Gait in Eight: Efficient On-Robot Learning for Omnidirectional Quadruped Locomotion

Nico Bohlinger*, Jonathan Kinzel*, Daniel Palenicek, Jan Peters
*Equal contribution.
Department of Computer Science
Technical University of Darmstadt, Germany

Łukasz Antczak
MAB Robtics, Poznan, Poland

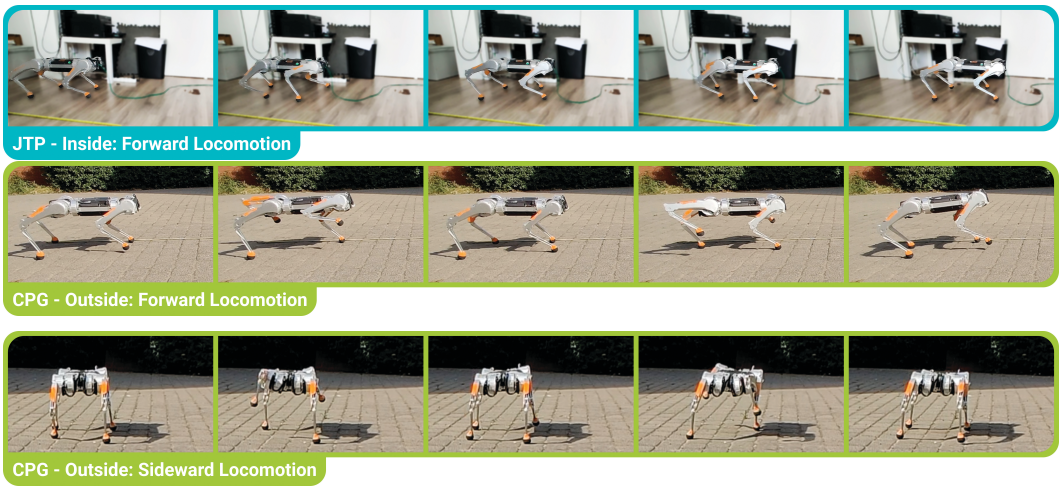

Figure 1: We demonstrate the effectiveness of the CrossQ algorithm in combination with a Joint Target Prediction (JTP) or Central Pattern Generator (CPG) architecture for learning omnidirectional locomotion on the MAB HoneyBadger quadruped.

Abstract

On-robot Reinforcement Learning is a promising approach to train embodiment-aware policies for legged robots. However, the computational constraints of real-time learning on robots pose a significant challenge. We present a framework for efficiently learning quadruped locomotion in just 8 minutes of raw real-time training utilizing the sample efficiency and minimal computational overhead of the new off-policy algorithm CrossQ. We investigate two control architectures: Predicting joint target positions for agile, high-speed locomotion and Central Pattern Generators for stable, natural gaits. While prior work focused on learning simple forward gaits, our framework extends on-robot learning to omnidirectional locomotion. Finally, we demonstrate the robustness of our approach in different indoor and outdoor environments.

18th European Workshop on Reinforcement Learning (EWRL 2025).

1 Introduction

Legged robot locomotion has long been an important research area in robotics, as achieving robust, agile, and adaptable gaits in unstructured environments is a challenging yet essential capability for many real-world applications. Traditional model-based controllers [27, 21], while effective in well-structured settings, often struggle to handle the inherent uncertainties and dynamic variations encountered in real-world terrains. In contrast, Deep Reinforcement Learning (DRL) offers a promising paradigm that allows robots to autonomously acquire locomotion skills directly through interaction with the environment. In recent years, DRL has shown remarkable success in learning complex and agile locomotion skills for many different legged robots [8, 3], such as high-speed running [4, 24], jumping and climbing in parkour-like courses [39, 11], navigating through challenging terrain [2, 38], and performing handstands and backflips [11, 20]. However, these works rely on scaling up on-policy DRL algorithms, mainly Proximal Policy Optimization (PPO) [30], through thousands of parallel simulated environments with GPU-based physics engines [28]. While a plethora of domain randomization is necessary to zero-shot transfer policies trained in simulation to the real world [35, 18], this creates a significant embodiment gap between the widely randomized and approximated dynamics of the simulated robot and the specific, nuanced dynamics of the real robot. On-robot learning promises to bridge the embodiment gap by learning directly on the real system. This enables the DRL agent to be aware of its physical embodiment and the specific hardware constraints. It can continuously adapt to changes, such as wear and tear, battery depletion, hardware modifications, or environmental changes. Recent advances in off-policy DRL have made first steps toward this paradigm by improving the learning efficiency enough to enable training quadruped locomotion in real-time directly on the robot [33, 34, 22]. However, these works must simplify the learning task to plain forward locomotion with a fixed target velocity. They only achieve crawling gaits and rely on powerful laptops with dedicated GPUs to run the training process fast enough to be feasible. Our goal is to lift computational constraints by improving the learning efficiency and update speed with the recently proposed CrossQ DRL algorithm [6] and extend the learning task to omnidirectional locomotion, allowing any desired velocity in the xy-plane.

Our main contributions are as follows:

- We introduce an efficient on-robot learning framework based on CrossQ
- We learn forward and omnidirectional locomotion while reaching higher maximum velocities, training significantly faster, and doubling the action frequency compared to previous works.
- We investigate and compare two control architectures based on Joint Target Prediction (JTP) and Central Pattern Generators (CPGs), highlighting the trade-offs between achieving aggressive, high-speed gaits and maintaining stable, natural locomotion with high foot clearance.
- We provide comprehensive empirical evaluations in simulation and in two real-world environments, demonstrating the practical viability and effectiveness of our framework.

2 Preliminaries

In this section, we introduce the necessary background and notational foundation for the remainder of the paper. First, we describe the Reinforcement Learning (RL) framework and efficient algorithms to learn in it. Then, we introduce the quadruped robot platform used in our experiments in both simulation and the real world.

2.1 Reinforcement learning

We formulate the problem of learning on-robot locomotion as training an RL agent that interacts with an environment defined by a Partially Observable Markov Decision Process (POMDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, P, O, R, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{O}$ is the observation space, P is the transition dynamics, O is the observation function, R is the reward function, and γ is the discount factor. Due to partial observability and noisy sensors, the agent does not have access to the true state of the environment $s \in \mathcal{S}$, but instead receives observations $o \in \mathcal{O}$. To learn a control policy $\pi(a|o)$ that solves the POMDP, the agent needs to explore the environment sufficiently.

Therefore, we employ the Maximum Entropy RL framework [40]. The goal is to learn a policy that maximizes the expected discounted return while also maximizing the entropy of the policy $J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t(r_t - \alpha\mathcal{H}(\pi(\cdot|o_t)))\right]$, where $\tau = (o_0, a_0, r_0, o_1, a_1, r_1, \ldots)$ is a trajectory generated by rolling out the policy π in the environment and $\mathcal{H}(\pi(\cdot|o_t))$ and α are the entropy of the policy and the temperature parameter, respectively.

2.2 Efficient model-free off-policy reinforcement learning

Soft Actor-Critic (SAC) [14] is a popular choice of model-free off-policy DRL algorithms for tasks with continuous state-action spaces. SAC is an actor-critic algorithm, formulated for the Maximum Entropy RL framework. As such, it learns a soft Q-function

$$Q_\phi^\pi(o, a) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t(r_t - \alpha\log\pi(a_t|o_t))\right],$$

with $o_0 = o$ and $a_0 = a$, which models the expected discounted return of a policy π when taking action a based on the current observation o. This is done by minimizing the Bellman error $[Q_\phi^\pi(o_t, a_t) - r_t - \gamma Q_\phi^\pi(o_{t+1}, a_{t+1})]^2$. Simultaneously, SAC learns a parameterized policy $\pi_\theta(a|o)$, with the objective of maximizing expected discounted return

$$\theta = \arg\max_\theta Q_\phi^\pi(o, \pi_\theta(o)).$$

To reduce the number of environment interactions, authors have mainly proposed to increase the Update-To-Data (UTD) ratio, which refers to the number of gradient updates performed per agent environment interaction [25, 12, 10, 15]. Naturally, this results in increased compute costs and wall-clock time, which can be problematic for on-robot learning in real-time, especially on a constrained compute budget. Previous work on on-robot learning for quadruped locomotion has relied on the Dropout Q-Functions (DroQ) algorithm [15] with a UTD ratio of 20 [33, 34] to achieve sample-efficient but compute-intensive learning.

In this work, we build on the recently proposed CrossQ algorithm [6] that is based on SAC and achieves state-of-the-art sample efficiency while maintaining the original UTD of 1. The authors achieve this by carefully using Batch Normalization (BN) [17] within the critic network and removing target networks. The main insight is to compute the BN statistics on the joint current state-action (o, a) and next state-action $(o', a' \sim \pi_\theta(o'))$ distribution. In practice, this is implemented via a joint forward pass of the current and next state-action batches through the Q-function.

2.3 Quadruped platform

All experiments were performed on the HoneyBadger 4.0 quadruped robot from MAB Robotics (Figure 2), a 12 DoF platform with three actuated joints per leg. The robot measures 60 cm in length, 40 cm in width and height, and has a mass of 12 kg. Each joint is driven by a torque-controlled, quasi-direct drive actuator with a 9:1 gear ratio while weighing 0.5 kg each. The actuators are controlled by MAB MD80 servo drives and deliver a nominal torque of 9 Nm and a peak of 18 Nm. Furthermore, the robot is equipped with a dual-computer system, a VectorNav VN-100 AHRS IMU, and is powered by a 42 V Li-Ion battery. The robot's software is built on ROS 2 that enables the necessary low-level joint control for our experiments.

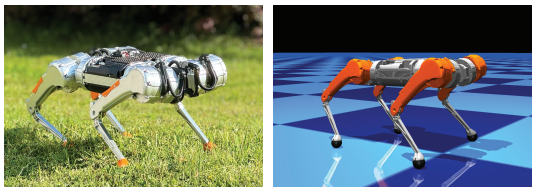

Figure 2: MAB Robotics HoneyBadger quadruped robot in the real world (left) and in the MuJoCo simulation (right).

3 Efficient on-robot learning for legged locomotion

We propose learning legged locomotion directly on the HoneyBadger quadruped robot using the CrossQ algorithm and a carefully designed learning framework. First, we define the task setting and reward design for learning forward and omnidirectional locomotion. Then, we introduce two control architectures based on JTPs and CPGs to efficiently learn agile, stable and natural gaits.

3.1 Locomotion tasks & reward design

We first consider the task of learning forward locomotion only as a simplified version of the full locomotion task, since it is commonly used in on-robot locomotion learning [33, 34, 22]. We formulate the task as learning to track a desired x-velocity $\bar{v}_x \in \mathbb{R}$ with the robot's trunk. The reward function is designed to encourage the robot to move in a straight line forward at the target velocity while keeping the body orientation upright and minimizing energy consumption by penalizing high torques. Ablations on the reward function can be found in appendix B.1. Besides tracking a desired velocity, we also consider training the robot to walk forward as fast as possible and change the reward function to simply encourage higher forward velocity while penalizing any velocity in the y-direction. Finally, forward-only locomotion with a fixed target velocity is a common limitation when learning on-robot locomotion, therefore, we extend the task to omnidirectional locomotion by considering random desired velocities in the xy-plane $\bar{v}_{xy} \in \mathbb{R}^2$. Although we omit a target yaw velocity, this setting matches the sim2real literature in robot locomotion more closely [29] and enables a more versatile gait. We modify the tracking reward to also consider the y-velocity by penalizing any deviation from the target velocity in the y-direction. Table 1 summarizes the reward functions for the fixed forward, maximum forward, and omnidirectional locomotion tasks, defined by $r_{\text{track-x}}$, $r_{\text{max-x}}$, and $r_{\text{track-xy}}$, respectively.

One of the main challenges of training RL policies directly on a real robot is the availability of key quantities needed in reward terms and the inherent noise in estimates of these quantities. Unlike in simulation, where the complete and true state of the robot is readily available, real-world experiments must rely on state estimation techniques that introduce significant uncertainty through sensor noise and may not provide all the necessary information to begin with. We limit our reward function to rely only on proprioceptive information, such as accurate torque measurements from the joint encoders and noisy data from the onboard IMU. The orientation of the trunk is estimated by fusing the magnetometer and accelerometer data. For the linear velocity of the trunk, we use a Kalman filter to fuse the acceleration data with the av-

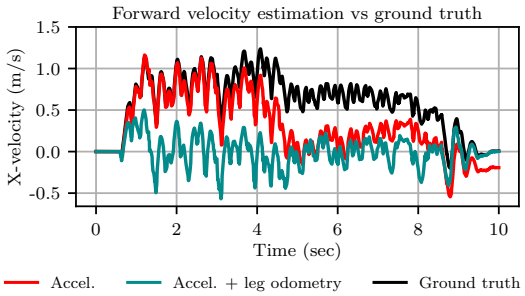

Figure 3: Comparison of the linear x-velocity estimation with only integrating the acceleration data from the IMU with a Kalman filter, with the Kalman filter fusing of the accelerations and the leg odometry, and the ground truth.

Table 1: We build upon the tracking reward $r_{\text{track-x}}$ from Levy et al. [22] and add penalty terms for improving the gait quality and energy efficiency. v_x, v_y, and v_{xy} are the x-, y-, and combined xy-velocity of the trunk of the robot, respectively. $(R(\theta)^\top v_{xy})_y$ is the y component of the trunk velocity rotated into the target direction defined by the target velocity $\bar{v}_{xy}$. θ is the orientation and ω is the angular velocity of the trunk. τ is the torque applied to the joints.

Reward	**Term**
$r_{\text{track-x}}$	$\begin{cases} 1, & \text{for } v_x \in [\bar{v}_x, 2\bar{v}_x] \\ 0, & \text{for } v_x \in (-\infty, -\bar{v}_x] \cup [4\bar{v}_x, \infty) \\ 1 - \frac{\lvert v_x - \bar{v}_x \rvert}{2\bar{v}_x}, & \text{otherwise} \end{cases}$
$r_{\text{max-x}}$	$v_x - \lvert v_y \rvert$
$r_{\text{track-xy}}$	$r_{\text{track-x}} - \left\lvert \left(R(\theta)^\top v_{xy} \right)_y \right\rvert$
r_{yaw}	$\lvert \omega_{\text{yaw}} \rvert^2$
r_{upright}	$\lvert \theta_{\text{pitch, roll}} \rvert^2$
r_{energy}	$\lvert \tau \rvert^2$
$r_{\text{total-track-x}}$	$\max(r_{\text{track-x}} - 0.1 r_{\text{yaw}} - 10 r_{\text{upright}} - 0.0003 r_{\text{energy}}, 0)$
$r_{\text{total-max-x}}$	$\max(2 r_{\text{max-x}} - 0.1 r_{\text{yaw}} - 10 r_{\text{upright}} - 0.0003 r_{\text{energy}}, 0)$
$r_{\text{total-track-xy}}$	$\max(r_{\text{track-xy}} - 0.1 r_{\text{yaw}} - 10 r_{\text{upright}} - 0.0003 r_{\text{energy}}, 0)$

eraged hip velocity inferred from a leg odometry module that relies on the forward kinematics of the robot. The linear velocity estimation is the most crucial part of the robot state, as it is the main reward signal driving the learning process, but also the most challenging to estimate accurately due to the lack of foot contact sensors on the robot and the integration of noisy acceleration data. Figure 3 shows that integrating only acceleration data works well after an initial calibration but is prone to drift away from the ground truth after a few seconds. Combining the acceleration data with the leg odometry grounds the estimation. This prevents significant drift in the estimates, but systematically underestimates the velocity due to the lack of foot contact sensors, which requires the assumption that all four feet are always in contact with the ground. In general, the learning process is always restricted by the quality of the linear velocity estimation and the ability of the RL algorithm to extract useful information from the noisy and biased signal. A motion capture system could provide a more accurate estimate of the linear velocity [13] but is only available in a controlled lab setting and, hence, is not suitable for experiments in uncontrolled environments, such as outdoors.

3.2 Control architecture

The first control architecture that we consider is the JTP, which is commonly used in different variations in legged robot locomotion [28, 33, 34]. Here, the actions of the policy $a_{\text{JTP}} \in [-\varphi, \varphi]$ are offset joint angles to a nominal standing position q^{nominal} and are clipped to a maximum deviation of φ. This ensures a minimal but sufficient joint range for learning a viable gait quickly by reducing the search space of the policy. The resulting target joint angles $q^{\text{target}} = q^{\text{nominal}} + a_{\text{JTP}}$ are first processed by a filter before being tracked by a PD controller. The filter manages the Gaussian noise used for policy exploration during the early training phase. Without filtering, early policies exhibit uncoordinated, jittery movements, leading to inaccurate velocity estimation and oscillations that hurt the learning process and compromise the robot's safety. Although the low-pass filter is a common choice for smoothing trajectories [31, 33, 34], it also reduces the system's responsiveness to sudden changes, thereby decreasing the robot's potential agility and speed. Therefore, we employ the One-Euro filter [9] as it can significantly enhance responsiveness by balancing the trade-off between low-pass filtering during low velocities and no filtering after a velocity threshold. Ablations on the choice of the filter can be found in appendix B.1.

The second control architecture we consider uses the CPG framework originating from biology [23], where rhythmic patterns are generated by neural circuits in the spinal cord of animals. In robot locomotion, CPGs provide an intuitive formalism to define natural gaits by generating smooth, periodic trajectories for the robot's feet [16, 32, 19]. We configure a CPG to generate a stable in-place trot pattern by defining sinusoidal feet height trajectories $p_{\text{CPG}} = [f(t_i)]_{i=1}^4$ using a spline function

$$f(t_i) = \begin{cases} h(-2t_i^3 + 3t_i^2), & t_i \in [0, \pi/2) \\ h(2t_i^3 - 3t_i^2 + 1), & t_i \in [\pi/2, \pi) \end{cases}$$

where t_i is the normalized phase of the gait cycle of the i-th foot with a fixed frequency. In case of the trot gait, the phases of the right and left legs are shifted by $\pi/2$. The maximum foot height h is set to $0.15\,\text{m}$, which we empirically validated on the HoneyBadger for robustness on rough terrain with slight inclinations (up to $3°$). The action of the policy $a_{\text{CPG}} = [x_i, y_i, z_i]_{i=1}^4$ modifies the CPG trajectory by predicting an offset position in the Cartesian space for each foot. Similarly to JTP, the predicted offsets are restricted to a Cartesian subspace to reduce the complexity of exploration. The final feet positions are calculated by $p = p_{\text{CPG}} + a_{\text{CPG}}$, so they can be converted to the corresponding joint angles using analytical inverse kinematics and applied using a PD controller. While the CPG provides a stable in-place trot, the RL policy refines this gait, enabling the robot to walk in any direction and dynamically adapt to the environment.

In both control architectures, the agent has access to the following observations: joint angles, joint velocities, previous action, linear and angular accelerations of the trunk, linear and angular velocities of the trunk, desired trunk velocity, and the gravity vector. As discussed for the reward function, the linear velocities of the trunk are crucial for the learning process, but are the most noisy and biased estimates of the observations. For the CPG approach, the observation space is extended to include the normalized CPG phase variable $[l_1, l_2] \in [0, 1]$, which tracks the current progress within the gait cycle for the right and left legs.

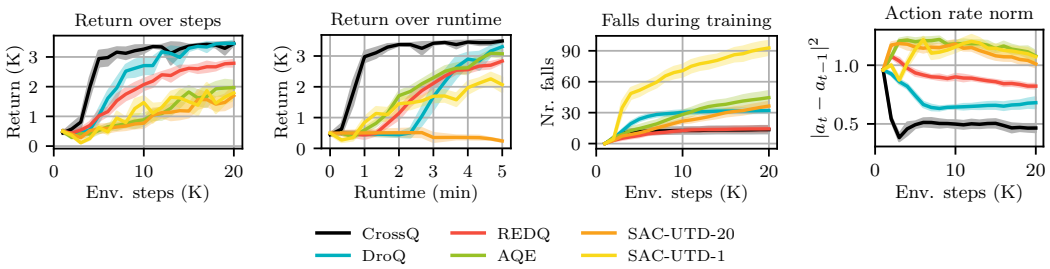

Figure 4: Learning curves of the different algorithms in terms of return over environment steps (left) and runtime (center left), number of falls (center right), and squared action rate norm (right).

4 Experimental results

In this section, we empirically evaluate our framework in both simulation and on the real HoneyBadger robot. The simulation results compare CrossQ with state-of-the-art off-policy methods in our locomotion setting, focusing on learning efficiency and stability. Building on these insights, real-world experiments in different environments demonstrate the practical viability of our framework and compare the performance of the proposed control architectures.

4.1 Simulation

Before we deploy our learning framework on the real robot, we first evaluate and ablate its performance in simulation. We use the MuJoCo physics engine [36] to simulate the HoneyBadger robot (see Figure 2) and the locomotion tasks on flat terrain with randomized action delays. We build on the DRL library RL-X [7] to integrate the simulation environment with the algorithm in JAX, and to run the experiments with 10 seeds for each setting.

First, we perform an ablation study on the learning efficiency of CrossQ [6] by comparing it to other off-policy algorithms, namely SAC [14], Aggressive Q-Learning with Ensembles (AQE) [37], Randomized Ensembled Double Q-Learning (REDQ) [10], and DroQ [15]. We use the default hyperparameters proposed in the original papers and the same network sizes for all algorithms. For the UTD ratio, we use 1 for CrossQ, 1 and 20 for SAC, 5 for AQE, and 20 for REDQ and DroQ. Appendix A summarizes all hyperparameter choices. We evaluate the algorithms on learning forward locomotion with a target velocity of $\bar{v}_x = 0.5$ m/s and combine them with the JTP control scheme. Figure 4 highlights the superior learning speed of CrossQ compared to the other algorithms in terms of environment steps and pure training time. CrossQ learns a good locomotion policy after 1 minute of training, while DroQ, the second-best algorithm and used in previous works [33, 34], requires close to 5 times more training time to reach the same performance. During the evaluation, the final gait of CrossQ appears to be smoother, which is reflected in the squared action rate norm $\sum_j^{\text{joints}} |a_t^j - a_{t-1}^j|^2$, which is significantly lower for CrossQ compared to the other algorithms. Importantly, CrossQ also achieves the lowest number of falls during training. The lower action rate norm and the fall rate indicate a better-tempered exploration strategy, which we hypothesize to be due to the removal of target networks, leading to more accurate value estimates. Fewer falls are highly beneficial for real-world experiments later on, as every fall requires manual intervention and can lead to hardware damage.

Next, we compare the two control architectures: JTP and CPG. We train both using CrossQ and first evaluate their ability to learn high-speed agile locomotion on the maximum forward velocity task. Figure 5 shows that JTP reaches a maximum forward velocity of 1.5 m/s at the end of training, while the CPG achieves only around 0.75 m/s. This is expected, as the JTP has more direct control over all joints and can learn a more aggressive gait. However, the learning process with the CPG is much more stable and leads to zero falls during training, while the JTP approach falls multiple times. When training the agent on the target forward velocity task with $\bar{v}_x = 0.5$ m/s, both approaches learn to track the target velocity quickly and show a similar end performance. Using the CPG leads to a smaller variance in the learning curves and no falls during training.

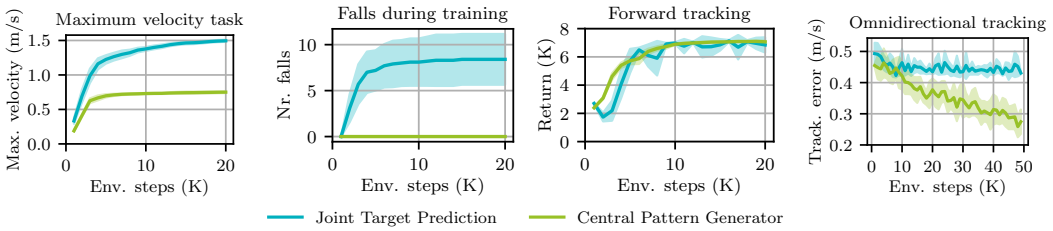

Figure 5: Left – Average maximum velocity reached in an episode during training on $r_{\text{max-x}}$. Center left – Number of episodes terminated due to falls during the maximum velocity training. Center right – Average return on the target forward velocity task. Right – Absolute tracking error of the target velocity on the omnidirectional task.

Finally, we evaluate the omnidirectional locomotion task with target velocities $\bar{v}_{xy}$ independently sampled from $\mathcal{U}(-0.5, 0.5)$ for x and y. Figure 5 shows that the CPG learns to track target velocities in both directions, while the JTP struggles to learn the task at all and falls up to 30 times during training. It should be noted that we were able to help the JTP approach learn by applying a curriculum [5] strategy on the sampled target velocities based on the tracking error, but we omit this strategy, as noisy estimates of the tracking error in real-world experiments make it unreliable. We refer the reader to appendix B for details.

In summary, in our experiments the CPG approach is more stable and robust. It does not fall at all during training and its initial trot pattern is easily adaptable for the policy to learn omnidirectional locomotion. The JTP approach, on the other hand, learns more agile and aggressive gaits and can achieve higher velocities. This difference in gait style can also be seen in the different footstep patterns shown in Figure 6. The agile JTP policies produce more irregular patterns and higher frequency gaits, compared to the CPG ones with a fixed gait frequency.

4.2 Real-world

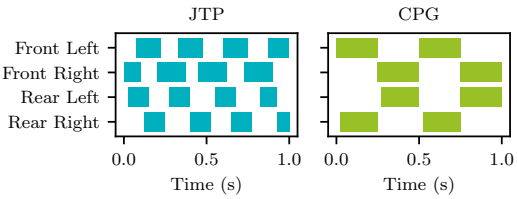

Figure 6: Footstep patterns of the learned policies for the JTP (left) and CPG (right) approaches.

We evaluate our learning framework on the real HoneyBadger robot in two settings: a small office environment that provides a smooth, level surface with low friction and no obstacles, and a spacious outdoor environment with uneven cobblestone terrain, high friction, and small obstacles. The experiments for the forward locomotion task are carried out in both environments, while learning omnidirectional locomotion is only evaluated outdoors due to space constraints in the office environment (see Figure 1). The training is performed on a M3 MacBook Pro that is directly connected to the robot via Ethernet and ROS 2. In contrast, previous works rely on a laptop with a dedicated NVIDIA GPU [33, 34]. Furthermore, we can double the action frequency of our policy to 40 Hz, compared to 20 Hz used in previous works [33, 34], due to the reduced update time of CrossQ with the UTD ratio of 1 and the removal of the target network. Like in simulation, we train the policies for 20,000 steps which corresponds to 500 seconds or around 8 minutes of raw training time. For the omnidirectional locomotion task, we increase the training time to 50,000 steps to handle the increased difficulty of the task. The actual training duration is significantly longer due to the need for manual intervention, including resetting the robot after falls, reorienting it when it reaches the boundaries of the training area and additional safety triggers that prevent the robot from damaging itself (e.g., joints being close to their angle limits). We note that using a self-resetting policy that can recover from falls autonomously and is learned prior to the main task could reduce the training time of our experiments [33]. After training, we evaluate the performance of the agent by removing the exploration noise and rolling out the deterministic policy.

First, we evaluate learning forward locomotion in the indoor office environment. In addition to combining CrossQ with the JTP and CPG control architectures, we also include a SAC baseline

with a UTD ratio of 20, a low-pass filter and using the JTP approach. The results of this and all the following real world experiments are summarized in Table 2. The SAC baseline was only able to reach a very low velocity of 0.013 m/s and learned a strategy of heavily jumping with the front legs, leading to harsh movements and poor grip on the ground. This resulted in the robot being unable to learn a straight and stable gait. The training took up to 40 minutes to complete, due to the 20 Hz action frequency limited by the high UTD ratio and a total of 43 falls during training. CrossQ with the JTP approach and a One-Euro filter was able to reach a maximum velocity of 0.85 m/s after completing the training in 25 minutes with only 15 falls, which is the fastest gait learned in a few minutes directly on a quadruped robot to our knowledge. The agent initially took small steps while maintaining balance, gradually improving its gait, and increasing its step size over time. Although the agent developed a fast-paced trotting strategy, it struggled with occasional backward falls and walking in a straight line. CrossQ with the CPG reached a maximum velocity of 0.3 m/s after 30 minutes of training with 38 falls. The CPG triggered our safety constraints regularly, which led to many unnecessary falls, nevertheless the agent was able to learn a straight and stable gait with high foot clearance.

Next, we test the forward locomotion task in the outdoor environment, which introduces additional complexity due to uneven cobblestones, increased friction, and small obstacles such as curbs. The JTP approach reached a maximum velocity of 0.25 m/s after 19 minutes of training with 19 falls. The agent initially focused on balancing its trunk by deliberately falling backward to prevent tipping forward, but over time leveraged front-leg coordination to maintain stability. This initial focus on balance led to a slower final gait speed, but the agent was able to adapt to terrain variations effectively. The CPG approach reached a maximum velocity of 0.33 m/s after 17 minutes of training with 39 falls. The final policy showed robustness against environmental disturbances and avoided unnecessary safety activations after an initial phase of struggle with inclinations.

The omnidirectional locomotion experiment introduced random target velocities in both the x- and y-direction. Due to the strong noise and drifting in the linear velocity estimation, curriculum learning was not feasible, requiring the agent to adapt without a difficulty progression for the target velocities. Like in simulation, the JTP was unable to learn with the full range of target velocities in the xy-plane and failed to achieve any meaningful directional movement, resorting to a standing behavior. The CPG achieved a maximum velocity of 0.25 m/s in the x-direction and 0.15 m/s in the y-direction after 33 minutes. After overcoming early instabilities that resulted in 43 falls, the agent adapted to the outdoor environment, achieving forward, left, and right movements during the evaluation. However, the learned gait was not perfectly straight and the maximum velocities fell short of the target velocities, leaving room for future work.

In summary, like in simulation, the JTP control architecture proved to be more agile, while the gait of CPG looked more natural and had better yaw control. But unlike in simulation, the CPG suffered from

Table 2: Comparing real-world experiments with different environments, tasks and control architectures. **Eval. Vel.**: Maximum velocity reached during evaluation. For omnidirectional locomotion, separate x and y velocities are reported. **Yaw Ctl.**: Deviation from initial yaw orientation. Very Good ($<10°$), Good ($10° - 30°$), Medium ($30° - 60°$), Poor ($> 60°$). **Nr. Falls**: Total number of falls during training. **Durat.**: Time required to complete the training and evaluation.

App.	Eval. Velocity	Yaw Ctl.	Nr. Falls	Durat.
Office: Forward Locomotion				
SAC	0.013 m/s	Poor	43	40 min
JTP	0.85 m/s	Medium	15	25 min
CPG	0.3 m/s	Good	38	30 min
Outside: Forward Locomotion				
JTP	0.25 m/s	Good	19	19 min
CPG	0.33 m/s	Very Good	39	17 min
Outside: Omnidirectional Locomotion				
JTP	N/A	N/A	6	25 min
CPG	x: 0.25 m/s, y: 0.15 m/s	Poor	43	33 min

triggering the joint limit safety constraints, leading to more terminations during training. Depending on the environment and the desired locomotion task, both control architectures have their advantages and disadvantages, with a trade-off between agility and stability.

5 Conclusion

In this work, we presented a framework for efficiently learning quadruped locomotion directly on the HoneyBadger quadruped robot using the CrossQ algorithm. Our approach leverages CrossQ's sample efficiency and minimal compute overhead to achieve maximum velocities of up to 0.85 m/s in just 8 minutes of raw training. We combine CrossQ with two control architectures: a JTP scheme for agile, high-speed gaits and a CPG scheme for stable, natural gaits. Lastly, we extended on-robot locomotion learning to omnidirectional locomotion with different target velocities in the xy-plane. Our real-world experiments in indoor and outdoor environments showed the practicality of our framework and the robustness of the learned policies to terrain variations and sensor noise. Future work will focus on improving the linear velocity estimation with visual-inertial odometry [1], exploring visual observations, algorithmic improvements of the base algorithm [26], and fine-tuning powerful pre-trained policies from simulation to adapt to new environments and the specific robot embodiment that is changing through wear and tear or hardware modifications.

Acknowledgments

We sincerely thank MAB Robotics for providing the HoneyBadger robot and all the extensive support and guidance during the experiments. We especially thank Jakub Matyszczak, Jakub Bartoszek, Krzysztof Kwapisz and Michał Raszewski.

References

[1] D. Aditya, J. Huang, N. Bohlinger, P. Kicki, K. Walas, J. Peters, M. Luperto, and D. Tateo. Robust localization, mapping, and navigation for quadruped robots. *arXiv preprint arXiv:2505.02272*, 2025.

[2] A. Agarwal, A. Kumar, J. Malik, and D. Pathak. Legged locomotion in challenging terrains using egocentric vision. In *Conference on robot learning*, 2023.

[3] B. Ai, L. Dai, N. Bohlinger, D. Li, T. Mu, Z. Wu, K. Fay, H. I. Christensen, J. Peters, and H. Su. Towards embodiment scaling laws in robot locomotion. *Conference on Robot Learning (CoRL)*, 2025.

[4] G. Bellegarda, Y. Chen, Z. Liu, and Q. Nguyen. Robust high-speed running for quadruped robots via deep reinforcement learning. In *International conference on intelligent robots and systems*, 2022.

[5] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *International conference on machine learning*, 2009.

[6] A. Bhatt, D. Palenicek, B. Belousov, M. Argus, A. Amiranashvili, T. Brox, and J. Peters. CrossQ: Batch normalization in deep reinforcement learning for greater sample efficiency and simplicity. In *International conference on learning representations*, 2024.

[7] N. Bohlinger and K. Dorer. Rl-x: A deep reinforcement learning library (not only) for robocup. In *Robot world cup*, pages 228–239. Springer, 2023.

[8] N. Bohlinger, G. Czechmanowski, M. Krupka, P. Kicki, K. Walas, J. Peters, and D. Tateo. One policy to run them all: an end-to-end learning approach to multi-embodiment locomotion. In *Conference on robot learning*, 2024.

[9] G. Casiez, N. Roussel, and D. Vogel. 1 € filter: a simple speed-based low-pass filter for noisy input in interactive systems. In *Conference on human factors in computing systems*, 2012.

[10] X. Chen, C. Wang, Z. Zhou, and K. Ross. Randomized ensembled double Q-learning: Learning fast without a model. In *International conference on learning representations*, 2021.

[11] X. Cheng, K. Shi, A. Agarwal, and D. Pathak. Extreme parkour with legged robots. In *RoboLetics: Workshop on robot learning in athletics @ CoRL*, 2023.

[12] P. D'Oro, M. Schwarzer, E. Nikishin, P.-L. Bacon, M. G. Bellemare, and A. Courville. Sample-efficient reinforcement learning by breaking the replay ratio barrier. In *International conference on learning representations*, 2022.

[13] J. S. Furtado, H. H. Liu, G. Lai, H. Lacheray, and J. Desouza-Coelho. Comparative analysis of optitrack motion capture systems. In *Advances in Motion Sensing and Control for Robotic Applications*, pages 15–31. Springer, 2019.

[14] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, 2018.

[15] T. Hiraoka, T. Imagawa, T. Hashimoto, T. Onishi, and Y. Tsuruoka. Dropout q-functions for doubly efficient reinforcement learning. In *International conference on learning representations*, 2021.

[16] A. J. Ijspeert. Central pattern generators for locomotion control in animals and robots: a review. *Neural networks*, 21(4):642–653, 2008.

[17] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, 2015.

[18] G. Ji, J. Mun, H. Kim, and J. Hwangbo. Concurrent training of a control policy and a state estimator for dynamic and robust legged locomotion. *Robotics and automation letters*, 7(2): 4630–4637, 2022.

[19] M. Kasaei, M. Abreu, N. Lau, A. Pereira, and L. P. Reis. A cpg-based agile and versatile locomotion framework using proximal symmetry loss. *arXiv preprint arXiv:2103.00928*, 2021.

[20] D. Kim, H. Kwon, J. Kim, G. Lee, and S. Oh. Stage-wise reward shaping for acrobatic robots: A constrained multi-objective reinforcement learning approach. *arXiv preprint arXiv:2409.15755*, 2024.

[21] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous robots*, 40:429–455, 2016.

[22] J. Levy, T. Westenbroek, and D. Fridovich-Keil. Learning to walk from three minutes of real-world data with semi-structured dynamics models. In *Conference on robot learning*, 2024.

[23] E. Marder and D. Bucher. Central pattern generators and the control of rhythmic movements. *Current biology*, 11(23):R986–R996, 2001.

[24] G. B. Margolis, G. Yang, K. Paigwar, T. Chen, and P. Agrawal. Rapid locomotion via reinforcement learning. *International journal of robotics research*, 43(4):572–587, 2024.

[25] E. Nikishin, M. Schwarzer, P. D'Oro, P.-L. Bacon, and A. Courville. The primacy bias in deep reinforcement learning. In *International conference on machine learning*, 2022.

[26] D. Palenicek, F. Vogt, J. Watson, and J. Peters. Scaling off-policy reinforcement learning with batch and weight normalization. *arXiv preprint arXiv:2502.07523*, 2025.

[27] J. Pratt, J. Carff, S. Drakunov, and A. Goswami. Capture point: A step toward humanoid push recovery. In *International conference on humanoid robots*, 2006.

[28] N. Rudin, D. Hoeller, P. Reist, and M. Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on robot learning*, 2022.

[29] N. Rudin, D. Hoeller, P. Reist, and M. Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on robot learning*, 2022.

[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[31] I. Selesnick and C. Burrus. Generalized digital butterworth filter design. *Transactions on signal processing*, 1998.

[32] Y. Shao, Y. Jin, X. Liu, W. He, H. Wang, and W. Yang. Learning free gait transition for quadruped robots via phase-guided controller. *Robotics and automation letters*, 7(2):1230–1237, 2021.

[33] L. Smith, I. Kostrikov, and S. Levine. A walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning. In *Robotics: Science and systems*, 2023.

[34] L. Smith, Y. Cao, and S. Levine. Grow your limits: Continuous improvement with real-world rl for robotic locomotion. In *International conference on robotics and automation*, pages 10829–10836, 2024.

[35] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *International conference on intelligent robots and systems*, 2017.

[36] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *International conference on intelligent robots and systems*, 2012.

[37] Y. Wu, X. Chen, C. Wang, Y. Zhang, and K. W. Ross. Aggressive q-learning with ensembles: Achieving both high sample efficiency and high asymptotic performance. In *Deep reinforcement learning workshop @ NeurIPS*, 2022.

[38] C. Zhang, N. Rudin, D. Hoeller, and M. Hutter. Learning agile locomotion on risky terrains. In *International conference on intelligent robots and systems*, 2024.

[39] Z. Zhuang, Z. Fu, J. Wang, C. Atkeson, S. Schwertfeger, C. Finn, and H. Zhao. Robot parkour learning. In *Conference on robot learning*, 2023.

[40] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, 2008.

A Learning and filtering hyperparameters

We summarize the hyperparameters for the different off-policy algorithms used for the experiments in Table 3. The parameters for the action filtering are listed in Table 4.

Table 3: Hyperparameter Configurations

Parameter	**SAC**	**SAC-20**	**DroQ**	**REDQ**	**AQE**	**CrossQ**
Learn. Rate	0.003	0.003	0.001	0.0003	0.0003	0.005
Batch Size	256	256	256	256	256	128
Frequency	20 Hz	20 Hz	20 Hz	20 Hz	20 Hz	40 Hz
Neurons	256, 256					
Nr. critics	2	2	2	10	10	2
Gamma	0.99					
Optimizer	Adam					
UTD ratio	1	20	20	20	5	1

Table 4: Filter Parameters

Parameter	**None**	**Low-pass filter**	**One-Euro filter**
mincutoff	–	0.4	2.5
beta	–	–	0.1
dcutoff	–	–	100

B Omnidirectional curriculum

To enable the JTP approach to learn omnidirectional locomotion, we introduce a curriculum strategy. The curriculum is designed to systematically guide the learning agent from easy-to-learn backward locomotion to stable omnidirectional movement in the xy-plane. We model movement directions as a circular space, partitioned into two half-circles, representing leftward and rightward directions. Each half-circle is further divided into multiple bins, corresponding to incremental directional expansions. Initially, the agent is trained exclusively in backward movement, exploiting its natural tendencies. As training progresses, adjacent bins are sequentially introduced, expanding the range of movement directions the robot can reliably execute. Performance tracking determines the progression of the curriculum. Each bin is considered learned once the robot achieves at least 95% of the target velocity over an episode. Once the robot satisfies these conditions for a given bin, the curriculum introduces the next adjacent bin, continuing until the full circle of movement directions is covered. When selecting a bin, a specific direction within it is uniformly sampled to ensure full coverage.

B.1 Filter and reward ablations

We carry out ablation studies to investigate the impact of different filter setups and reward terms on the learning performance. The filter setups are compared in Figure 7, showing the average return and the number of falls for the One-Euro filter, the low-pass filter, and no filtering. The One-Euro filter achieves a balance between the high performance of no filtering and the low amount of falls of the low-pass filter.

Different compositions of reward terms are compared in Figure 8, illustrating the impact of additional reward penalties, especially on the number of falls during training, while maintaining the same target velocity. The additional penalties significantly reduce the amount of falls, achieving a 50% reduction compared to only using the tracking reward terms.

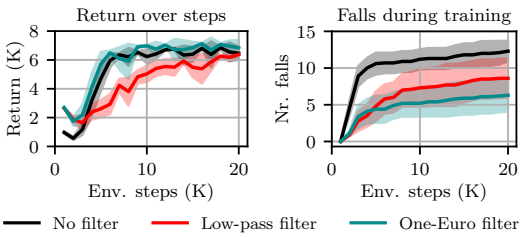

Figure 7: Ablation study of filter setups, illustrating their impact on the average return (left) and the number of falls (right).

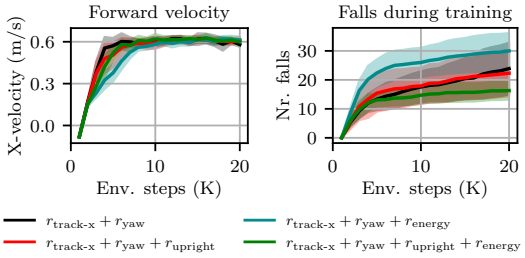

Figure 8: Ablation of reward terms, showing the impact on the achieved velocity (left) and the number of falls during training (right).