ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

REINFORCEMENT LEARNING FOR AGILE QUADRUPEDAL LOCOMOTION OVER CHALLENGING TERRAINS

ROBOTICS SEMESTER PROJECT AT EPFL XPLORE

Mathieu Schertenleib - Supervised by Auke Ijspeert

7th June 2024

Contents

1 Introduction **3**
- 1.1 Context . . . 3
- 1.2 Goals . . . 3
- 1.3 Framework . . . 3

2 Methods **6**
- 2.1 Markov Decision Process . . . 6
  - 2.1.1 Introduction . . . 6
  - 2.1.2 Observations . . . 7
  - 2.1.3 Actions . . . 8
  - 2.1.4 Policy neural network . . . 9
  - 2.1.5 Rewards . . . 9
  - 2.1.6 Domain randomization . . . 11
- 2.2 Terrain curriculum . . . 11

3 Results **13**
- 3.1 Introduction . . . 13
- 3.2 Flat terrain . . . 13
- 3.3 Challenging terrain . . . 18
  - 3.3.1 Velocity tracking . . . 18
  - 3.3.2 Position tracking . . . 22
  - 3.3.3 DAgger training . . . 25

4 Conclusion **28**

ABSTRACT

In recent years, Deep Reinforcement Learning has enabled the development of robust and agile locomotion policies that can be used on challenging terrains. Impressive results with parkour-like behavior have shown the potential of this field of research, as well as continuously opening new solutions to explore. This works aims to train in simulation several of such policies, experimenting across different parameters and exposing the best solutions, while exploring novel combinations of commands and sensors for a custom robot. It also validates design parameters for this quadruped by providing insightful data from trained policies, and ensures sim-to-real transfer by careful design and randomization of environment parameters.

CHAPTER 1

INTRODUCTION

1.1 CONTEXT

Deep reinforcement learning has emerged as a powerful tool for enabling autonomous agents to learn complex behaviors through interaction with their environments. In the realm of legged locomotion, DRL has shown promise in teaching quadruped robots to navigate diverse terrains robustly. These advancements are particularly significant for tasks requiring high levels of coordination and adaptability, such as parkour-like movements [1], [2]. An essential component in achieving robust legged locomotion is the effective use of sensory inputs. Among these, height scans from built elevation maps and depth input from sensors like depth cameras play a critical role in perceiving the environment, allowing the robot to understand and react to three-dimensional obstacles and terrain features [3], [4], [5], [6]. [7] introduces a different formulation for actor commands, using a target position rather than a velocity command. The resulting policy can traverse significantly harder terrains, being free to optimize its trajectory, and opening up new unexplored solutions.

1.2 GOALS

Chienpanze is a research project of the EPFL Xplore association. The goal is to develop a quadruped robot, with notably custom motors, and a fully self-developed structure, electronics and software. In this context, this project aims to develop a policy using Deep Reinforcement Learning for the robot. Since a physical prototype is not available yet, the goal is to develop a policy in simulation only, while taking measures to ensure the smallest possible sim-to-real gap for future work. The main target is to achieve close to state-of-the-art performance on challenging terrains, while exploring new methods, notably by integrating position tracking commands and depth input, and perform a comparison of the advantages and disadvantages of several observation spaces.

Due to the early nature of this quadruped research project, some technical aspects are not yet fully settled. For this reason, a secondary goal of this work is to gather insightful data that could help better understand the design constraints. This will mainly consist of the torques the robot requires, as well as the necessity for foot contact sensors, and the usability of a front-facing depth camera.

1.3 FRAMEWORK

From a technical point of view, the release of Isaac Gym in 2021 [8] and the results of [9] for massively parallel deep reinforcement learning enable developers to train a complete end-to-end policy in a few

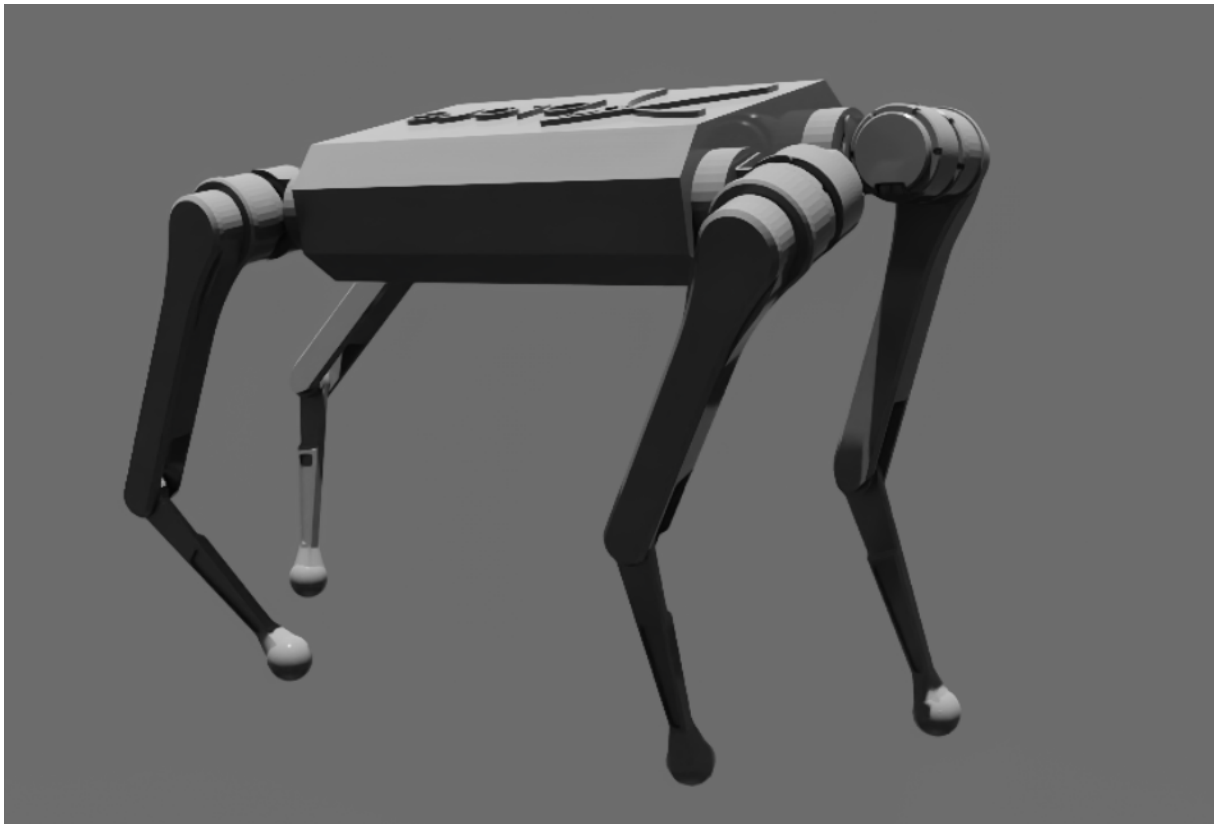


FIGURE 1.1
The Chienpanze robot. It is about 60-70 cm tall and weighs 35-40kg (fig. 1.1). Its joints are driven by brushless DC motors with a moderate reduction ratio, giving a relatively high output torque capability for its weight.

minutes or hours. More recently, the migration of Isaac Gym to Orbit [10] (built on top of NVIDIA Isaac Sim) allows for faster development iteration. For this reason, this work is done in Orbit, to ensure compatibility in the future years as well as ease of comprehension for people used to Isaac Gym.

CHAPTER 2

METHODS

2.1 MARKOV DECISION PROCESS

2.1.1 INTRODUCTION

A Markov Decision Process is a mathematical model describing the sequential decision-making process of an agent interacting with its environment, where the transitions are partly stochastic.

COMPONENTS

- State space S: the set of all possible states the environment can be in at any given time.
- Action space A: the set of all possible actions the agent can take.
- Transition probability function $P(s'|s, a)$: defines the probability of transitioning from state s to state s' given that the agent takes action a, and encapsulates the dynamics of the environment.
- Reward function $R(s, a, s')$: assigns a numerical reward after transitioning from state s to state s' due to action a.

PROCESS

- At each time step t, the agent extracts an observation o_t from the current state s_t.
- Based on the current state, the agent selects an action a_t according to a policy $\pi(a_t|o_t)$.
- The environment responds to the action by transitioning to a new state s_{t+1} according to the transition probability function $P(s'|s, a)$.
- The agent receives a reward r_t from the reward function $R(s_t, a_t, s_{t+1})$.

OBJECTIVE

The goal of the agent is to find an optimal policy π^* that maximizes the expected cumulative reward over time. The return G_t is defined as the sum of discounted future rewards:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

The optimal policy π^* is the one that maximizes the expected return from any starting state.

VALUE FUNCTIONS

- State value function $V^{\pi}(s)$: the expected return starting from state s and following policy π.

$$V^{\pi}(s) = \mathbb{E}\left[G_t | s_t = s, \pi\right]$$

- Action value function $Q^{\pi}(s, a)$: the expected return starting from state s, taking action a, and then following policy π.

$$Q^{\pi}(s, a) = \mathbb{E}\left[G_t | s_t = s, a_t = a, \pi\right]$$

DEEP REINFORCEMENT LEARNING

We use neural networks to learn the policy and value functions and find the optimal policy π^*, specifically using Proximal Policy Optimization [11], an on-policy policy gradient optimization method.

2.1.2 OBSERVATIONS

The observation space can be separated into four groups: proprioception, height scan, depth from the camera, and commands. See chapter 3 for a detailed explanation of which experiments use which observations. We also add uniform noise on observations, with ranges specified in table 2.3.

PROPRIOCEPTION

- Base linear velocities (in base space) $v_b \in \mathbb{R}^3$
- Base angular velocities (in base space) $\omega_b \in \mathbb{R}^3$
- Projected gravity vector (in base space) $g_b \in \mathbb{R}^3$
- Joint positions $q_t \in \mathbb{R}^{12}$
- Joint velocities $\dot{q}_t \in \mathbb{R}^{12}$
- Previous actions $a_{t-1} \in \mathbb{R}^{12}$
- Binary foot contacts $f_t \in \mathbb{R}^4$

Regarding the state of the base of the robot, we only include in the observation space the components that are not dependent on an absolute 2D position (x, y, yaw) in the world, namely because the commands are always relative to the robot's pose (see section 2.1.2). For the angular DOFs, we therefore only include an encoding of pitch and roll through the projected gravity vector, because yaw is absolute. No position DOFs are included in the observation space, because x and y are absolute, and z is ambiguous for rough terrain with abrupt vertical changes. Velocities in all 6 DOFs are included.

At the joint level, we include the states that are available on the robot, namely angles and velocities. We omit joint torques, because they can not be measured and would be hard to predict.

The previous actions are included to provide a rudimentary form of recurrence to the network, which should allow it to learn smooth gaits more easily.

Finally, we include binary foot contact indicators. Since we want to validate whether these sensors are necessary on the robot, we will perform experiments with and without these included (see chapter 3).

HEIGHT SCAN

The height scan consists of a rectangular grid of 17*7 height scan samples, which extends around the robot to the approximate range of positions the feet can reach on a relatively flat terrain. The samples extend further in front of the robot, in order to capture obstacles up to around 1m in front (mostly dictated by the maximum targeted gap width), see fig. 2.1.

DEPTH CAMERA

The depth image consists of 18*12 samples, with a field of view of 87° by 58°, see fig. 2.2. It is inclined 45° downwards. A depth camera pointing straight ahead would not be able to see the terrain close enough to the front feet: at around 60cm above the ground, a vertical field of view of 58° would give a minimum horizontal distance of $\tan(90° - 58°/2) * 60\text{cm} \approx 108\text{cm}$, which is too far for any policy that is not limited to walking straight ahead, and which would require much longer term memory in the policy network.

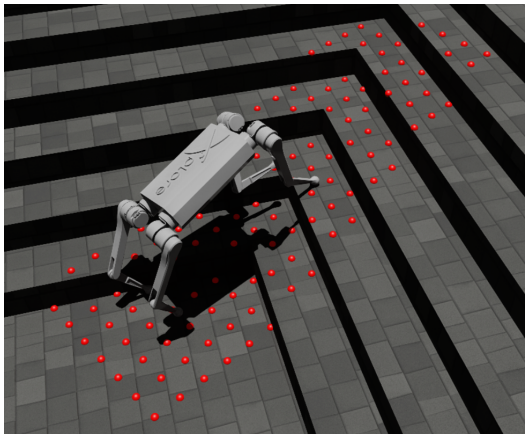


FIGURE 2.1
The 17*7 height scan samples around the robot.

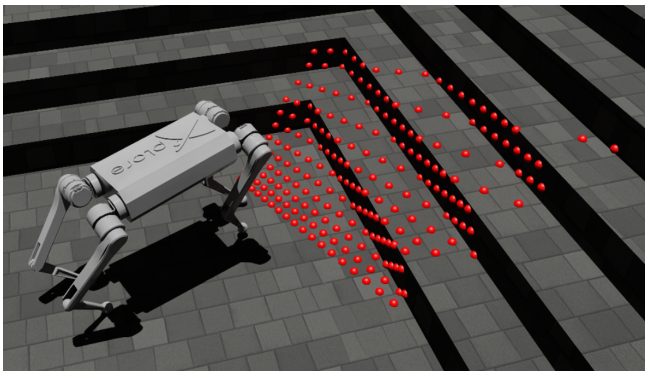


FIGURE 2.2
The 18*12 depth samples in front of the robot.

COMMANDS

We test two types of commands (see chapter 3 for more details):

- Velocity commands are specified as (v_x, v_y, ω_{yaw}) of the base frame (so v_x is forward/backward and v_y is left/right).
- Position commands are specified as the relative position (x, y, γ_{yaw}) of the goal in the base frame. The goal is the desired position of the base at the end of the episode. Additionally, we also include in the observations the time left to reach the goal, to have control over the velocity of the robot.

2.1.3 ACTIONS

The action space $\in \mathbb{R}^{12}$ corresponds to the desired joint positions. Linear outputs from the policy neural network (see section 2.1.4) are first scaled by 0.4, then clipped to [-1, 1], and finally scaled to the joint limits and added to the default joint angle:

- ± 0.5 radians for hip abduction-adduction
- ± 1 radian for hip flexion-extension

- ± 1 radian for knee flexion-extension

The desired joint positions q_d are updated at 50Hz and sent to a joint-level PD controller that converts them to joint torques.

$$\tau = K_p * (q_d - q) - K_d * \dot{q}$$

In simulation, this is an implicit continuous-time PD controller, with gains K_p and K_d sampled from the range detailed in table 2.2. The output torques are clipped to 40Nm for hip abduction-adduction and flexion-extension, and to 70Nm for knee flexion-extension, then applied to the articulation. Additionally, the angular velocities of the joints are limited to 15rad/s.

2.1.4 POLICY NEURAL NETWORK

We model the policy function using neural networks. For experiments without depth observations (see chapter 3 for more details), the network is a simple 3-layer MLP with hidden sizes [512, 256, 128], that takes as input the vector of concatenated observations, and outputs the action vector.

For experiments using depth, the agent needs to remember latent embeddings of the visual input for later steps, when the obstacles are out of its field of view and under its hind feet. For this, we concatenate all observations including depth samples, feed them first into a 1- or 2-layer recurrent GRU with 256 hidden units, then feed the output of the GRU to a 3-layer MLP of sizes [512, 256, 128]. Defining the number of GRU layers (1 or 2) is part of the experiments (chapter 3).

2.1.5 REWARDS

The reward function design process is progressive. To validate the initial code implementation of the MDP in the simulator as well as simulation parameters, initial training was done with only velocity tracking rewards and no penalties. The velocity tracking reward used is an exponential (Gaussian) kernel, with a sstandard deviation σ of 0.5:

$$r_{v_{xy}} = \alpha_1 * \exp\left(\frac{||v_{xy_d} - v_{xy}||}{\sigma^2}\right) \qquad r_{\omega_z} = \alpha_2 * \exp\left(\frac{||\omega_{zd} - \omega_z||}{\sigma^2}\right)$$

Once it was validated that the training converged to a stable policy with good tracking, the first penalty added was mechanical energy, since it has been shown that minimizing mechanical energy leads to the emergence of smoother gaits [12]. It also reduces the torques applied and the joint velocities, which is particularly crucial for the transfer to the real system. We choose to add a single term combining both torque and velocity rather than two separate terms, because it removes a tuning parameter in the tuning process, without sacrificing effectiveness of the penalty.

$$r_{\text{energy}} = -\alpha_3 * \sum_{i=1}^{12} (\tau_i * \dot{q}_i)^2$$

Once the energy term was added, the resulting gait was smoother and the applied torques were lower, but the agent still moved its feet very quickly, tapping the ground rather than taking long steps. This is due to the velocity tracking reward: in order to maximize the tracking reward, the agent must keep its base at a constant linear velocity, which does not lead to a natural gait (the torso velocity of an animal oscillates while walking). Therefore, we add a term rewarding feet air time, ensuring the agent takes longer steps. This proves to be a particularly crucial element in obtaining a natural-looking gait. The reward at time t is

defined as the sum of the air times of the last steps of the N_f feet that just touched the ground at time t. The reward is set to zero if the commands are small, meaning the agent should not take a step in this case.

$$r_{\text{air time}} = \alpha_4 * \sum_{i=1}^{N_f} (t_f - 0.5)$$

This current reward gives good results, however the base tends to oscillate up and down a lot, and the policy tends to converge to a bound or pronk gait even at low speeds. To mitigate this, we penalize vertical linear velocity as well as pitch and roll rates of the base.

$$r_{v_z} = -\alpha_5 * v_z^2 \qquad r_{\omega_{xy}} = -\alpha_6 * ||\omega_{xy}||^2$$

Finally, we want to minimize the number of collisions of the legs with the ground (especially on rough terrain). Therefore, we add a term penalizing the number of contacts N_c of the thigh or shank with the ground that exceed 1N of force.

$$r_{\text{contacts}} = \alpha_7 * N_c$$

For position tracking commands, we remove the velocity tracking terms $r_{v_{xy}}$ and r_{ω_z} and add three new terms, following the results from [7]. First, we add a term minimizing the distance to the goal at the end of the command period. More specifically, if the position commands are resampled at interval T seconds, we give the following reward for the last T_r seconds of this interval, where x_g, y_g and γ_g are the goal x, y and yaw expressed in the base frame. It is important to give the reward only at the end of the episode, such that the agent is free to optimize its trajectory (both in space and time) without penalties. This naturally removes some of the unwanted tapping behavior encountered when introducing velocity tracking commands.

$$r_{\text{final position}} = \alpha_8 * \frac{1}{1 + 0.5 * (x_g^2 + y_g^2 + \gamma_g^2)}$$

Since the reward is sparse, it may be difficult for the agent to explore and start walking towards the goal by itself. For this reason, we add a direction bias rewarding movement in the direction of the goal. However, this term is removed after some training iterations (specifically once the position tracking reward is high enough), to not introduce a bias in the final policy (which leads to the problems of velocity tracking again). In the term below, v_b is the (v_x, v_y) velocity of the base (in base frame) and x_g is the (x, y) position of the goal in base frame.

$$r_{\text{bias}} = \alpha_9 * \min(v_b \cdot \frac{x_g}{||x_g||}, 1.0)$$

Finally, since PPO minimizes the discounted sum of rewards, it is beneficial for the agent to wait without moving (to minimize penalties) and sprint towards the goal at the end of the command period, to push penalties as ar as possible in the future. This leads to unnecessary high torques, and increases the risk of falling over. To mitigate this issue, we add a penalty if the agent waits still while being far from the goal (v_b is the (v_x, v_y) linear velocity of the base, and d_g is the distance to the goal).

$$r_{\text{stall}} = -\alpha_{10} * \begin{cases} 1 & \text{if } ||v_b|| < 0.1 \text{ m/s and } d_g > 0.5\text{m} \\ 0 & \text{else} \end{cases}$$

The coefficients α_i are specified in table 2.1.

Coefficient	Value
α_1	1.0
α_2	1.0
α_3	-5e-6
α_4	1.0
α_5	2.0
α_6	0.05
α_7	1.0
α_8	12.0
α_9	1.0
α_{10}	1.0

TABLE 2.1
Reward function coefficients.

2.1.6 DOMAIN RANDOMIZATION

In order to improve the robustness of the learned policy to environmental variations, we randomize several simulation variables, detailed in table 2.2. Additionally, on each episode reset the robots are given a random initial velocity (from the ranges [-0.5, 0.5] m/s or [-0.5, 0.5] rad/s on each axis), and their joints are reset to the default position multiplied by a factor sampled from the range [0.5, 1.5].

Variable	Range	Units
Static friction	[0.6, 1.0]	-
Dynamic friction	[0.4, 0.8]	-
Added base mass	[-5, 10]	kg
Joint stiffness K_p	[40, 80]	Nm/rad
Joint damping K_d	[0.8, 2]	Nms/rad

TABLE 2.2
Randomized environmental variables and their ranges.

We also add uniform noise on the observations, with ranges detailed in table 2.3

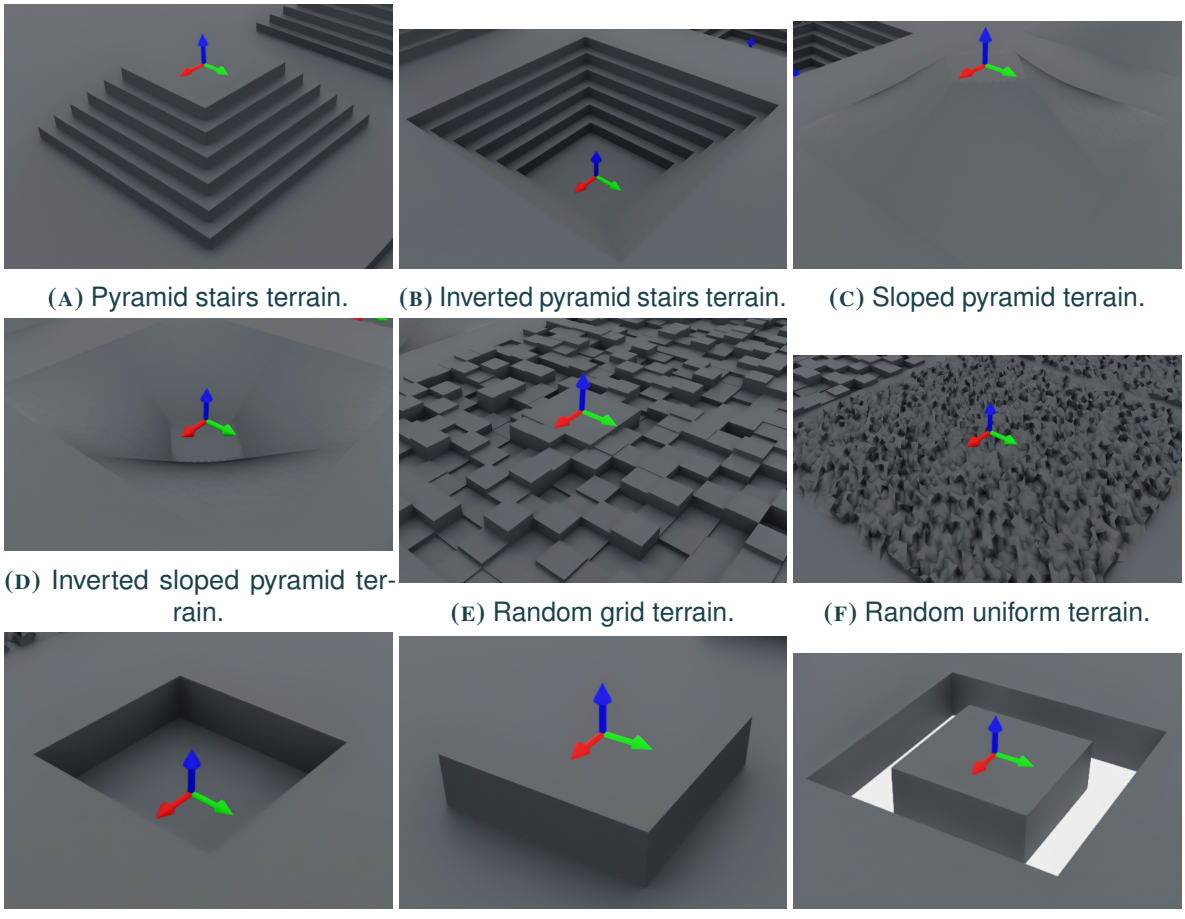
2.2 TERRAIN CURRICULUM

The simulation environment consists of a grid of 8*8 meters square terrains, where each column is one of 9 different terrain types shown on fig. 2.3, and each row is a set of 16 linearly increasing difficulties for that terrain type (fig. 2.3 shows the maximum difficulty level). The variety of terrains aims to mimic real-world situations, namely stairs, slopes, rough uneven ground, high steps and gaps in the ground, and should ensure robustness of the trained policy over a wide range of scenarios.

We always train with 4096 agents in parallel. They are uniformly distributed over all terrain types. As training starts, the agents all spawn on the easiest level of their respective terrains (row 0). When agents walk more than 3 meters from their starting position (for velocity tracking commands), or when they end up less than 50cm from the goal (for position tracking commands), they are promoted to harder terrains. If they fall, or if they walk less than half of the distance required by their commands, the are moved to easier terrains.

Component	Noise range	Units
Base linear velocity	[-0.1, 0.1]	m/s
Base angular velocity	[-0.2, 0.2]	rad/s
Projected gravity	[-0.05, 0.05]	m/s^2
Joint positions	[-0.02, 0.02]	rad
Joint velocities	[-1.0, 1.0]	rad/s
Previous actions	-	-
Foot contacts	-	-
Height scan	[-0.1, 0.1] (see chapter 3)	m
Depth	[-0.04, 0.04]	m
Commands	-	-

TABLE 2.3
Added uniform noise on observations.



(A) Pyramid stairs terrain. (B) Inverted pyramid stairs terrain. (C) Sloped pyramid terrain.

(D) Inverted sloped pyramid terrain. (E) Random grid terrain. (F) Random uniform terrain.

(G) Pit terrain. (H) Box terrain. (I) Gap terrain.

FIGURE 2.3
Terrain types, seen at maximum difficulty. The agents spawn at the terrain origin in the center.

CHAPTER 3

RESULTS

3.1 INTRODUCTION

With a complete environment and a well-defined MDP, we can start preparing experiments to answer our research questions. There are three groups of experiments:

- Policies trained with RL on flat terrain, allowing for a more precise analysis of gaits, torques or energy, and a comparison between velocity tracking and position tracking commands.
- Policies trained with RL on challenging terrains, allowing for a comparison of performance between velocity tracking and position tracking commands, as well as observation spaces.
- Policies trained with DAgger from a good-performing expert trained with RL, allowing for an analysis of performance when the depth camera is used.

Each individual experiment (a single training run, for a given set of parameters) is repeated 5 times. On all training curves in this document, we always plot the mean of the 5 experiments as a line, and draw the 100% percentage interval as a color patch. The choice of displaying the 100% percentage interval (which represents minimum and maximum values), contrary to a more common standard deviation, is because we have only 5 runs of each experiment, and it is more important to know the range limit of training results. Additionally, the variations on the curves are often strongly asymmetrical. For high-noise curves such as mean episode rewards, the plots are smoothed using a moving average of 30 samples.

The PPO hyperparameters are defined in table 3.1.

3.2 FLAT TERRAIN

We train two policies, the first one with velocity tracking commands and the second one with position tracking commands. The policies are each trained using PPO for 1000 learning iterations, with the hyperparameters defined in table 3.1.

Episodes last for 300 steps, and commands are sampled once at the beginning of the episode. The velocity commands are sampled from the range [-1, 1] m/s for v_x and v_y, and [-2, 2] rad/s for ω_z. For position tracking, the goal is sampled uniformly around the starting position of the agent, in the range [-4, 4] meters, and the goal yaw is randomly distributed over 360°.

If we look at fig. 3.1, we immediately notice that the position tracking policy takes 2-3 times more iterations than velocity tracking to converge to a stable gait, which is indicated by the episode length reaching its maximum value of 300. This can be explained by the sparsity of the reward, making it a

Environments	4096
Steps per environment	24
Batch size	98304 = 4096 * 24
Mini-batch size	24576 = 4096 * 6
Epochs	5
Clip range	0.2
Entropy coefficient	0.005
Discount factor γ	0.99
GAE discount factor λ	0.95
Desired kl	0.01
Max gradient norm	1.0

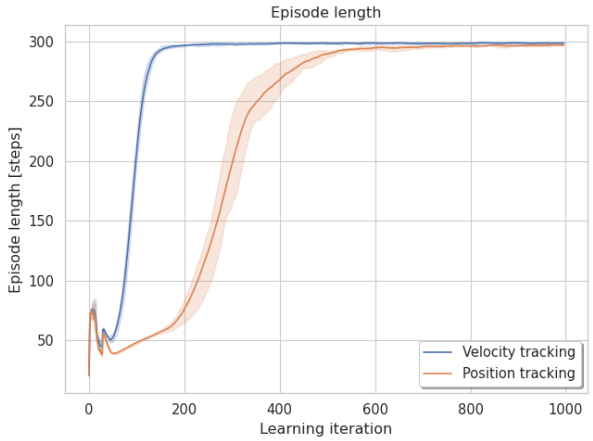
TABLE 3.1
PPO hyper-parameters. Similarly to [9] and [13], the learning rate is adaptive, based on KL-divergence.

harder optimization problem. Additionally, the direction bias term, introduced to mitigate this problem, only rewards linear movement of the base, without tracking yaw. For this reason, the agent has to learn to orient itself in the right direction on top of learning to walk. In contrary, for velocity tracking, the reward covers all three 2D pose DOFs, and is provided continuously, making it way easier for the optimizer to pick up on small improvements in velocity tracking.

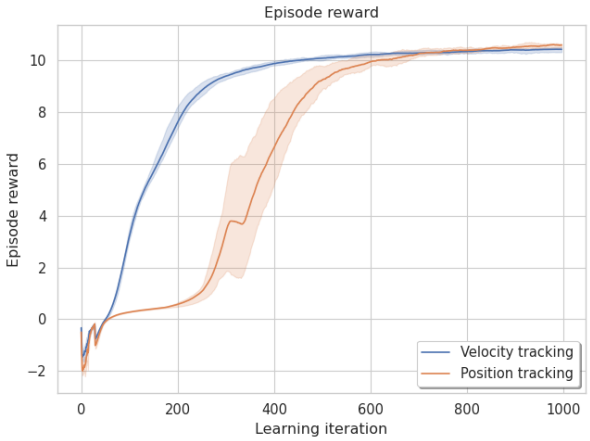
By looking at fig. 3.2, we get confirmation that velocity tracking leads to fast convergence. We note that both tracking terms (both are Gaussian exponentials) reach about 93% of their maximum value. By looking at the simulated resulting policy, tracking is almost perfect visually. The gap to 100% is explained by very small velocity oscillations, as well as the short moments at each command resampling where the tracking error suddenly increases until the agent catches up.

If we now look at fig. 3.3, we first notice a higher general variance across runs, indicated by a bigger [min, max] interval (light blue patch). For this experiment, the direction bias reward is turned off after 300 steps, when the position tracking reward has reached a sufficiently high value. This deactivation of the direction bias can also be noticed on the total reward plot of fig. 3.1, with a small dip at step 300. The stall penalty is generally quite small; it seems like any amount of significance compared to the other terms is sufficient to refrain the agent from waiting while far from the goal. The important dip around step 400 is due to the agent learning to stay upright and move towards the goal towards the end of the command resampling period. It then learns to start walking straight away.

Regarding torques, we can see on fig. 3.4 that the continuous RMS torques follow an expected distribution, meaning the hip abduction-adduction is lowest at 7.5Nm, hip flexion-extension is at 14Nm, and knee flexion-extension is at 19Nm. This is well within the torque range allowed by our motors, and therefore validates the sizing of the actuators with an actual locomotion task.

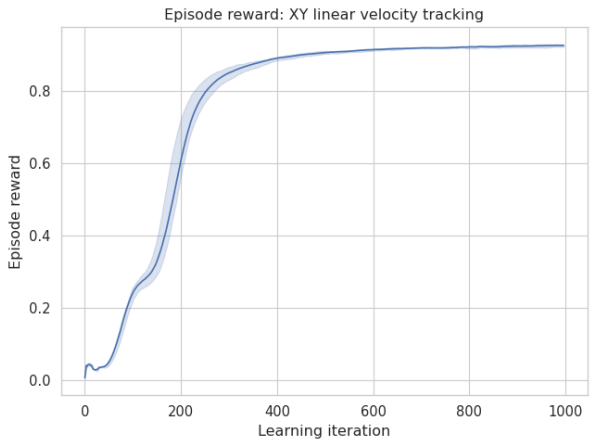


(A) Mean episode length.

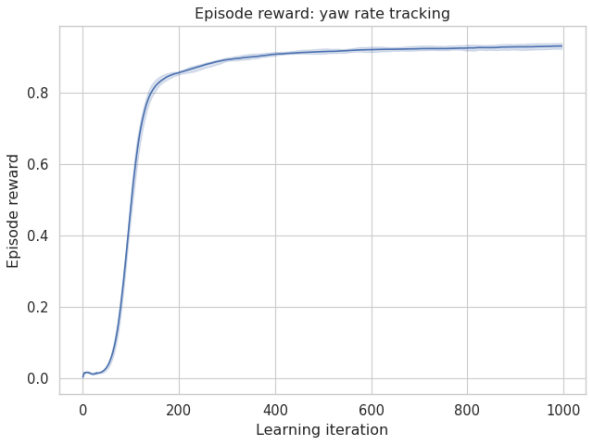


(B) Mean episode rewards.

FIGURE 3.1
Comparison of training curves for flat terrain policies, for velocity tracking and position tracking commands.

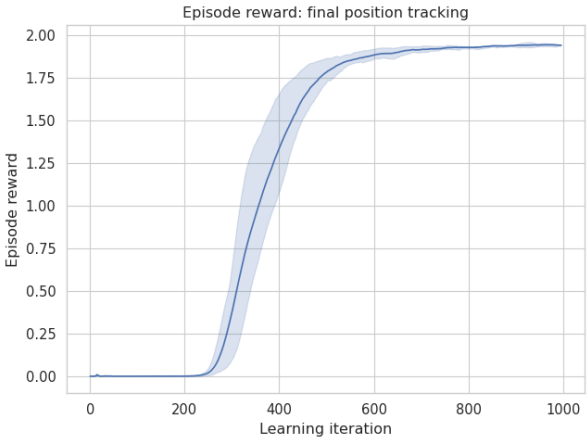


(A) Mean episode rewards: linear velocity tracking.

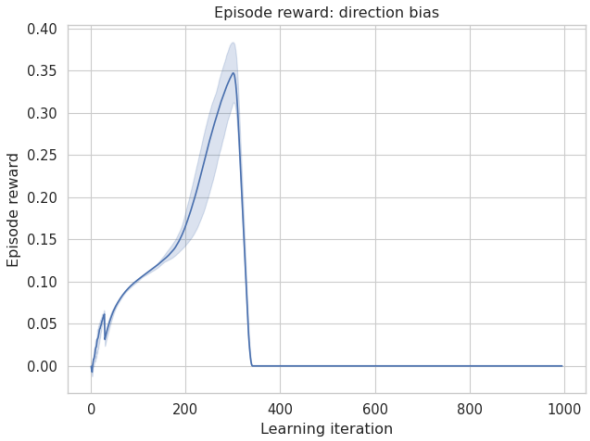


(B) Mean episode rewards: angular velocity tracking.

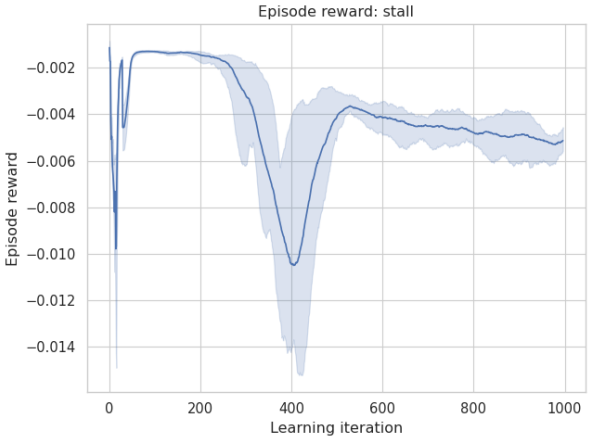
FIGURE 3.2
Detailed training curves of rewards for the velocity-tracking flat terrain policy.



(A) Mean episode rewards: final position tracking.



(B) Mean episode rewards: direction bias.



(C) Mean episode rewards: stall penalty.

FIGURE 3.3
Detailed training curves of rewards for the position-tracking flat terrain policy.

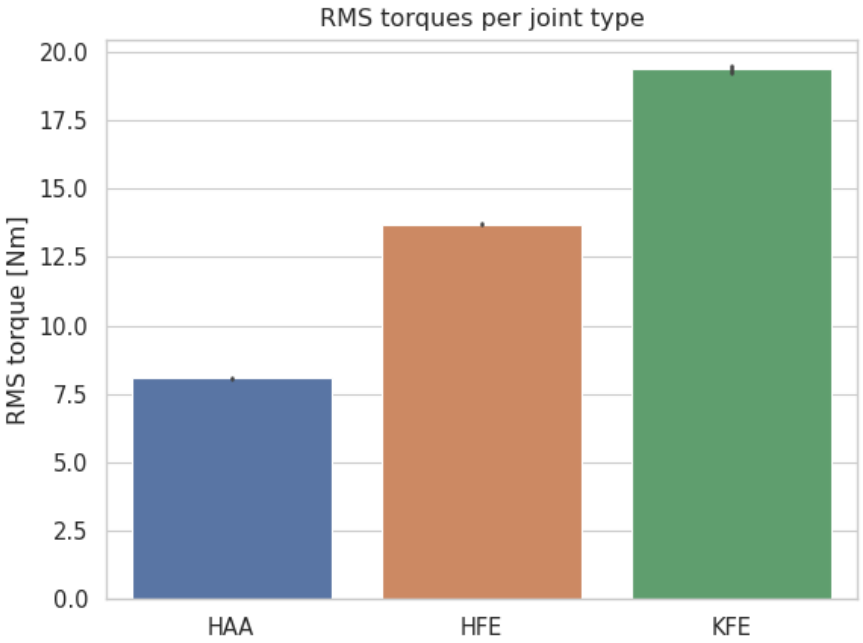


FIGURE 3.4
RMS torques while walking at 1m/s. HAA: hip abduction-adduction, HFE: hip flexion-extension, KFE: knee flexion-extension.

3.3 CHALLENGING TERRAIN

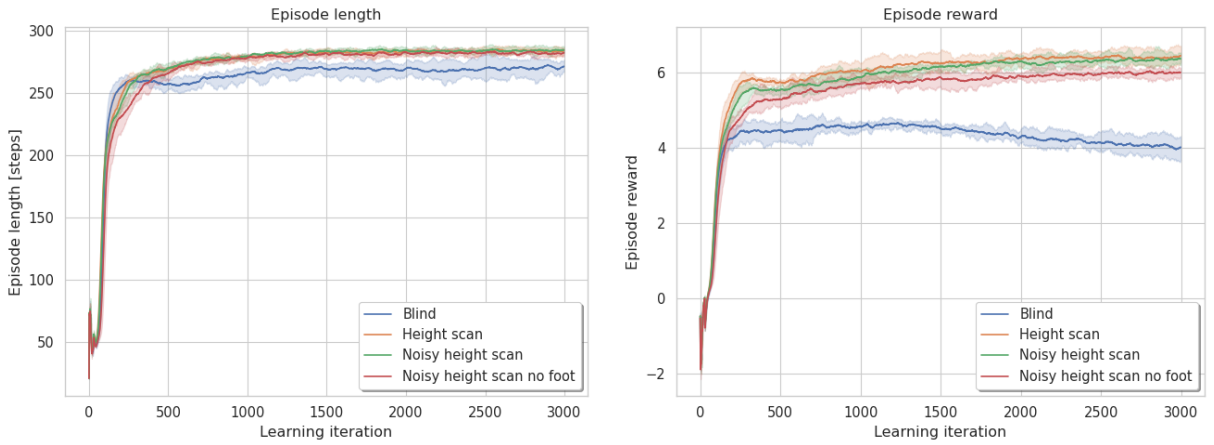
For both velocity and position tracking, we train four policies each with PPO, for 3000 learning iterations, with the hyperparameters defined in table 3.1. The differences between the four environments are the following:

- Blind: only proprioception is used as the observation space.
- Height scan: the observation space contains proprioception as well as a noiseless height scan.
- Noisy height scan: the observation space contains proprioception as well as a noisy height scan (the noise level is indicated in table 2.3).
- Noisy height scan no foot: same as Noisy height scan, but the binary foot contact sensors were removed from the observation space.

Episodes last for 300 steps, and commands are sampled once at the beginning of the episode. For these experiments, we simplify the problem by limiting the range of commands. This is also necessary to ensure a fair comparison between velocity and position tracking. The initial heading of the robot is in the range [-45°, 45°]. The velocity commands are sampled from the range [0.6, 1.0] m/s for v_x, always zero for v_y, and [-2, 2] rad/s for ω_z. For position tracking, the goal is sampled on a relatively flat patch of ground from 2.7 to 4 meters in front of the robot, over the whole width of the terrain (8 meters). The goal yaw is in the direction of the goal, to simplify the yaw tracking problem.

3.3.1 VELOCITY TRACKING

If we look at fig. 3.5, we observe that the general behaviour is similar for all policies having access to height scan. For the blind policy, it falls more often (indicated by a lower mean episode length) and has generally lower rewards due to less accurate velocity tracking.



(A) Mean episode length.

(B) Mean episode rewards.

FIGURE 3.5
Comparison of training curves for challenging terrains, for velocity tracking commands.

In order to assess the performance of the trained policy, we plot the mean terrain levels during training. Due to the curriculum, the mean terrain level at the end of the training is a direct measure of the maximum difficulty level that can be solved by the policy in a stable manner.

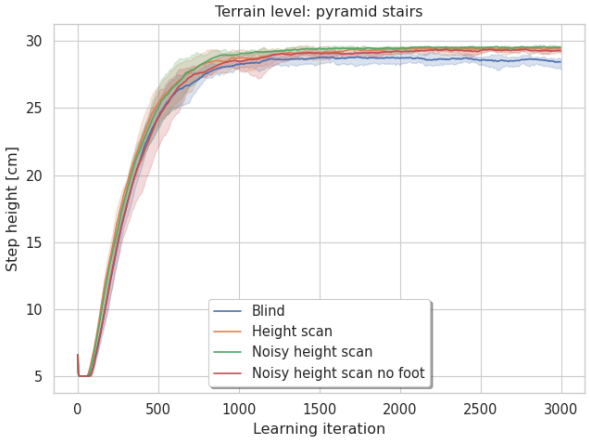
On challenging terrain, the performance is highly dependent on the type of terrain. We can group pyramid stairs, slope pyramid (fig. 3.6) and box (fig. 3.7) together, because they only require the agent to walk

down, which is significantly easier than going up. We note that the agent can almost solve the maximum level for all four experiments, with the blind one having a bit more trouble on the box. Since it can not see the drop from the box, it learns a robust reception method once it detects that it is falling.

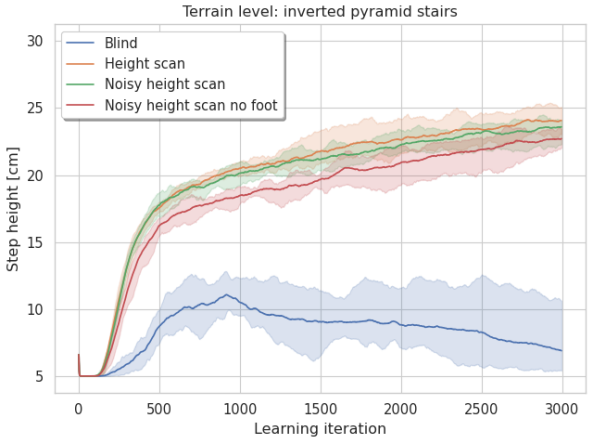
Inverted pyramid stairs and slope, random grid and random uniform (fig. 3.6) all have the same characteristics. Height scan and noisy height scan perform the best with very little variability between the two (which is quite impressive with a $\pm$10cm noise!), followed by noisy height scan without foot contacts. The blind policy is generally stopped by the limitations of having no exteroception: on stairs or random grid, it can not learn to lift the feet higher up, because this behavior would hinder its rewards on other terrains. Interestingly, for the uniform noise terrain, it completely fails for some training runs. The leading hypothesis is that the high frequency noise is too hard to navigate by simply carefully stepping, since it can cause abrupt changes in balance. On the random uniform terrain, the policy without foot contacts is also having trouble, because it can not use its feet to sense the ground and mitigate the other noisy observations, like the other policies do.

On fig. 3.7 we can see the terrain levels for the three special obstacle types which are not common in other works, namely the pit, box and gap. We already discussed the box above, so it is interesting to take a look into the simulation for the pit and gap. We can see on fig. 3.8 the noisy height scan policy climbing out of a pit. This type of behavior tends to appear late in the training, because the agent needs to learn a multi-step process. First, it moves its front right foot as high as possible, to put it out of the pit. We note that this is always the resulting policy for the best performing runs, so the maximum traversable pit depth is limited by the range of motion of the foot. Robots that are lighter and proportionally stronger could learn to jump out of the pit, but this seems to be unfeasible with Chienpanze. Secondly, it lifts itself up and puts the second front foot down on the edge. Thirdly, it gives a strong impulse on the ground, while lifting one of its rear legs to also step out. Finally, it fully lifts itself up and brings the fourth foot out of the pit.

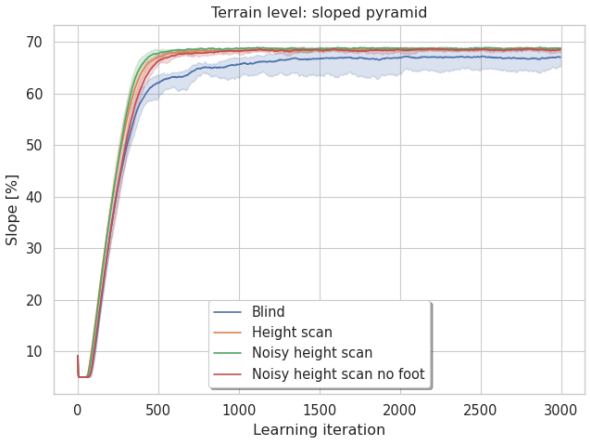
Concerning the gap, easier levels can be traversed by simply walking over it. We can see that the blind policy is limited to those levels, at around 10cm gaps. For harder gaps, the agent has to learn to accept some reduced velocity tracking rewards while it positions itself and does a big step over the gap, akin to a leap for some exceptionally performant cases (see fig. 3.9). Finally, we observe that the policies with noisy height scans are surprisingly better than the noiseless one. This is probably due to a more cautious policy when the scan has noise, leading to the agent stepping less close to the edge and being more cautious about balance.



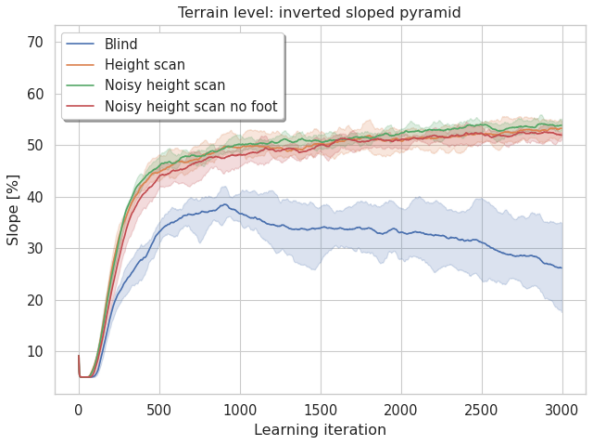
(A) Pyramid stairs terrain.



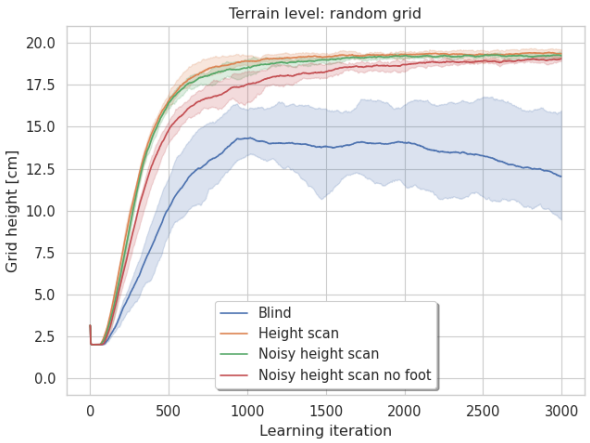
(B) Inverted pyramid stairs terrain.



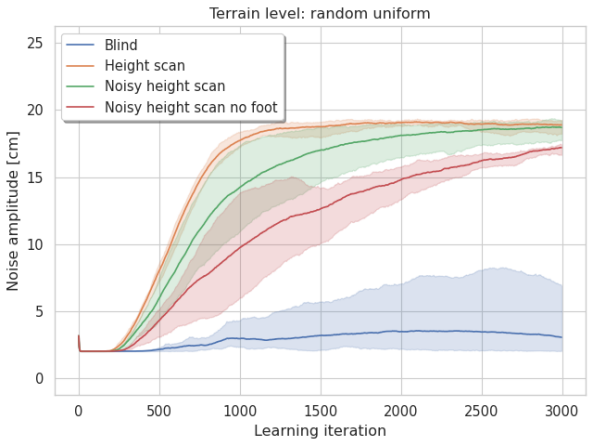
(C) Sloped pyramid terrain.



(D) Inverted sloped pyramid terrain.

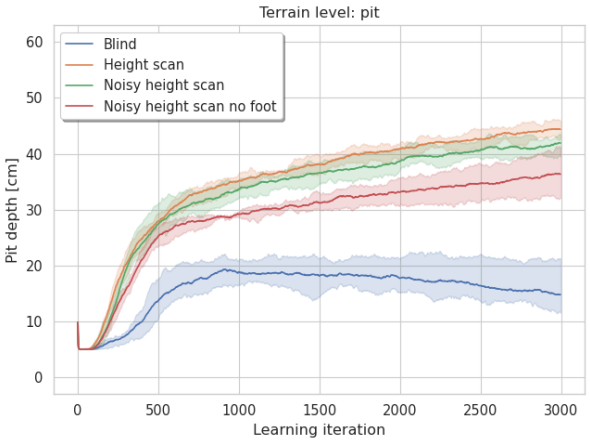


(E) Random grid terrain.

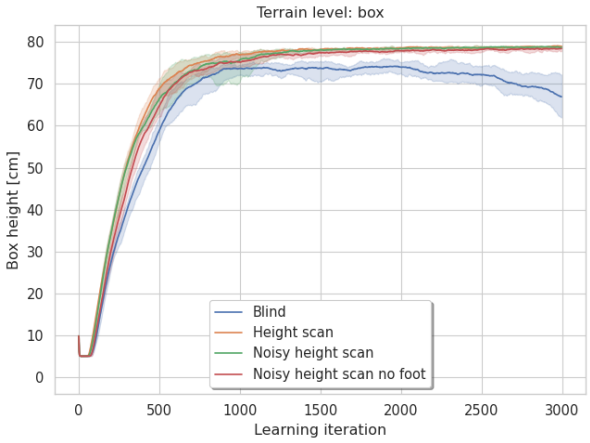


(F) Uniform random noise terrain.

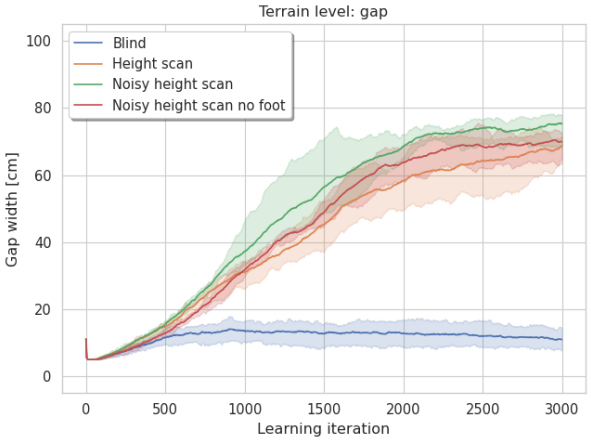
FIGURE 3.6
Terrain levels during training, for velocity tracking commands.



(A) Pit terrain.



(B) Box terrain.



(C) Gap terrain.

FIGURE 3.7
Terrain levels during training for special obstacles, for velocity tracking commands.

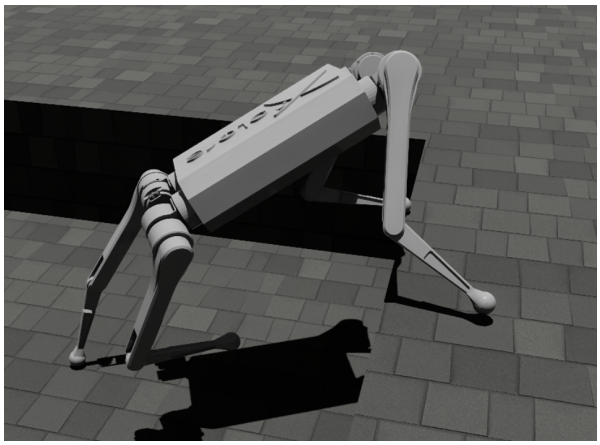


FIGURE 3.8
Velocity tracking policy climbing out of a pit.

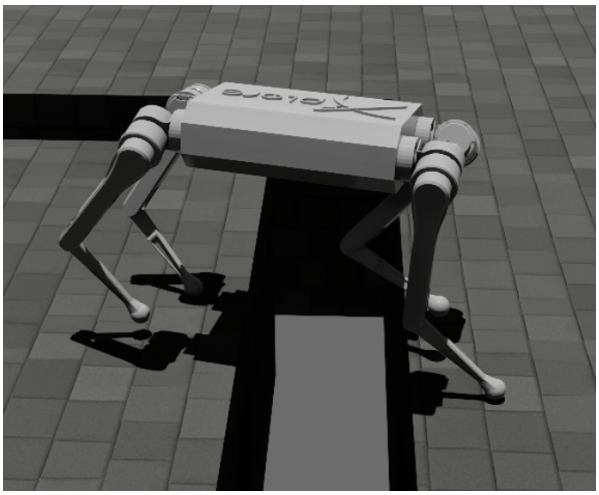
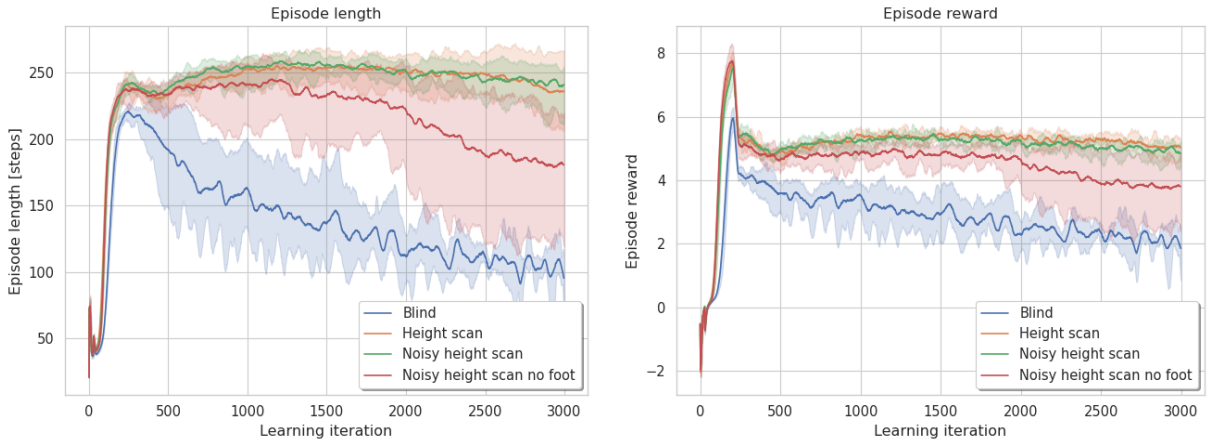


FIGURE 3.9
Velocity tracking policy walking over a gap.

3.3.2 POSITION TRACKING

For position tracking, all discussions regarding velocity tracking still hold. The tendencies noted above are still present here, but the main difference is a greatly increased variance between runs, as well as a general lower achieved final level (fig. 3.11). The two height scan policies with foot contacts are still head-to-head on the lead, but this time the policy without foot contacts tends to fall behind more often than velocity tracking.

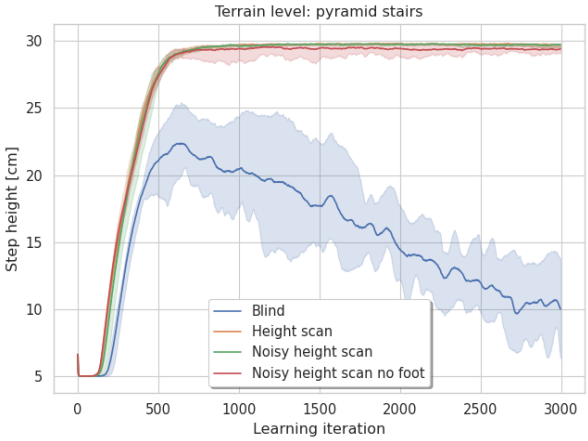
More importantly, the blind policy tends to regress during training as seen on fig. 3.10, fig. 3.11 and fig. 3.12. The main hypothesis is that the policy first learns to walk towards the goal on the easiest terrains, which gives it quite high rewards. It also learns to go down slopes and stairs well. Then, as training progresses, it arrives at harder terrains, which can not be solved and lead to lower position tracking rewards. Furthermore, the set of skills required to solve all terrains is quite diverse. For this reason, it is possible that the agent has to give up terrain performance, to try to increase its rewards again. Trying to solve larger gaps or deeper pits, for example, require tradeoffs whose effects must be minimized later.



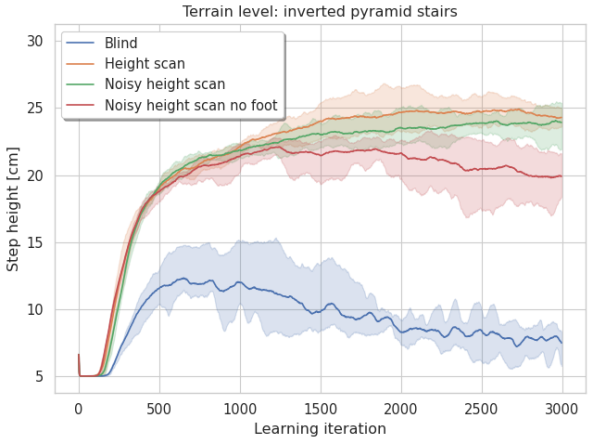
(A) Mean episode length.

(B) Mean episode rewards.

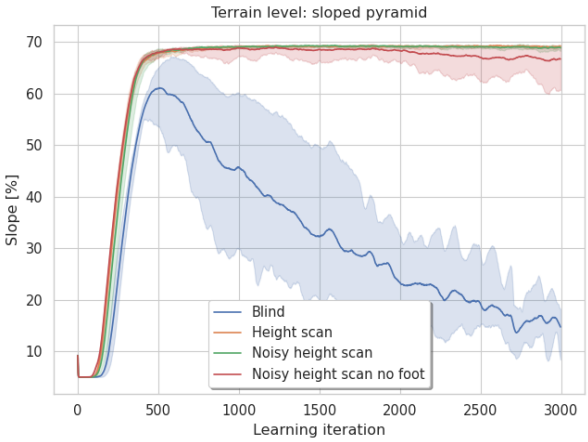
FIGURE 3.10
Comparison of training curves for challenging terrains, for position tracking commands.



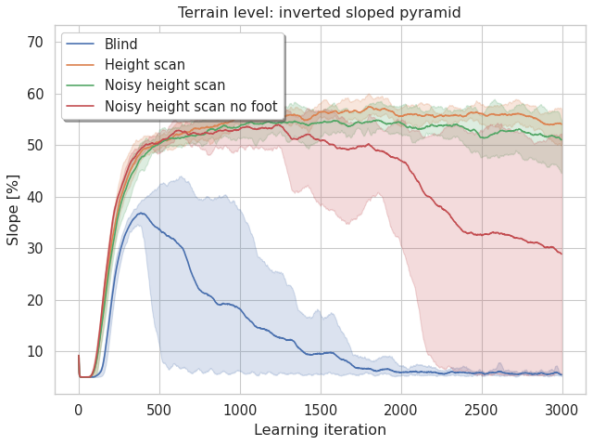
(A) Pyramid stairs terrain.



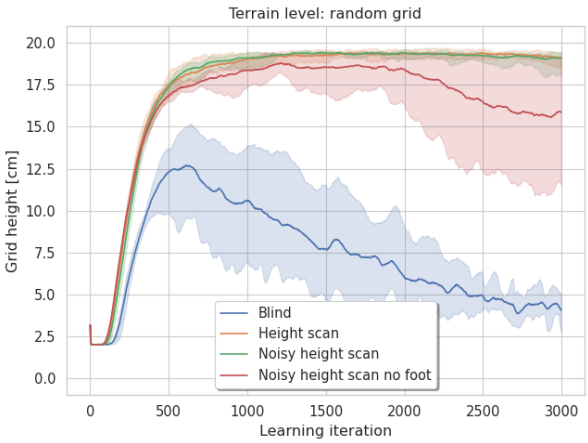
(B) Inverted pyramid stairs terrain.



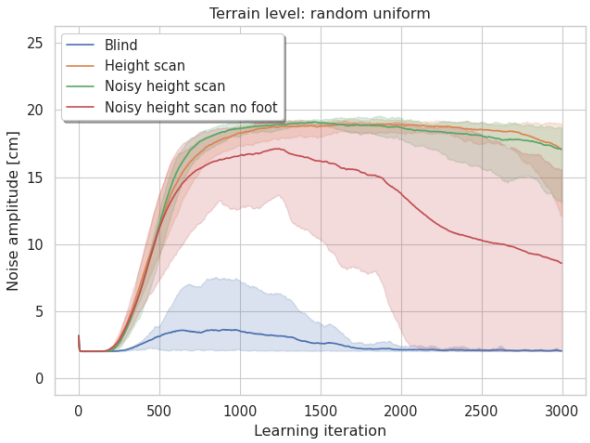
(C) Sloped pyramid terrain.



(D) Inverted sloped pyramid terrain.

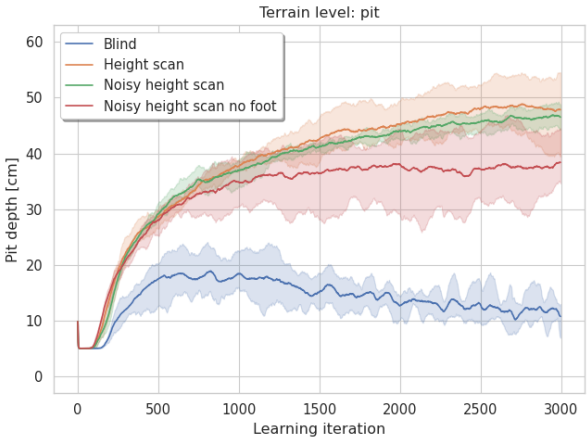


(E) Random grid terrain.

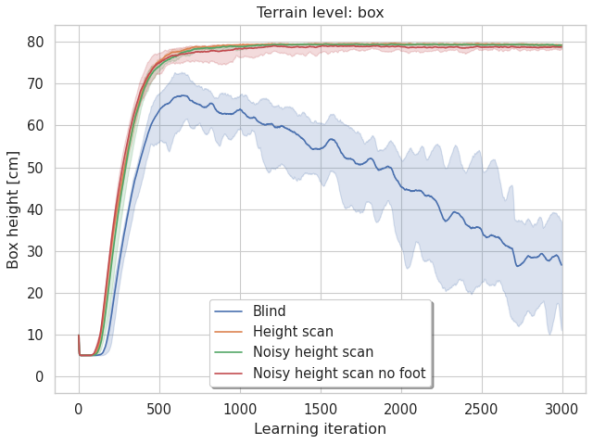


(F) Uniform random noise terrain.

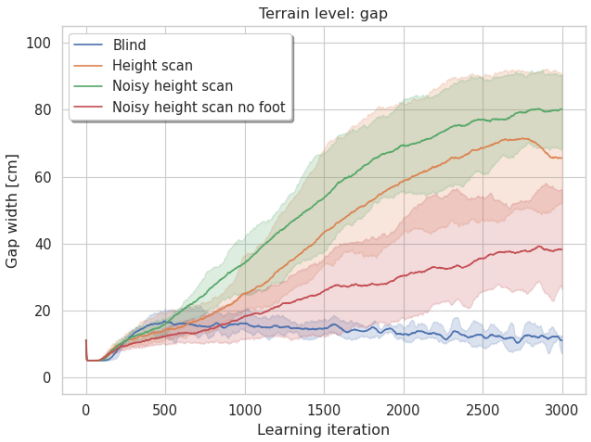
FIGURE 3.11
Terrain levels during training, for position tracking commands.



(A) Pit terrain.



(B) Box terrain.



(C) Gap terrain.

FIGURE 3.12
Terrain levels during training for special obstacles, for position tracking commands.

3.3.3 DAGGER TRAINING

All policies in previous experiments used height scan as exteroceptive input. In order to train a policy that could be deployed on the real robot, we need to use proprioception and the depth camera only, using a teacher-student method. To achieve this, we start from a good-performing teacher policy that uses noiseless height scan inputs, and we use supervised learning with the DAgger [14] (Dataset Aggregation) algorithm.

For this part of the training, 256 environments are simulated in parallel. The terrain curriculum is deactivated and the agents are uniformly distributed on all terrain types and difficulties. For each learning iteration, the student policy is unrolled in the simulator for 24 steps. At each step, we also query the teacher policy for good actions, and store the teacher-student action pairs in a buffer. After unrolling, we use standard supervised learning (namely Adam optimizer with learning rate 0.001 and backpropagation) with the teacher-student action batch, stepping the student policy towards better actions. We train for 5000 learning iterations, for both velocity tracking commands and position tracking commands. We can see on fig. 3.14 that velocity tracking performs significantly better than position tracking. This is due to the harder learning task and observation-action mapping, as well as a sometimes more agressive position tracking velocity, when it comes to surpassing obstacles, which leads to easier falls, especially for the student at the start of the supervised training.

From a qualitative standpoint, the trained student reaches performance levels comparable to the teacher on most terrains. The most notable exception is the gap, where the student seems to have more difficulties learning the particular observation-action mapping, while having only an estimate of terrain under its feet with the recurrent depth. Nonetheless, the policy is still able to achieve impressive results, such as a 70cm gap on fig. 3.13.

Furthermore, we see that adding a second GRU layer does not improve performance. On the contrary, it does not improve terrain traversability in our tests and slightly slows down training.

FIGURE 3.13
Robot leaping over a gap of 70cm.

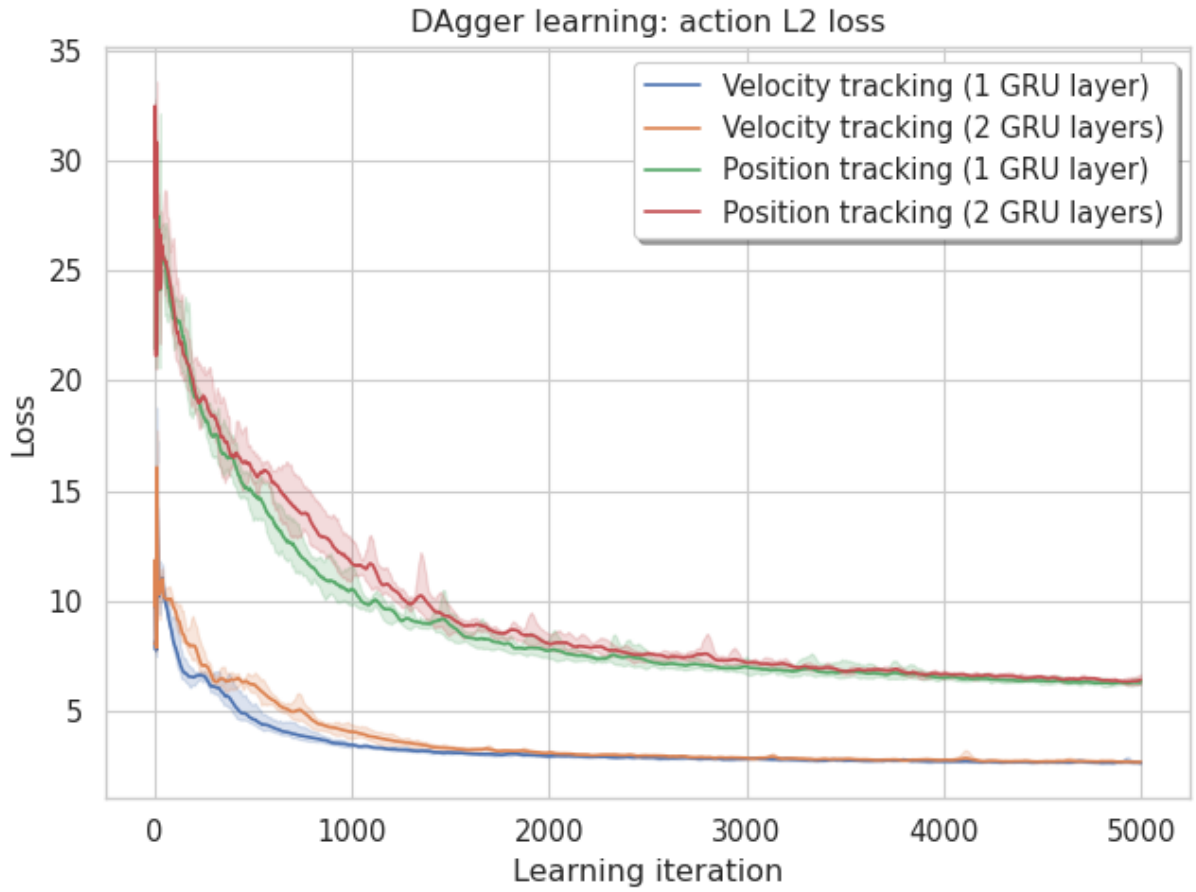


FIGURE 3.14
Action L2 loss during DAgger training.

CHAPTER 4

CONCLUSION

We obtain two main policies that could be transferred realistically to the real robot. First, the policy taking noisy height scan input could be implemented if the higher-level locomotion controller (which sends velocity or position tracking commands) builds an elevation map for path planning. The high noise amplitude of $\pm$10cm make it realistically feasible. We note that this would be the highest performing policy in terms of robustness as well as terrain traversability. However, sensor drift during the construction of the height map could lead to dramatic results if not compensated properly.

The second usable policy is the one using depth input. The parameters and noise level used correspond to that of the Intel D435 camera, and therefore should work without modification. This policy is slightly less performant than the height scan ones, but has the main advantage of not requiring a height map, as well as being mostly safe from sensor drift, since it does not require any information on the absolute position of the robot.

One aspect that was not treated in this work is the integration of an RMA architecture [15], which adds a learned environmental factor encoder to the policy, increasing robustness to external factors such as friction, extra mass or motor strength. Such an architecture is probably necessary to achieve state-of-the-art robust locomotion on hardware. However, since we use a high range of domain randomization parameters, such as an added mass up to 10kg as well as large friction coefficient ranges, and we add high noise to observations, we argue that it is probably doable to transfer the current policy as-is and have a working system.

To conclude, we successfully developed several policies for quadrupedal locomotion using DRL. We achieve robust locomotion over a variety of challenging terrains, notably 30cm stairs, 60% slopes, 20cm uniform noise, 50cm pits and 80cm gaps. We were also able to identify the torque usage, as well as the necessity of foot sensors and the usability of a depth camera for locomotion over hard terrains, and answered all our research questions.

BIBLIOGRAPHY

[1] Ziwen Zhuang et al. *Robot Parkour Learning*. 2023. arXiv: 2309.05665 [cs.RO]. URL: https://robot-parkour.github.io/.

[2] Xuxin Cheng et al. *Extreme Parkour with Legged Robots*. 2023. arXiv: 2309.14341 [cs.RO].

[3] David Hoeller et al. *ANYmal Parkour: Learning Agile Navigation for Quadrupedal Robots*. 2023. arXiv: 2306.14874 [cs.RO].

[4] Takahiro Miki et al. 'Learning robust perceptive locomotion for quadrupedal robots in the wild'. In: *Science Robotics* 7.62 (Jan. 2022). ISSN: 2470-9476. DOI: 10.1126/scirobotics.abk2822. URL: http://dx.doi.org/10.1126/scirobotics.abk2822.

[5] Ruihan Yang et al. *Learning Vision-Guided Quadrupedal Locomotion End-to-End with Cross-Modal Transformers*. 2022. arXiv: 2107.03996 [cs.LG].

[6] Ananye Agarwal et al. *Legged Locomotion in Challenging Terrains using Egocentric Vision*. 2022. arXiv: 2211.07638 [cs.RO].

[7] Nikita Rudin et al. *Advanced Skills by Learning Locomotion and Local Navigation End-to-End*. 2022. arXiv: 2209.12827 [cs.RO].

[8] Viktor Makoviychuk et al. *Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning*. 2021. arXiv: 2108.10470 [cs.RO].

[9] Nikita Rudin et al. *Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning*. 2022. arXiv: 2109.11978 [cs.RO].

[10] Mayank Mittal et al. 'Orbit: A Unified Simulation Framework for Interactive Robot Learning Environments'. In: *IEEE Robotics and Automation Letters* 8.6 (2023), pp. 3740–3747. DOI: 10.1109/LRA.2023.3270034.

[11] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[12] Zipeng Fu et al. *Minimizing Energy Consumption Leads to the Emergence of Gaits in Legged Robots*. 2021. arXiv: 2111.01674 [cs.RO].

[13] Nicolas Heess et al. *Emergence of Locomotion Behaviours in Rich Environments*. 2017. arXiv: 1707.02286 [cs.AI].

[14] Stephane Ross, Geoffrey J. Gordon and J. Andrew Bagnell. *A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning*. 2011. arXiv: 1011.0686 [cs.LG].

[15] Ashish Kumar et al. *RMA: Rapid Motor Adaptation for Legged Robots*. 2021. arXiv: 2107.04034 [cs.LG].

[16] Guillaume Bellegarda et al. 'Robust High-Speed Running for Quadruped Robots via Deep Reinforcement Learning'. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2022. DOI: 10.1109/iros47612.2022.9982132. URL: http://dx.doi.org/10.1109/IROS47612.2022.9982132.

[17] Joonho Lee et al. 'Learning quadrupedal locomotion over challenging terrain'. In: *Science Robotics* 5.47 (Oct. 2020). ISSN: 2470-9476. DOI: 10.1126/scirobotics.abc5986. URL: http://dx.doi.org/10.1126/scirobotics.abc5986.

[18] Russ Tedrake. *Underactuated Robotics. Algorithms for Walking, Running, Swimming, Flying, and Manipulation*. 2023. Chap. 10. URL: https://underactuated.mit.edu/trajopt.html.

[19] Dian Chen et al. *Learning by Cheating*. 2019. arXiv: 1912.12294 [cs.RO].

[20] Denys Makoviichuk and Viktor Makoviychuk. *rl-games: A High-performance Framework for Reinforcement Learning*. https://github.com/Denys88/rl_games. May 2021.

[21] Jun Yamada et al. *TWIST: Teacher-Student World Model Distillation for Efficient Sim-to-Real Transfer*. 2023. arXiv: 2311.03622 [cs.RO].

[22] Jonah Siekmann et al. *Blind Bipedal Stair Traversal via Sim-to-Real Reinforcement Learning*. 2021. arXiv: 2105.08328 [cs.RO].

[23] Samarth Sinha et al. *D2RL: Deep Dense Architectures in Reinforcement Learning*. 2020. arXiv: 2010.09163 [cs.LG].

[24] Guillaume Bellegarda, Milad Shafiee and Auke Ijspeert. *Visual CPG-RL: Learning Central Pattern Generators for Visually-Guided Quadruped Locomotion*. 2024. arXiv: 2212.14400 [cs.RO].

[25] Ruihan Yang, Ge Yang and Xiaolong Wang. *Neural Volumetric Memory for Visual Locomotion Control*. 2023. arXiv: 2304.01201 [cs.RO].

[26] Chieko Sarah Imai et al. *Vision-Guided Quadrupedal Locomotion in the Wild with Multi-Modal Delay Randomization*. 2022. arXiv: 2109.14549 [cs.RO].

[27] Xuxin Cheng, Ashish Kumar and Deepak Pathak. *Legs as Manipulator: Pushing Quadrupedal Agility Beyond Locomotion*. 2023. arXiv: 2303.11330 [cs.RO].

[28] Simar Kareer et al. *ViNL: Visual Navigation and Locomotion Over Obstacles*. 2023. arXiv: 2210.14791 [cs.RO].