Learning to walk with a soft actor-critic approach

Gijs de Jong[3]*, Leon Eshuijs[1,2,3]*, and Arnoud Visser[3]

[1] Intelligence Systems Group, Universiteit Utrecht
[2] Computational Linguistics and Text Mining Lab, Vrije Universiteit
[3] Intelligent Robotics Lab, Universiteit van Amsterdam

Abstract. Quadruped robots offer a promising alternative to wheeled robots 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 modulation. The SpotMicroAI and mini-Pupper-2 robots are used as a case study to demonstrate the generalizability of the reinforcement learning algorithms. Afterward, the learned policies of the SpotMicroAI are applied to a physical robot to inspect the sim-to-real gap. Our results show that both SAC and ARS can effectively be used to learn legged locomotion that provides a robust and versatile solution for navigating diverse terrains.

Keywords: reinforcement learning · gait generation · robotics

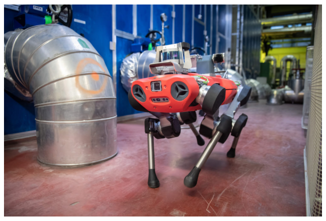
1 Introduction

Legged robotics have the ability to walk through rough terrain, due to their small selection of contact points [5]. 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 [18,15,7].

Until recent years, research on learning-based methods for locomotion has mostly been limited to simulation [10]. One of the primary reasons is the high cost and complexity associated with real-world electronics. Training on real-world robotics is also significantly slower, since they need to be manually reset whenever the robot falls over, whereas simulations can reset automatically [14]. Additionally, the discrepancies between the simulated environment and the real world hinder the transfer of learned behavior, in a problem known as the sim-to-real gap [9]. For instance, real-world physics are comprised of complex interactions between objects such as friction, collisions, deformations, and contact forces

* Equal contribution

making them hard to simulate accurately. Secondly, actuator dynamics are also difficult to model due to latency in the physical response, and non-linear behavior meaning their responses are not proportional to the applied input. Lastly, the 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. Typical examples of such high-end platforms include the ANYmal robot by ANYbotics (Fig. 1a) and the Spot robot by Boston Dynamics (Fig.1b).


(a) ANYmal

(b) Spot

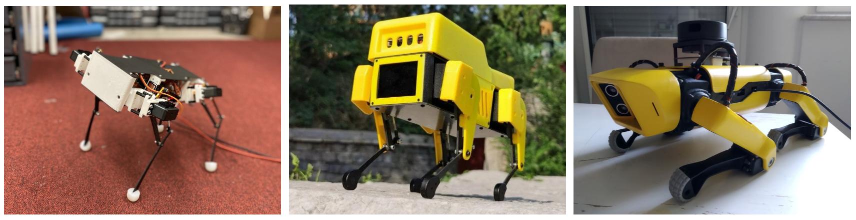
Fig. 1: Two examples of robots with state-of-the-art actuators and sensors.

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 [16] as seen in Fig. 2a. This low-cost open-source platform is designed to reduce complexity and cost, thereby improving the accessibility of robotics research. Importantly, recent work [20] has demonstrated that learning-based models can be robust enough to apply to hobby robotics. By randomizing the dynamics of the robot and the environment in simulation, their approach named D^2-GMBC bridged the sim-to-real gap on a variation of the open-source robotics project *SpotMicroAI*[4], as seen in Fig. 2c. However, their Reinforcement Learning (RL) algorithm, called Augmented Random Search (ARS), consists of only a single-layer Neural Network. To investigate how the D^2-GMBC framework benefits more complex RL algorithms, we apply it using the Soft Actor-Critic (SAC) algorithm and use the mini-Pupper and SpotMicroAI as research platforms in this study.

The primary contribution of this paper is to demonstrate the versatility of the D^2-GMBC learning approach to other Reinforcement Learning algorithms. Moreover, we highlight some of the limitations of the original setup such as the ARS learning scheme and the reward function. The main findings from our experiments are:

– In the simulation, the SAC agent demonstrated competitive performance and improved learning capabilities when deployed on the Pupper platform.

[4] `https://spotmicroai.readthedocs.io/en/latest/`



(a) Stanford Pupper (b) mini-Pupper (c) SpotMicroAI

Fig. 2: Three low-cost open-source robot platforms. At the left is the Stanford Pupper developed by [16]. At the center is the mini-Pupper robot developed by MangDang Robotics based on the Stanford Pupper. At the right is the SpotMicroAI robot developed by Deok-yeon and further improved by [20].

– On the SpotMicroAI platform, the SAC algorithm exhibited faster learning, whereas ARS achieved a higher training reward. However, ARS enjoys certain advantages during training, such as policy updates based on parallel variations and the ability to reset the environment, making direct comparisons of training rewards challenging.
– When tested on the physical SpotMicroAI robot, the SAC agent demonstrated dynamic falling prevention behavior, although neither agent managed to enhance their speed beyond the fixed baseline gait.

2 Related work

In the RoboCup 4-Legged League, machine learning has been used to optimize a quadruped gait [18]. In this work, policy gradient reinforcement learning was used to find a set of parameters, which significantly outperformed a variety of existing manually-tuned solutions. They demonstrated this for the Sony Aibo ERS-210 robot. While the robot achieved a faster gait, it resulted in unsteady camera motions that degraded the robot's visual capabilities [21]. To address this issue, [21] implemented the policy gradient algorithm with a learning objective that optimizes for both speed and stability. The resulting gait was considerably more stable while maintaining a sufficient speed compared to the gait developed by [18].

The emergence of complex behavior in reinforcement learning agents has been explored by Nicolas Heess *et al* [10]. Their work suggests that training agents in rich environments, can facilitate the learning of complex behaviors without relying on carefully designed reward functions. In the context of locomotion, [10] 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 successfully transferred to unseen terrain types.

Such a transfer to unseen terrain is for instance demonstrated by Biao Hu *et al.* [12] for the Doggo quadruped robot. The gait learned with reinforcement learning not only increased the speed on flat ground, but also allowed the robot to move uphill. This approach shows the potential of learning a quadruped gait in simulation, although the authors are not explicit on how the validate the learned gait on the physical robot.

Yet, as Jie Tan *et al.* [22] demonstrated, the reality gap is a major obstacle to applying deep RL in robotics. They showed that the reality gap can be narrowed, but this requires considerable effort to reduce the model discrepancies with comprehensive evaluations to the effect of incorporating information on for instance weight distributions, frictions, and latency. For their Minitaur quadruped robot, they showed that both trotting and galloping can be learned as gait.

3 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. A typical gait alternates ground contact, moving the diagonal legs concurrently, as seen in Fig. 3. Here, the front left (FL) and back right (BR) legs initiate in the stance phase, while the front right (FR) and back left (BL) legs start off in the swing phase.

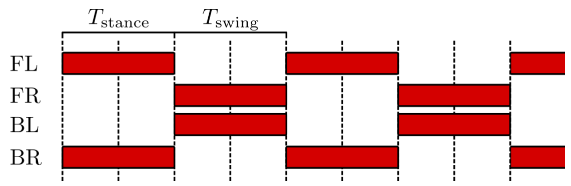


Fig. 3: Leg phases for trotting from [20], where red is the *stance* phase.

Generating this gait requires creating coordinated limb movements that ensure stability, energy efficiency, and adaptability to various terrains [6]. This process requires the integration of sensor feedback, control algorithms, and biomechanical principles to generate robust and versatile gaits.

Methods for finding suitable gaits for legged locomotion can be broadly classified into methods that require handcrafted tuning [13], evolutionary approaches [19,11] and learning-based approaches [18,7]. Our focus lies on a learning-based method proposed by [20], which combines an extended version of the Bezier curve-based gait generator initially proposed by [13] with reinforcement learning techniques. A detailed explanation of this technique can be found in Section 4.

4 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 [20]. The framework builds upon previous work by [13], 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.

4.1 Bezier Curve Gait

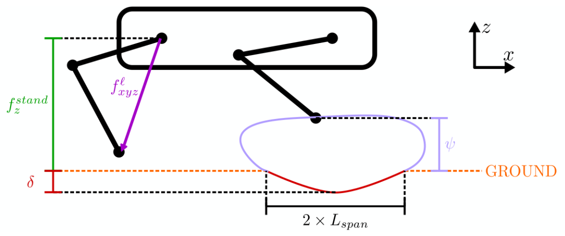


Fig. 4: Schematic of foot placement adapted from [20].

A gait trajectory of one leg is described by the path the foot takes over a closed curve, as illustrated by the purple curve in Fig. 4. The trajectory is parameterized 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 [20], Rahme *et al* use a trajectory that includes a Bezier curve during the swing phase and a sinusoidal curve during the stance phase.

The Bezier curve generator, $\mathit{\Gamma}$, is controlled using three control inputs: $\zeta = \left[\rho \, \overline{\omega} \, L_{\text{span}}\right]$. Here, ρ represents the rotation angle of the trajectory relative to the robot's forward direction within its frame, and ranges from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$. The variable $\overline{\omega}$ denotes the robot's yaw velocity, while L_{span} corresponds to half of the stride length. The two last parameters include ψ, which shapes the trajectory of the swing phase, while δ controls the shape of the sinusoidal curve trajectory applied during the stance phase. The generator outputs $f_{xyz}^{\mathit{\Gamma}}$, consisting of the 3D foot positions of each foot, relative to the body:

$$f_{xyz}^{\mathit{\Gamma}} = \mathit{\Gamma}(S(t), \zeta, \psi, \delta) \tag{1}$$

By manipulating these control inputs, planar trajectories are converted into 3D foot-position trajectories which are then converted into a frame relative to each leg's rest position to get the final output foot trajectory for leg l, denoted as f_{xyz}^{l} in Fig. 4.

4.2 Problem Statement: Gait Modulation

To adaptively modify the gait of the robot during locomotion, the problem is phrased as that of gait modulation which we will first formulate as a Partially Observable Markov Decision Problem (POMDP). A POMDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, r(s,a), P(s\prime|a,s))$, consisting of the state space $\mathcal{S}$, the action space $\mathcal{A}$, the observation space $\mathcal{O}$, a reward function $r(s,a)$ and the transition probability $P(s\prime|a,s)$.

The policy of the model, π as parameterized by the learnable parameters θ, is then tasked to find the best action for each state to maximize the expected reward. For each observation $o \in \mathcal{O}$, the agent takes the action $a \in \mathcal{A}$ based on the policy, formalized as $a = \pi_\theta(o)$.

To adjust the position of the individual legs, the action space contains a 3-dimensional residual position is defined by $\Delta f^l_{xyz} \in \mathbb{R}^3$, resulting in the total residual set Δf_{xyz}. Additionally, the policy also outputs the mentioned parameters ψ and δ for the Bezier curve generator, which are used to generate the foot positions f^{Γ}_{xyz}. After adding the residuals, the final position of all the foot endpoints is:

$$f_{xyz} = f^{\Gamma}_{xyz} + \Delta f_{xyz} \tag{2}$$

4.3 Reward

For the reward and training setup, we follow the procedure of [20] with a few adjustments. To reduce the training computation, we reduce the maximum length of an episode from 5000 steps to 2000 steps, which translates to 20 seconds per episode. An episode is terminated prematurely if the robot hits the ground or exceeded a roll or pitch of 60 degrees. The reward function for each time step as described in the paper of D^2-GMBC is defined as:

$$r_t = \Delta d - 10(|r| + |p|) - 0.03 \sum |\omega| \tag{3}$$

The term Δd is the distance traveled in the forward direction in one step, and the penalty terms for r and p encourage the robot's body to remain perpendicular to the ground. The ω term further promotes a stable walk by punishing any angular motions of the body.

However, the code implementation as provided by the authors[5] uses d in the reward instead of Δd. This difference makes sense since the roll and pitch are many orders of magnitude larger than the traversed distance per step, Δd, which is in meters. The downside of this approach is that the reward per step for a given state becomes infeasible to determine with decent accuracy, since the reward at the end is much higher than at the beginning, and the time is not observed by the agent. To illustrate this problem, consider a speed of 0.2 m/s, the distance part of the reward can differ by 4.0 between the beginning and end state of an episode. This is in the same range as the roll and pitch reward.

[5] `https://github.com/moribots/spot_mini_mini`

For the training of the ARS agent, his discrepancy is not a problem because, as the following sections explain, the policy is updated only by the total reward of an episode. Contrary, the SAC agent updates its policy based on the reward obtained at each step. For better comparison of the RL agents, we train them with the same objective and modify the reward to be:

$$r_t = 100\varDelta d - 10(|r| + |p|) - 0.03 \sum |\omega| \tag{4}$$

This scaling factor was chosen to put the distance reward in the same range as the orientation penalty and resulted in the best convergence on both agents during early experiments.

4.4 Dynamics and Domain ($\mathbf{D^2}$) Randomization

To promote the transferability of the learned policy from simulation to reality, Rahme *et al* [20] combine different randomization techniques in their framework D^2-GMBC to address the significant differences between the simulated and real-world dynamics. This involves randomization of the mass of each robot's link and the friction between the feet and the ground, which are crucial factors affecting locomotion. Additionally, domain randomization is applied by introducing variability in the terrain geometry encountered by the robot during training.

By incorporating both randomization techniques of Rahme *et al* [20], 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 randomization strategies in improving the generalization capability of the trained agent.

5 Augmented Random Search for legged locomotion

Rahme *et al* applied the Augmented Random Search (ARS) algorithm to learn the policy used to modulate the gait, achieving decent results [20]. 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.

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 Randomization, 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 $\varDelta\theta_i$ from a zero mean normal distribution with a standard deviation of 0.05, each rollout i uses a unique 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 $\eta = 0.03$ as [20].

An undocumented feature of ARS implementation is an additional action filter. This filter works by slowly interpolating between two actions in order to reduce the risk of abrupt motions during training, however, this makes it impossible for the agent 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 action values. Unfortunately, this alteration complicates the comparison of training procedures between SAC and ARS due to the difference in their training environment.

Moreover, the requirement of ARS to train using multiple rollouts of the same environment, makes it computationally expensive and impossible to scale to settings where the environment cannot be controlled. Moreover, since ARS is restricted to learning a linear policy, it has an inherent limitation on the policy's complexity.

6 Soft Actor-Critic for learning legged locomotion

An Actor-Critic algorithm is an off-policy method that computes both the policy and the value function over states. The policy defines a distribution over the actions for a given state. The value function $V(s)$ estimates the expected return for each state under the current policy. Similarly, the Q-function estimates the expected return of each state-action pair and relates to the value function by $V(s) = \mathbb{E}_{a\sim\pi(s)} Q_\pi(s, a)$.
In this report, we apply the Soft Actor-Critic (SAC) algorithm as proposed by [8], where the critic is the Q-function. To promote exploration, SAC introduces an extra constraint to the loss function of the value and policy function by maximizing the entropy $\mathcal{H}$ of the policy.

During each step of the training process, the replay buffer $\mathcal{D}$ stores the step tuple $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{r}_t)$. After enough steps have been stored, a batch of steps is sampled from the buffer to update the functions. A common problem with learning a Q-function is the influence of a positivity bias, where the model puts more emphasis on positive rewards than negative rewards. To minimize the positivity bias and improve learning speed, double Q-learning is applied, a method where two Q-functions are trained in parallel, parameterized by μ_1, μ_2. More specifically, we apply the double Q-learning procedure as proposed in [4], by using the minimum value of the two Q-functions for the value gradient. Minimizing the loss:

$$J_Q(\mu_i) = \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t,\mathbf{s}_{t+1},\mathbf{r}_t)\sim\mathcal{D}} \left[\left(Q_{\mu_i}\left(\mathbf{s}_t, \mathbf{a}_t\right) - \left(\mathbf{r}_t + \gamma V_{\mu_1,\mu_2}\left(\mathbf{s}_{t+1}\right)\right)\right)^2 \right] \tag{5}$$

This loss minimizes the squared error between the current state-action value and what it should be according to the minibatch, namely the real reward plus the state value of the next state, according to the current policy. The value function in equation 5 is implicitly defined by the policy and the Q-function as:

$$V_{\mu_1,\mu_2}(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t \sim \pi_\theta} \left[\min_{i \in \{1,2\}} Q_{\mu_i}(\mathbf{s}_t, \mathbf{a}_t) - \alpha \log \pi_\theta (\mathbf{a}_t \mid \mathbf{s}_t) \right] \tag{6}$$

The policy is learned by minimizing the Kullback-Leibner distance between the action distribution and the state-action distribution, for a sampled minibatch of states. This policy loss, as defined below, is the same as the policy loss of the original SAC [8] but uses the explicit temperature parameter α, which relates to the entropy.

$$J_\pi(\theta) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}, \mathbf{a}_t \sim \pi_\theta} \left[\alpha \log \pi_\theta (\mathbf{a}_t \mid \mathbf{s}_t) - \min_{i \in \{1,2\}} Q_{\mu_i}(\mathbf{s}_t, \mathbf{a}_t) \right] \tag{7}$$

Because the action batch is sampled from the policy, the loss could not be back-propagated to update θ. Therefore, the action is sampled using the reparameterization trick [17], where the stochasticity of the sampling arises from an external sampled noise variable. Given this noise variable, the action resulting from the policy is obtained in a deterministic manner. Lastly, the temperature parameter is updated by minimizing:

$$J(\alpha) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}, \mathbf{a}_t \sim \pi_\theta} \left[-\alpha \log \pi_\theta (\mathbf{a}_t \mid \mathbf{s}_t) - \alpha \mathcal{H} \right] \tag{8}$$

7 The robots

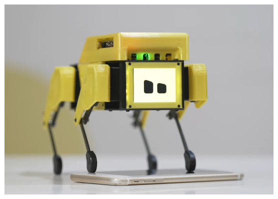
The first robot utilized in this study is the mini-Pupper 2, a recent creation by MangDang Robotics, as depicted in Fig. 5a. It is an enhanced version of Stanford's original mini-Pupper, which was introduced in 2021 [16]. The mini-Pupper 2 incorporates several improvements to enhance its functionality 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.

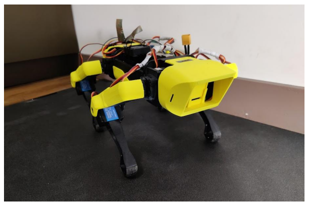
The second robot is a modified open-source hardware design of the SpotMicroAI (see Fig. 5c). The body of the robot is 3D printed from Polylactic Acid (PLA) material except for the feet, which are made of rubber for better shock absorption. To handle the weight of the robot all the servos have a maximum torque of 20kg/cm. Although plenty of schematics of the SpotMicroAI are available online, there is no good solution that merges the Jetson Nano with the actuators and IMU sensor. Through careful examination, a new schematic is created (see [2] for details).

The simulation used in this study is based on the PyBullet physics engine[6]. As the mini-Pupper 2 is an open-source platform, MangDang Robotics provides the

[6] `http://pybullet.org/`


(a) Physical mini-Pupper 2

(b) Simulated mini-Pupper 2


(c) Physical SpotMicroAI

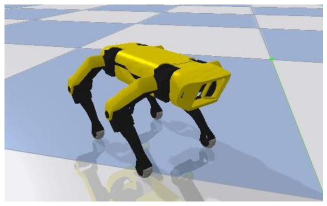

(d) Simulated SpotMicroAI

Fig. 5: The mini-Pupper 2 robot and SpotMicroAI, both real and simulated.

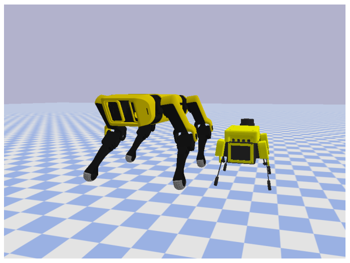


Fig. 6: SpotMicroAI on the left vs. mini-Pupper 2 on the right.

mesh of the robot body alongside a physical description of each component of the robot in URDF format, enabling an accurate representation of the robot in the simulator. To reduce potential adverse effects of inaccurate physical parameters, we employ the aforementioned technique of randomizing the robot's dynamics.

The SpotMicroAI is simulated with the same PyBullet physics engine. The mesh used to 3D print the components is transferred into a 3D model. Additional weight is placed inside the robot to mimic the weight influence of the Jetson Nano and the battery. Because these are estimates, again a domain randomizing technique is used to reduce the dependence on the precise values. By doing so, we aim to develop a policy that exhibits sufficient generality to effectively transfer to the real-world environment.

8 Experiments

In this study, a reinforcement learning agent is trained using both SAC and ARS for both robots, whereafter the results are compared. The Bezier Curve gait provides 3D foot-position trajectories, which can be converted into joint angles for each of the actuators with inverse kinematics. For the mini-Pupper 2, Mangdang Robotics provides an inverse kinematics implementation[7], which was adapted to work with the D^2-GMBC gait generator. Similarly, an inverse kinematics implementation for the SpotMicroAI exists[8] and we make the code for this platform available online[9].

The two robots also differ in size (see Fig. 6), which means that obstacles and slopes are relatively more difficult for the smaller mini-Pupper 2. Therefore, the domain randomization part of the DD^2-GMBC training is adjusted to compensate for the height differences. For the mini-Pupper the offset from the base-plane ranged within $[-0.035, 0.035]$, while the SpotMicroAI was implemented using a range of $[-0.08, 0.08]$. See [1] for more details.

9 Results

The experiments are performed for the two robots in simulation and for the SpotMicroAI also with the real platform. Unfortunately, the kickstarter program from MangDang robotics[10] failed to finalize the design of the mini-Pupper 2 Pro in the timeline advertised at the beginning of the campaign, so the sim-to-real transfer could not yet be demonstrated for this platform. At the conference the learned walking-behavior of both robots [3] will be demonstrated.

9.1 mini-Pupper 2 simulation

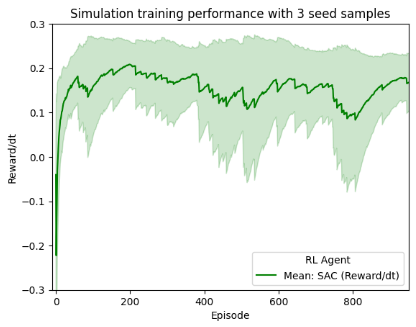
The training performance on the mini-Pupper 2, for both the SAC and ARS agents using a stable gait is illustrated in Fig. 7. The dark lines represent the moving average window over 50 episodes.

The results demonstrate 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. 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 completed.

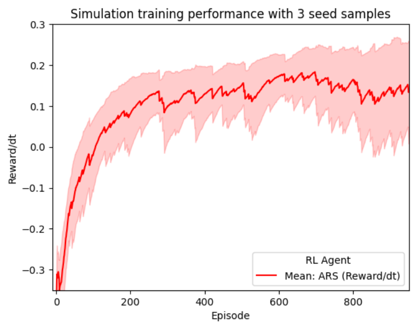
9.2 SpotMicroAI simulation

The training performance on the SpotMicroAI, for both the SAC and ARS agents using a stable gait is illustrated in Fig. 8. Again, the dark line represents

[7] `https://github.com/mangdangroboticsclub/StanfordQuadruped/`
[8] `https://github.com/OpenQuadruped/spot_mini_mini/`
[9] `https://github.com/watermeleon/spot_micro`
[10] https://www.kickstarter.com/projects/336477435/mini-pupper-2-open-source-ros2-robot-kit-for-dreamers/description



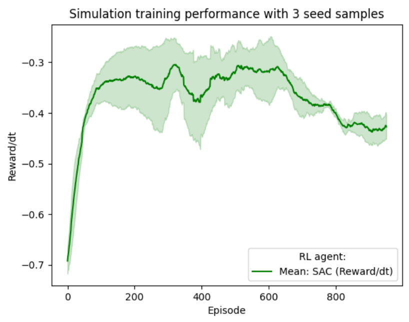
(a) SAC



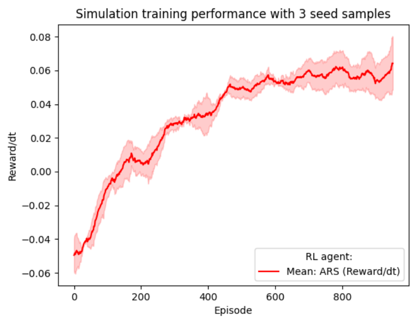
(b) ARS

Fig. 7: Training curves of the mini-Pupper 2 in simulation for the two agents using D^2 Randomization. Left in green the agent trained with SAC, right in red the agent trained with ARS.

the mean of these experiments, and the lighter area represents the standard deviation.



(a) SAC



(b) ARS

Fig. 8: Training curves of the SpotMicroAI in simulation for the two agents using D^2 Randomization. Left in green the agent trained with SAC, right in red the agent trained with ARS.

From the learning curves in Fig. 8 we see that although the starting reward of SAC is much lower, it learns much faster than ARS. However, we can see that ARS achieves a much higher reward than SAC and that the reward of ARS always increases over time. Both this non-decreasing reward and the much higher reward of ARS training are a result of the multiple rollouts. For each of the episodes, one of the 16 rollouts will likely lead to a policy with a higher reward. To estimate the quality of these agents we need to investigate if ARS retains its advantage beyond the benefit of its privileged information provided by the rollouts.

9.3 Sim-to-real transfer

To test if the walking behavior also works when applied to a real robot, the gait learned for the SpotMicroAI in a simulated environment is now transferred to the real SpotMicroAI robot. This is a replication of the experiment performed by Rahme *et al* [20], but now extended with our implementations of both reinforcement learning (RL) algorithms: the ARS and SAC method. The goal of the experiment is to let the robot walk a track of one meter. To reduce the impact of the legs touching the floor, the track consists of a thin flexible mat with a height of 0.5 cm. The robot is tasked to walk this path ten times for each RL agent and for the baseline model which uses a fixed gait. Details about the calibration of the IMU values of the same movement applied to the simulated and the physical robot are provided in Appendix A. Table 1 summarizes the results of the experiments. For each agent, the mean and standard deviation of the results are shown. For each RL agent, one run was considered failed because the robot became stuck in continual erratic behavior that prevented it from going forward.

Table 1: The results of real-world experiments of traversing one meter.

	mean	**standard deviation**	**failed runs**
Baseline	15.20	2.56	0
ARS	15.33	0.94	1
SAC	21.22	1.47	1

From Table 1 we notice that neither of the agents was able to improve the forward walking speed above that of the baseline. When looking at the standard deviation we do note that the ARS agent has a more predictable walking speed, as indicated by the low standard deviation. The SAC agent is significantly slower than the other two and has an average walking speed of 21 seconds per meter compared to the 15 seconds per meter of the other two models. However, the walking speed was only one part of the reward signal, so to evaluate the walking stability we need to inspect the quality of the walk itself.

The quality of the walk can be inspected visually. Three short representative clips of each gait are available[11]. A closer inspection of those clips shows that the baseline implementation demonstrates a suboptimal walk. Even after meticulous calibration and reconfiguration of the servos, the robot falls through its hind legs on multiple occasions, likely due to the weight of the body from the addition of both a Jetson Nano control board and the extra battery.

[11] https://youtu.be/Ji8rXjD60mU

The ARS clip shows adaptive behavior when the robot threatens to fall backward or forward. However, in the video it is also visible that the agent sometimes overcompensates, causing the body to swing from left to right or from front to back. Therefore, we hypothesize that the reduced traversal time likely arises from the falling prevention behavior.

The SAC agent displays even more complex falling prevention behavior. In the video, this is visible when the roll or pitch becomes too high, causing the agent to stand still for a short term. So, the agent can not only modify one or two legs based on the body angles but when the IMU values diverge significantly enough, the agent even seems to halt all movement. However, from the start of the SAC clip, we see this falling prevention can significantly impede the gait, as it encounters difficulties in starting the gait. Yet, after the agent has achieved a stable gait, the SAC agent demonstrates significant adaptive behavior.

10 Conclusion

In this study, the work of Rahme *et al* was extended with another reinforcement learning method (SAC algorithm). It was applied on two different robot platforms.

For both robot platforms, the training in simulation showed the same pattern: 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.

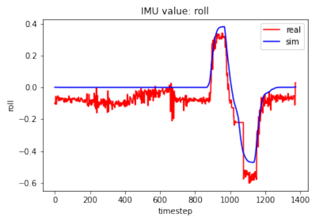
The experiments with the real SpotMicroAI robot showed that the learned gaits exhibited falling prevention behaviors. The resulting walking behavior were not faster than the baseline gait, but potentially more stable. Future research should demonstrate if an increase in the traversal speed is possible by adjusting the reward the agent receives so that is less concerned with minor deviations in the body orientation.

References

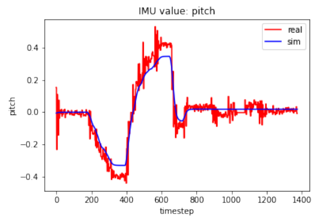
1. de Jong, G.: A soft-actor-critic approach to quadruped locomotion. Bachelor thesis, Universiteit van Amsterdam (Jun 2023)
2. Eshuijs, L.: Spotmicroai: a micro step towards intelligent locomotion. Project report, Universiteit van Amsterdam (Feb 2023)
3. Eshuijs, L., de Jong, G., Visser, A.: Demonstrating reinforcement-learned gaits with two small quadrupeds. In: Proceedings of the 35th Benelux Conference on Artificial Intelligence (BNAIC) (Nov 2023)
4. Fujimoto, S., van Hoof, H., Meger, D.: Addressing function approximation error in actor-critic methods. In: Proceedings of the 35th International Conference on Machine Learning. PMRL, vol. 80, pp. 1587–1596 (Jul 2018)
5. Fukuhara, A., Gunji, M., Masuda, Y.: Comparative anatomy of quadruped robots and animals: a review. Advanced Robotics **36**(13), 612–630 (Jun 2022)

6. Fukuoka, Y., Kimura, H., Cohen, A.H.: Adaptive dynamic walking of a quadruped robot on irregular terrain based on biological concepts. The International Journal of Robotics Research **22**(3-4), 187–202 (Mar 2003)
7. Haarnoja, T., Ha, S., Zhou, A., Tan, J., Tucker, G., Levine, S.: Learning to walk via deep reinforcement learning. In: Robotics: Science and Systems XV (June 2019)
8. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: Proceedings of the 35th International Conference on Machine Learning. PMLR, vol. 80, pp. 1861–1870 (Jul 2018)
9. Hanna, J.P., Desai, S., Karnan, H., Warnell, G., Stone, P.: Grounded action transformation for sim-to-real reinforcement learning. Machine Learning **110**(9), 2469–2499 (May 2021)
10. Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S.M.A., Riedmiller, M., Silver, D.: Emergence of locomotion behaviours in rich environments. preprint arXiv 1707.02286 (Jul 2017)
11. Hornby, G., Fujita, M., Takamura, S., Yamamoto, T., Hanagata, O.: Autonomous evolution of gaits with the sony quadruped robot. In: Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation. vol. 2, pp. 1297–1304 (Jul 1999)
12. Hu, B., Shao, S., Cao, Z., Xiao, Q., Li, Q., Ma, C.: Learning a faster locomotion gait for a quadruped robot with model-free deep reinforcement learning. In: International Conference on Robotics and Biomimetics (ROBIO). pp. 1097–1102 (2019)
13. Hyun, D.J., Seok, S., Lee, J., Kim, S.: High speed trot-running: Implementation of a hierarchical controller using proprioceptive impedance control on the mit cheetah. The International Journal of Robotics Research **33**(11), 1417–1445 (Aug 2014)
14. Ibarz, J., Tan, J., Finn, C., Kalakrishnan, M., Pastor, P., Levine, S.: How to train your robot with deep reinforcement learning: lessons we have learned. The International Journal of Robotics Research **40**(4-5), 698–721 (Apr 2021)
15. Iscen, A., Caluwaerts, K., Tan, J., Zhang, T., Coumans, E., Sindhwani, V., Vanhoucke, V.: Policies modulating trajectory generators. In: Proceedings of the 2nd Conference on Robot Learning. pp. 916–926 (Oct 2018)
16. Kau, N.: Stanford pupper: A low-cost agile quadruped robot for benchmarking and education. preprint arXiv 2110.00736 (Feb 2022)
17. Kingma, D.P., Salimans, T., Welling, M.: Variational dropout and the local reparameterization trick. Advances in neural information processing systems **28** (2015)
18. Kohl, N., Stone, P.: Policy gradient reinforcement learning for fast quadrupedal locomotion. In: IEEE International Conference on Robotics and Automation (ICRA '04). vol. 3, pp. 2619–2624 (Apr 2004)
19. Lipson, H., Malone, E.: Evolutionary robotics for legged machines: From simulation to physical reality. In: Proceedings of the 9th International Conference on Intelligent Autonomous Systems (IAS-9) (Mar 2006)
20. Rahme, M., Abraham, I., Elwin, M.L., Murphey, T.D.: Dynamics and domain randomized gait modulation with bezier curves for sim-to-real legged locomotion. preprint arXiv 2010.12070 (Oct 2020)
21. Saggar, M., D'Silva, T., Kohl, N., Stone, P.: Autonomous learning of stable quadruped locomotion. In: RoboCup 2006: Robot Soccer World Cup X. Lecture Notes in Artificial Intelligence, vol. 4434, pp. 98–109. Springer (Dec 2006)
22. Tan, J., Zhang, T., Coumans, E., Iscen, A., Bai, Y., Hafner, D., Bohez, S., Vanhoucke, V.: Sim-to-real: Learning agile locomotion for quadruped robots. In: Proceedings of Robotics: Science and Systems XIV (June 2018)

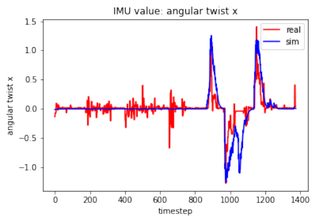
Appendix A: IMU values



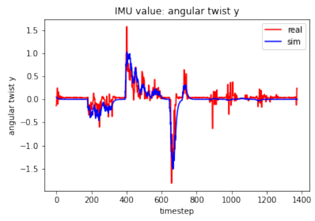
(a) Roll



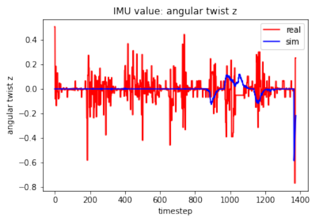
(b) Pitch



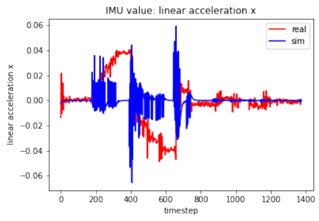
(c) Angular twist: x-direction



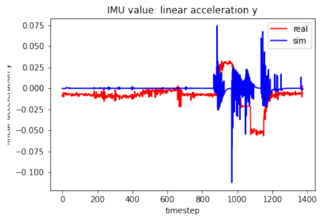
(d) Angular twist: y-direction



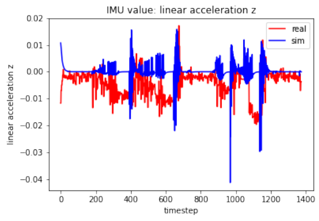
(e) Angular twist: z-direction



(f) Linear Acceleration: x-direction



(g) Linear Acceleration: y-direction



(h) Linear Acceleration: z-direction

Fig. 9: IMU results of the same movement in simulation and on the real robot, for the SpotMicroAI. In red are the results of the real robot and in blue are the results of the simulation. The performed movement consisted of a negative and positive pitch movement, followed by a positive and negative roll movement

Before the quality of the RL agents on the real robot can be inspected we first compare the IMU data between the simulation and the real robot. To achieve this comparison, we use IK to make the robot apply a roll in both directions followed by a pitch in both directions. All displayed values were as returned by the IMU, except for the linear acceleration, which was reduced by a factor of 10 to match the values in the simulation. As seen from the Figures, the roll and pitch measurements are similar to those in the simulation. The angular twist also follows the same values as the simulation but already contains much more noise, especially in the z-direction. Lastly, the linear acceleration looks much more chaotic in real life than in simulation but importantly has spikes at the same points in the graph which suggests it still contains useful information. Even though the angular twist and linear acceleration contain more noise on the real sensor than in the simulation, the results in the simulation also show erratic behavior, we thus expect the agents to put more weight on the values of the roll and pitch. From these results, we conclude that the sensor data provided by the IMU is faithful enough for the sim-to-real transfer of the RL agents.