Toward Realistic Locomotion: Domain and Goal Randomization for Quadruped Policy Learning

Romain Emanuele Salvi
DAUIN
Politecnico di Torino
Turin, Italy
s339304@studenti.polito.it

Mattia Sernia
DAUIN
Politecnico di Torino
Turin, Italy
s339829@studenti.polito.it

Riccardo Vittimberga
DAUIN
Politecnico di Torino
Turin, Italy
s344810@studenti.polito.it

Giorgio Zoccatelli
DAUIN
Politecnico di Torino
Turin, Italy
s349395@studenti.polito.it

***Abstract*—This paper presents a study that begins with a basic locomotion environment and explores a possible evolution through environment redesign, new tasks, and reward function shaping. The goal is to extend reinforcement learning applicability by transitioning from a simple agent-environment setup to a more complex, realistic scenario.**

The new disconnected environment features a quadruped tasked with reaching a fixed or randomly chosen target, incorporating goal randomization. Additional complexity includes mass randomization of the quadruped's limbs, simulating real-world variability and enhancing policy robustness. This work examines how terrain slope, four-limb use, indirect paths, and physical changes affect performance.

Source code is available at this *link* and this *link*.

I. Introduction

Reinforcement Learning (RL) is a machine learning paradigm where an agent learns to make decisions through interaction with an environment, receiving rewards or penalties as feedback. Its goal is to discover a policy that maximizes long-term reward by balancing exploration and exploitation. RL is especially suited for robotics, where agents must learn robust, adaptive behavior in dynamic and uncertain conditions.

This work addresses progressively more challenging episodic control tasks, structured as follows:

- Understanding and testing baseline policy optimization methods, such as REINFORCE (Vanilla Policy Gradient), Actor-Critic (AC), Proximal Policy Optimization (PPO), and Uniform Domain Randomization (UDR). All experiments in this stage are conducted in the MuJoCo-Py environment, with a single Hopper robot as the controlled agent.
- Applying the most performing algorithm to a quadruped robot operating on disconnected terrain, significantly increasing both agent and environment complexity.
- Introducing random target points, requiring the quadruped to generalize its policy to variable goal locations and adapt its motion accordingly.
- Varying the mass distribution of the quadruped, limited to scenarios where the target point remains fixed, with the aim of improving policy robustness to physical variations such as manufacturing tolerances or payload changes.

II. Related Work

A. REINFORCE and Actor Critic

1) REINFORCE: The first model trained in the Hopper environment uses one of the simplest reinforcement learning algorithms: REINFORCE, a Vanilla Policy Gradient (VPG) method based on Monte Carlo estimation. However, Monte Carlo estimators typically have very high variance, which makes the direction of the updates to the policy noisy and unreliable.

So, measures were taken to ensure the agent was learning steadily and not getting stuck in local optima. Normalizing the advantages [1] and the use of batching improves both learning stability and performance. [2].

Furthermore, an entropy regularization is introduced in the policy loss formula. This encourages the agent to remain uncertain and explore more overlooked possibilities. The entropy coefficient was selected based on a grid search over a range of values.

2) Actor Critic: Another significant algorithm in RL is Actor-Critic. This approach involves two neural networks:

- The "Actor", which manages the policy that the agent, in our case a hopper, will follow at each step.
- The "Critic", which updates the Actor's neural network based on the advantage of the selected action.

Unlike the REINFORCE algorithm, which updates weights only at the end of the episode, both the Actor and the Critic weights are updated at each step. This should lead to faster convergence speed and to more satisfying results.

Although this algorithm improves upon the VPG method, it can still suffer from high instability in critic and policy losses. To mitigate these issues, Mean Squared Error (MSE) loss was used for the critic, effectively penalizing large deviations between predicted and target values to enhance value estimation accuracy. Learning rates were also fine-tuned to improve training stability. Softplus was chosen over tanh-based activations to avoid saturation and vanishing gradients.

Initial experiments with three-layer networks for both Actor and Critic validated the update mechanism. The subsequent adoption of deeper architectures led to significant improvements in reward magnitude and convergence speed, at the cost of increased computational complexity.

B. PPO and Uniform Domain Randomization

Proximal Policy Optimization (PPO) is a policy gradient method designed to improve training stability by limiting the deviation between the new and the old policies during updates. This is achieved through a clipped surrogate objective that discourages large policy updates, enabling reliable performance without the need for complex optimization constraints [3].

In this project, the PPO-Clip variant was employed using the `stable-baselines3` library, which includes a clipped objective with $\varepsilon = 0.2$, an Adam optimizer, and a fixed learning rate [4]. Uniform Domain Randomization (UDR) was implemented on the link masses of the Hopper robot. The values of the three masses in the source environment were sampled over a uniform distribution, excluding the torso, whose mass is fixed at -30% with respect to the target environment.

The goal is to train the agent to perform well across a range of environments, ensuring its ability to generalize. By exposing it to varied dynamics, its behavior becomes robust to changes, as is typical in real-world scenarios.

C. Test results comparison

The results across the various algorithms are satisfactory and align with expectations. Figure 1 shows the rewards achieved by agents trained with three different algorithms over 500 test runs. To improve readability and reduce clutter, a moving average was applied to the data. The variance-reduction techniques applied to the REINFORCE algorithm achieved the intended effect: its results are notably consistent.

AC outperformed the REINFORCE algorithm by using a value function (the Critic) to reduce variance in the policy gradient estimate. Despite this, it did not perform as well as PPO due to the lack of stable and constrained policy updates. Training on more episodes reduced variance, but the dense network made it computationally costly. While rewards surpassed REINFORCE, this came with greater variance.

To evaluate the performance of the highest-performing algorithm, PPO, UDR was applied. The test rewards in this source environment were relatively high and tightly clustered, indicating low variance and suggesting that the policy effectively adapted to mass randomization. The agent was initially trained in the source environment. When tested in the target environment, where only the torso mass was shifted by 30%, the rewards demonstrate that while the learned policy generalizes reasonably well, its performance is not fully robust to unseen mass configurations.

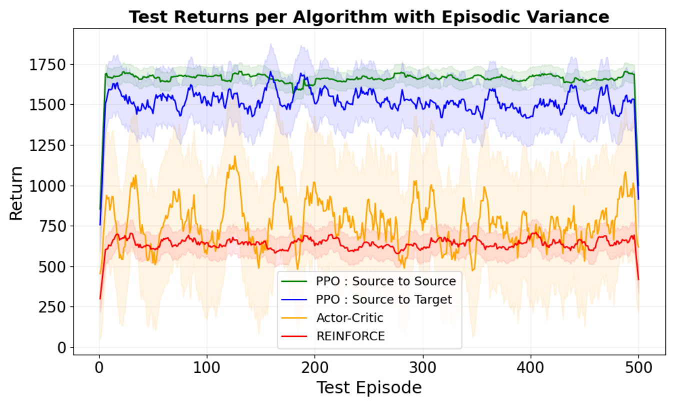

Fig. 1. Test Returns per Algorithm with Episodic Variance

III. The Quadruped Environment

A. Introduction

After learning the essential concepts of RL in the simple Hopper environment, where the single-legged agent allows focusing almost exclusively on stability and control, the focus of the research shifted to a setting that better simulates real-world needs.

Consequently, this report focuses on the task of teaching a quadruped agent to reach a predefined point in a mountainous environment. This task has several real-world applications, such as a rescue robot performing mountain search and rescue operations. The transition from the Hopper to a quadruped introduces a new set of challenges: maintaining equilibrium now depends on the interaction of four limbs. Furthermore, relocating the agent from a flat environment to a mountainous one increases the complexity of the task, since the agent needs to learn how to face different scenarios.

Since building such an environment from scratch is beyond the scope of the project, the environment available in the *quadruped-rl-locomotion* repository was adopted as the basis for the work.

B. Base Environment

The starting environment of the project was a custom Gymnasium environment, featuring a quadruped agent moving through a simulated infinite flat terrain.

The quadruped consists of a central trunk and four identical legs. Each leg has 3 degrees of freedom: abduction/adduction, hip flexion/extension, and knee flexion/extension. Its structure is defined in a custom MuJoCo XML model, including physical properties such as dimensions, masses, inertia, joint limitations, and contact parameters. The model also includes other detailed parameters such as joint damping, friction loss, and actuator strength. Table III-B shows the masses and the geometrical properties of representative components.

The robot is controlled using a torque-based actuation system. Each of its 12 joints is driven by an actuator that receives a torque signal, which is later computed by the control policy. All relevant information about the agent and its components, including joint and link positions, velocities, and angular velocities, is provided to the optimizer. This setup allows the agent to perform realistic locomotion behaviors under physically plausible dynamics.

Component	Mass (kg)	Geom Type	Geom Size (m)
Trunk	5.204	Box	0.125 × 0.04 × 0.057
Hip	0.680	Cylinder	0.046 × 0.02
Thigh	1.009	Capsule	0.015
Calf	0.196	Capsule	0.01

TABLE I
MASS AND SHAPE OF EACH COMPONENT

C. Environment Redesign

As mentioned in Section III-B, the environment simulates an infinite, flat terrain. However, the goal of the project was to train a quadruped agent and evaluate its policies on an uneven mountainous terrain. This constitutes a typical sim-to-sim scenario: to enable eventual sim-to-real transfer, control policies must generalize across diverse simulated environments. To overcome the limitations of the base environment, it was decided to simulate a real-world location. Given the small physical dimensions of the quadruped agent, it was not feasible to source a real-world height map at a matching scale. Instead, a region of an actual mountain range was selected and converted into a height map to approximate a realistic terrain.

The full updated map spans from -40 to +40 meters in both directions, length and width. The elevation values across the map have been scaled so that the highest point reaches exactly 6.3 meters. This value was arbitrarily chosen to limit slope steepness, ensuring that the quadruped did not encounter terrain beyond its climbing capabilities. As the initial environment did not include a map with elevation data, there was no need to retrieve the height of a point from its (X, Y) coordinates. The geographic data and the height map were obtained from a height map generation tool [5]. The (X, Y) coordinates of each point on the map were mapped to a height value.

Finally, the image was recolored using a color map that simulates a barren, mountainous terrain. This was implemented for aesthetic reasons and also to aid visual analysis of the results, adding depth to an otherwise flat, all-white background. Figure 2 is a sample frame collected from the rendering of the newly designed environment. The red pole shows the position of the goal, while the quadruped stands at the center of the map.

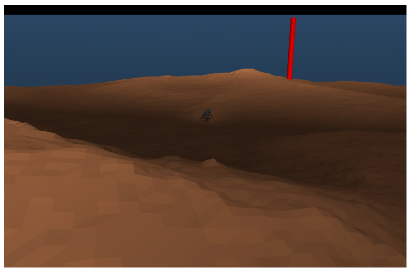

Fig. 2. Sample frame from the redesigned environment.

D. The Model

The training procedure used in the initial project was studied to teach the quadruped to move forward at a fast pace. This was achieved using a PPO algorithm implemented through the `stable-baselines3` library. To accelerate learning and improve efficiency, multiple instances of the environment were launched in parallel. This allowed the agent to gather experience from several episodes simultaneously. Together with PPO the model uses a multilayer perceptron policy, which can be used to fine-tune an existing policy or to create one from scratch. Since the existing training pipeline proved to be effective for the previous task, it was also applied to the new project. Table II shows the parameters used in most training runs:

Parameter	Value
Policy	MlpPolicy
Learning Rate	0.0003
Number of Environments	12
Total Time steps	10,000,000
Optimizer	PPO

TABLE II
DEFAULT TRAINING PARAMETERS USED IN THE PPO MODEL

IV. PROJECT EXTENSION

A. Task description

The main objective of this work is to train a quadrupedal robot to walk toward a designated goal, initially fixed and later randomized, across varying terrain conditions. The training procedure is structured progressively, beginning with locomotion on uneven terrain while keeping the target position fixed. This allows the agent to develop a stable and reliable walking gait that can handle terrain variations before introducing additional complexity, such as randomized goal positions. To evaluate robustness, physical perturbations were introduced by randomizing the robot's mass distribution within plausible bounds, but only during the phase where the goal position remained fixed. In this work, goal randomization was also applied by varying the direction in which the robot was required to walk, which also meant that the goal could lie uphill, downhill, or on flat terrain. This required the robot to adapt its gait and control strategy to the slope it encountered, improving its robustness across a range of walking conditions.

B. Observation Space, Termination Conditions and Reward Function Shaping

1) Observation Space: The observation space is crucial in RL, as it defines the information an agent receives from the environment, directly impacting its ability to learn and to make effective decisions. It provides the agent with information that helps it understand the current state of the environment and make informed decisions. The observations provided to the step function only include the most relevant features that are necessary for the agent to learn the task. A large observation space with irrelevant components can slow down learning, while a space with too little information may prevent the agent from learning the optimal policy. These are the components that were included in the space:

- The relative position of the joints with respect to their initial positions. This information tells the agent, at a given step, how much its joints differ from their default configuration.
- The angular velocity of the joints. A higher value means the quadruped is moving the limb faster.
- The previous action performed by the agent, giving insight into recent choices and movements of the agent.
- The velocity of the robot projected onto the direction of the goal. This is computed by first finding the direction from the robot to the goal, defined in the (X, Y) plane, then measuring how much of the robot's current planar velocity aligns with that direction. It allows the agent to understand whether it is moving effectively toward the target at each step. The result is then divided by the maximum possible distance to the goal to normalize the value, making the reward scale-invariant and aiding learning regardless of the starting distance.
- The relative direction of the quadruped with respect to the goal. It is defined as the angle (in radians) between two unit vectors: the agent's forward direction and the vector to the goal. Early experiments showed that without heading awareness, the agent struggled with orientation, often falling or failing to link similarly aligned goals. Replacing Cartesian goal coordinates with polar ones, including the relative direction angle, improved generalization by enabling direction-based strategies.
  Generalization was further improved by reducing the problem to two dimensions, computing relative direction solely in the (X, Y) plane, as the z-coordinate is terrain-dependent. Including the goal's height was avoided, since a low z-value may still correspond to a difficult path, potentially misleading the agent.
- To overcome this lack of information about slope and z-coordinates, a matrix representing the relative elevation of the terrain directly ahead of the quadruped is included as well. This serves as a simplified model of a partial and limited-range sensor, mimicking what a real quadruped robot might perceive through onboard sensing in a real-world scenario. Such information is essential in an uneven, mountainous terrain characterized by slopes and steps such as the one described in Section III-C. It enables the agent to detect vertical and horizontal discontinuities, allowing it to anticipate and adapt to various terrain configurations.
  The sampled terrain spans 1 meter forward and 60 centimeters laterally, providing a focused yet relevant local map. The field of view is restricted, but it is enough to capture the most immediately actionable portion of the environment.

Figure 3 qualitatively illustrates the goal vector, the direction vector, and the height information matrix. It must be noted that the observation space represents only a subset of the complete state; for example, the height grid covers just the immediate visible terrain rather than the entire map. Consequently, the agent must learn to infer the broader state of the environment from these limited and local observations.

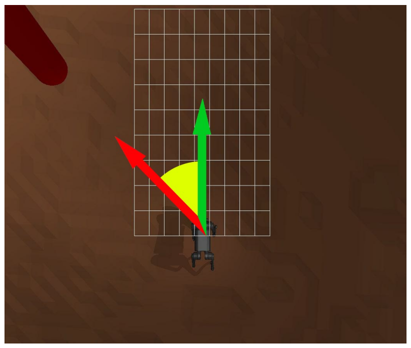

Fig. 3. Illustration of Relative Direction and the Slope Matrix

2) Termination Conditions: Episode termination conditions define when a single trial or interaction sequence between the agent and its environment ends. After each episode, the agent is reset to its initial state with added noise to introduce variability. This prevents the agent from always starting in the same position, forcing it to generalize across different scenarios. Some termination conditions used in the project were inherited from the original repository. These conditions included a numerical check to ensure the robot's state remained within valid bounds, and limits on the torso's roll and pitch to keep them within a predefined healthy range. These constraints effectively terminate the episode if the robot falls—sideways or forward/backward. Yaw was originally part of the termination criteria but was later removed, as the quadruped needed the flexibility to turn toward targets that were not directly in front, fixed or randomly placed. The three rotation directions are shown in Figure 4.

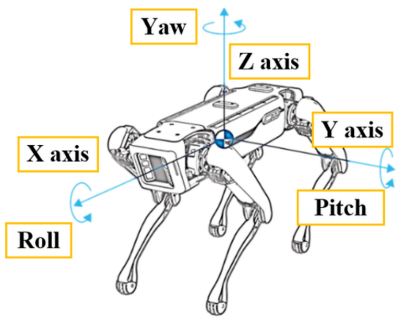

Fig. 4. Illustration of pitch, roll, and yaw of the quadruped robotic structure (adapted from [6]).

Since the objective of this work is to train the agent to walk toward a specified goal, two primary scenarios must be handled: one in which the agent successfully reaches the goal, and another in which the agent fails to make meaningful progress over an extended period. The first is straightforward to detect, as it involves checking whether the agent's distance to the target falls below a predefined threshold. The latter case necessitates an early termination condition, introduced to avoid wasting training time on ineffective or stagnating trajectories. At any step, if the reduction in distance to the goal over the previous 100 steps is less than 20 centimeters, the episode is terminated.

Imposing these conditions prevents reinforcement of ineffective policies and accelerates convergence by directing learning toward more relevant, goal-oriented behavior.

3) Reward Function Shaping: In RL, a reward function is a crucial component that guides an agent's learning process by providing feedback on its actions. The reward should be smooth, consistent, and ideally scale-invariant across different scenarios to support stable learning. These are the components that were included in the reward function:

- The progress computed using the old distance from the goal and the new one. It is a key parameter since it measures the agent's progress in reducing its distance to the goal at each step. This is computed as the difference between the Euclidean distance from the previous position to the target and the distance from the current position to the target.
  In a complex, mountainous terrain where the path to the goal is not always straight, this progress-based reward helps the agent focus on moving in the right direction instead of just trying to stay balanced or walking without a clear purpose.
- The orientation of the agent with respect to the objective point. At every timestep, this parameter is updated using the relative direction described in Section IV-B1.
  The orientation term must encourage both turning toward the goal and holding the correct direction once the robot is aligned. To achieve this, the term is composed by two contribution: the current alignment, computed as $\cos\theta$, and the rotation, computed as $\cos\theta_t - \cos\theta_{t-1}$. When the agent is already aligned with the goal, the direction term approaches 1 ($\cos\theta \xrightarrow[\theta\to 0]{} 1$). Both components are weighted equally, a choice validated by preliminary experiments.
- The velocity component aligned with the direction of the goal. At each step, the agent's velocity in the plain is projected onto the vector pointing from the agent's current position to the target.
  This measures how much of the agent's movement is actually helping it advance toward the goal. The normalization ensures that only the direction matters, not the distance to the goal. Beside, a small constant is added to the denominator during normalization to avoid numerical instability or division by zero if the agent is extremely close to the goal.
  Including this term in the reward encourages the agent not just to move, but to move efficiently in the right direction. This is especially important in mountainous terrain, where poorly directed movements can waste time or increase the risk of falling.
- To address a common issue in RL, namely, optimizing the agent for short-term rewards at the expense of long-term objectives, a survival-based reward component was introduced.
  Initially, the reward function prioritized goal-directed progress and time efficiency, measured by the velocity towards the goal. However, this led the agent to adopt reckless strategies, such as leaping toward the goal without regard for survival, resulting in unstable landings and frequent early terminations.
  To mitigate this behavior, a small positive reward was added at each timestep in which the agent remained "alive". Combined with a penalty applied when an episode ends before the agent has reached the objective, this modification incentivizes the agent to balance goal-seeking behavior with episode longevity.
- Episode-termination rewards were included to address two critical cases. The survival-oriented reward logic successfully encouraged the agent to avoid falling. However, it introduced a new failure mode: the agent learned to stand still indefinitely to accumulate stable rewards and avoid the risk of falling due to movement.
  To counter this, a substantial penalty was imposed for terminations caused by prolonged inactivity. This kind of termination is detailed in Section IV-B2. The penalty was deliberately set to exceed the cumulative reward obtainable from standing still or making no meaningful progress over 100 episodes.
  The final component of the reward function is a substantial bonus awarded when the agent successfully reaches the goal. This approach aligns with common RL practice, where achieving the primary objective is incentivized with a significant terminal reward.

In the reward function, weights are carefully designed to guide the learning process toward the desired behavior. When the target is fixed, greater emphasis is placed on forward progress and time efficiency, particularly when the agent is already well oriented toward the goal (i.e., when the relative direction is small).

Once the angle created by the direction faced by the quadruped and the actual direction to the goal lies under a specific threshold, its primary objective should be reaching the target quickly and smoothly, as the travel direction is stable and predictable. In this case the quadruped focuses more on efficient locomotion and purposeful motion rather than spending effort on continuous reorientation. Additional small rewards for joint motion and velocity are included to encourage continuous movement and prevent the agent from stopping or hesitating unnecessarily.

Although the overall reward structure remains consistent with that of the fixed goal setting, randomized target positions increase the importance of accurate and timely orientation adjustment. Given that the agent always begins in the same orientation but the target is randomly located, the agent must initially reorient itself toward the goal before effective forward motion is possible.

In this setting, the orientation reward maintains a substantial weight even when alignment is nearly optimal, promoting continuous refinement of the agent's heading. While progress and velocity toward the goal remain important, the relative emphasis on orientation is increased to foster robust reorientation strategies under dynamically changing target positions. This careful reward shaping accelerates learning and leads to more robust and generalizable locomotion policies.

V. Experiments and Results

A. Domain and Goal Randomization

The main goal of the project is to develop an agent that can operate effectively in simulated real-world conditions. Collecting real-world data is both costly and time-consuming, so the policy is trained entirely in simulation. However, the reality gap between simulation and the physical world often causes such policies to underperform when applied in real-world scenarios. To mitigate the sim-to-real gap, the robustness of the model must be increased. The strategies chosen are Domain Randomization and Goal Randomization.

Exploiting these techniques allows to generate a wide range of simulated environments with randomized parameters and obtain a policy capable to generalize on new, unseen cases. Two distinct training conditions are considered:

- During training, the agent is not informed of the robot's actual mass at test time, necessitating policies that are robust to variations in physical dynamics.
- Goal randomization during training is introduced to ensure generalization, as the agent will not have prior knowledge of the target position during testing

1) Mass Randomization: In real-world applications, the exact masses of individual robot components are rarely known with absolute precision. To make the control policy robust to such variation, training is performed on a source domain. More specifically, the mass of each component was scaled by a random multiplicative factor sampled from a Uniform distribution, $f \sim \mathcal{U}(0.8, 1.2)$. This allows the masses to vary up to 40% of their nominal weights.

Training is performed in the source domain, where key parameters are randomized, while deployment occurs in the target domain, whose parameters remain fixed. Uniform Domain Randomization is applied, where each parameter α_i is bounded to an interval and sampled uniformly within the range [7].

Since the robot is assumed to remain geometrically similar under this perturbation, other physical properties are adjusted. The mass moment of inertia must be multiplied by $f^{\frac{5}{3}}$ since $I \propto m^{\frac{5}{3}}$. Also, the actuator gear must be scaled accordingly, this is due to compensate for the different inertia load.

After each training episode, all the perturbed parameters are reset to their nominal values. This prevents the learning algorithm from overfitting any particular combination of parameters and promotes generalization to unseen physical configurations.

The resulting mass intervals for each individual component are summarized in Table III.

Component	Standard Mass	Min Mass	Max Mass
Trunk	5.204	4.1632	6.2448
Hip	0.680	0.544	0.816
Thigh	1.009	0.8072	1.2108
Calf	0.196	0.2352	0.1568

TABLE III
Mass Ranges

2) Goal Randomization: This technique involves randomizing the goal at the start of each episode by sampling it from a defined goal space. Rather than training the agent to reach a fixed point, the goal location is drawn from a probability distribution at each episode, requiring the agent to generalize across a range of possible targets.

In this setup, goal positions are sampled uniformly from a square region measuring 10 meters per side, with the quadruped positioned at the center. This only applies to the (X, Y) coordinates of the goal, as the z-coordinate is implicitly determined once the X and Y positions are fixed. A uniform distribution was chosen to prevent the agent from overfitting to specific goal locations; in contrast, a normal distribution would bias training toward nearby targets and reduce the diversity of behaviors learned.

To implement this behavior, the agent was provided with explicit information about the goal and its relative configuration. With this inclusion, the setup falls under Goal-Conditioned Reinforcement Learning (GCRL). As detailed in Section IV-B1, the policy receives observations encoding the robot's position relative to the goal, expressed in polar coordinates: the distance and the direction to the target. This representation is also applied to the robot's velocity relative to the goal, allowing the policy to reason about motion in goal-centric terms. [8]

B. Weights Fine-Tuning

Recalling that the general shape of a reward function has the form

$$r(s, a) = w_1 \cdot r_1(s, a) + w_2 \cdot r_2(s, a) + \cdots + w_n \cdot r_n(s, a)$$

and given the highly stochastic nature of our application (as per almost every RL implementation), a grid search strategy is adopted to explore a predefined discrete set of values. For each weight w_i referred to as a different reward contribution as exploited in Section IV-B3. For each candidate weight configuration, the agent was trained on a fixed number of environment steps, and its performance was evaluated across a suite of validation scenarios. The evaluation metrics included the distance traveled in an episode, average episode length, and average episode reward.

The model's performance is trained and tested on different setups, partitioned into four groups based on their values: similar configurations are put in the same group. These were named based on what behavior they would promote the most. The weights used for the manual grid search are shown in Table IV. The table distinguishes between two phases: the reward weights applied before the agent has properly oriented toward the goal ('pre') and those applied afterward ('post').

Name	Weights pre Orientation	Weights post Orientation
Progress	$[0.5, 1, 1.5, 3]$	$[1, 3, 4, 5]$
Orientation	$[0.5, 1.5, 2, 3]$	$[0.5, 1, 1.5, 2, 3, 4]$
Time Eff	$[0.5, 1, 2]$	$[1.5, 2, 3, 4]$
Survival	$[0.5, 1, 2]$	$[0.5, 1, 2]$
Death	$[0.5, 1, 2]$	$[0.5, 1, 2]$

TABLE IV
WEIGHT RANGES

The optimal weight configurations were determined by their capacity to generalize effectively across the fixed and randomized goal cases, exhibiting robust and interpretable behavior. In both fixed and randomized goal settings, all selected weight configurations were trained for a limited number of time steps to enable early identification of the most promising setups, a process known as early pruning. Evaluation was not based on absolute reward values, since these vary with different weight scales, but rather on the distance traveled by the agent. The results explained in the following Table V acknowledge the so-called dynamic configuration as the highest-performing one.

Name	Distance traveled
Efficient	0.8633
Dynamic	5.9317
Cautious	0.8707
Balanced	1.2924

TABLE V
BEST MODEL FROM EACH CONFIGURATION BASED ON THE DISTANCE TRAVELED (MAX 8 METERS)

For the randomized goal scenario, resulting models were assessed on eight equidistant points positioned at 45° intervals around a circle centered on the robot's initial position. Within each group of weight configurations, the best-performing model was selected for comparison against the others.

Figure 5 reports the distance traveled by the robot prior to episode termination. It refers to the random goal case. The x-axis labels denote the corresponding weight setups. Among the tested models, the Dynamic configuration demonstrated the strongest performance by a significant margin. Despite exhibiting high variance, it achieved the highest mean distance and maximum value across trials. Based on these outcomes, this configuration was selected for subsequent use.

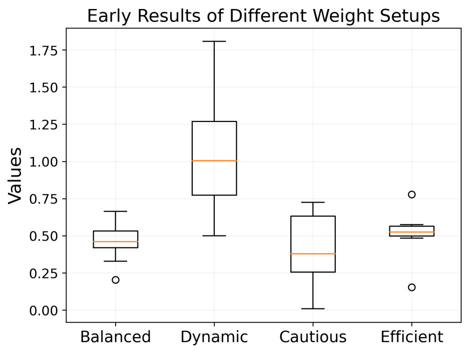

Fig. 5. Early Results of best model for each configuration type, randomized goal case.

Table VI reports the weight configuration that achieved the best performance in both scenarios. In the post-orientation phase, weights associated with progress are increased, reflecting the shift in priority toward advancing efficiently toward the target once correct alignment has been achieved.

Name	Selected Weight pre	Selected Weight post
Progress	3	5
Orientation	0.5	0.5
Time Eff	0.5	1.5
Survival	0.5	0.5
Death	2	2

TABLE VI
SELECTED WEIGHTS

C. Results

The experiments were conducted over several millions of time steps, with the most relevant results presented in Table VII. The evaluation metrics were carefully chosen to capture different aspects of the agent's performance. These include episode length (indicative of the time required to reach the target), cumulative reward (reflecting the internal logic of the reward function), success rate (percentage of goals reached), and total training time steps.

Under the fixed-goal scenario, the quadruped reached the target in fewer time steps with fixed masses compared to randomized ones, demonstrating more efficient behavior. Nonetheless, the success rate was 100% for the fixed target point with both fixed and randomized masses. The point was set arbitrarily at (10,10), which was selected after verifying consistent performance across multiple positions.

The final phase of training involved navigating to randomly sampled target positions. To support this more complex scenario, the number of training time steps was significantly increased. This proved to be the most challenging setup, as the goal changed with every episode, demanding strong generalization from the policy.

Performance evaluations, conducted over 500 distinct goal locations, revealed satisfactory results for nearby targets: the robot reliably reached its destination, exhibiting gait patterns comparable to those observed in fixed-point experiments. However, performance deteriorated for more distant targets. In many cases, the quadruped advanced partway toward the goal but failed to maintain balance or trajectory, yielding a success rate of only 24%. The average reward was correspondingly lower, given that significant reward was only granted upon successful arrival at the target.

VI. DISCUSSION AND CONCLUSIONS

This project presents a progression from a basic hopper to a quadruped robot capable of traversing mountainous terrain, with adaptability to both dynamic goal positions and variations in body mass distribution. The development followed a gradual approach: first, a selection of an appropriate environment, then the design of an informative observation space, followed by the construction of a reward function capable of guiding the learning process through increasingly complex tasks.

The observation space combines internal state information (such as joint positions, velocities, and previous actions) with an external representation of the terrain. Since no physical sensors were used, a grid-based sampling of the terrain in front of the robot was introduced. This allowed the agent to "perceive" the environment and anticipate slope changes.

The reward function was carefully constructed to reflect multiple aspects of successful behavior.

It rewarded forward progress and penalized misalignment with the target direction, while encouraging time-efficient movement and penalizing instability or failure. Different reward weightings were used, depending on the robot's alignment with the goal. This allowed a gradual transition from rough guidance to fine control as the robot approached the target. Additional components, such as survival bonuses, helped refine the behavior over time.

Using this setup, policies were successfully trained to walk in mountainous terrain, first toward a fixed point, then with varying mass configurations, and finally toward random goal locations. Each new challenge required only minor modifications to the system, showing that the architecture was flexible and generalizable.

Subsequently to a manual grid search procedure, the best configurations were identified for the fixed target with randomized masses and random target settings.

Test	**Ep. length**	**Ep. Reward**	**Reached (%)**	**Time steps**
fixed masses	245	1415.9417	1.00	$2 * 10^7$
rand masses	251	2509.0007	1.00	$2 * 10^7$
rand points	$175(avg.)$	$346(avg.)$	$0.24(avg.)$	$6 * 10^7$

TABLE VII
BEST CONFIGURATION RESULTS FOR FIXED POINT, RANDOMIZED MASSES AND RANDOMIZED POINT

As previously discussed, the fixed-mass scenario leads to more efficient goal-reaching behavior, although the randomized-mass setup yields higher cumulative rewards due to longer episode durations. A potential improvement for both settings involves introducing a regularization penalty on leg abduction to discourage excessive outward splaying, which may promote a more natural, stable, and energy-efficient gait.

The robot, when tasked with reaching random points, demonstrated reliable performance for nearby targets but struggled as target distance increased. The average reward remained low, as successful goal completion was infrequent at longer ranges. A promising direction to address this limitation is to gradually increase target distances throughout training. This approach, known as curriculum learning, would allow the robot to first master short-range navigation and progressively adapt to more complex, long-distance tasks. By structuring the learning process incrementally, the agent is more likely to develop robust policies capable of handling a wider range of goal placements.

Further extensions and refinements to the reward function could significantly improve the realism and generalization ability of the learned behavior. Possible improvements involve the incorporation of periodic motion constraints to promote more regular and natural locomotion patterns [9], and the enforcement of a steady, fluid speed constrained within a safe operational range.

Another important step would be to jointly train the model under varying mass conditions and random goal placements. In the current setup, these two sources of variation were addressed separately.

Additionally, adding obstacles or dynamic elements to the environment would push the robot toward true autonomy. This would require expanding the observation space (e.g., with obstacle detection) and introducing new reward components related to collision avoidance or efficient detouring [10].

Given sufficient computational resources, supplemental improvements could also be achieved by increasing the number of training episodes and employing more complex neural network architectures.

These directions could help reduce the gap between simulation and real-world deployment, supporting the development of quadruped robots capable of adapting to complex environments and unpredictable tasks.

REFERENCES

[1] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347v2*, 2017, https://arxiv.org/pdf/1707.06347.

[2] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," *arXiv preprint arXiv:1502.05477v5*, 2017, https://arxiv.org/pdf/1502.05477.

[3] OpenAI, "Proximal policy optimization," 2018, https://spinningup.openai.com/en/latest/algorithms/ppo.html#proximal-policy-optimization.

[4] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021, http://jmlr.org/papers/v22/20-1364.html.

[5] MantiCorp, "Unreal heightmap generator," https://manticorp.github.io/unrealheightmap/.

[6] V. Stenchlák, M. Císar, and I. Kuric, "Inclination stability controller for spot micro robot based on artificial neural networks," 2022.

[7] L. Weng, "Domain randomization for sim2real transfer," *lilianweng.github.io*, 2019. [Online]. Available: https://lilianweng.github.io/posts/2019-05-05-domain-randomization/

[8] D. Jain, A. Iscen, K. Caluwaerts, J. Tan, T. Zhang, and V. Sindhwani, "Hierarchical reinforcement learning for precise motor control," *arXiv preprint arXiv:1905.08926*, 2019. [Online]. Available: https://arxiv.org/pdf/1905.08926

[9] Z. Zhang, W. Li, Y. Yang, P. Liu, W. Huang, and Y. Chua, "Residual-guided curriculum learning for robust locomotion of quadruped robots," *IEEE Transactions on Robotics*, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10606552

[10] T. Liu, Z. Tang, Z. Zhou, Y. Zhu, X. Ma, J. Liu, P. Wang, and D. He, "Robust terrain-aware locomotion control for quadruped robots via terrain curriculum and sub-policy ensemble," *arXiv preprint arXiv:2404.15096*, 2024. [Online]. Available: https://arxiv.org/abs/2404.15096