing hand-coded solutions. Specifically Kohl *et al.* [2] uses reinforcement learning to tune a parameterised gait for the Sony Aibo ERS-210 robot. While this resulted in a fast gait it resulted in unsteady camera motions which degrade the robot's visual capabilities [12]. To address this issue, Saggar *et al.* [12] presented an implementation of the policy gradient algorithm with a learning objective that optimises for both speed and stability. The resulting gait was reasonable fast but considerably more stable compared to the gait developed by Kohl *et al.* [2].

The emergence of complex behaviours in reinforcement learning agents has been explored by Heess *et al.* [13]. Their work suggests that training agents in rich environments, can facilitate the learning of complex behaviours without relying on carefully designed reward functions that incentivise specific behaviours. In context of locomotion, Heess *et al.* [13] demonstrated this principle by training simulated bodies on a variety of challenging terrains and obstacles. By exposing the agents to diverse environmental conditions, the agents were able to learn robust and adaptive locomotion skills that could handle unseen terrain types.

Chapter 3

Theoretical Background

This chapter provides a concise introduction to quadruped gait generation and reinforcement learning. Additionally, a brief introduction is given to the Soft Actor-Critic learning algorithm.

3.1 Quadruped Gait Generators

A quadruped gait refers to the pattern of movement exhibited by a four-legged robot. Typically a gait consists of two phases, *swing* and *stance*. During the *swing* phase, the foot moves through the air to its next position and during the *stance* phase, the foot contacts the ground and moves the robot using ground reaction forces. The gait alternates ground contact between the diagonal legs as seen in Figure 3.1. Here, the front left (FL) and back right (BR) legs initiate in the stance phase, meanwhile the front right (FR) and back left (BL) legs start off in the swing phase.

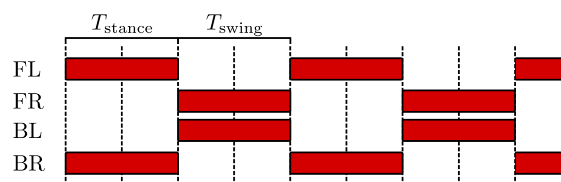


Figure 3.1: Leg phases for trotting from [8], where red is the *stance* phase.

Generating this gait requires creating coordinated limb movements that ensure stability, energy efficiency, and adaptability to various terrains. This process requires the integration of sensor feedback, control algorithms and bio-mechanical principles to generate robust and versatile gaits.

Methods for finding suitable gaits for legged locomotion can be broadly classified into methods that require modeling and tuning [14], evolutionary approaches [11], [15] and learning approaches [2] [4]. Our focus lies on a learning-based method proposed by Rahme *et al.* [8], which combines an extended version of the Bezier curve-based gait generator initially proposed by [14] with reinforcement learning techniques. A detailed explanation of this technique can be found in Chapter 4

3.2 Reinforcement Learning Preliminaries

Reinforcement learning (RL) is a machine learning technique that focuses on training agents to make optimal decisions and take actions in an environment to maximise a cumulative reward. In RL the agent interacts with the environment by observing its current state and taking actions based on that state, the environment then responds with a reward or penalty based on the chosen action. The objective of the agent is to learn a policy, denoted as π. This policy is a function mapping from states to actions, and it is optimised to maximise the expected cumulative reward over time.

Throughout the RL training process, the agent implements an exploration-exploitation strategy. This strategy seeks an equilibrium between investigating novel actions and capitalising on established actions previously associated with high rewards. To accomplish this, various techniques could be used such as value functions [16] and Q-learning [17], both of which enable the agent to forecast the potential rewards of specific actions, thereby facilitating more informed decision-making processes.

To formalise this decision-making problem and effectively handle the complex dynamics of environments, RL often relies on describing the problem with mathematical structures. One such structure commonly used in RL is the Markov Decision Process (MDP).

3.2.1 Markov Decision Process

Markov Decision Processes are a widely-used mathematical approach for modelling decision-making problems in RL. MPDs are built on the concept of an environment consisting of different states. An agent can perform various actions to transition between these states, and each action has a certain probability of leading to a particular new state. Additionally, the agent receives a reward or penalty associated with each transition.

Formally, an MDP is represented as a tuple $(\mathcal{S}, \mathcal{A}, R(s,a,s'), P(s'|s,a))$. Here, $\mathcal{S}$ is the state space, which comprises all possible states of the system. $\mathcal{A}$ is the action space, encompassing all actions the agent can take. $R(s,a)$ is the reward

function; it denotes the reward (or cost) the agent receives when it transitions from state s performing action a. $P(s'|s,a)$ is the state transition probability function, which indicates the probability that action a in state s will lead to the new state s'. The policy is used by the agent to make decisions and is a mapping from the state space to the action space, $a = \pi(s)$.

The goal of an MDP is to find a policy π, which maximises the expected sum of rewards over time:

$$\sum_{t=0}^{\infty} \mathbb{E}\left[\gamma^t R(s_t, a_t)\right] \tag{3.1}$$

Where γ is a discount factor which determines how much rewards in the distant future should impact the decision, a_t is chosen by the agent $a_t = \pi(s_t)$, using the current policy.

Implementing an MDP requires access to comprehensive state information for decision-making. In simulation, this information is typically readily available. However, in real-world scenarios, obtaining the full state of a robot is often impractical or challenging due to limitations such as noisy or incomplete sensor data.

This is why consider a generalisation of an MDP, a Partially Observable Markov Decision Process (POMDP). With a POMPD, the agent is not given access to the actual state of the environment, but observations that provide partial information about the state.

A POMDP is formally defined by introducing the term $\mathcal{O}$ to the MDP tuple, $(\mathcal{S}, \mathcal{A}, \mathcal{O}, R(s,a), P(s'|s,a))$ where $\mathcal{O}$ is the observation space, containing a set of all possible observations the agent can receive, and unlike the policy function in an MDP, the policy in a POMDP is a mapping from the observation space to the actions.

3.3 Soft Actor-Critic

Soft Actor-Critic (SAC) is a maximum entropy reinforcement learning algorithm designed to find optimal policies in environments with high dimensional state and action spaces.

3.3.1 Maximum Entropy Reinforcement Learning

In contrast to standard RL, which maximises the expected sum of rewards, the SAC algorithm proposed by Haarnoja *et al.* [18] considers a more general maximum entropy objective. This objective incorporates the expected entropy of the policy,

which favours stochastic policies:

$$\sum_{t=0}^{\infty} \mathbb{E}\left[\gamma^t R(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)\right] \tag{3.2}$$

Where $\mathcal{H}$ is the entropy and α denotes the temperature parameter which determines the relative importance of the entropy term against the reward, thus controlling the stochasticity of the optimal policy.

This augmented objective has the advantage that the policy is incentivised to explore more widely during the training process, while giving up on clearly unpromising avenues. This is shown to considerably improve the learning speed over methods that optimise for the conventional RL objective [19].

3.3.2 Soft Actor-Critic

Building on the principles of Maximum Entropy Reinforcement Learning, the SAC algorithm employs an actor-critic architecture which consists of an actor and a critic. The actor, also known as the policy takes the current state of the environment as input and produces a continuous action as output. The critic, also known as the value function evaluates the quality of the actions taken by the actor by approximating the reward function. To learn and update the policy, typically policy iteration is used, which alternates between a policy evaluation step and a policy improvement step. During the policy evaluation step, the value function is computed for the current policy. After which, during the policy improvement step, this new value function is used to optimise the policy.

During the training process, the algorithm steps through the environment by using the current policy and stores each step tuple (s_t, a_t, s_{t+1}, r_t) in the replay buffer $\mathcal{D}$. This is done because value functions estimate the expected reward associated with particular actions in specific states. By sampling from the replay buffer, we obtain tuples comprising of a state, action, and the corresponding reward, enabling us to compute the error and update the value functions through gradient descent [20]. This makes the SAC algorithm significantly more sample efficient than other popular RL algorithms such as TRPO [21] or PPO [22] which require new samples to be collected for each gradient step.

The algorithm as proposed by Haarnoja *et al.* [18] uses a parameterised state value function $V_\psi(s_t)$ and a soft Q-function $Q_\theta(s_t, a_t)$ as critic. The state value function represents the value of being in a state, thus providing an estimate of how good a state is for the agent. The soft Q-function is used to estimate the reward the agent can achieve by choosing action a_t in a given state s_t.

The actor is a tractable policy $\pi_\phi(a_t|s_t)$.

The value function $V_\psi(s_t)$ is trained to minimise the squared residual error objective in Equation 3.3.

$$J_V(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\frac{1}{2} (V_\psi(s_t) - \mathbb{E}_{a_t \sim \pi_\phi}[Q_\theta(s_t, a_t) - \log \pi_\phi(a_t|s_t)])^2 \right] \tag{3.3}$$

Where a state s_t is sampled from the replay buffer $\mathcal{D}$. a_t is sampled from the action distribution given by the current policy π_ϕ. The soft Q-function is an approximation of the reward function, and can therefore be used to compute the expected value achieved by taking action a_t under the current policy.

The soft Q-function is trained by minimising the squared error between the current state-action value and the expected value. The expected value is based on a sample drawn from the replay buffer, which consists of the actual reward received and the value of the next state, as per the current policy:

$$J_Q(\theta) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}} \left[(Q_\theta(s_t, a_t) - (r_t + \gamma V_\psi(s_{t+1})))^2 \right] \tag{3.4}$$

The policy is learned by minimising the KL distance between the action distribution and the state-action distribution, for a sampled mini-batch of states:

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}, a_t \sim \pi_\phi} [\log \pi_\phi(a_t|s_t) - Q_\theta(s_t, a_t)] \tag{3.5}$$

In this case, the action batch is sampled from the current policy, which means the loss cannot be back-propagated to update ϕ. However, this problem is mitigated by employing the reparameterisation trick, as proposed by [23]. Here, the stochasticity of the sampling arises from an externally sampled noise variable, rather than from the action batch itself. Thus, the action outputted by the policy is obtained in a deterministic manner.

Chapter 4

Method

4.1 Problem statement

The locomotion task of a quadruped robot can be formulated as a POMDP. Which is formulated by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, R(s,a), P(s'|s,a))$ as described in Chapter 3. The state space encompasses all possible permutations of the robot's state. This includes its position within the environment, orientation, and velocity.

The action space comprises all feasible actions that the robot can undertake. Each action corresponds to a specific set of control inputs for the GMBC-gait system, as described in the next section. These control inputs dictate the desired leg movements and coordination, enabling the robot to execute different gaits and adapt its locomotion strategy.

The observation space consists of a set of all possible observations that can be made by the robot. Each observation includes data collected from the robot's Inertial Measurement Unit (IMU) in its current state. By limiting the observations to the IMU data the gap between simulation and reality is minimised [8].

We approach this problem, by training an RL agent to learn a policy π, which is parameterised by learnable parameters ϕ. The objective of this policy is to select the best action for each state to maximise the expected reward. Therefore, for each observation $o \in \mathcal{O}$, the agent selects an action a.

4.2 Dynamics and Domain Randomized Gait Modulation with Bezier Curves

At the core of their method lies the Dynamics and Domain Randomized Gait Modulation with Bezier Curves (D^{2}GMBC) framework as proposed by Rahme *et al.* [8]. The framework builds upon previous work by Hyun *et al.* [14], and

uses an extended and open-loop variation of the Bezier curve gaits. They achieve transverse, lateral and rotational motion by combining multiple 2D gaits into a single 3D gait.

4.2.1 Bezier Curve Gait

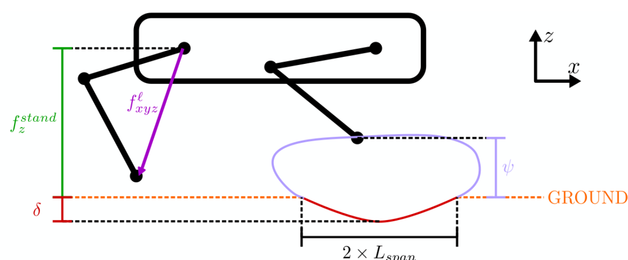


Figure 4.1: Schematic of foot placement adapted from Rahme *et al.* [8].

A gait trajectory, as illustrated by the purple curve in Figure 4.1, is a closed curve followed by a foot during locomotion. The trajectory is parameterised by a phase variable $S(t) \in [0, 2)$. Specifically, the leg is in stance when $S(t) \in [0, 1)$, and it is in swing when $S(t) \in [1, 2)$. In their work, Rahme *et al.* [8] use a trajectory that includes a Bezier curve during the swing phase and a sinusoidal curve during the stance phase.

The trajectory is controlled using three control inputs: $\zeta = \begin{bmatrix} \rho & \varpi & L_{\text{span}} \end{bmatrix}$. Here, ρ represents the rotation angle of the trajectory relative to the robot's forward direction within its frame, ranging from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$. The variable ϖ denotes the robot's yaw velocity, while L_{span} corresponds to half of the stride length. Additionally, the trajectory also depends on curve parameters $\beta = \begin{bmatrix} \psi & \delta \end{bmatrix}$. Here, ψ shapes the Bezier curve trajectory used during the swing phase. Concurrently, δ controls the shape of the sinusoidal curve trajectory applied during the stance phase.

By manipulating these control inputs, planar trajectories are converted into 3D foot-position trajectories which is then converted into a frame relative to each leg's rest position to get the final foot trajectory for leg l, denoted as f^l_{xyz} in Figure 4.1.

4.2.2 Gait Modulation

The generated trajectory is further influenced by a policy $\pi_\phi(o_t)$, where ϕ represents the learnable parameters. This policy takes the observation o_t of the robot's current state as input and produces a delta value Δf_{xyz} that serves to modulate the foot positions generated by the Bezier curve generator. Additionally, the policy also outputs parameters β for the Bezier curve generator, which is then used

to generate the foot positions f_{xyz}. These two values are then added together, resulting in the final foot positions that are used to control the leg motors of the robot as illustrated in figure 4.2.

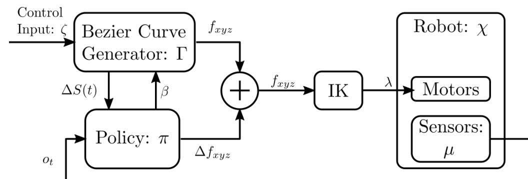


Figure 4.2: GMBC System diagram from Rahme *et al.* [8]

4.2.3 Dynamics and Domain Randomisation

To promote the transferability of the learned policy from simulation to reality, Rahme *et al.* [8] introduces randomisation techniques to address the significant differences between the simulated and real-world dynamics. This involves randomisation the mass of each robot's link and the friction between the feet and the ground, which are crucial factors affecting locomotion. Additionally, domain randomisation is applied by introducing variability in the terrain geometry encountered by the robot during training.

By incorporating these randomisation techniques, the agent is exposed to a wide range of dynamic and environmental conditions during training, leading to the acquisition of a more robust and adaptable gait. Experimental results demonstrate that the learned gait exhibits enhanced stability and achieves greater distances when transferred to real robot platforms. The successful transfer of the learned policy from simulation to reality highlights the effectiveness of these randomisation strategies in improving the generalisation capability of the trained agent.

4.3 Augmented Random Search

Rahme *et al.* [8] originally used the Augmented Random Search (ARS) algorithm to learn the policy used to modulate the gait, achieving decent results. Similar to the study conducted by Eshuijs [9], we adopt the Augmented Random Search (ARS) algorithm as a baseline in this thesis. By employing ARS as a benchmark, we aim to compare and evaluate the performance of the Soft Actor-Critic (Section 4.4), in a consistent manner.

ARS is a method for training static, linear policies aimed at continuous control problems and is as the name suggests based on Random Search but with a few augmentations.

4.3.1 Random Search

Random search is a method which explores the parameter space rather than the action space, which essentially makes it derivative-free optimisation. It works by selecting a direction uniformly at random in the parameter space and then optimising the function along this direction. A primitive version of random search simply computes a finite difference approximation along the random direction and then takes a step along this direction. ARS is based on this exact simple strategy.

4.3.2 Augmentations

ARS introduces three augmentations to the Basic Random Search (BRS) algorithm to improve it, the first improvement consists of scaling its update steps by the standard deviation of the rewards collected at each iteration. The second augment normalises the trained policies using a mean and standard deviation which are updated during the training process. The last enhancement is that ARS is able to drop perturbation directions that yield the least improvement of the reward [24].

ARS applies different rollouts in parallel, where the sampled environment variables remain the same. For training in simulation this means that for each episode, 16 rollouts are performed under the same parameters for the dynamics and the domain. Each rollout contains a slightly different permutation of the previous best policy and by using the same dynamics and domain randomisation the best mutated policy is found. The policy function of ARS consists of a single layer neural network θ that maps from the observation space $\mathcal{O}$ to the action space $\mathcal{A}$. By sampling a set of residual parameters $\Delta\theta_i$ from a zero mean normal distribution with a standard deviation of 0.05, each rollout i gains its own policy function $\overline{\theta}_i = \theta + \Delta\theta_i$. We then use the best policy to update the current policy, using the same learning rate η as [8], 0.03.

By examining the source code[1] of the ARS implementation used in [8] we discover that an additional action filter was used for the the implementation of ARS.

This filter works by slowly interpolating between two actions in order to reduce the risk of abrupt motions during training, however this does result in the agent being unable to get a good set of actions during the start of an episode. To solve this the agent is kept in a fixed position for the first 20 steps of each episode, in order to obtain a decent baseline for the actions values. The interpolation is done as follows:

$$\text{action} = \alpha \cdot \text{prev_action} + (1 - \alpha) \cdot \text{new_action} \tag{4.1}$$

with α being the interpolation parameter, set to 0.7.

[1] `https://github.com/OpenQuadruped/spot_mini_mini`

Unfortunately this change makes comparing the training procedures between SAC and ARS difficult. As the observation space, environment and action space the same for both models are no longer the same. Eshuijs [9] showed that using the action filter with the SAC algorithm prevented learning. Eshuijs [9] explains that the reason for this failure, is the alpha filter breaks the Markov Property [25], which states that each next state is only dependent on the current state, action and observation, and independent of other previous states and actions. The alpha filter makes the model dependent on older actions, thus breaking the Markov Property. Yet, removing the action filter from the ARS implementation resulted in the ARS algorithm not learning.

4.4 SAC for learning legged locomotion

The second RL algorithm used to learn a policy for the GMBC parameters, is an extended version of SAC algorithm as described in Chapter 3, proposed by Haarnoja *et al.* [4] specifically to learn a robot gait using reinforcement learning. This extension eliminates the requirement for manual tuning of the temperature parameter α, which can be labour-intensive due to the unpredictable variability of entropy during the training process.

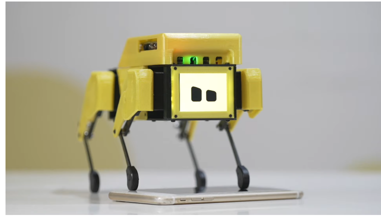
This is achieved by modifying the RL objective, by treating the *expected* entropy as an additional constraint. This results in a slightly altered version of objective function used to learn the policy, in the standard soft actor-critic algorithm (Equation 3.5) by including a dynamic temperature parameter a.

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}, a_t \sim \pi_\phi} \left[\alpha \log \pi_\phi(a_t|s_t) - Q_\theta(s_t, a_t) \right] \tag{4.2}$$

The dynamic temperature parameter is learned exactly like the other parameters in the standard soft actor-critic algorithm, by minimising:

$$J(\alpha) = \mathbb{E}_{s_t \sim \mathcal{D}, a_t \sim \pi_\phi} \left[\alpha \log \pi_\phi(a_t|s_t) - \alpha \mathcal{H} \right] \tag{4.3}$$

4.5 The robot



(a) Physical mini-Pupper 2

(b) Simulated mini-Pupper 2

Figure 4.3: The mini-Pupper 2 robot and the simulated version.

4.5.1 Physical robot

The robot utilized in this thesis is the mini-Pupper 2, a recent creation by Mangdang Robotics, as depicted in Figure 4.3a. It is an enhanced version of Stanford's original mini-Pupper, which was introduced in 2021 [7]. The mini-Pupper 2 incorporates several improvements to enhance its functionality and performance.

One notable enhancement is the upgraded servo system, which enables the robot to collect real-time feedback from its environment. This feedback allows for better control and interaction with the surroundings. Additionally, the mini-Pupper 2 is equipped with an Inertial Measurement Unit (IMU) that includes a gyroscope to measure angular rate and an accelerometer to measure acceleration. This IMU provides crucial data about the robot's orientation and motion.

The mini-Pupper 2 is powered by a Raspberry Pi 4B, supplemented by an additional CM4 compute module, ensuring efficient processing capabilities. It is powered by a 1000mAh battery, providing ample power for its operations.

4.5.2 Simulation

The simulation used in this thesis is based on the the PyBullet physics engine [26]. As the mini-Pupper 2 is an open-source platform, Mangdang Robotics provides the mesh of the robot body alongside a physical description of each component of the robot in URDF format[2], enabling an accurate representation of the robot in the simulator. While the simulation does not perfectly replicate real-world conditions, we acknowledge its limitations. To address this, we employ the aforementioned technique of randomising the robot's dynamics and the environment we train the robot in. By doing so, we aim to develop a policy that exhibits sufficient generality to effectively transfer to the real-world environment.

[2] `http://wiki.ros.org/urdf/XML/model`

Chapter 5

Experiments and Results

In this section the we will discuss the experiments conducted and the achieved results. In order to verify that the techniques explored in Eshuijs [9] generalise to different quadruped robots we replicate the simulation environment used in that study. We then train an RL agent using both SAC and ARS and compare the results to those achieved by Eshuijs [9].

5.1 Experiment setup

In order to compare the results achieved by Eshuijs [9] we aim to replicate the simulation environment as closely as possible. Therefore we use the code[1] provided by Eshuijs [9] and adapt it to the much smaller mini-Pupper robot, as seen in Figure 5.1 .

5.1.1 Simulated environment

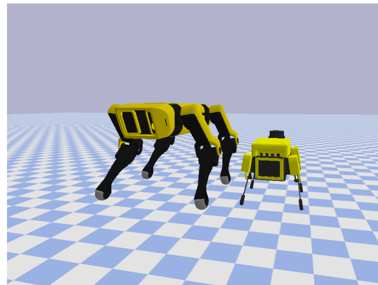
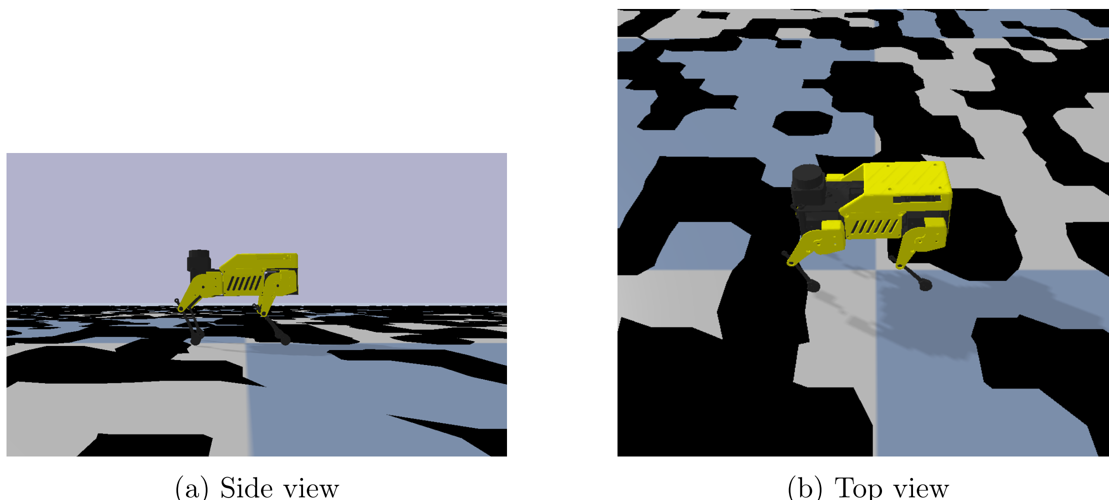


Figure 5.1: SpotMicroAI on the left vs. mini-Pupper 2 on the right.

[1] `https://github.com/watermeleon/spot_micro`

The simulation environment used for the experiments is based on the environment used by Eshuijs [9] but with two modifications. First, the mini-Pupper 2 robot was implemented replacing the SpotMicroAI robot. This was done by adjusting the gait controller slightly, ensuring the correct joints and motors were selected when setting the simulation dynamics. The original inverse kinematics (IK) implementation for the SpotMicro robot did not work for the smaller Pupper robot, due to the significant difference in joint lengths as seen in Figure 5.1. Luckily Mangdang Robotics provides an IK implementation[2], which we altered to work with the gait generator.



(a) Side view (b) Top view

Figure 5.2: Domain randomisation of the simulated environment.

The second modification involves adapting the domain randomisation to suit the smaller size of the Pupper robot. The original terrain randomisation posed challenges for the smaller legs of the Pupper, particularly due to steep inclines. To address this, the parameters used for terrain generation was altered. The height-field used as terrain is generated using a grid-based algorithm, wherein a uniformly distributed random offset is added to each point on the base plane. For the SpotMicroAI, Eshuijs [9] used a range of $[-0.08, 0.08]$. For the mini-Pupper we used a range of $[-0.035, 0.035]$ as seen in Figure 5.2a. Early experimentation showed that ranges higher than the proposed range resulted in the Pupper being unable to climb over some of the terrain.

[2] `https://github.com/mangdangroboticsclub/StanfordQuadruped/blob/mini_pupper/pupper/Kinematics.py`

5.1.2 Reward function

The reward function used in the original report is a simple weighted sum, incorporating multiple metrics to assess the effectiveness of the agent's actions. These metrics include:

- Forward reward: This metric measures the relative distance between the previous position and the current position. It encourages the agent to make progress in the forward direction.
- Orientation reward: This metric imposes a penalty for non-zero pitch and yaw values. It incentivises the agent to maintain a stable and upright orientation.
- Rate reward: This metric penalises non-zero velocity. It discourages excessive or undesired movement speed, promoting smoother and controlled motions.

5.1.3 Parameters

Apart from the number of simulation steps, the original report does not provide specific details regarding the learning rate or the weights assigned to different metrics in the reward function. Despite attempting to use the parameters from the original code, our attempts to replicate the reported results using the SpotMicroAI were unsuccessful. Due to the lack of specific parameter values mentioned in the original report, comparing the results directly proved challenging. However, by iteratively adjusting the parameters, we were able to identify values that yielded a similar learning pattern as observed in the original study. This iterative process involved fine-tuning the parameters to approximate the desired learning behaviour and achieve comparable results. Although not identical, the observed learning patterns allowed for a meaningful comparison between our findings and the outcomes reported in the original report.

5.2 Experiments

5.2.1 Stable gait

The first experiment focused on reproducing the stable gait achieved in the original work by [9]. To attain this, we carefully selected the weights in the reward function to prioritise traversal distance while ensuring gait stability. The objective was to replicate their findings and establish a baseline for comparison.

The final weights are described in Table 5.1

Reward	**Weight**
Forward	500
Orientation	2
Rate	0.03

Table 5.1: Metric weights for the reward function used while training the stable gait.

The forward reward has a relatively high weight in the reward function due to its computation at each simulation step, capturing the relative position difference. On the other hand, the orientation and rate metrics are absolute values and have a larger range. Consequently, to prioritise the forward metric as the primary performance indicator, it requires a significantly higher weight.

By assigning a higher weight to the forward reward, the agent is encouraged to prioritise forward progress in its decision-making process. This weight adjustment allows for a stronger emphasis on achieving desired forward motion, while still considering the orientation and rate metrics to maintain stability and control.

5.2.2 Fast gait

The goal of the second experiment was to learn a gait with a higher traversal speed. To achieve this we took the parameters of the first experiment and increased the weight of the forward reward. The final weights are described in Table 5.2.

Reward	**Weight**
Forward	625
Orientation	0.5
Rate	0.09

Table 5.2: Metric weights for the reward function used while training the fast gait.

By increasing the weight of the forward reward, the agent is motivated to prioritise decisions that result in significant forward movement at each simulation step. Conversely, by reducing the weight assigned to the orientation objective, the agent places less emphasis on maintaining a precise orientation. This intentional adjustment aims to encourage the agent to explore gait patterns that may be less stable but potentially lead to higher traversal speeds.

Additionally, the weight of the rate objective has been significantly increased. This adjustment is intended to incentivise the agent to maintain a consistent non-zero velocity throughout the locomotion task. By assigning a higher weight to the

rate objective, the agent is encouraged to achieve smoother and more controlled forward motion, contributing to overall stability and improved traversal speed.

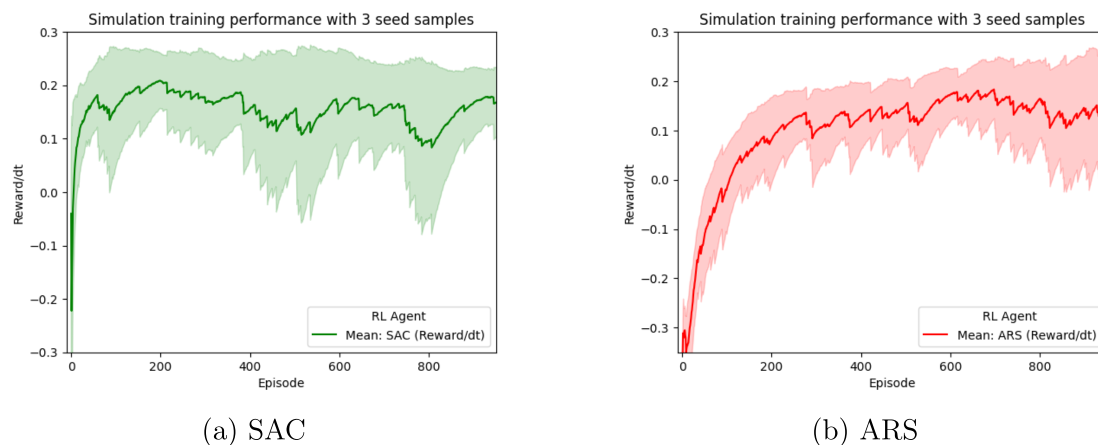
These weight adjustments were made strategically to influence the agent's decision-making process and gait generation. By re-balancing the priorities within the reward function, we aim to steer the agent towards the desired outcome of achieving a higher traversal speed, even if it comes at the expense of some stability.

5.3 Results

5.3.1 Stable gait

The training performance of both the SAC and ARS agents in various environments is illustrated in Figure 5.3. Similar to the figure of the original study, we publish the moving average window over 50 episodes.

The results closely align with those reported in the original study, demonstrating that the SAC agent exhibits notably faster learning compared to the ARS agent. This outcome is consistent with our expectations, as the ARS agent updates its policy only once per episode, whereas the SAC agent initiates updates as soon as the replay buffer contains a sufficient number of samples. This disparity in the update frequency contributes to the SAC agent's accelerated learning rate, enabling it to converge more quickly towards optimal performance. The accelerated learning rate also results in the ARS agent starting out with a slightly lower reward, because the SAC agent is able to adapt before the first episode is complete.



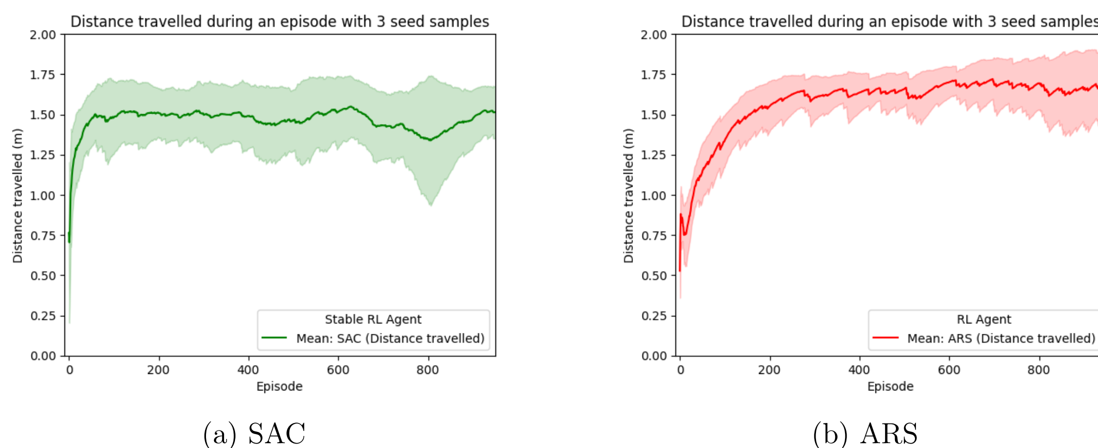
(a) SAC (b) ARS

Figure 5.3: Training curves in simulation for the two agents using D^2 randomisation. Left in green the agent trained with SAC, right in red the agent trained with ARS.

Figure 5.4 illustrates the distance covered during each episode throughout the training process. It is evident that the SAC agent makes rapid progress in terms of distance covered during the initial stages of training, reaching a peak of approximately 1.50 meters per episode. However, as the training progresses, the agent's performance plateaus, and there is limited improvement in the distance travelled.

Upon analysing the training performance in relation to the distance travelled, it becomes apparent that the agent faces challenges with respect to the other metrics. Despite the distance travelled remaining relatively constant, the reward achieved by the agent exhibits fluctuations and occasional drops. This suggests that the agent may struggle with optimising the orientation and rate metrics, potentially impacting its overall performance and hindering further improvement in the distance travelled.

The ARS agent achieves a higher distance traveled, approximately 1.65 meters, compared to the SAC agent. However, it is important to note that the ARS agent's training reward is slightly lower than that of the SAC agent. This suggests that the ARS agent may have incurred higher penalties for orientation and rate metrics, indicating a less stable gait. Although the ARS agent covers more distance, the lower training reward implies that it may have sacrificed stability for speed.



(a) SAC

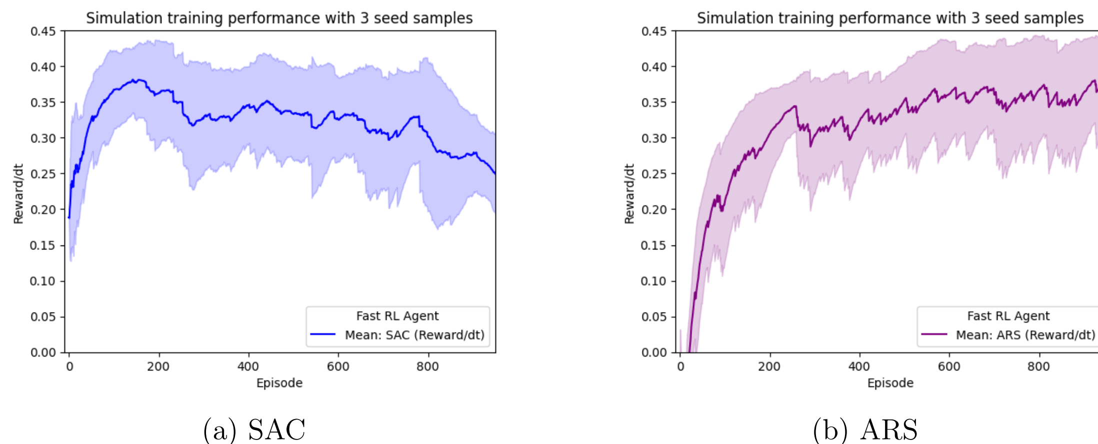
(b) ARS

Figure 5.4: Distance travelled during the training process in simulation for the two agents. Left in green the agent trained with SAC, right in red the agent trained with ARS.

5.3.2 Fast gait

The training performance of both the SAC and ARS agents for the fast gait is illustrated in Figure 5.5. Similar to the results of the stable gait, we publish the moving average window over 50 episodes.

The training performance of the fast gait using the SAC algorithm exhibits a similar pattern to the stable gait, with the agent quickly reaching a high reward and then plateauing. On the other hand, the ARS agent achieves a higher overall reward, but requires a significantly longer number of episodes to reach it.



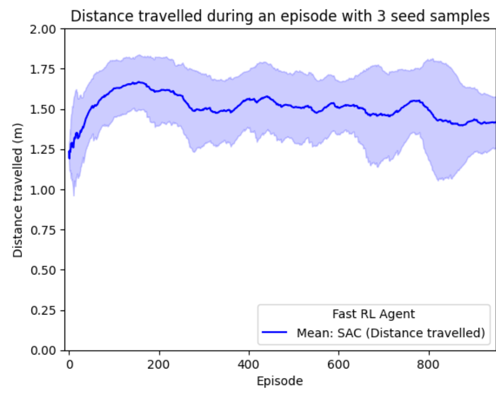
(a) SAC

(b) ARS

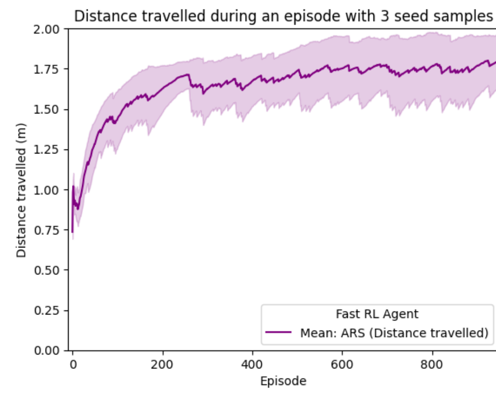
Figure 5.5: Training curves for a fast gait in simulation for the two agents. Left in blue the agent trained with SAC, right in purple the agent trained with ARS.

By comparing the distance travelled during an episode for the fast gait, as depicted in Figure 5.6, to the distance travelled by a stable agent, as shown in Figure 5.4, it becomes evident that the fast gait yields a slightly greater distance covered. Since the episode time steps are consistent, this finding indicates that the fast gait enables a higher traversal speed.

The observed increase in distance travelled implies that the fast gait exhibits a more efficient locomotion pattern, allowing the agent to cover more ground within the same time-frame.



(a) SAC



(b) ARS

Figure 5.6: Distance travelled during the training process in simulation for the two fast agents. Left in blue the agent trained with SAC, right in purple the agent trained with ARS.

Chapter 6

Conclusion

In this thesis, our primary objective was to investigate the generalisability of RL techniques explored by Eshuijs [9]. Specifically, learning policies for the D^2GMBC framework with both SAC and ARS. Although making direct comparisons with the results presented in the original report was challenging due to the absence of specific training parameters, the performance exhibited by both the SAC and ARS agents aligns with our anticipated outcomes. During the simulation training, the SAC agent demonstrated a faster learning rate, while the ARS agent achieved a higher overall reward. This discrepancy is anticipated since the SAC algorithm begins updating the policy as soon as the replay buffer contains a sufficient number of actions, whereas the ARS agent updates its policy once per episode after completing 16 rollouts.

The parameters selected for the fast gait experiment yielded training performance that closely aligns with the findings reported by Eshuijs [9] in their original study. This suggests that the parameters employed in our stable gait experiment were more similar to those utilized in their study. This suggests that the parameters employed in the fast gait experiment were more similar to those used in their study, compared to the ones used in our stable gait experiment.

In conclusion, the results obtained in this study provide evidence that the techniques investigated by Eshuijs [9] have the potential to generalise successfully to different quadruped platforms.

Chapter 7

Discussion

Regrettably, due to unforeseen circumstances, the arrival of the real robot was delayed, preventing us from conducting tests to evaluate the learned locomotion on it. In the original report by Eshuijs [9], a section on the performance of the agents on the real robot was included, which indicated that the gaits learned in simulation performed slightly worse than the baseline.

Future work will involve conducting experiments once the robot arrives, enabling us to test the learned gaits on the physical robot and compare the results with those reported in the original study. This additional evaluation will provide valuable insights into the transferability of the learned gaits from simulation to the real-world setting and further enhance our understanding of the performance of the implemented RL techniques on the physical robot platform.

Upon the arrival of the real robot, another aspect that can be explored is the traversal speed. Initial experiments indicated that increasing the weight of traversal speed in the reward function led to an unstable learning process. This instability is likely attributable to the randomization of the terrain, which introduced excessive bumpiness, thereby hindering the robot's ability to maintain a consistent traversal speed and incurring significant penalties.

In future research, it would be worthwhile to investigate the impact of reducing the noise in the randomised terrain. By creating an environment with smoother terrains, it may be possible to facilitate higher traversal speeds without compromising stability. This avenue of exploration holds potential for optimising the locomotion capabilities of the robot and further refining the reward function to achieve desired performance in terms of traversal speed.

Bibliography

[1] J. Tenreiro Machado and M. Silva, "An overview of legged robots," in *International symposium on mathematical methods in engineering*, 2006, pp. 1–40.

[2] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 3, 2004, 2619–2624 Vol.3.

[3] A. Iscen, K. Caluwaerts, J. Tan, *et al.*, *Policies modulating trajectory generators*, 2019.

[4] T. Haarnoja, A. Zhou, S. Ha, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," *CoRR*, vol. abs/1812.11103, 2018.

[5] J. Hwangbo, J. Lee, A. Dosovitskiy, *et al.*, "Learning agile and dynamic motor skills for legged robots," *CoRR*, vol. abs/1901.08652, 2019.

[6] A. Bouman, M. F. Ginting, N. Alatur, *et al.*, *Autonomous spot: Long-range autonomous exploration of extreme environments with legged locomotion*, 2020.

[7] N. Kau, *Stanford pupper: A low-cost agile quadruped robot for benchmarking and education*, 2022.

[8] M. Rahme, I. Abraham, M. L. Elwin, and T. D. Murphey, "Dynamics and domain randomized gait modulation with bezier curves for sim-to-real legged locomotion," *CoRR*, vol. abs/2010.12070, 2020.

[9] L. Eshuijs, *Spotmicroai: A micro step towards intelligent locomotion*, Feb. 10, 2023.

[10] P. Biswal and P. K. Mohanty, "Development of quadruped walking robots: A review," *Ain Shams Engineering Journal*, vol. 12, no. 2, pp. 2017–2031, 2021.

[11] G. Hornby, M. Fujita, S. Takamura, T. Yamamoto, and O. Hanagata, "Autonomous evolution of gaits with the sony quadruped robot," in *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 2*, 1999, pp. 1297–1304.

[12] M. Saggar, T. D'Silva, N. Kohl, and P. Stone, "Autonomous learning of stable quadruped locomotion," in *RoboCup 2006: Robot Soccer World Cup X 10*, Springer, 2007, pp. 98–109.

[13] N. Heess, D. TB, S. Sriram, *et al.*, *Emergence of locomotion behaviours in rich environments*, 2017.

[14] D. J. Hyun, S. Seok, J. Lee, and S. Kim, "High speed trot-running: Implementation of a hierarchical controller using proprioceptive impedance control on the mit cheetah," *The International Journal of Robotics Research*, vol. 33, no. 11, pp. 1417–1445, 2014.

[15] H. Lipson and E. Malone, "Evolutionary robotics for legged machines: From simulation to physical reality.," in *Intelligent Autonomous Systems 9 - IAS-9, Proceedings of the 9th International Conference on Intelligent Autonomous Systems, University of Tokyo, Tokyo, Japan, March 7-9, 2006*, 2006.

[16] M. L. Littman, "Value-function reinforcement learning in markov games," *Cognitive Systems Research*, vol. 2, no. 1, pp. 55–66, 2001.

[17] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992.

[18] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *CoRR*, vol. abs/1801.01290, 2018.

[19] J. Schulman, X. Chen, and P. Abbeel, "Equivalence between policy gradients and soft q-learning," *arXiv preprint arXiv:1704.06440*, 2017.

[20] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[21] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, *Trust region policy optimization*, 2017.

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

[23] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[24] H. Mania, A. Guy, and B. Recht, "Simple random search provides a competitive approach to reinforcement learning," *CoRR*, vol. abs/1803.07055, 2018.

[25] F. Grabski, “1 - discrete state space markov processes,” in *Semi-Markov Processes: Applications in System Reliability and Maintenance*, F. Grabski, Ed., Elsevier, 2015, pp. 1–17.

[26] E. Coumans and Y. Bai, *Pybullet, a python module for physics simulation for games, robotics and machine learning*, `http://pybullet.org`, 2016–2021.